MULTI-TARGET CLASSIFICATION AND REGRESSION IN WINEINFORMATICS

by

James Palmer

A thesis presented to the Department of Computer Science
and the Graduate School of the University of Central Arkansas in partial
fulfillment of the requirements for the degree of

Master of Science
in
Applied Computing

Conway, Arkansas
May 2018

TO THE OFFICE OF GRADUATE STUDIES:

The members of the committee approve the thesis of

James Palmer presented on Apr 5, 2018.

Bernard Chen, Committee Chairperson

Victor Sheng

Victor Sheng

Sinan Kockara

PERMISSION

Title: Multi-Target Classification and Regression in Wineinformatics

Department: Computer Science

Degree: Applied Computing

In presenting this thesis in partial fulfillment of the requirements for a graduate degree from the University of Central Arkansas, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by the professor who supervised my thesis work, or, in the professor's absence, by the Chair of the Department or the Dean of the Graduate School. It is understood that due recognition shall be given to me and to the University of Central Arkansas in any scholarly use which may be made of any material in my thesis.

James Palmer

April 17, 2018

Date

Abstract

Wineinformatics is the field which uses machine learning and data mining techniques to glean useful information from wine. In this work, attributes extracted from wine reviews are used to make predictions on three key targets/ labels: price per 750mL bottle, quality based on a 100 point scale, and style derived from the region of origin. Using support vector machines and support vector regression, binary and multiclass classification and regression schemes are applied to price and grade, while binary classification is applied to origin. Several different implementations and techniques are explored to achieve good performance, including both single label and multi-label approaches. Additionally, a class imbalance problem is explored and partially mitigated for the grade label. This work is able to achieve binary classification accuracies of 85.9%, 75.89%, and 88.52% on grade, price, and region, respectively; multi-class classification accuracies of 76.21% and 48.71% on grade and price, respectively; and mean absolute error for regression of 1.59 points and $12.94 for grade and price, respectively.

Contents

List of Tables

List of Figures

Chapter 1

Introduction to Wineinformatics and Data Science

Wine is one of the most popular drinks in the world. In the year 2015, over twenty-eight billion liters of wine were produced across 63 countries [1]. Wines are made on every continent on Earth except Antarctica, and they differ in color, aroma, vintage, flavor, aging process, alcohol content, ingredients used, and other attributes. Because there are so many different types of wines, it is a challenging research area to understand what gives wines their unique characteristics, and particularly what leads to characteristics which are more or less desirable. Knowing the effects these characteristics have would be beneficial to all parties in the wine industry: producers, consumers, distributors, critics, and enthusiasts. Producers would be able to better tailor their recipes and vineyards to meet the conditions favorable for high quality wine, distributors would have an easier time getting high quality wine in the hands of consumers as more high quality wine would be available to them, and consumers, critics, and enthusiasts would have better tasting experiences.

Perfecting the wine tasting experience for everyone is not a trivial problem, or it would have been done long ago. A systematic approach is required, one that looks at wine in all its aspects. These aspects can be captured in many forms from many sources: from the critics who taste the wine, from the markets which price the wine, from the regions where the wine is produced, just to name a few examples. If this information is compiled together properly, useful data on wine becomes available for researchers to analyze. Wineinformatics is the field which aims to develop methods to understand this wine data and answer these and other research questions.

Wineinformatics is a fairly new field, enabled due to the advent of computing

technologies and techniques, advances in statistics, and the successes of data science, particularly in data mining, machine learning, and natural language processing. Wineinformatics is not the only field enabled by these advances in technology and knowledge. Bioinformatics is a very popular field that has attracted a significant amount of attention. Wineinformatics can be compared in some ways to bioinformatics: like bioinformatics, wineinformatics is an umbrella term which covers many approaches and topics, with a strong emphasis on statistical methods. Obviously, the subject of research is different, but because they are related, the successes in one field may be applicable in the other. This means that not only does good research in wine improve the experiences for those who enjoy wine, but it may also improve and advance other informatics fields as well. Although wineinformatics occupies just one corner of the fields in data science, data science itself is an emerging branch of research that is powering advances everywhere.

1.1 Data Science

Data science is a very broad field, so much so that it has become a cornerstone in understanding the world. Data science itself is at the intersection of several related fields: artificial intelligence, statistics, business strategy, software engineering, and information science. When it is applied, it is paired with domain knowledge related to the application. This means that data science is applicable to a diverse range of fields: social and political sciences, physics and chemistry, biology and medicine, communication and marketing, business and finance, engineering and manufacturing, entertainment and sports, defense and security, and any field in between. Because of this, it has become a massively popular field.

While data science has many successes, its popularity has had its drawbacks. "Data science" is now a buzzword that is championed as the harbinger of success,

without there necessarily being any understanding of what data science actually is. This has led to investments in data scientists and technologies that have not been able to actualize on the goal their deployment was supposed to bring. Indeed, turning data analytics into something actionable, such as increasing revenue, has been one of the chief problems businesses have encountered. Ironically, there are now businesses which promise to turn this around, often by offering data science services of their own. While data science may be spectacularly successful in many fields, it is clearly not a silver bullet which brings success overnight.

Keeping these pitfalls in mind, it is worth discerning between what data science actually is and what it is not. To that end, let us pin down what data science is. Data science is simply a field which uses scientific methods to extract knowledge from raw data. Data may come in many forms from any or multiple of the fields listed earlier. All of the methods in data science revolve around, predictably, the data itself. Without data, there would be nothing for data scientists to do. Fortunately, data is generated in massive quantities every day.

To get an understanding of what forms data may take on, it helps to categorize it into different types. One way to categorize data is by how structured the data is. If data is unstructured, it could look like almost anything. It could be the text from a book, or a collection of books. There could be ambiguity in the data, statements of facts when the facts are truly false, or items with meaning which cannot be deciphered. For these reasons, structured data is often preferred and more useful. Structure provides an organized framework of storing specific types of data. A simple and very common method of storing data is with a table, like a spreadsheet. Even then, rules must be in place to ensure there is a specific meaning for every cell in the table. Often, each row in the table will describe a different object. These objects are called observations, instances, records, and/or trials. Each column will usually correspond to a different characteristic about the objects. They are called

```
<header class="site-header" itemscope="" itemtype="https://schema.org/WPHeader">
  ::before
  <div class="wrap">
    ::before
    <div class="title-area">
      <h1 class="site-title" itemprop="headline">
        <a href="http://uca.edu/">University of Central Arkansas</a>
      </h1>
      <p class="site-description" itemprop="description">UCA</p>
    </div>
    ::after
  </div>
  ::after
</header>
<h2 class="screen-reader-text">Main navigation</h2>
```

Figure 1.1: An example of HTML, taken from the University of Central Arkansas' home page. Content types are specified by a schema of tags. The content between the "<h1>" and "</h1>" tags is a heading of the first type. Heading types with larger numbers will generally have smaller font sizes.

variables and/or features. Collections of these tables which make up databases is also structured data. In between these two extremes there is semi-structured data (which may also be called, with some slight differences in meaning, quasi-structured and quasi-unstructured data). Semi-structured data is data which is partially structured, but contains unstructured data within it. Unlike raw, disorganized unstructured data, semi-structured data contains some organization to it, enough so that it is able to describe itself to some extent. Web pages are laid out with HTML code, which contains tags to markup which type of object something is. An example is shown in Figure 1.1. This makes web pages an example of semi-structured data. Other documents made with other markup languages are also semi-structured.

With this understanding, we can begin to understand the general workflow of a data scientist. A data scientist must first have a goal, something they wish to achieve. They then either are provided with data or they collect it themselves. The data scientist will then transform their data into some structure that is useful to them, most often a table or collection of tables. Once the data is in a usable form, the data scientist can then apply methods to the data to extract information about

it. When the data scientist has the results they want, they will then do something to make their results applicable. This could be as simple as creating a visual for others to understand the results or as complicated as an actionable plan to cause change on an organizational or larger scale. On projects of this scale, there will generally be multiple teams involved to craft and enable the plan, so communication is one of the most important skills a data scientist can have. It is also not uncommon for the answers provided by a data scientist's research to lead to more questions and more avenues for data science to explore.

1.2 Machine Learning and Data Mining

The data scientist is confronted with an enormous challenge in the middle of the analytical process: the choice of methods they wish to use. This choice of method (and often many different methods working together) will depend heavily on the kind of data they have and the goal they wish to achieve. Two categories of the data science methodologies are data mining and machine learning. While there are distinctions between these two, there is also considerable overlap. Data mining is more focused on exploratory analysis. That is, the data scientist does not really know what gems of knowledge they might find within the data, and they will apply different techniques to break down the data to better understand it. Data visualization, pattern analysis, and other techniques that fall under a category known as "unsupervised learning" are cornerstones of data mining [2] [3] [4]. Machine learning is focused on accomplishing a specific task, such as making accurate predictions or recognizing known patterns within the data. A widely known example is the linear regression [5]. These methods fall under a category known as "supervised learning" [6] [7] [8] [9]. Of course, as mentioned before, data mining and machine learning are not mutually exclusive, and

there are examples of methods from one category being used in the other domain. Indeed, there is also a hybrid between supervised and unsupervised learning, known as semi-supervised learning [10] [11] [12] [13].

At the heart of all of these methods is the idea of learning. While humans are certainly adept at learning, computers are another matter entirely. Data scientists have to rely on computers to do the tasks required of them, for the size of the data and computations involved are too large or tedious for humans to do manually with any hope of efficiency or reliability. Those familiar with programming will know that computers do not learn on their own; they must be made to. For those unfamiliar, humans create a series of instructions for computers to execute. Assuming the instructions do not contain any flaws and that the computer is operating properly, the computer will execute those instructions exactly. How then does a computer get the flexibility to learn? It actually turns out that learning programs are not fundamentally any different from other programs. When a computer is learning, it is still executing instructions verbatim. The difference is that the instructions are made in such a way as to allow the computer to generate rules and recognize patterns based on some input – the data. The rules will be generated often according to some mathematical optimization problem. This kind of learning is a special case of artificial intelligence, not the general, strong AI capable of human level or beyond thinking. It remains to be seen if general AI can be constructed from these weaker forms or if a new approach entirely will be needed.

There are several types of unsupervised learning. The first type is known as clustering. In clustering, observations in the data is grouped together based on some metric, often by how similar the data is to each other. Each grouping is known as a cluster. Data may belong to one or more clusters, depending on the exact clustering method employed. If the data is numeric, it is possible to find the distances between the points of data. If the data is not numeric (or not all of the variables are numeric),

there may be other methods of grouping the data based on similarity. This may be possible if there is some notion of order. For example, t-shirt sizes are not numerical, but they are ordinal ('S', 'M', 'L', 'XL', etc.).

Identifying t-shirt sizes would be a good application for clustering. Imagine that a brand new t-shirt manufacturer is setting up shop. They have managed to acquire a data set containing the height, weight, waist size, and size measurements of millions of people's torsos. They want to know what sizes of t-shirts they need to make. For economic reasons, it is not feasible for the shirt company to make shirts for every possible size of person, so they cannot simply make t-shirts for every sized person in the data set. To solve this problem, they hire a data scientist. The scientist expects that larger people will require larger shirt sizes, so the data set should be adequate for solving the manufacturer's problem. The scientist is going to try to identify groups of people that can share the same shirt size. If there are too few groups, there will not be enough sizes to fit everyone. Conversely, if there are too many groups, the t-shirt company's profits will suffer from the increased logistics costs. Depending on the choice of algorithm, the data scientist may get to choose the number of clusters formed or the algorithm may decide as it constructs the clusters. In either case, the data scientist will need to identify the size of each cluster that is formed. A cluster that is too small maybe is not worth developing t-shirt sizes for, and a cluster that is too big may need to be split apart so that the shirts fit those people better. This is an unsupervised problem because there is not necessarily any right answer. Different data scientists, and indeed, different companies, may come to a different conclusion about what the sizes should be to fit certain groups of people. Fortunately there are now standard clothing sizes in place to guide manufacturers, but those sizes could not just be chosen arbitrarily if we expected different companies to follow them.

For a supervised learning problem, a company may want to identify which shirt size a specific person falls into. Rather than clustering the dataset to determine

what the shirt sizes should be, this problem assumes that the shirt sizes are already defined and that the people's shirt sizes are listed in the data set. In this case, the data scientist wishes to be able to use height, weight, waist size, and torso size measurements to predict which category of shirt size a person belongs to (such as 'S' or 'L'). This type or problem is known as classification, because there are a finite and discrete number of categories an object may belong to. If the output could be a real number instead, the problem would be known as regression. In this problem, since the data set already contains the shirt sizes for all of the people, the goal is to learn from the data set to make predictions for people which are not included in the data set. In this case, the data scientist will pick algorithms which learn from the data set. It may learn that if a person is above a certain height, they will always need an 'XL' sized shirt. The rules may be more complicated than this. The structure of the rules may not be based on simple conditions, either. Positions of the data points in space or the output of a complicated mathematical function on the data point may determine the shirt size instead. Regardless of the method of how the algorithm works, the data scientist knows that this problem does have a right answer. Because of this, it is possible to find a solution that is better than all others. Thus, the data scientist's job will be to find the algorithm that best learns from the data set. Because the algorithms get to see the answer when it is building the model to match the data set, it is actually possible for the algorithms to cheat. There are methods researchers can use to prevent the cheating. In the simplest case, the data scientist will split the data set into two parts: a training set and a testing set. The data scientists will give the algorithm the training set to learn the rules. Then, the algorithm will be evaluated based on its performance on the testing set (the algorithm is not allowed to change the rules when given the testing set). This ensures that the data scientist picks an algorithm that not only learns the rules well, but also an algorithm which is capable of generalizing to new data points.

When these practices are utilized correctly, models which can explain the data are generated. These models give us knowledge about the underlying trends in the data and can be used to solve real world problems. As mentioned before, one of the problem domains is wine. Wine informatics uses these techniques to find patterns about wine that may otherwise not be possible for humans to detect without the aid of computers. Many techniques in both supervised and unsupervised learning have been employed in order to learn about wine, and this work aims to add to that.

1.3 Prior Work in Wine Informatics

While much work has been done in analyzing the physical and chemical properties of wine [14] [15] [16], there are disadvantages to this approach. Specifically, there is disconnect between human perception and the chemical analysis. Humans are unable to intuitively appreciate a wine based on chemical knowledge of wine. Knowing that a wine has more or less of a particular chemical does nothing to describe what humans perceive when tasting it. Instead, humans intuitively understand what they perceive through sensation. Representing this understanding creates qualitative information and not quantitative information. This makes the analysis of human perception more difficult: there is subjectivity. Fortunately, there are known experts in the field of wine who make wine reviews. These wine reviews are considered to be good descriptors for the wine. An example which compares a wine's review and its chemical properties are shown in Figure 1.2, adapted from [17]. Wine reviews may be generated by experts according to different policies. Wine Spectator is a popular source for wine reviews, generating reviews for about 15,000 reviews each year, and their experts follow a specific tasting guide [18]. In order to avoid bias, all of their tastings are blind tastings. Additionally, reviewers tend to focus on specific types of wines, so that they may become better experts on them. Some tastings may be made by multiple

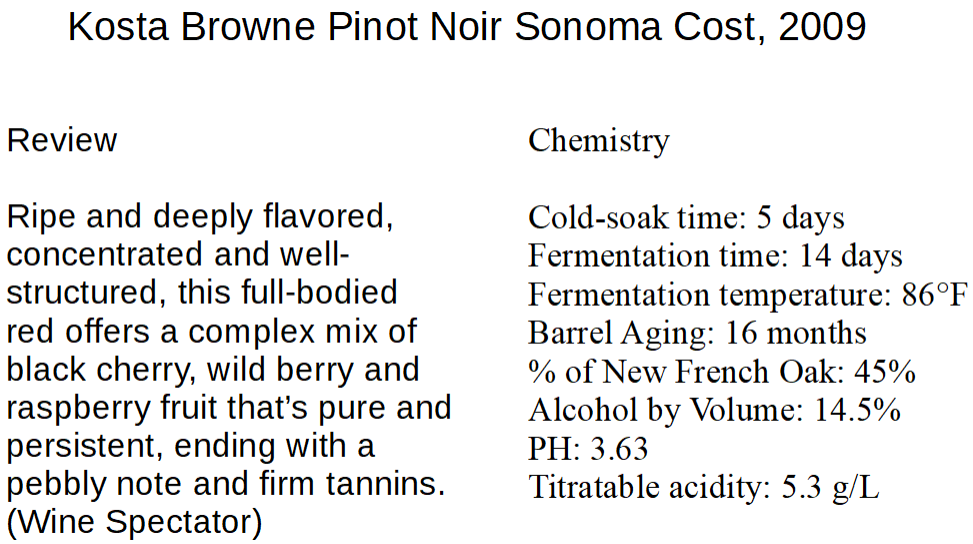


Figure 1.2: An example of a wine's chemical and sensory properties. The review is passed through the Computational Wine Wheel when constructing the key phrases. Figure adapted from *Advances in Data Mining. Applications and Theoretical Aspects: 16th Edition*.

reviewers to ensure consistency and accuracy.

Many works have used wine reviews to analyze wines, and there has also been efforts to allow non-experts to contribute to building a qualitative description of wine [19] [20] [21] [22]. But in order for wine reviews to be useful, the unstructured data must be made structured. In order to construct a representation of wines from these reviews, a systematic process needs to be in place. We chose to use the Computational Wine Wheel 2.0 for this process [17]. The original Computational Wine Wheel was inspired by an earlier wine wheel, the Wine Aroma Wheel by Ann Noble [23] [24]. The Wine Aroma Wheel clustered different wine aromas into a hierarchy. More general descriptions of the aroma are found in the middle of the wheel, while more specific descriptions that fall under the general description branch out towards the outside of the wheel. The Computational Wine Wheel is similar, except that instead of clustering wine aromas, it clusters important words found in the wine reviews. To build the wheel, reviews for the top 100 wines in 2011 from Wine Spectator were analyzed [24]. Important key words, such as "blackberry", were extracted from the

reviews. Unimportant words, such as articles, were discarded. The second version of the wheel improved upon the first wheel by using 10 years worth of top 100 wine reviews. This allowed many more categories of normalized wine attributes to be identified in this Wine Wheel 2.0 [17]. Out of 1,000 wine reviews, 985 different flavors and descriptions of high quality wine were found to have significance.

1.4 Goal of this Work

Using the key words from the reviews found with the Wine Wheel, we wish to predict the wines' price, quality, and region of origin. These three variables are interesting for several reasons. First, humans want to have a pleasurable experience when drinking wine. Whether or not a person has a positive experience and the strength of that experience is best described by a wine's quality. Although not everyone may enjoy a particular example of a high quality wine, it is expected that the majority of people who like wine would enjoy it. The second characteristic we wish to predict is the wine's price. The price of the wine is of high interest to consumers, producers, and distributors alike. The more expensive a wine is, the less likely a person is going to want to buy it. It would make sense that higher quality wine also costs more than lower quality wine, so if we are able to predict the wine's quality, it would be hoped that the price could also be predicted. Lastly, we wish to predict the region where the wine was made, its region of origin, but we will do so in a roundabout way. Wines from a particular region of the world may have a particular taste not found in other wines. In particular, different regions of the world employ different techniques to make wine. These techniques can broadly be grouped into two distinct style: old world and new world. Where these techniques are employed roughly aligns with the corresponding geographic regions, hence the name. Since the style of wine is clearly important to the wine tasting experience, wine drinkers are naturally curious

as to which style of techniques was used to make their wine. Rather than predict which country a wine comes from, we will instead focus on its style, but because the stylistic names and patterns of usage are based in geography, we will still refer to this as region.

It is with these considerations in mind that led to the decision to predict these characteristics. Because this information about wine is known, the type of problem is a supervised learning problem. In order to find the best performance, many different techniques will be explored throughout the remaining chapters. And, as the reader will see, unsupervised learning techniques will also be used in an attempt to enhance some of the the results. It is hoped that all of these efforts will some day be applied to improve the experiences of the many people throughout the world of wine and, where possible, beyond.

Chapter 2

The Basis of the Work: Data

Without data, there is nothing for the data scientist to do. Our source data, as mentioned before, is from the same place used for the Computational Wine Wheel: the website Wine Spectator. The data we used is much larger in scope than that used to construct the Wine Wheel. Instead of using the Top 100 Wines over a decade, we used all high quality wine over the period of a decade. Ranging over a period from 2006 to 2015, we found and processed over 100,000 wine reviews. Of course, getting data in a raw form is not enough: the data must be cleaned and preprocessed in order to become usable. How we did this will be described in the sections below. After that process was completed, we had a total of 105,085 wine reviews that we could effectively use.

2.1 Preprocessing and Cleaning

Once the raw data is acquired from the website, it must be cleaned and prepared. Information that we do not need must be discarded. Wines without a portion of the information we do need must also be discarded. We cannot use a wine if we do not know which part of the world it came from, for example, because one of the goals of this work is to learn how to predict that very thing. We also need concrete definitions for the variables we wish to predict. As the goal is to predict wine quality, price, and region of origin, we need these variables to be set to values in such a way that the machine learning method can understand it. For instance, wine may come from Napa Valley, California; Bordeaux, France; or any other of a number of places. We cannot simply use the words for where the wine came from: the words must be transformed into something an algorithm can understand. This applies to all

three of our response variables. Additionally, once the reviews are passed through the Computational Wine Wheel, we have a list of keywords which are important for that wine review. These keywords must also be transformed into something that a machine learning algorithm can understand.

We can therefore consider the preprocessing to occur in three stages: cleaning, building the response variable representations, and building the wine review representations. Cleaning is removing wines which don't have the information we need, removing excess information from the data, and transforming malformed information into something we can use. Building the response variable representations is the stage where the categories of information we want to predict are transformed. Specifically, the three categories must be transformed into three response variables that an algorithm can understand. Building the wine review representations is the stage where the keywords extracted by the Wine Wheel are transformed into the attributes that will make up the bulk of the data.

2.1.1 Cleaning

After the raw data is passed through the Computational Wine Wheel, a raw data set consisting of the keywords, price, grade, region of origin, and other information is formed. An excerpt of this data is shown in Figure 2.1. Each record must be split into its fields. Fortunately the fields are delimited for us already, and we only need to keep the relevant fields. Additionally, odd characters must be removed from the review.

```
ACORN Sangiovese Russian River Valley Alegr□a Vineyards|2008|
82|$28|Country: California □ Region: Sonoma □ Issue Date: Web Only - 2012|
CHERRY|SPICE|JUICINESS|MODEST|EARTHY
```

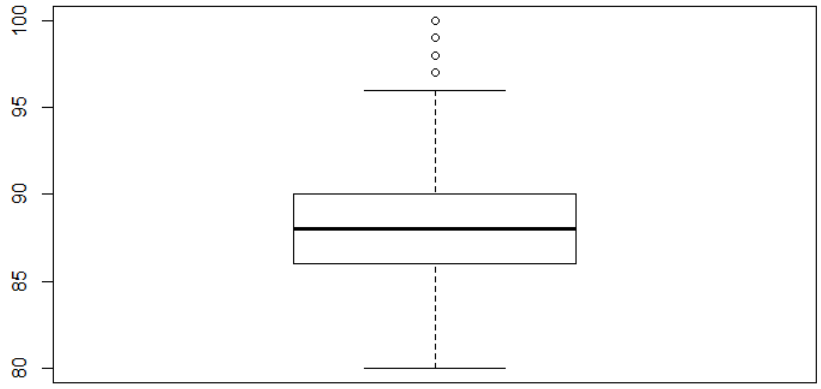
Figure 2.1: An example of an instance of the data before cleaning and after passing through the Computational Wine Wheel. The fields are pipe (|) delimited.

As one can see, there are odd characters in the field denoting the wine's region. In the later stages of data preprocessing, this would cause a problem in transforming the response variables. The currency symbol also needs to be removed so that the field can be converted into a number. With the currency symbol there, the computer will think the field is a string of characters, not a number, which would prevent arithmetic operations from being performed on it. Interestingly, there were some wines with a price of $0. Although it is odd that some wines were essentially listed as free, we decided to keep them in the data set since they are essentially just cheap wines and otherwise are like any other wine in the data set. The Grade must also be converted to a numeric data type. Lastly, there are several keywords at the end of the record. The number of keywords may change between different records. This must also be dealt with in order to build a structured table. Once the cleaning stages are finished, we can then transform the data into useful representations.

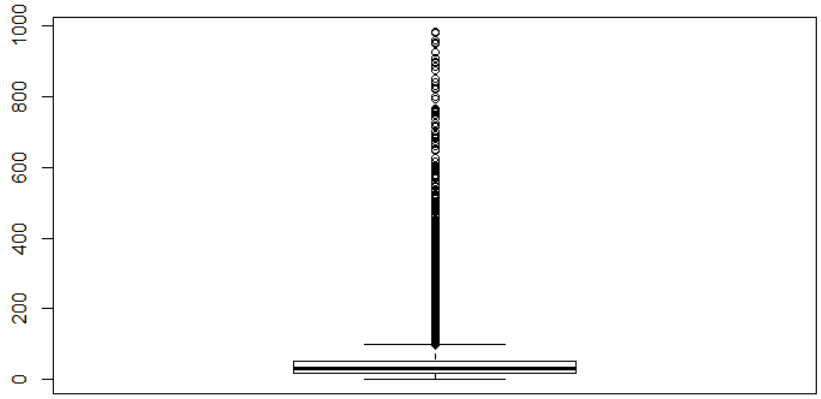
2.1.2 Building the Response Variable Representations

In order to make predictions, there are three pieces of information that need to be transformed: grade, price, and region. First, let us consider grade, which is the quality of the wine. The quality of the wine is judged based on Wine Spectator‘s 100 point scale. Quality ranging from 80-84 is good, while 85-89 is very good, 90-94 is outstanding, and 95-100 is classic [25].

There are two ways this data can be represented: numerically and categorically. As mentioned in Chapter 1, the choice of representation for a variable changes the problem type. A numerical representation makes the problem regression, and a categorical representation makes the problem classification. Thus, a numerical representation simply uses the point value itself to represent the variable. The advantage of this representation is that the machine learning algorithm can learn to pinpoint a wine's grade precisely. This representation has two disadvantages, however.



(a) Grade Boxplot



(b) Price Boxplot

Figure 2.2: (a) The boxplot of the grades. Most of the wines are in the "Very Good" range. (b) The boxplot of the prices. Most wines cost less than $50 per 750mL, but there are many expensive outliers.

First, the boundaries between the different categories of wine, such as the distinction between "good" and "very good" wine, is not obvious with a numerical representation. Indeed, a human may think a wine is outstanding, potentially even a classic, and yet, they may be hesitant to rank it as highly as a classic wine due to the prestige often associated with such high quality wines. A machine learning algorithm seeing the numerical representation may have a harder time seeing such a distinction. The second disadvantage is that it is harder to interpret the performance of an algorithm on a numerical value. One cannot simply say if a wine was predicted correctly: there will always be some level of error with this approach. Conversely, it is possible to tell if a classification approach gave a correct or incorrect prediction, as well as being better able to find the subtleties between the categories that humans may use when they grade wine. This comes at the price of loosing precision. Additionally, there may be more wines in one class than another. This problem will be described later in this chapter. In this work, both representations will be used, but classification will be used the majority of the time. To transform the numerical value to a categorical value, we simply assign a number to each of the categories: '1' for good, '2' for very good, '3' for outstanding, and '4' for classic.

The second variable of interest is price. This variable presents to us the same question that grade did: regression or classification? Once again, we will use both approaches in this work, with classification being used most often. How then, do we classify price? First, all prices were normalized to U.S. dollars per 750mL, which is a common volume for a bottle of wine. Then, we found the quartiles and categorized the wines into one of these quartiles. The prices for the quartiles are <\$18, \$18-\$29, \$29-\$50, and >\$50. Quartiles were chosen as the method of categorizing the prices in order to balance the wines evenly between price groups. Like with grade, we simply assign a number to each of the quartiles. To get an idea of what the classes for grade and price look like, it helps to see the distribution of wines across these classes.

Therefore, the boxplots of these distributions are shown in Figure 2.2.

Last, we need to consider the region of origin. Unlike with grade and price, this problem is inherently a classification problem, as we cannot assign ordered numbers to the countries. Additionally, there are far too many places a wine may come from to simply attempt to use every region as its own class. The results would be fairly useless to us if we tried it. To make the problem easier, we categorized origin based on wine style. There are two widely recognized styles of producing wine: an old world style and a new world style [26]. Historically, these styles tended to reflect where a wine was made as well as the techniques used to make it. Old world wines come primarily from Europe and Israel. New world wines come from anywhere else. Sometimes North Africa and parts of Asia may also be included in this group, but sometimes they are not. For the purpose of this work, these regions are considered New World. Today, this trend largely still holds true, although there are some exceptions. Old world techniques may be applied in new world regions, and vice versa. For simplicity, this work assumes that the historical trend is valid. We then used two classes for region of origin: if a wine originates in the old world, it is given the category of '1' for this variable. Otherwise, it is given the category of '0'.

Grade and price are both multi-class problems, while region is a two class problem. As will be described in later chapters, a multi-class dataset is harder to work with than a two class dataset. Thus, in addition to the multi-class dataset, we also use a modified data set with two classes each for grade and price. These two classes are formed using the lower two and upper two classes. For grade, the 'good' and 'very good' classes are merged into one class. 'Outstanding' and 'classic' are merged into the other class. These classes may also be known as '90-' and '90+', respectively. For price, the lower two quartiles are merged into one class and the upper two quartiles are merged into the other class. These classes may also be known as '$29-' and '$29+', respectively. The breakdown of these classes for each dataset is shown in Table 2.1.

(a) Four class data

Category	Grade	Category	Price
1	≤ 84	1	$\leq$ \$18
2	85-89	2	\$18-\$29
3	90-94	3	\$29-\$50
4	95-100	4	>\$50
Category	Region		
0	New World		
1	Old World		

(b) Two class data

Category	Grade	Category	Price
1	< 90	1	$\leq$ \$29
2	≥ 90	2	$>$ \$29
Category	Region		
0	New World		
1	Old World		

Table 2.1: (a) Response variables and their class categories for the four class dataset. Note that region remains two class. (b) Response variables and their class categories for the two class dataset.

This will allow us to compare results when there is and is not a multi-class problem. We will call this dataset the 'two class' dataset for simplicity, while the dataset with four classes for grade and price will be called the 'four class' dataset, even though there are still only two classes for region.

2.1.3 Building the Wine Review Representations

The data has been cleaned and we have a schema for predicting the information we want, but we have yet to transform the wine review keywords into something that can be useful. A structured table requires the same number of columns in every row, where each row is a different wine. However, a review for one wine may have 4 keywords for example, while another review may contain 8 keywords. If each keyword is to be a variable, a wine must have a column for each keyword that a wine may have, not only the keywords that that particular wine has. We then must construct a set of all the unique keywords that the Wine Wheel found in our data set. In this

Statistics	Value
Minimum	4
Median	13
Mean	13.57
Max	29
Standard Deviation	2.77

Table 2.2: Statistics on the frequency of attributes with a value of "1" per wine.

data set, it turns out that there are 799 different keywords. This means that there are 799 predictor variables. Each variable will be represented with a binary value: '1' if the wine has that keyword in its review, '0' if a wine does not have that keyword in its review.

Because there are 799 predictor variables, and most wines do not have that many keywords in their review, the majority of the values will be 0. This makes the data very sparse. Indeed, the number of attributes any particular wine is associated with is fairly few, only about 13 in 799 on average. Some additional statistics for the number of attributes that a wine has are shown in Table 2.2. This information will be useful when we choose the machine learning algorithm in the next chapter.

2.1.4 Putting it All Together

When all the preprocessing is completed, the data has been structured into a table with 105,085 rows and 802 columns. 799 of the columns are for the keywords and are known as the attributes or predictor variables. The remaining three columns are for grade, price, and region, and are known as the response variables or labels.

2.2 Descriptive Analysis

Understanding the data is what a data scientist's goal is all about, and the first steps to understanding the data is a descriptive analysis of the data. While we would like to be able to visualize the data, it is not possible to visualize a dataset with so many

Figure 2.3: A word cloud of all of the keywords that occur on at least 2,000 different wines. The size of the word corresponds to the frequency. Color is used to help visually separate the words from each other.

dimensions. Instead, we can visualize pieces of the data and use some statistics to better inform us.

It would be interesting to note what some of the most popular wine attributes are in the dataset. Figure 2.3 visualizes the most common keywords found in the data set. As we can see, there is a huge variety of very common wine. Even though only a small subset of the words are pictured, the reader should note that the words primarily reflect the flavor of the wine and not its price, quality, or region of origin. Although there are some phrases that would suggest the quality, such as "good" or "excellent quality", these words were considered important by the Computational Wine Wheel in describing the wine. These words are also not guaranteed to be in very high quality wines. Thus, the machine learning algorithm will need to learn to distinguish how these words correspond to quality, price, and location.

It is also a good idea to visualize the distribution of our classes. Since price was

(a)

Grade Category	Mean Price	Maximum Price	Standard Deviation
Good	$ 20.95	$ 370	$ 15.53
Very Good	$ 30.66	$ 850	$ 24.58
Oustanding	$ 64.21	$ 952	$ 60.33
Classic	$ 163.04	$ 985	$ 158.13

(b)

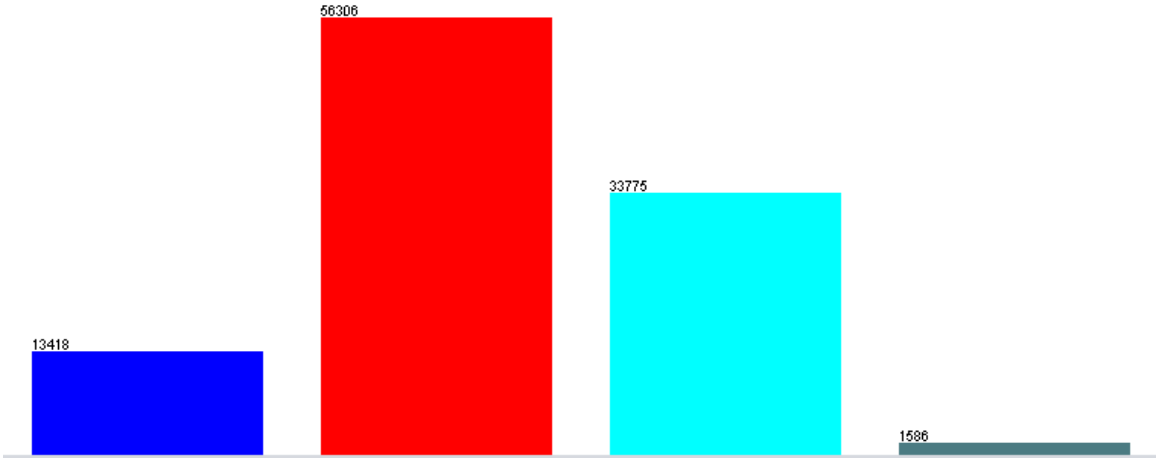
Grade Category	Percent under $50	Percent over $200
Good	95.8935 %	0.0223 %
Very Good	88.0883 %	0.1439 %
Oustanding	54.5285 %	2.8246 %
Classic	15.5107 %	23.3922 %

Table 2.3: (a) Statistics of prices over the grade categories. (b) The percent of relatively cheap and expensive wines in each grade category. Prices are per 750mL.

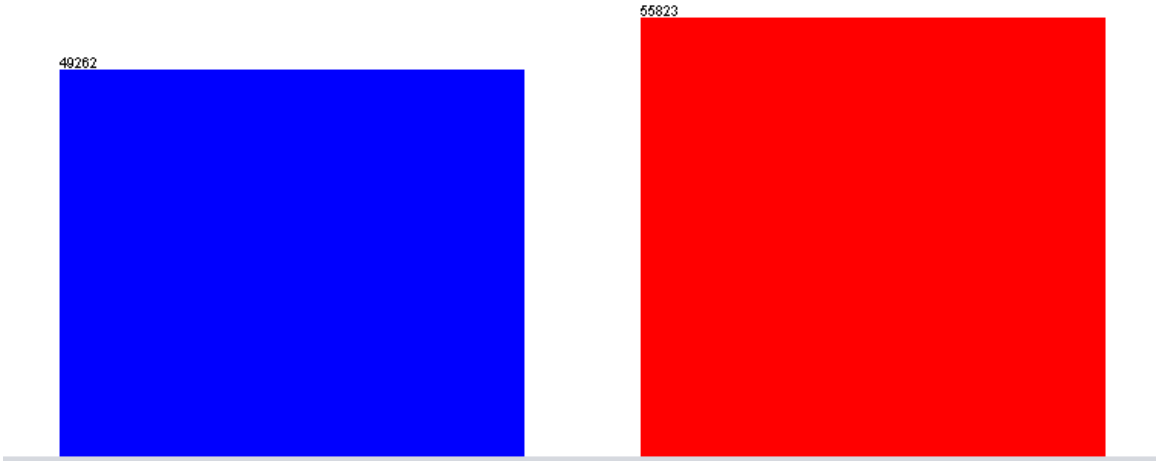
divided up according to quartiles, each quartile is roughly the same size. The mean price is $42 per bottle, while we know from the quartiles that the median price is $29 per bottle. We would expect the price to increase as the quality of the wine increases, and indeed they do. However, the standard deviation in the price sharply increases as the quality increases. This is shown in Table 2.3 (a). This suggests that higher quality wine does not always cost more. Indeed, it is very possible to find classic and outstanding wines for under $50.

It is generally preferable for each class of our data within a variable to contain the same number of wines. Since we used quartiles for dividing the prices of wines, this is done for us. However, the grade variable is divided based on Wine Spectator's 100-point scale and region is divided based on region type. There is no guarantee for balance in either variable. The distribution showing the number of wines in each category for grade and region are shown in Figure 2.4. There are more old world wines than new world wines, but the imbalance in region is not that bad. The real problem is with grade: the imbalance between the classes is alarming. There are only 1,586 wines in the "Classic" category. It is expected that wines of this caliber would be few and far between, but because the other classes outnumber it by nearly 100 to 1, our

ability to make predictions on that class will suffer. Essentially, a machine learning algorithm can completely ignore the class and only suffer about a 1% decrease in performance, while attempts to classify members of this class correctly can lead to greater decreases in performance elsewhere. Thus, the classifier will tend to ignore the class. Solving the imbalance problem will be a major focus of Chapter 6.



(a) Distribution of grades



(b) Distribution of regions

Figure 2.4: (a) The distribution of wines across the grade classes. The "Classic" class only has 1,586 wines. (b) The distribution of wines across the region classes. There are slightly more old world wines than new world wines.

Chapter 3

The Workhorse: Prediction Methodology

In the past two chapters, we explored two critical components in order for this work to proceed: the goal of this work and the data the work will be built upon. One final foundational piece remains: the entirety of this work depends on the ability to make predictions. Without any method of mapping the keywords in the wine reviews to the grade, price, and region of origin, there is no point in knowing either from the perspective of our objective. The purpose of this chapter is to introduce this last foundational component of the work.

3.1 White Box vs Black Box Algorithms

As the analysis will be conducted via a machine learning algorithm, we need to pick one to use. There are many machine learning algorithms, with many different properties. They can broadly be classified into two types: white box and black box. Both types of algorithms are capable of performing classification and/or regression tasks. The difference lies in the interpretability of the model built by the algorithm.

A white box algorithm produces a model which is easily interpreted by humans. A decision tree is an example of a white box algorithm [27]. It produces a series of a decisions based on the data in the shape of the tree, with the first decision being the root of the tree, and subsequent decisions being stems off of the previous decision. Each decision is decided by a condition, such as the value of a variable is greater than some threshold. When there are no more decisions to make, one has arrived at the leaf of the model, and the model outputs a prediction based on all of the past decisions. This schema makes it very easy to see why the model outputs a prediction based on a given input.

Black box algorithms still produce outputs, but the reason for their outputs are not so easy for humans to understand. Usually, a black box algorithm's focus is on greater predictive power at the expense of interpretability. While knowing which wine review attributes yield better or worse grades, for example, might be a desirable thing to know, this work is focused on ensuring that we can predict grade based on the attributes in the first place. In several prior works, many algorithms were compared for performance in many different fields, and a black box algorithm was one of the best performers [28]. This algorithm is the Support Vector Machine [29]. In addition to performing well in Wine Informatics, Support Vector Machines have performed well in other domains [30] [31] [32] [33]. It is for these reasons that it is our choice of algorithm.

3.2 The Algorithm: Support Vector Machines

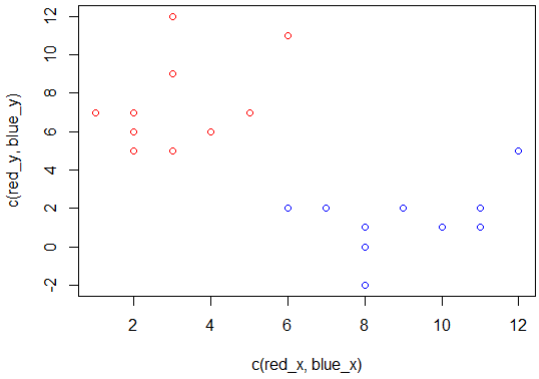
3.2.1 Background

Support Vector Machines (SVMs) were developed over a period of several decades. In 1963, Vapnik and Lerner introduced a method of pattern recognition using generalized portraits, one of the first contributions specifically towards inventing the SVM [34]. Even so, it was not until 1992 when Boser, Guyon, and Vapnik published a paper that advanced SVMs close to their current form by introducing an algorithm for optimal margin classifiers [35]. Finally, the modern SVM arrived with the idea of the kernel trick, as well as support for regression, which was also added in the 1990s [36] [37]. Of course, none of these advancements mean anything without an understanding of what SVMs actually do.

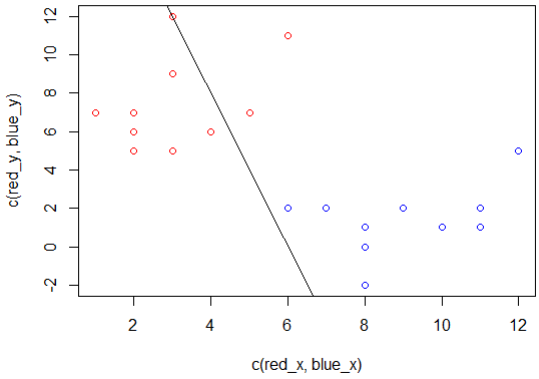
3.2.2 Basic Operation

To understand how SVMs work, let us consider that we have two groupings of data. Each grouping can be represented by a collection of points in space. For simplicity, let's assume that the groups are well separated from each other and are fairly close to members of the same group, as shown in Figure 3.1(a). We want to classify the points into the two groups: red in the upper left corner, and blue in the lower right corner. A SVM accomplishes this by drawing a line separating the two groups. SVMs operate in high dimensional space, so the high dimensional version of a line is a hyperplane. The hyperplane separating the two groups of points should be drawn such that the margin, or space between the hyperplane and the points, is at a maximum. The rationale for this is easy to see when considering the following the visuals. In Figure 3.1(b), a hyperplane which fails to separate the points is shown, while a hyperplane which is barely able to separate the points is shown in Figure 3.1(c). A SVM would not draw either of these hyperplanes. In the first case, the hyperplane simply fails to separate the classes. In the second case, the hyperplane may not separate the classes when new data points are added. If a red data point were added with an x-axis value of 6 or greater and still appear on the graph, the hyperplane would fail to separate the classes. Similarly, if a blue data point with an x-axis value less than four were added to the graph, the hyperplane would fail. A hyperplane that is much more likely to be successful when new data is added is shown in Figure 3.1(d). This hyperplane better takes into account both x-axis and y-axis values that are likely to be found for a class of data. In other words, it is more likely to be more successful because the margin between the hyperplane and the points is much larger.

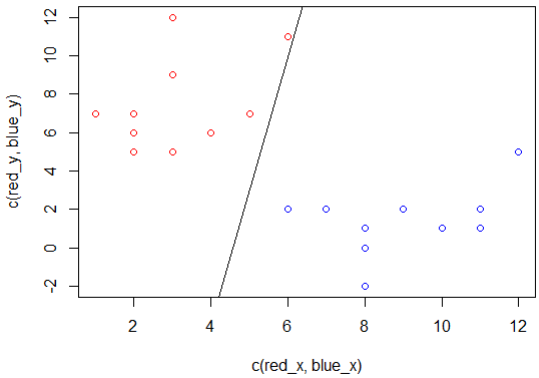
Mathematically, an SVM simply solves an optimization problem. If the points are represented as $(\vec{x}_1, y_1), \ldots (\vec{x_n}, y_n)$, the formula for the hyperplane can be given by $\vec{w} \cdot \vec{x} - b = 0$, where $\vec{x}$ are the attributes, y is the response variable, $\vec{w}$ is the "slope" of the hyperplane, and b is the intercept [38]. The margin which represents



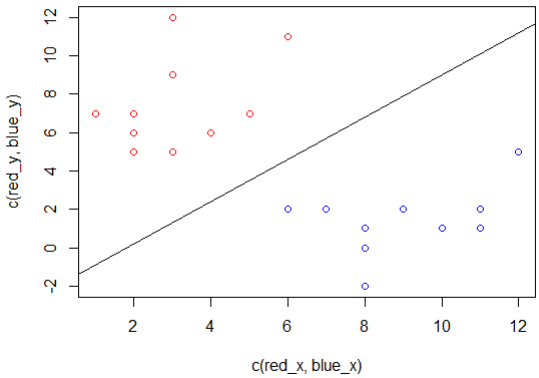
(a) A collection of points, classed by color



(b) A non-separating hyperplane



(c) A poor, separating hyperplane



(d) A good, separating hyperplane

Figure 3.1: A collection of points and several hyperplanes

the closes a point gets to the separating hyperplane can be represented with two additional hyperplanes, one for each class of data. Each hyperplane would then form a boundary where no points of either class are found between the margin hyperplane and the separating hyperplane. The formula is $y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1$. Subject to this constraint, the optimum $\vec{w}$ can then be given by the minimizing $\frac{1}{2}\|\vec{w}\|^2$ [38]. These formulas assume a hard margin, that is, a margin without points between the margin and separating hyperplane. In general, this is not always possible. A soft margin allows for some overlap. With a regularization term, the goal of an SVM is to minimize

$$\frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{n}\epsilon_i,$$

where n is the number of points, C is a parameter which controls how hard or soft the margin is, and ϵ_i is a "slack variable" in the optimization which is an upper bound on the number of errors [38].

When the SVM constructs a model, it essentially needs to define only three things: the separating hyperplane and the two margins on either side. This is defined by what are called support vectors. Support vectors are simply the data points which lie on or within the margins and also data that was misclassified. That is, data that is on the wrong side of the hyperplane also becomes a support vector. The model is composed entirely of these support vectors, and all non-support vector data points are discarded. This means that SVM models are somewhat sparse by selecting important data points automatically. Still, it is difficult to interpret what each support vector means, even though their purpose is clear. This makes the SVM a black box algorithm.

3.2.3 Kernels

In the previous section, the idea of a separating hyperplane, which is a generalization of a line, was introduced. Because a hyperplane is effectively a line, a basic SVM

would only be able to separate linearly separable data. However, it is entirely possible for the data to not be linearly separable, yet still have a somewhat simple boundary between the classes. Consider a case where we have an x-y plane with two sets of data: red and blue. Let the red data be near the origin and let the blue data orbit the origin, but further away, to form a donut around the red data. No line will be able to separate this data as is. In order for a successful separation to occur, the data should be transformed so that a linear separation is possible. This transformation is applied via a process called the kernel trick [38] [39] [40] [41]. A kernel is just a function which defines how this transformation is performed. Other than the linear kernel, which is the kernel an SVM should use to separate data that is already linearly separable, common kernels include the polynomial, radial basis function, and sigmoid kernels.

Mathematically, these kernels simply transform the data by putting it into higher dimensional space. For demonstration, let us return to the donut example with red and blue data. If x_i and y_i were to represent the x- and y-coordinates of the data, with i being an index to represent each data point, the red data would have fairly small absolute values of x_i and y_i, as they are close to the origin. Meanwhile, the blue data would have large x_i and y_i values. One could then use this information to come up with a kernel which would linearly separate the data in three dimensional space. If the function $z_i = (x_i + y_i)^2$ were used, for example, the red data would only go up a little bit in the z-axis direction because of its comparatively small x_i and y_i coordinate values. The blue data would raise much further up. This would allow for the SVM to draw a hyperplane parallel to the XY-plane that separates the data linearly. Some popular kernels are shown in Table 3.1.

Kernel	Definition	References
Linear	$K(\vec{x_i}, \vec{x_j}) = \vec{x_i} \cdot \vec{x_j}$	[40] [42] [43]
Polynomial	$K(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j} + c)^d$	[40] [42] [43] [44]
Gaussian RBF	$K(\vec{x_i}, \vec{x_j}) = \exp(-\gamma \|\vec{x_i} - \vec{x_j}\|^2)$	[40] [41] [42] [43] [44]
Hyperbolic Tangent	$K(\vec{x_i}, \vec{x_j}) = \tanh(k\vec{x_i} \cdot \vec{x_j} + c)$	[43] [44] [45]

Table 3.1: Mathematical definitions of some popular kernels

3.2.4 Regression

While the separating hyperplane can be used for classification, it is also possible to run an SVM in regression mode, known as Support Vector Regression (SVR). Intuitively, the idea is to draw a hyperplane that follows the data and keeps all the data within the margins as best as possible. Any error within the margin is ignored, while errors outside of the margin are considered in the optimization [38]. This is similar to how support vectors are chosen in classification mode: by ignoring some errors within the margin the SVR achieves some sparsity [38].

Because SVMs can be used for both classification and regression, their applicability to our use case is further enhanced, as in later chapters both approaches are used. While this does not make the results between classification and regression directly comparable, it will be easier to compare the two since the same basic principles are being applied to both datasets.

3.2.5 Hyperparameters

A parameter is a value found within the model that is used to calculate a prediction based on given input, and these are the values that are optimized by the training process when building the model. Conversely, a hyperparameter is a value set before the model building process begins. The choice of kernel is one such hyperparameter. Depending on the kernel, there might be different hyperparameters associated with that kernel. For example, a polynomial kernel can be set with different degrees, such as 2 or 3, to form a quadratic or cubic polynomial kernel, respectively. Kernels may

also have a hyperparameter to control an intercept term or some other constant value found within their definition.

In addition to the kernel hyperparameters, the SVM has several other hyperparameters as well. Specifically when classifying, the hardness or softness of the margin, known as the costs of constraints violation, can be changed. In regression, the loss penalty, which controls the width of the margins, can be set. These hyperparameters collectively influence how well the SVM performs on any given data set.

3.3 Multiclass Classification Approaches

When doing a classification problem, the SVM is inherently a binary classifier. That is, by separating data with a hyperplane, it divides the data into two sets only. This is adequate for classifying region of origin into the two groups: new world and old world. It is also adequate for when price and grade are treated as a two class problem. However, it is not adequate for when grade and price are multi-class problems. There are a couple of approaches that can be used to make this easier. One is to use an SVM explicitly modified to handle the multi-class case. However, some of the implementations we chose to use in the later chapters do not support this, so we will instead talk about the transformation methods that we do use, since that is supported in all of the implementations.

3.3.1 Hierarchical

When the classes are ordinal, it is actually sensical and straightforward to divide up a multi-class problem into several multi-staged two-class problems. In our case, both the grade and price categories are ordinal, as it makes sense to order them from smallest to largest. Region, on the other hand, is not ordinal, as old world as not

greater or lesser than new world. (Of course, region is only a two class problem, and is discussed here only as an example.) In the first stage, the data is split into the two broadest categories, with the lowest values making one class and the highest values making the other class. An SVM is then used to divide the data accordingly. In the second stage, there are two sets of data: data categorized into the lower category and data categorized into the higher category. These datasets are also split based on their classes. Two SVMs are then trained, one for each dataset. This produces four classes in stage three. This continues until all of the multiple classes are represented.

This method requires training several SVMs. If the number of classes is a power of 2, the number of stages required is $S = log_2(C)$, where C is the number of classes. As the number of stages increases, the number of required SVMs doubles. This leads to the formula for the total number of SVMs: $SVMs = \sum_{i=1}^{log_2(C)} 2^{i-1} = C - 1$.

3.3.2 One versus All

Another approach is to have as many SVMs as there are classes. Each SVM is trained to recognize if an object is a member of that class or not. In addition to identifying if an instance is a member of the class, the SVM must put out a confidence score for how confident it thinks an instance is a member of its class. The confidence outputs from all of the SVMs are compared, and the one that output the highest confidence is the class that the instance is assigned to.

This method is called one versus all (or one versus rest) because a single SVM trumps the predictions of all the other SVMs. While this method focuses on achieving high confidence, the fact that only one SVM is outputting the confidence reduces the stability of any single result.

3.3.3 One versus One

To solve the stability problem, the SVMs can be arranged in such a way as to vote on the class an instance belongs to. For this to work, the SVM must always be voting for a class, not just if an instance belongs to a particular class. Otherwise, there would be many multi-way ties. For this to work, both outputs of the SVM must correspond to a class. To satisfy all of the combinations, there needs to be $C(C-1)/2$ SVMs, where C is the number of classes. The class that gets the most votes for a particular instance is the class that instance is assigned to.

Both the One versus One and One versus All methods can result in ties. There can be the same number of votes between classes or two SVMs can share identical confidences, respectively. How this ambiguity is resolved is implementation dependent. However, both of these methods have the advantage over the hierarchical approach in one crucial way: they can work on non-ordinal data. Additionally, the one versus one approach also has the advantage that a misclassification by a single SVM does not necessarily prevent a correct classification, like it does in both the hierarchical and one versus all methods. Except where otherwise noted, this method was the method that our implementations used when dealing with the multi-class problem.

3.4 Building and Using the Model

Although the SVM is a method for modeling the data, there is still one crucial piece of the methodology that is missing: the process of constructing, using, and evaluating the model. There are two aspects to this process: training for model building and testing for model evaluation. Training and testing can then be built up into a more rigorous approach of model evaluation: cross-validation.

3.4.1 Training and Testing

The construction and evaluation of a model can be broken down into two phases: training and testing. Training is the construction of the model. In this phase, a randomly selected part of the data set is fed to the algorithm. This data set is called the training set. In our case, the support vector machine analyzes the training set and builds a separating hyperplane from that training set. The SVM then outputs a model which is said to have been trained on the training set. At this point, it is possible to evaluate how well the model performs on the training set by using the evaluation metrics, such as seeing how many of each class is correctly classified (in case of regression, other metrics such as mean square error are used). These metrics have a serious problem, however: they will be too good. It may not make sense that an evaluation metric can be too good: we do want to build a model that performs very well, after all. However, the reason these metrics are too good is the same reason as a student with an answer sheet to a test can get a score on the test that is too high: the algorithm, like the student, have effectively cheated.

To understand why cheating has occurred, consider how the SVM builds the separating hyperplane: it builds the hyperplane such that there is a maximum margin between the different classes of data. In order for it to succeed, however, it must know where every point in the training set is and what class they belong to. This means that the SVM is able to evaluate how well it is doing while it is building the model and change the model accordingly. Similarly, a student with the answer sheet is able to see how highly their answers will score and can adjust their answers to score higher. Because the SVM actually needs the answers in order to build any model that is useful to us, the solution to this cheating problem is to evaluate the model on data that it has not seen yet. This phase is called testing.

In the testing phase, the remainder of the data set that was not used for training is fed to this model that was built earlier in the training phase. This data set is called

the testing set. When this data is given to the model, the model is fixed so that it cannot change. The only thing that is done with the model is to use it to predict which class each data point in the testing set belongs to. For SVMs, this involves testing to see which side of the hyperplane each point belongs to. Because the hyperplane cannot change, there is no chance for cheating. The evaluation metrics are then able to properly assess how well the model should perform on real world data, that is, data the model has not seen before. Even though evaluating the model with the training set gives inflated results, it is still useful to compare the results from the training set with the testing set. First, it is expected that the results from the testing set will not be as good as the results from the training set. How far apart these results are may indicate if there are problems in the model. If the training accuracy is very high, such as 95% for example, while the testing accuracy is significantly lower, such as 60% for example, there is evidence the model overfit. Overfitting is what happens when the model effectively memorized the training set and is unable to generalize to new data in the testing set. This is analogous to a student that memorized the homework problems and performs very well on them, but never learned how to solve the problems and therefore performs poorly on the test. Conversely, underfitting is when the model did not learn how to classify the data well at all, and both the training and testing performance will be very low.

Overfitting may be caused by several things, such as not having enough training data or having too much model complexity. Reducing overfitting can be done by modifying the model hyperparameters or adding more data to the training set. Underfitting is usually caused by having data that is too complex for the algorithm to represent, and can usually be solved by increasing the complexity of the model. It is also possible that there is in fact no relationship between the dependent and independent variables, in which case increasing model complexity will only produce spurious results.

3.4.2 Cross-Validation

When splitting the dataset into training and testing, some of the data is wasted: the testing set is not used to build the model. This is problematic because data is valuable. Without data, it is not possible to build any model, and it is not always easy or cheap to acquire more data. We therefore want to maximize our usage of data. Simultaneously, we do not want to cheat in the evaluation metrics. It is possible to do both with a process called k-fold cross-validation.

In k-fold cross-validation, the data set is randomly partitioned into k equally sized sets. If we choose $k = 5$, for example, then the data is randomly partitioned into five equally sized sets. We then apply five training-testing phases on this data. Each one of these training-testing phases is called a fold. A fold works as follows: take $k - 1$ sets of the data, in this example that is four of the five sets, and use it for training a model. Then use the remaining partition to test the model. In the next fold, the sets which are used for training and testing are rotated. In our example, three of the four sets used for training in the first fold would be used for training in the second fold. The testing set used in the first fold would also become part of the training set in the second fold. The remaining subset, the one used for training in the first fold that is not used for training in the second, would become the testing set in the second fold. By performing this rotation k times, one for each fold, k different models are built. Each model is trained with a different training set made from the $k - 1$ subsets of data. Each model is also tested with a different, last remaining, subset of data. In this way, k-fold cross-validation achieves both objectives: the entire data set is used to build a model at some point, and cheating is avoided because a testing set is used to evaluate each model.

Once all k models have been constructed and cross-validation is finished, how does one evaluate the models or test them on new data? The simplest way to evaluate them is to simply average the models together. Averaging the performance metrics gives

a central measure for how well all of the k models perform when working together. To make predictions on new data, each model is given the new data and makes a prediction. A voting scheme can then be used to classify the new data, or an average can be used if doing regression.

Even though cross-validation is more computationally expensive, it has an additional advantage. Since each model is trained on a slightly different training set, cross-validation is not only more efficient with data usage, it is also able to produce more robust results than what is possible by a simple train/test split. Due to these advantages, cross-validation is heavily employed throughout this work.

Chapter 4

Classifying Grade, Region and Price Plus Multiclass Classification with the Hierarchical Approach

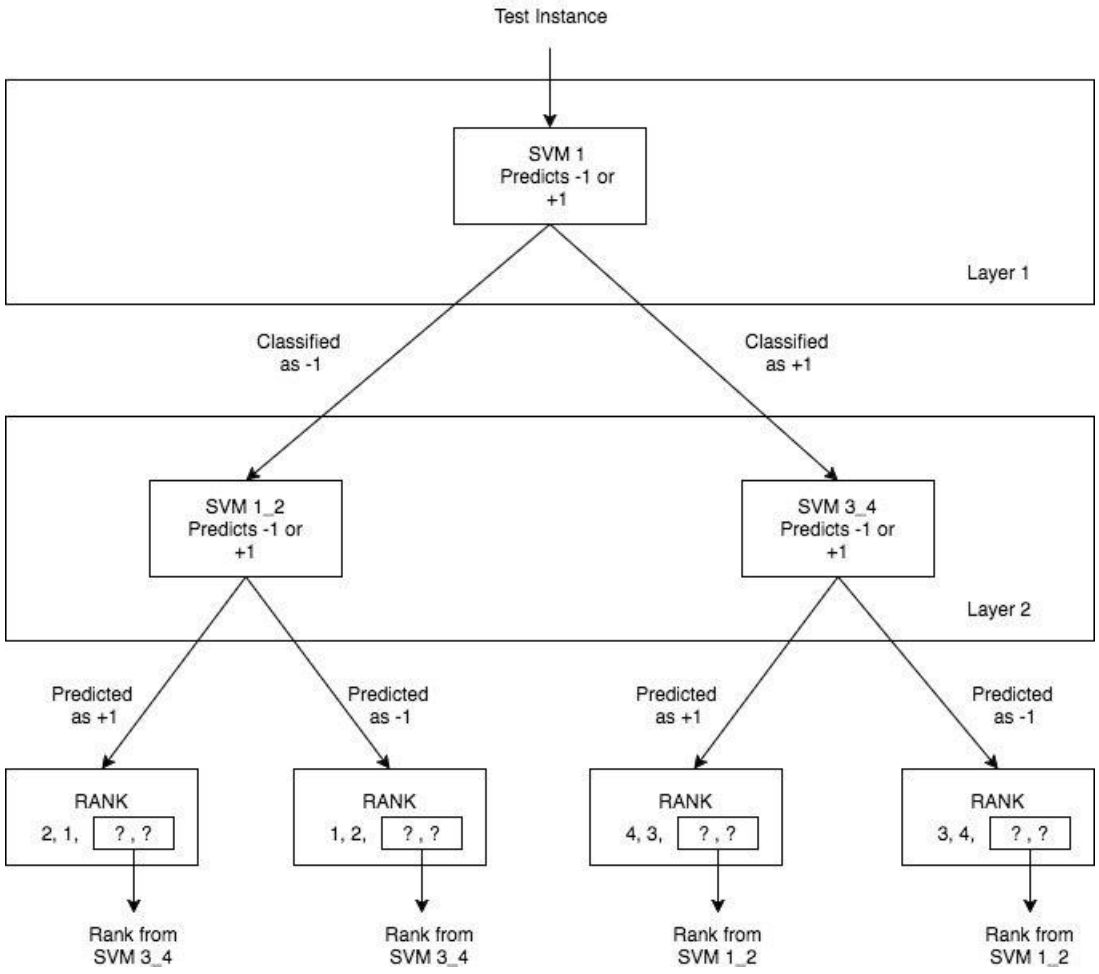
The previous chapters have laid the groundwork; we are now ready to learn from the data. In this chapter, we start by classifying the wine into their region, grade, and price categories. Since region is a two class problem, only one SVM is needed to classify between new world and old world wines. To solve the multi-class problem, we employ the hierarchical approach, which was introduced in Chapter 3, and will be discussed in detail here. The hierarchical approach is exclusively for classifying grade and price, as both are multi-class problems.

4.1 Design and Methodology

4.1.1 Hierarchical Approach

The entire philosophy of the hierarchical approach is that a complicated, multi-way decision problem can be reduced into many different pathways of binary decisions. This approach hinges on the idea that each decision provides more specific information about the object in question. Fortunately, the structure of the grade and price multi-class data allows this approach to do precisely that: the classes are ordered, and ordered classes can be grouped together such that each decision picks between the "high" and "low" classes. Doing this several times should allow the SVMs to predict the wine categories accurately.

The data has four classes for grade and four classes for price, and both of these



(a) Hierarchy

Category	Grade	Category	Price
1	≤ 84	1	$\leq$ \$18
2	85-89	2	\$18-\$29
3	90-94	3	\$29-\$50
4	95-100	4	>\$50

(b) Class Breakdown

Figure 4.1: (a) This shows the hierarchical approach for multi-class classification using SVMs arranged in a two-layer tree-like structure. (b) Table of class breakdowns

variables are considered independently. This means that there needs to be two hierarchies of SVMs, one for each variable. Without loss of generality, consider how a hierarchy of SVMs classifies a wine's grade. First, we arrange several SVMs into a tree-like structure, shown in Figure 4.1. Since there are four grade classes, we need three SVMs arranged into two layers. The first layer contains one SVM, which makes a binary decision. Each decision represents a pathway to the next SVM, found in the second layer. Since there are two possible pahtways from the first layer to the second layer, there are two SVMs in the second layer. Each of these SVMs is also a binary classifier. Two outputs from two SVMs creates a total of four possible outputs from the second layer. Each one of these four outputs will correspond to one of the four grade classes. A similar setup is used for price.

4.1.2 Hierarchy in Practice

The operation for the system can be broken down into the two phases of training and testing. In the training phase, each SVM will be trained independently. The SVM in the first layer is trained on the entire training set. The training set is modified to combine the lower two and upper two classes from the four classes. That is, the lower two classes are grouped together to form a class and the upper two classes are grouped together to form a different class. The two combined classes are the 90- and 90+ grades. The first layer SVM is trained to distinguish between these two classes. The second layer SVMs are each trained on a subset of the training set. One of the second layer SVMs is trained on the 90- data set. It is taught to distinguish between the 80-84 and 85-89 classes of wine. We will denote this SVM as the Layer 2_- SVM. Using the 90+ data set, the other second layer SVM is taught to distinguish between the 90-94 and 95-100 classes of wine. We will denote this SVM as the Layer 2_+ SVM.

The testing phase is slightly more complicated. Each SVM can be tested independently to see how well they learned to split their part of the training set.

For example, testing the Layer 2_+ SVM on a 90+ testing set will evaluate how well the Layer 2_+ SVM does in splitting the data between the two classes. However, this does not tell us the performance of the entire system. In fact, evaluating the SVMs independently only give an upper bound for the performance of the system.

To evaluate the entire system, we need to consider how the system actually classifies when testing. First, an instance will be fed to the Layer 1 SVM. This will classify the instance as 90- or 90+. Second, the instance is fed to the second layer. If the classification is 90-, for example, the instance will be fed to the Layer 2_- SVM, which will classify the instance as 80-84 or 85-89. If the first layer SVM is wrong, it is not possible for the second layer to be correct.

4.1.3 Ranking

Rather than trying to achieve a perfect classification, a ranking of classifications can be constructed, which is also shown in Figure 4.1. In a ranking of classifications, the class that is thought most likely to correspond to the instance is given the highest rank, followed by the second, third, and least most likely classes. To assign this ranking, each instance that has been classified by the first layer is fed to both second layer SVMs. The combined outputs from both the first and second layer can then be used to construct the ranking.

The output from the first layer will be used to determine if the lower or upper classes should come first in the ranking. As shown in the figure, if the first layer outputs a negative prediction, the lower classes will come first in the ranking. Determining which of the two lower classes comes first is determined by the Layer 2_- SVM. Similarly, the ranking of the upper two classes is determined by the Layer 2_+ SVM. If we denote each class as "1", "2", "3", and "4", with "1" and "2" being the lower classes, a negative, positive, and negative prediction from the Layer 1, Layer 2_-, and Layer 2_+ SVMs respectively would result in a ranking of "2, 1, 3, 4". With this

setup, the list of all possible rankings is "1, 2, 3, 4", "1, 2, 4, 3", "2, 1, 3, 4", "2, 1, 4, 3", "3, 4, 1, 2", "3, 4, 2, 1", "4, 3, 1, 2", and "4, 3, 2, 1". Rankings such as "3, 2, 4, 1" and "2, 3, 1, 4" are not possible with this setup, so any difficulty in distinguishing between classes "2" and "3" will harm the overall ranking more severely. In order to evaluate the ranking, a performance metric other than accuracy is required. Ideally, the very first ranked class is the correct one. When the first ranked class is incorrect, it is preferable for the second ranked class to be correct over the third or fourth ranks. The metric which measures how far down the ranking one has to traverse to get to the right class is called coverage, and can be defined by the following formula:

$$CV = \frac{1}{p}\sum_{i=1}^{p} \max \text{ rank}_{y \in Y_i}(x_i, y) - 1,$$

where x_i is the attributes for instance i, y is the correct class for that instance, Y_i is the set of possible classes, and p is the number of instances [46].

4.1.4 Implementation

There are several SVM implementations available for use. In this chapter, SVMLight was the choice of implementation [47]. It is written in the C programming language and is supported on several different platforms. Although SVMLight supports multiple kernels, we solely used the linear kernel. The other hyperparameters were left at their defaults. Additionally, five fold cross validation was utilized for all training and testing.

Fold	Accuracy
1	84.57%
2	83.96%
3	84.26%
4	84.43%
5	84.17%
Average	84.23%

Table 4.1: New/Old World Classification Accuracy

4.2 Results

4.2.1 Region

The results for region are simpler to report and are therefore presented first. The classification accuracy for each fold of the cross validation is shown in Table 4.1. On average, the SVM is able to achieve 84.23% accuracy. For comparison, if the model were to guess randomly, we would expect a 50% classification accuracy. This indicates that there is significant correlation between the words used to describe a wine and where it comes from.

4.2.2 Grade

Let us now consider how the SVMs can predict grade. The results for grade are divided into two sections, because there are two phases of classification. We will first report the results at each layer followed by the overall results of the system. Let us first consider when each SVM is tested independently. The first stage of classification is in Layer 1, and in Table 4.2, the ability of the first layer to classify a wine as 90+ or 90- wine is shown. The first layer is able to classify a wine as either 90+ or 90- with 84.09% accuracy, on average. For comparison, we would expect a 50% accuracy with random chance. Note that the Layer 1 classification accuracy establishes an upper bound for the accuracy of the entire hierarchical classification system. Even if all

Fold	Accuracy
1	84.49%
2	84.03%
3	84.06%
4	83.94%
5	83.92%
Average	84.09%

Table 4.2: Grade Layer 1 Classification Accuracy

Layer 2 SVMs classify at 100% accuracy, we would only achieve the 84.09% overall accuracy that Layer 1 achieves.

In addition to establishing an upper bound for the system, the classification accuracy at Layer 1 is important to note for another reason: it is the classification accuracy of grade on the two class dataset, where grade was transformed into the two class problem. This dataset is used again in Chapters 5, 6, and 7. Thus, the classification accuracy for Layer 1 is directly comparable to the two class accuracies that will be achieved in other implementations in future chapters. It could thus serve as a reference point to compare implementations, whereas the accuracy of the system serves as a reference point to compare multiclass approaches (provided the implementation remains constant). The same will be true for the price accuracy at Layer 1, as well.

Next, let us consider the classification accuracy at Layer 2. The results for Layer 2 are shown in Table 4.3. As the results show, it is easier to distinguish between the class 3 and class 4 wine than it is to distinguish between class 1 and class 2 wine. We attribute the greater accuracy of the Layer 2_+ SVM to the number of instances of Classic Wines: there are very few of these wines. Thus, the SVM is able to misclassify the majority of class 4 wines without significantly impacting the overall classification accuracy. The same cannot be said for Layer 2_-, and the classification accuracy is therefore lower. Clearly, the imbalance is a problem for the upper two classes, as the higher accuracy masks what is really happening. This imbalance will be dealt with

Fold	Accuracy – 12	Accuracy – 34
1	86.12%	95.69%
2	85.78%	95.57%
3	85.01%	95.53%
4	85.55%	95.45%
5	85.43%	95.33%
Average	85.58%	95.51%
Upper Bound	90.54%	

Table 4.3: Grade Layer 2 Classification Accuracy

in Chapter 6.

In addition to getting an accuracy upper bound from Layer 1, we can also get an upper bound by averaging the accuracy of the Layer 2 SVMs. By taking the average of both SVMs average per-fold accuracy, the upper bound is calculated as 90.54% – provided that the Layer 1 SVM has a 100% classification rate and that there are an equal number of 90- and 90+ wines. Combining the upper bounds of both layers, we expect an overall classification accuracy of no greater than 76.14%. Since Layer 1 does not classify at 100% accuracy, we cannot, of course, achieve the upper bound accuracy at Layer 2, the classification accuracy of the entire system is reported next.

To get the overall classification accuracy, the SVMs were not treated independently, but operated in the hierarchy as described earlier. At this stage, we can report both accuracy and coverage, shown in Table 4.6. The overall accuracy for the system is 73.07% – which is very good, being only 3% less than the upper bounds established from Layer 1 and Layer 2. This is corroborated by the value of coverage: 0.4294. A coverage value of 0.4294 means that for every test instance we travel on average less than 1 rank to find the correct class for that instance. The maximum theoretical coverage for a 73.07% accuracy result is when all incorrectly classified instances have the maximum coverage value: that is, the percentage of incorrectly classified instances multiplied by the maximum coverage value, 3. This maximum is 0.8079. Using this formula, we can derive the average number of steps

down the ranking that is taken when an instance is incorrectly classified: 1.59. We can attribute this value to the fact that Layer 1 classification accuracy is lower than the theoretical best Layer 2 classification accuracy. In other words: more of the error for misclassified instances can be accounted for in Layer 1 than in Layer 2. Because Layer 1 controls the order of the lower and higher classes, an error in Layer 1 is more likely to cause a greater coverage, as the correct class is guaranteed to have position 2 or 3 in the ranking (as opposed to position 0 or 1 for a correct Layer 1 classification).

4.2.3 Price

The last variable we want to predict from wine reviews is the price. The setup for price is the same as the setup for grade. First, consider an upper bound on accuracy from the Layer 1 results, shown in Table 4.4. Immediately it is obvious that while there is correlation between the wine attributes and price, the correlation is more difficult to predict than with grade. Specifically, the SVM has a 74.65% accuracy when classifying wines as $29- or $29+. We would expect a 50% accuracy with random chance.

An upper bound can also be established by considering Layer 2, shown in Table 4.3. The results are less accurate than in Layer 1: the accuracy is only 12-14% better than random chance. Combining the two SVMs, we see the upper bound set by Layer 2 is 63.36%, assuming 100% in Layer 1 and an equal distribution of wines between

Fold	Accuracy
1	74.47%
2	74.78%
3	74.46%
4	75.71%
5	74.84%
Average	74.65%

Table 4.4: Price Layer 1 Classification Accuracy

Fold	Accuracy – 12	Accuracy – 34
1	62.81%	65.09%
2	62.61%	64.87%
3	62.78%	64.81%
4	62.08%	64.54%
5	62.19%	64.53%
Average	62.69%	64.77%
Upper Bound	63.36%	

Table 4.5: Price Layer 2 Classification Accuracy

the two price classes.

While the results for price are less promising than for grade, there is still enough information to predict quite better than random chance. Note that by random chance, there would be a 25% classification accuracy for both grade and price. Theoretically, the maximum classification accuracy for the whole system (based on the upper bounds established in Layers 1 and 2) is 47.30%. The actual classification accuracy our system predicts at is 45.61%, which is less than 2% worse than the theoretical maximum. This results is initially unexpected, considering that the difference between the theoretical best accuracy and the actual accuracy for grade is greater. However, this result can be explained by considering that the upper bounds were established by assuming that there is a uniform distribution of wines between the different classes. While this is nearly true for price, there is an imbalance problem for grade.

It is also notable that the coverage for price is worse: 0.8698. Considering only misclassified test cases, the coverage is 1.6. While the coverage for misclassified test cases to be worse for price than grade, it is remarkably similar to grade, even though grade has a much higher overall classification accuracy and lower overall coverage. This can be explained by considering the relationship between Layer 1 and Layer 2. Specifically, Layer 2 is a worse performer for price than for grade. In the case of grade, with a higher Layer 2 accuracy, a misclassification at the Layer 1 level is a more frequent occurrence. Thus, when a misclassification does happen, it is more

Variable	Accuracy	Coverage	Misclassified Coverage
Origin	84.23%		
Grade	73.07%	0.4294	1.5
Price	45.61%	0.8698	1.6

Table 4.6: Overall Results Summary

likely to push the correct rank to position 2 or 3. For price, however, the Layer 2 accuracy is bad enough that a misclassification at Layer 1 occurs less frequently than a misclassification at Layer 2. Therefore, since more misclassifications occur at Layer 2 than at Layer 1, there is a greater chance that a misclassification only swaps the correct class to the second rank, as opposed to pushing the correct class to a rank of 2 or 3. Nevertheless, since the overall classification accuracy is less at all three layers, overall coverage must still be greater for price than it is for grade.

4.3 Remarks

These results are very interesting for three reasons. First, and most important, they show that there is a relationship between wine review keywords and our variables of interest: grade, price, and region. This is remarkable in and of itself. The style of a wine, if it is old world or new world, can not only be determined from its description, but it can be determined reliably. The quality of wine can also be determined reliably, with the exception of the 95+ class due to the class imbalance. And while price is much harder to predict, there is clearly evidence of a relationship between the wine reviews and their price. This evidence validates that the approach used for this work, the usage of reviews to make predictions, is a tenable one.

Second, these results show that there is significant difference in wine reviews based on the kind of country that a wine originates from. Not only that, but the way in which regions were grouped together to form new and old world wines bears significance. This was to be expected for grade and price as both are variables are

based on scales that have clear meaning. The significance of the specific regional groupings done for this work was less clear, as each region has a complex history that often intermingle with the other regions. These results shows that there is indeed a difference between them. New world regions must have some characteristics about them that distinguish them from old world regions, and these differences are neither superficial nor arbitrary.

Third, the results show that a binary classifier like the support vector machine can perform well on the multi-class problem using the hierarchical approach. While not surprising, it gives credence to the idea that more complicated classification structures can be built up from several simpler ones. We will see this idea again in future chapters, successfully constructing more complicated classification systems from basic SVMs to solve both multi-class and other problems.

In summary, these results are encouraging and form the gateway for the remainder of this work. We not only have managed to do multiclass classification, but we also have results for the two class problems as well. The focus will be to improve upon these predictions, address some of the shortcomings in this chapter, and expand upon the kinds of predictions that can be made on wine by utilizing their reviews.

Chapter 5

Continuing the Classification Problem with The One Versus One Approach

The primary aim of this chapter is to use a different multiclass approach to handle the multiclass problem on grade and price. In doing so, additional SVM implementations are introduced, allowing us to compare the binary, two class problems for grade, price, and region as well. In doing these things, this chapter will lay the groundwork for the next several chapters in this work.

5.1 Methodology

As described in Chapter 3, the one versus one method uses several classifiers arranged in such a way as to vote on the correct class. Grade and price are four class problems, so there needs to be six models, one for each combination of two classes. Consider grade, for example, a different model will be built for each of these two class combinations: 80-84 and 85-89, 85-89 and 90-94, 90-94 and 95-100, 80-84 and 90-94, 80-84 and 95-100, and 85-89 and 95-100. An instance which receives more votes for a particular class than any other class is assigned to the class with the most votes. Of course, there is ambiguity if there are ties. How these ties are resolved is dependent on the implementation.

There are a couple of SVM implementations which have the one versus one approach ready to use out of the box, such as Kernlab and LibSVM [43] [48]. Both of these implementations support several different kernels, including linear, polynomial of degree two (quadratic), polynomial of degree three (cubic), and radial basis function. In addition to doing multiclass classification, the performances amongst

the different kernels will also be compared. The results for an additional kernel supported by Kernlab, but not LibSVM, will also be reported: the Laplacian RBF kernel [43]. Other kernels, such as the sigmoid kernels, were attempted during the early stages of this work, but were found to have abysmal performance, so they were excluded from the remainder of the work.

Kernlab and LibSVM are implementations of SVMs that operate in different environments. Kernlab is a package for the R programming language [49]. R has many built in data analytics capabilities which makes it a great choice for performing many of the operations required throughout this thesis. Unfortunately, R was designed to be single threaded, so to speed up the calculations, an alternative version of R was selected: Microsoft R Open, which uses the Intel Math Kernel Library to multithread matrix and other operations [50]. R will be used again in Chapters 6 and 8. LibSVM, on the other hand, is a library implementation that can be used with many programming languages and environments, including R. However, rather than using R, The Waikato Environment for Knowledge Analysis (WEKA) was chosen for this work [51]. WEKA is written in Java and has an easy to use user interface. WEKA will be used again in Chapter 7.

Both of the implementations mentioned here will form the backbone for the remainder of this work: they will both be used in Chapters 6 and 7 while Kernlab will be reused again in Chapter 8. Although the implementations used in these chapters remain the same, the hyperparameters must be kept the same in order to make direct comparisons between the implementations and approaches. The values of the hyperparameters, except for the kernel selection, will be left at their defaults in order to accomplish this. Additionally, five fold cross validation is employed again.

Although comparing the results for this chapter to the previous chapter would be ideal, we are unable to do so. First, note that the value of the cost of the constraints violation hyperparameter C defaults to 1 in Kernlab and LibSVM, but

defaults to the average of $\frac{1}{x_i \cdot x_i}$ in SVMLight. Additionally, the implementations themselves differ internally. Unfortunately, this means that we cannot directly compare which approach performs better: hierarchical or one versus one, without a way to directly compare and account for the differences in the implementations. The best comparisons that can be made between the two implementations are to consider the Layer 1 results in the previous chapter with the two class results in this chapter, as both results operate on the same data to achieve the same result: binary classification. That being said, the difference in how the default values are set only allows us to compare the implementations provided that the default values are used: we cannot compare the implementations as a whole. These limitations are acceptable as the goal of this work is to explore several methods to achieve good predictions, not exhaustively compare every approach and every implementation.

5.2 Results

Despite the limitation discussed in the prior section, we are still able to use the Kernlab and LibSVM implementations to determine how well the one versus one approach can achieve in terms of accuracy. All of these combinations – the two class dataset, four class dataset, implementation in R, implementation in WEKA, and the several kernels – create many results.

5.2.1 Two Class

First, let us consider the results for the two class dataset, which is where grade and price are broken up into two classes instead of four. These results are tabulated in Table 5.1.

First, let us compare the results for grade. LibSVM achieves the best results

overall, 85.92%, when using the linear kernel. Kernlab competes well when using the Laplacian and RBF kernels at 85.475% and 85.471%, respectively. Kernlab's linear kernel falls behind with 84.70% accuracy. Indeed, LibSVM's linear kernel outperforms Kernlab's in all three variables. However, LibSVM's polynomial kernel, both quadratic and cubic, suffer severe performance penalty, to the point that the results are not even close. Even though Kernlab's polynomial kernels do not perform as well as the linear, RBF, or Laplacian kernel, they at least remain competitive. This problem with LibSVM's polynomial kernel is present in the other two variables as well, suggesting that there may be a problem with its implementation.

Next, let us compare the results for price. Kernlab achieves the best results here with 75.890% accuracy using either the RBF or Laplacian kernels. LibSVM's linear kernel comes in second place with 74.86% accuracy, followed closely by Kernlab's linear kernel with 74.70% accuracy.

Lastly for the two class data is the region. Kernlab is victorious again with the Laplacian kernel at 88.518% accuracy, barely beating its own RBF kernel by a hair of 0.001%. LibSVM's and Kernlab's linear kernel again comes in second and third place with 84.4% and 84.36% accuracy, respectively, a whopping 4% behind the leader. Unlike before, Kernlab's cubic kernel remains competitive with the linear kernel, with an accuracy only 0.15% behind the linear kernel.

In a summary comparison of the two implementations, Kernlab's Laplacian RBF kernel, followed closely by its Gaussian RBF kernel, is able to get the best results in Price and Region. LibSVM's linear kernel performs the best of the two linear kernels, achieving the highest classification accuracy in Grade and beating Kernlab's linear kernel in the other two variables. Even so, LibSVM's RBF kernel tends to perform even worse than Kernlab's linear kernel, with grade as the sole exception.

These results also show that the two class results achieved in Chapter 4 can be outperformed by using a different implementation and/or a different default choice

Implementation	Kernel	Grade	Price	Region
R (Kernlab)	Linear	84.70%	74.70%	84.36%
	Quadratic	79.49%	68.41%	83.96%
	Cubic	82.81%	69.25%	84.21%
	RBF	85.471%	75.890%	88.517%
	Laplacian	85.475%	75.890%	88.518%
WEKA (LibSVM)	Linear	85.92%	74.86%	84.40%
	Quadratic	66.35%	50.96%	53.12%
	Cubic	66.35%	50.96%	53.12%
	RBF	84.94%	74.09%	82.92%

Table 5.1: The accuracies on the two class dataset broken down by implementation and kernel.

for the costs of constraints violation parameter. Only considering linear kernels (since we did not use SVMLight's non-linear kernels), the best results for region increased from 84.23% to 84.40%, the best results for grade increased from 84.09% to 85.92%, and the best results for price increased from 74.65% to 74.86%. As mentioned before, we cannot compare the implementations directly due to differences in the costs of constraints violation parameter. Regardless of the implementation used, it is clear that the Linear and RBF kernels perform well on the two class data.

5.2.2 Four Class

Next, let us consider the results for the four class dataset, which is where grade and price are left in the four class forms (and region remains two class). It is here where the one versus one approach is utilized by the implementations. These results are tabulated in Table 5.2. Region is omitted from the table since the number of classes remains the same. Also omitted are LibSVM's polynomial kernels, due to their exceedingly poor performance on the two class data.

Because this is a four class problem instead of a two class problem, we expect the performance for both grade and price to decrease significantly. Indeed, the models on price suffer a massive performance penalty, dropping by 30%. Kernlab maintains

the best performance with the Gaussian RBF kernel at 48.711%, followed by the Laplacian RBF kernel at 48.707%. Kernlab's linear kernel aslo barely beats LibSVM's by 0.01 percentage points. The other kernels perform worse, reflecting their worse performance on the two class problem.

Grade also suffers a performance hit, although not as severe as price does. We can attribute this to the class imbalance problem. With the highest grade class severely underpopulated compared to the other class, the models can effectively learn to classify each instance into one of three classes instead of into one of four classes without a significant performance penalty. This problem will be more thoroughly addressed in the next chapter, but for now, observe that Kernlab's RBF kernels are again the most performant with 76.211% and 76.206% accuracy for the Guassian and Laplacian variants, respectively. In second and third place, LibSVM's linear kernel beats Kernlab's linear kernel by 0.09%. Compared to the two class problem, this represents a drop in performance by about 10%, although Kernlab's polynomial kernels and LibSVM's RBF kernel lose closer to 13%.

These results are not too surprising considering the two class results. Most unexpected is that Kernlab's RBF kernel now performs better than LibSVM's linear kernel on the grade problem, but this may be due to different sensitivities between the kernels to the class imbalance problem. The performance of the linear kernels here is again superior to the performance in Chapter 4: accuracy in grade increases from 73.07% to 75.22% while accuracy in price increases from 45.61% to 47.06%. Again, due to differences in both implementation and approach, neither can be said to be superior to the other from these results.

Implementation	Kernel	Grade	Price
R (Kernlab)	Linear	75.13%	47.06%
	Quadratic	66.98%	40.53%
	Cubic	70.58%	41.39%
	RBF	76.211%	48.711%
	Laplacian	76.206%	48.707%
WEKA (LibSVM)	Linear	75.22%	47.05%
	RBF	70.21%	44.85%

Table 5.2: The accuracies on the four class dataset broken down by implementation and kernel.

5.2.3 Summary

From both the results in Chapter 4 and in this chapter, it is clear that wine review attributes may be used to predict Grade, Price and Region, both in two class and multiclass setups. Although the multiclass problem is more difficulty by its very nature, it is possible to achieve significantly better than random performance by using both the hierarchical or the one versus one approach. While we cannot say which approach is superior in general for this problem, we can say that our specific choice of implementation and hyperparameters allows the one versus one approach to choice to perform very well. Thus, the one versus one approach will be used to solve all remaining multiclass problems in this work, both in this chapter and for Chapters 6 and 7.

5.3 Dimension Extraction and Selection

While the next chapter focuses on dealing with the class imbalance problem for grade, it would be nice to improve the performance for price. Price is harder to predict well in both the two class and four class problems versus the other variables. This can be attributed to the complex economic forces that govern a wine's price. Even so, it may be possible to improve the classification results by using dimension reduction and selection. There are many techniques to perform dimension extraction and selection,

but the essence of the approach lies in removing superfluous or highly self-correlated data that pollutes the model in hopes of reducing both computational complexity and increasing model performance. The difference between dimension extraction and dimension selection is subtle but important: dimension extraction, also known as feature extraction, aims to transform dimensions into fewer dimensions while still preserving some information contained in the extra dimensions (although this is not compression, as some information may be omitted), while dimension selection simply drops a subset of the dimensions out fo the data entirely. Both methods rely heavily on heavily on mathematical and statistical transformations, although it is possible to implement feature selection that exclusively relies on domain knowledge. Here, one technique for each approach is attempted.

5.3.1 Principal Components Analysis

Principal component analysis is a feature extraction technique which aims to linearly transform n variables into p dimensions such that each of the p dimensions are orthogonal to each other [52]. By making this transformation to the p orthogonal dimensions, a new coordinate system is introduced to the problem. The coordinates in this coordinate system are arranged in a particular way, such that the coordinate with the greatest variance comes first, followed by the coordinate with the second greatest variance, and so forth [52]. Variables with the least variance can then be discarded without loosing a significant amount of information.

Intuitively, PCA can be thought of as the data being reshaped from its n variables into a p dimensional, elongated ball, or ellipsoid. The dimension of the ball that is most elongated contains the most variance, while the dimension of the ball that is most squished, or short, contains the least amount of variance. The goal of classification is to distinguish between points that are different from each other. If all of the data points are removed of this tiny dimension with little variance, not much information

was lost because the tiny variance could only make the different points distinct from each other a little bit. Conversely, if one were to remove the component with the largest variance, there would be a lot of information loss, as a dimension that could tell big differences between the data points was lost.

PCA aims to find a middle ground: transform the data points into a smaller set of principal components and, optionally (but almost always) remove some of the principal components that have little variance, but not too many or important information will be lost. In this, choosing the number of principal components to keep is a very important hyperparameter when doing PCA.

5.3.2 Ratio of Attribute Occurrence

While we could use a complicated statistical technique for dimension selection, we instead chose a technique that is similar in some respects to PCA: while considering the class, remove the attributes that do not have many wines associated with them. This process is slightly more complicated than removing an attribute that simply does not have many wines associated with it. If we were to do that, than a hypothetical attribute that is shared by only the most expensive wines would be lost, which would of course make it harder, not easier, to distinguish between cheap and expensive wines. Instead, for each attribute, the percentage of wines having that attribute is calculated instead. Rather than discarding the attributes which simply do not occur often for each class on average, we want to discard the attributes which have the same percentage of attributes across the classes. The idea is that if each class has the same percentage of attribute occurrence for an attribute, than that attribute does not distinguish between those classes very well. Of course, this approach is limited in that it does not consider combinations of attributes.

After the frequency percentage for each attribute across the classes is computed, a ratio between these percentages per class are computed. In this way, it is possible

to identify which attributes have roughly the same frequency of occurrence regardless of class, and which attributes occur with different frequencies depending on the class. Because price is a multiclass problem, the maximum frequency of the four classes is used as the baseline to compute the ratio of difference between the multiple classes. For example, if the percentages of occurrence of an attribute is 2%, 3%, 3.1%, and 4%, the ratio is computed between 4% and 3.1%, 4% and 3%, and 4% and 2%. The minimum ratio amongst these four ratios is used to decide if an attribute is used. A higher min ratio suggests that there is greater difference between at least two of the classes, while a smaller min ratio suggests there is a lesser difference between all of the four classes. All attributes with the minimum ratio less than some threshold are removed from the dataset.

5.4 Results with Dimension Reduction

The dimension reduction methods were performed using the R language on the two class price problem. After performing the dimension reduction methods, Kernlab's implementation was first used to check the results, since Kernlab is accessible from R. Again, five fold cross validation and the cost of constraints violation parameter of 1 were used. The number of principal components used was tested at various points from 3 to 20 principal components. Based on the plot shown in Figure 5.1, we expect most of the variance to be captured within the first seven principal components, and almost no additional variance to be captured when the number is expanded beyond 10. The results are tabulated in Table 5.3.

Unfortunately, neither dimension reduction method yields improved results. The results are, in fact, discouraging: we cannot easily reduce the amount of information without compromising the quality of the predictions. The explanation for why lies in the limited number of attributes that any singular wine may have, coupled with

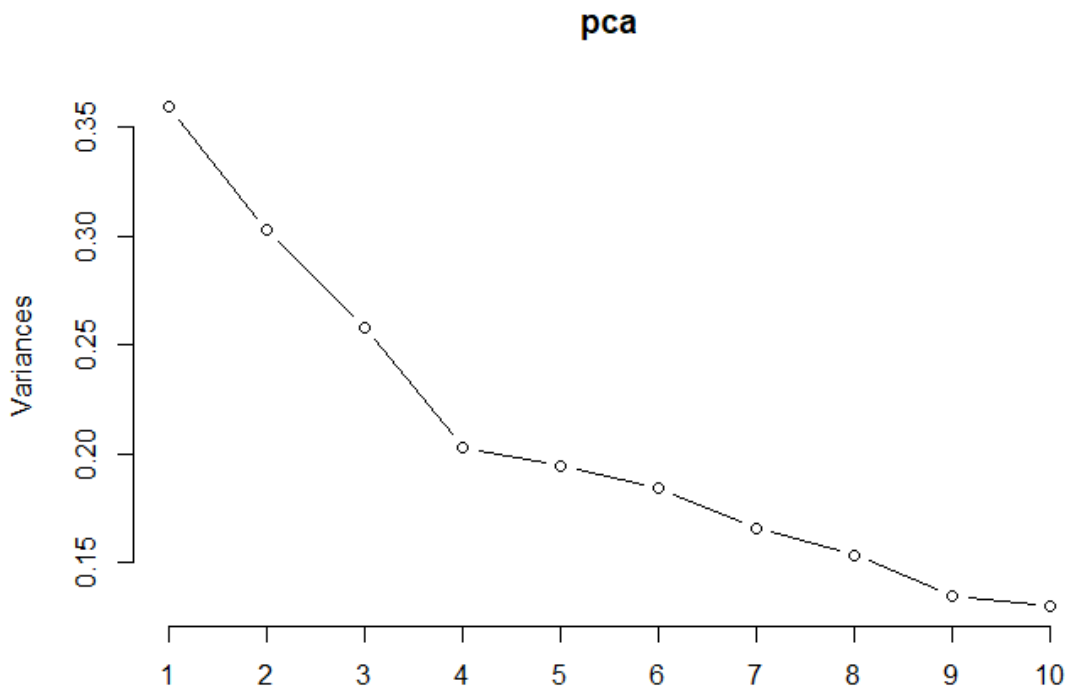


Figure 5.1: A plot of variance versus the number of principal components for the first ten components

Reduction Method	Hyperparameter	Best Kernel	Price Accuracy
PCA	3	Laplace	69.51%
	4	RBF	69.37%
	7	RBF	71.23%
	9	RBF	71.56%
	10	RBF/Laplace	71.64%
	20	Cubic	72.93%
Ratio Selection	90% (108)	RBF/Laplace	35.05%
	75% (193)	RBF/Laplace	69.10%
	60% (307)	RBF/Laplace	70.14%
	50% (401)	RBF/Laplace	70.64%
	40% (493)	RBF/Laplace	70.95%
	25% (627)	RBF/Laplace	71.19%
	10% (729)	RBF/Laplace	71.17%

Table 5.3: The results with dimension selection. The hyperparameter corresponds to the number of principal components when PCA was used and the ratio threshold when ratio selection was used. The number of attributes remaining after ratio selection is shown in parenthesis.

the binary nature of the attributes. Since most wines have on average only 13.57 attributes associated with it (see Table 2.2 in Chapter 2), removing any one of the attributes (or transforming them with the others into non-binary variables using PCA) makes it more difficult for the classifier to distinguish what type of wine it actually is, and therefore makes it harder to classify. Thus, we see diminished accuracies in all attempts. On a positive note, removing almost half of the variables still results in decent prediction accuracy with significant reductions in computational cost required to build the model. If one is pressed for time, ratio selection would be a good way to compromise between accuracy and computational cost. PCA, however, takes significantly longer for the model to build due to the non-binary nature of principal components, so it is not recommended for the same task.

These results do lend to an interesting question. While the Computational Wine Wheel was focused on extracting important keywords from reviews, these results suggest that greater accuracy is achievable when more words can be associated with wines. This conclusion is obvious even without these results: more information can always lead to better classifications. However, the Computational Wine Wheel was built from Top 100 Wines over a period of several years. If the Computational Wine Wheel were expanded to include all wines of 80+ quality (since the wines in this dataset are all of 80+ quality), additional keywords could be found, potentially increasing the number of attributes in this dataset. Would additional attributes for the wines in this dataset, even if some of the additions were not as significant as the keywords already used, lead to significant improvements in prediction accuracy? Additionally, would the inclusion of all words in the wine reviews, even the words related only to the grammar of the review, lead to improved results? This thesis does not answer these questions, but proposes them for future work. The expectation is that despite the increased cost in computation, without presuming which approach would be better, both should lead to better predictions since there is more information

available to the classifier. An advantage to trying these proposed questions to improve accuracy is that the wine reviews must already be available for this work to proceed, so conducting the above experiments do not require any additional information.

5.5 Remarks

In this chapter, two additional implementations could successfully classify wines based on their wine review attributes into different categories for quality, cost, and stylistic origin. In addition, the one versus one approach successfully extended the binary classify, allowing multi-class classification on quality and price. A summary of the best results from both this chapter and the last are shown in Table 5.4.

The quality of the results fluctuate depending on both the implementation and the kernel. The linear, Gaussian RBF, and Laplacian RBF kernels perform the best. Which kernel with the best results depends on the implementation. The linear kernel is the most performant when using LibSVM, while the two RBF kernels are the best when using Kernlab. Both implementations give good results when using their best kernels, while the other kernels may fall behind only a little or substantially. Between the two implementations, Kernlab outperforms LibSVM in all areas except for grade in binary classification. LibSVM manages about half a percent better in two class grade but falls behind between 0.05%-4.2% in the other tasks.

Despite the good results overall, some issues were encountered. The relative rarity of classic wines in comparison to lower quality wines led to a high multi-class classification accuracy in grade, even though the majority of classic quality wines were misclassified. Multi-class classification in price also proved challenging, and even though dimension reduction did not improve upon these results, it did lead to some interesting new research questions. In future chapters, some of these issues will be addressed. For now, these results are, to the best of the author's knowledge, state

Implementation	Approach	Kernel	Grade	Price	Region
SVMLight	Binary	Linear	84.09%	74.65%	84.23%
	Hierarchical	Linear	73.07%	45.61%	
LibSVM	Binary	Linear	85.92%	74.85%	84.40%
	One vs One	Linear	75.22%	47.05%	
Kernlab	Binary	Laplace	85.475%	75.890%	88.518%
	One vs One	RBF	76.211%	48.711%	

Table 5.4: Summary of the best results from Chapters 4 and 5. Note that results between SVMLight and the others are not directly comparable.

of the art.

Chapter 6

Dealing with the Imbalanced Problem on Grade

In the previous two chapters, we used support vector machines to classify wines by their grade using the hierarchical and one versus one approach. In both cases, the classifiers achieved good accuracy. However, the high accuracy masked a fundamental problem: the imbalance problem, shown in figure 6.1. The point of this chapter is to deal with this imbalance problem. But before we begin, why is imbalance between the classes a problem in the first place?

Let us consider the 90+ wines only. There are 35,361 wines in this dataset: 33,751 of them are outstanding and 1,586 of them are classic. In fact, there are 21.3 times as many outstanding (90-94) wines as there are classic (95+) wines. So why does this pose a problem to the classifier? Remember that the goal of the classifier is to achieve the highest accuracy possible. If it simply calls all wines in this dataset outstanding, it can achieve an accuracy of 95.4% by doing almost nothing at all. Indeed, 95% is a very high bar to beat. Therefore, it is very tempting for the classifier to call everything outstanding. Actually classifying the wines as classic leads to a high risk of reducing the overall accuracy: many outstanding wines could be called classic by mistake. Mathematically, one can consider that the SVM effectively draws the hyperplane such that everything falls on one side of the hyperplane: drawing a hyperplane in the middle of the data will lead to too many of the majority class instances on the wrong side of the hyperplane.

Unfortunately, the imbalance between the 90-94 and 95+ wines is not the only imbalance: 90- wines outnumber the 90+ wines 2 to 1. A multiclass imbalance problem is significantly harder than a binary class imbalance problem. This is because

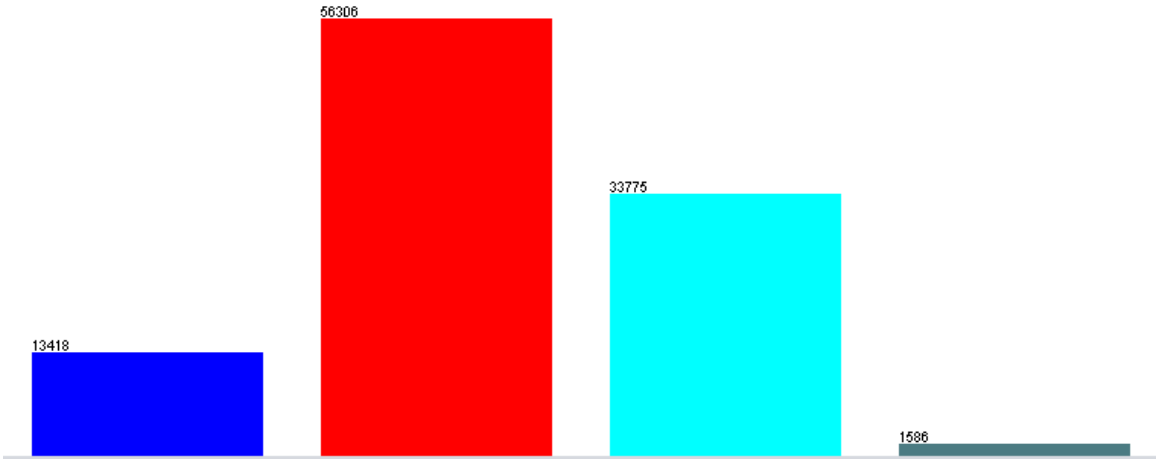


Figure 6.1: The distribution of wines across the grade classes.

the imbalances are different depending on the phase of the problem. If a hierarchical approach is being used, the imbalance at each level and in each SVM will be different. Similarly, if the one versus one approach is used, some of the SVMs will have an imbalance problem and some of them will not, depending on which combination of the classes each one is working on. To simplify the task greatly, with the exception of a single method, only the bigger imbalance will be considered: the imbalance between the 90-94 and 95-100 wines.

6.1 Imbalance Compensation Methods

There are many methods to deal with the class imbalance problem. While we do not try them all, we do try many methods. This section explains what methods we tried and how they work. Throughout this chapter, only the outstanding and classic wines will be considered unless otherwise noted. They will be referred to as the 95- and 95+ classes, but may also be called the majority and minority classes, respectively, to refer to their respective sizes.

6.1.1 Undersampling

The first method we explore is the method called "undersampling". Undersampling is when some randomly picked instances are simply thrown out of the dataset (the reason for the randomness is to avoid introducing biases). Specifically to deal with the imbalance problem, the majority class will be undersampled. The idea behind this is that, by removing many of the majority class instances, the dataset can be balanced without needing to create or collect any more data. Hopefully, even though there is information loss, the majority class can still be represented when it is shrunk down to the size of the minority class. In addition to dealing with the imbalance without needing more data, this method also has the fortunate side effect of being computationally efficient: a model can be built faster on the smaller dataset.

6.1.2 Oversampling

The second method is the converse of the first: instead of reducing the size of the majority class, we can increase the size of the minority class. Increasing the size of the minority class can be done in a couple of ways. One way would be to simply collect more data, but this method is not always feasible or even desirable: it may not reflect reality to have equally sized classes. Simply put: we expect there to be fewer classic quality wines in the world. After all, if high quality wine were in abundance, it would simply be, by definition, average quality. Oversampling is, like undersampling, a statistical technique we can use instead.

There are two ways to oversample the minority class. The first is to randomly take instances of the minority class and repeat them. This approach works by emphasizing the data that is already available. The second way is to randomly take instances of the minority class, tweak them in some way to create synthetic instances, and append them to the dataset. While synthetic samples will be used in a later oversampling method, for now, we will simply oversample by randomly repeating minority instances.

Unfortunately, oversampling has the side effect of being computationally inefficient: a larger dataset is created, requiring both more time and memory to build a model.

A method that is effectively the same as oversampling is class weighting. In this method, the SVM assigns weights to the penalty for misclassification. A weight of 2 for the minority class is the same as having two instances of the minority class in the data. The idea is that, if the weight on the minority class is chosen to be exactly the same as the imbalance ratio, the weight will "trick" the SVM into thinking an accuracy of 50% is achieved if it misclassifies all minority instances, rather than the 95% accuracy it actually achieves in such a case. This forces the SVM to pay just as much attention to classifying minority instances as majority instances. Unfortunately, due to implementation details, this method could not be tried due to exorbitant memory consumption. Even so, this method is worth noting for a future work.

6.1.3 Representative Clustering

The undersampling method worked by randomly selecting instances within the majority class. Because instances are being discarded, information loss is the biggest problem with this method. By randomly selecting majority instances to discard, the success of the method depends in part by the random selection.

Rather than simply randomly selecting information to discard, it is possible to randomly select data in a smart way. Specifically, consider that not all outstanding wines taste the same. In fact, within the class of outstanding wines, it is possible to subdivide them even further into groups via clustering. In this way, outstanding wines which share certain attributes will be clustered together, and different outstanding wines will be placed in a separate cluster.

Once the wines are clustered, we still want to discard some of the instances to balance the classes. However, rather than removing the instances randomly, we want to remove the instances such that these clusters are still represented. The intuition

behind this is that each cluster is a representative cluster for that class. That is, each cluster represents some taste profile in the outstanding wine category. We do not want to discard all of the instances within a cluster since that would effectively destroy that taste profile.

It is actually easier to select data to keep than it is to remove. The process of keeping instances after the majority class has been clustered goes as follows. First, set a desired number of instances to keep. When enough instances have been kept, stop adding more instances to keep. Until then, loop through each cluster. Find the point closest to the center of that cluster, then add it to the instances that are kept.

In this way, each cluster is upsized by 1 per loop until the desired number of points remains or until there is no data left to add. Note that by adding points in this way, larger clusters remain larger at the end of this process. This is sensical: wines with attributes that occur more often remain more frequent after this process is completed. Instead of randomly discarding information, this method attempts to preserve information representative of the different taste profiles across the majority class.

6.1.4 Cluster Based Undersampling

Oversampling and undersampling attempt to resize the classes to solve the imbalance problem. In representative clustering, undersampling is combined with flavor profiles to smartly represent the majority class. However, the clusters that are created in this technique may have different sizes. While it may make sense to preserve this information, it may also be desirable to balance the intraclass imbalances in both the majority and minority class. Intraclass imbalances are the differences that occur within the class itself, and these imbalances may make the interclass imbalance more severe from an attribute perspective. Consider that a certain flavor characteristic that occurs frequently in outstanding wines could very well be a very rare flavor

in classic wines. The common flavor is therefore a dominant characteristic of the majority class, but, since that flavor rarely exists in the minority class, any minority instance which has this characteristic might be mistakenly classified as the majority class. By balancing out the intraclass clusters, any effects caused by differences in the frequency of attributes between the classes is lessened. In this way, the relative rarity of the minority class is not also compounded by the relative rarity of a flavor within the minority class.

In this technique, the entire data set is clustered based on the attributes. Then, each cluster is undersampled so that the clusters are balanced. In this way, the intraclass imbalances and, to some extent, the interclass imbalances are reduced. The interclass imbalances are reduced by the nature that instances of different classes are less likely to be clustered together, and therefore clusters formed from the minority instances are smaller than the clusters formed by the majority instances. The clusters formed by the majority instances are then downsampled more heavily, although an imbalance between the classes is still possible afterwards, especially if the clusters do not cleanly separate the classes.

In addition to balancing out the attributes and hope that the classes are balanced as a side effect, we can make balancing the classes a more explicit endeavor by including the class label with the attributes when doing the clustering. In this way, we borrow an idea from granule computing. By slightly increasing the granularity of the data sent into the clustering algorithm by adding class information, we increase the likelihood that the clustering algorithm will better notice the differences between the classes.

6.1.5 Synthetic Minority Oversampling Technique

Synthetic Minority Oversampling Technique (SMOTE) is a method of oversampling the minority class by generating new, synthetic examples of the minority class [53].

Synthetic examples are not repeats of the minority instances, but new instances entirely. Of course, if we were to just make up instances and call them minority class, we could very well make up an instance that looks like it should be in the majority class. This would destroy any effort at classification. To solve this, SMOTE takes each minority instance and looks at its k nearest neighbors (of the same class), where k is a some integer that we set. Between every minority instance and its k nearest neighbors, an instance is created that lies between the two. This instance is created so that its distance from the original minority instance is random: it may be closer to the original instance or it may be closer to the neighbor.

This approach is based on the idea that members of an instance's class are more likely to neighbor each other than members of opposing classes. Creating synthetic examples between neighboring instances should thus prevent any new instances that look like the wrong class from being created.

6.1.6 Borderline SMOTE

SMOTE can be extended to focus only on data that defines the border between the majority and minority class. This method is known as borderline SMOTE [54]. First, it uses k nearest neighbors on the minority class, like before. It then calculates a metric to identify if a particular instance is on the border or not. If all of the instance's neighbors are of the majority class, the instance is considered noise and is skipped in future steps. If all of the instance's neighbors are of the minority class, the instance is not on the border and is also skipped. Otherwise, if the number of majority neighbors is larger than the number of minority neighbors, the instance is considered to be in danger of being misclassified. For each instance considered in danger, the m nearest neighbors between that instance and the other in-danger instances is calculated. Synthetic instances are then created between the instance and a random sample of its m nearest neighbors. In this way, the border between

the classes is strengthened, so it should be easier for the classifier to recognize the difference between the two classes. While Borderline SMOTE can be used to enhance any classifier, this method is particularly interesting for SVMs since SVMs try to build a hyperplane that defines the border.

6.2 Evaluation Metrics

Clearly, accuracy is not a good metric on an imbalanced data set if 95% accuracy can be attained by misclassifying the entire minority class. Instead, we will look at the overall accuracy and the percentage of minority instances which are correctly classified. In addition, we will also look at two other widely used metrics: precision and recall. Both of these metrics are built on four types of classifications: true positives, true negatives, false positives, and false negatives. True positives (TP) and negatives (TN) are when the classifier correctly classifies an instance into its positive or negative class. A false positive (FP) is when an instance is actually negative, but incorrectly called a positive by the classifier. Similarly, a false negative (FN) is when the instance is positive but incorrectly called a negative by the classifier. Both precision and recall consider the number of true positives. Precision compares the number of true positives to the number of true positives and the number of false positives. That is, of all the instances the classifier said were positive, precision measure how many of them were actually positive. Recall compares the number of true positives to the number of true positives and false negatives. That is, of all the instances that were actually positive, recall measures how many the classifier said were positive.

Mathematically, we can define precision as

$$P = \frac{TP}{TP + FP}$$

and we can define recall as

$$R = \frac{TP}{TP + FN}.$$

Intuitively, precision measures how honest the classifier is about what it says is positive. Meanwhile, recall measures how good the classifier is at finding positive instances. Both of these metrics are needed in an imbalanced scenario. If the minority class is considered the positive class, precision can tell us how many of the instances that were predicted as minority instances were actually minority instances. If there were lots of false positives, the precision will be lowered. Meanwhile, recall can tell us how many of the minority instances the classifier decided to predict correctly. A lower recall indicates that the classifier is still suffering under the class imbalance problem, while a lower precision indicates that too many of the majority classes are being classified as minority instances. Of course, to get a better idea of the overall precision and recall, we can also take the precision and recall for when the majority class is considered positive and average them with the precision and recall for when the minority class is considered positive. This average precision and recall are better metrics since they also indirectly consider true negatives and penalize false negatives.

It is likely that no matter which balancing method we use, the accuracy is going to decrease rather than increase because the imbalance is so bad. So long as the accuracy is still fairly high, this loss is acceptable. It is better to have a classifier which, even though it is wrong some of the time, can still tell us useful information when it is right, than it is to have a classifier that simply gives up, classifies everything as majority, and says nothing about the minority class. When considering the results, it makes sense to give the classifier some slack on the accuracy metric. After all, the accuracy metric is, on its own, insufficient to gauge the success of these implementations.

6.3 Implementation

Clustering requires the use of a distance metric to determine the similarity between objects. The default distance metric is Euclidean distance, which works well on data with real valued dimensions. However, our attributes are binary. A distance metric better suited for binary data is Jaccard distance [55], defined as

$$J(A, B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}.$$

All of the methods to deal with the class imbalance problem were coded in R, therefore the SVM implementation used was Kernlab's. As in the prior chapter, 5 fold cross validation was used, the cost of constraints violation parameter was set to 1, and five kernels were tried (linear, quadratic, cubic, Gaussian RBF, and Laplacian RBF). To perform clustering, k-means clustering was employed. The built in R function kmeans was used to perform the clustering. In addition, the kcca (K-Centroids Clustering Algorithm) function from the Flexclust package was used to perform clustering when using the Jaccard distance metric [55]. To perform SMOTE and borderline SMOTE, the respective functions from the smoteFamily package were used [56].

When using undersampling, the majority class was made the exact same size as the minority class. when using oversampling, the minority class was made the exact same size as the majority class. When using representative clustering and cluster based undersampling, the number of clusters was varied from a few up to 700 in various step sizes. When using representative clustering, the amount of undersampling of the majority class was also varied from the same size as the minority class to 16 times larger than the minority class. When using SMOTE and borderline SMOTE, the number of k nearest neighbors was varied from 3 to 21. When using borderline

SMOTE, the number of nearest neighbors in the second round of k nearest neighbors was also varied. In all cases, the default values for parameters were used unless otherwise specified (such as when changing the distance metric).

Cross validation was modified somewhat in order to handle the differing datasets created by the different sampling methodologies. Randomization of the dataset occurred first, and the same randomized dataset was used for every method (this is actually true for all methods tried in Chapters 5 through 7, while the data in Chapter 8 was randomized separately). The data was then split for cross validation. Cross validation went as follows: the entire 90+ dataset was split into the five folds. Each training set was then fed into the data transformation methodology, while the testing set was left untouched. In this way, each method is comparable to each other. All of the methods received the exact same training set to perform their operations on and all of the models were tested against all of the instances.

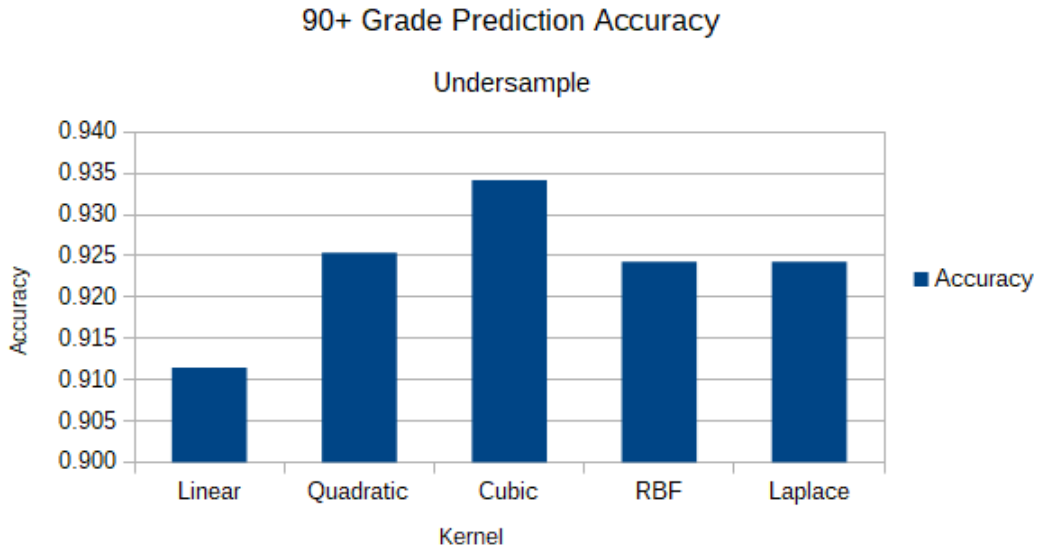
6.4 Results

In this section, we present the results across the various balancing schemes. As a comparison, the baseline classifier on the 90+ data misclassifies all minority class examples and correctly classifies all majority class examples.

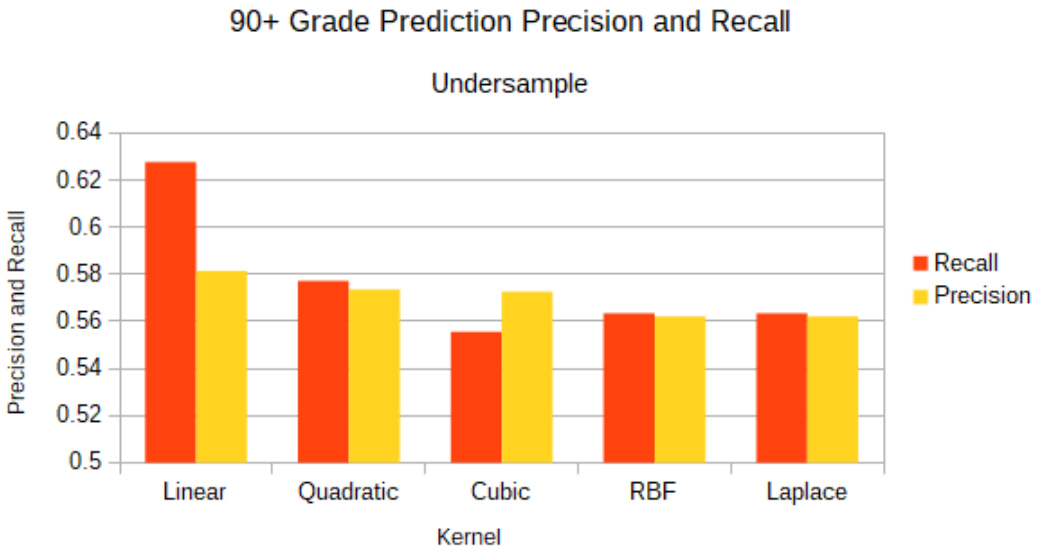
6.4.1 Undersampling

First, consider undersampling. Figure 6.2 (a), shows the accuracies of the different kernels. Remember that accuracy is insufficient to determine which kernel is best. Indeed, while the Linear kernel has the lowest accuracy, it actually performs the best on the minority class. To see this, consider Figure 6.2 (b), which shows the average precision and recall. Clearly, the linear kernels has the best recall with just over 62% recall and 58% precision.

If we consider just the precision and recall for when the minority class is considered the positive class, the values are 0.196 and 0.317, respectively. This means of all the instances predicted as minority, only 19.6% of them actually were minority. Additionally, of all the minority instances, only 31.7% of them were predicted correctly. While the linear kernel has the best precision and recall, it comes at a high price to the majority class: several thousand of them are misclassified at a gain of only 502 correctly classified minority instances. Overall, basic undersampling is a failure.



(a)

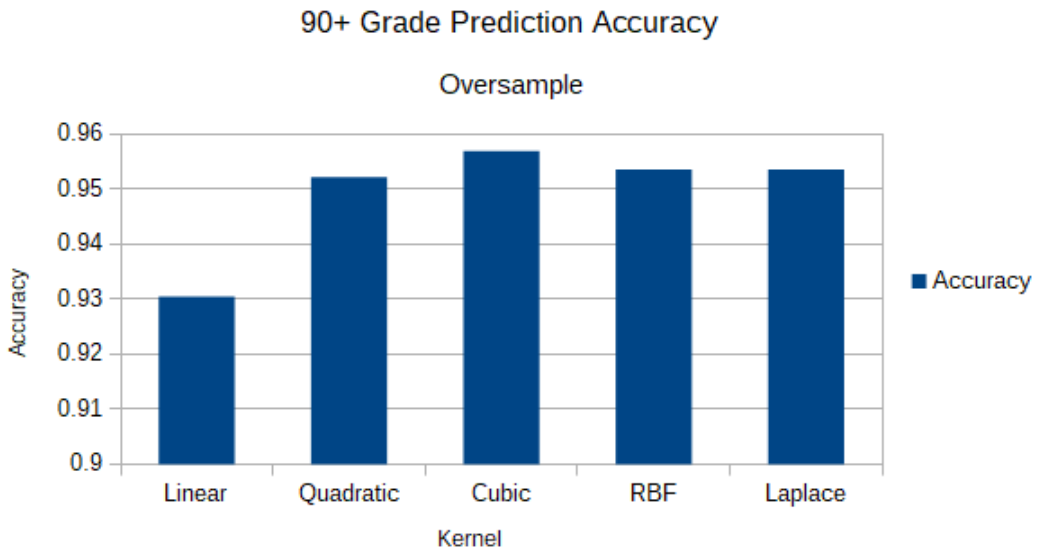


(b)

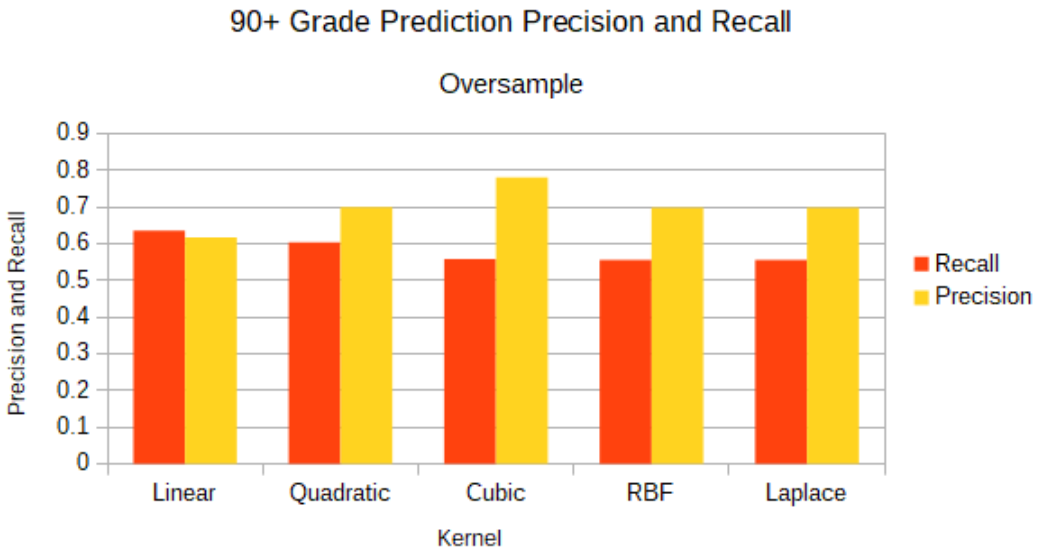
Figure 6.2: The accuracy, average precision, and average recall of the different kernels when undersampling the 90+ dataset.

6.4.2 Oversampling

Second, consider oversampling. In this case, performance across the kernels was more comparable. The accuracies still follow the same pattern as shown in Figure 6.3 (a), but the precision and recall graph is different as shown in Figure 6.3 (b). Even though the other kernels make some improvements, those improvements are primarily on classifying more majority instances right at the expense of the minority instances. The linear kernel still performs best overall. It is able to classify 495 of the minority instances and 32,408 of the majority instances. Precision is improved to 0.266, but recall decreases to 0.312, compared to undersampling. Oversampling therefore performs slightly better than undersampling, but not enough to justify the computational expense, and not enough to call any less of a failure than undersampling.



(a)



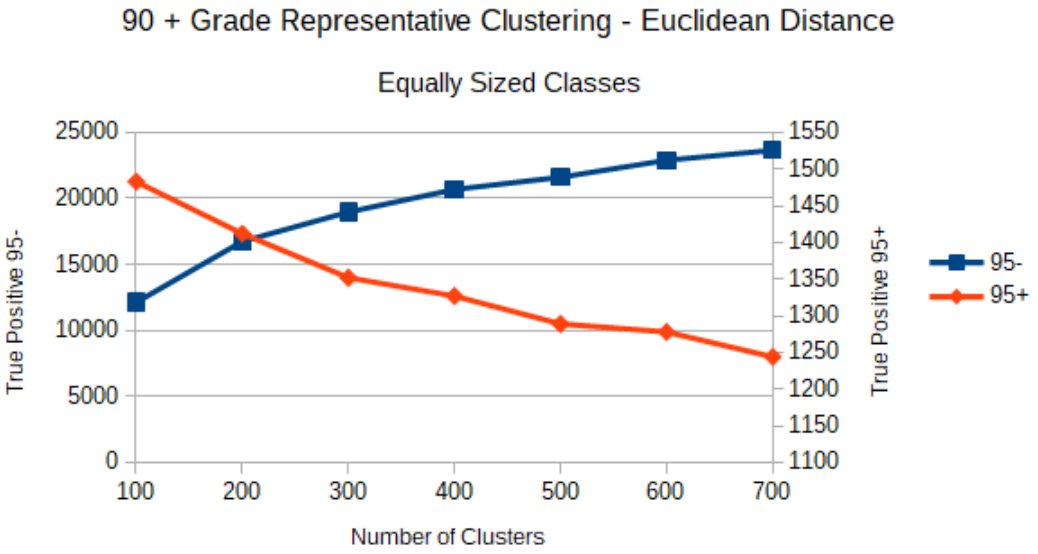
(b)

Figure 6.3: The accuracy, average precision, and average recall of the different kernels when oversampling the 90+ dataset.

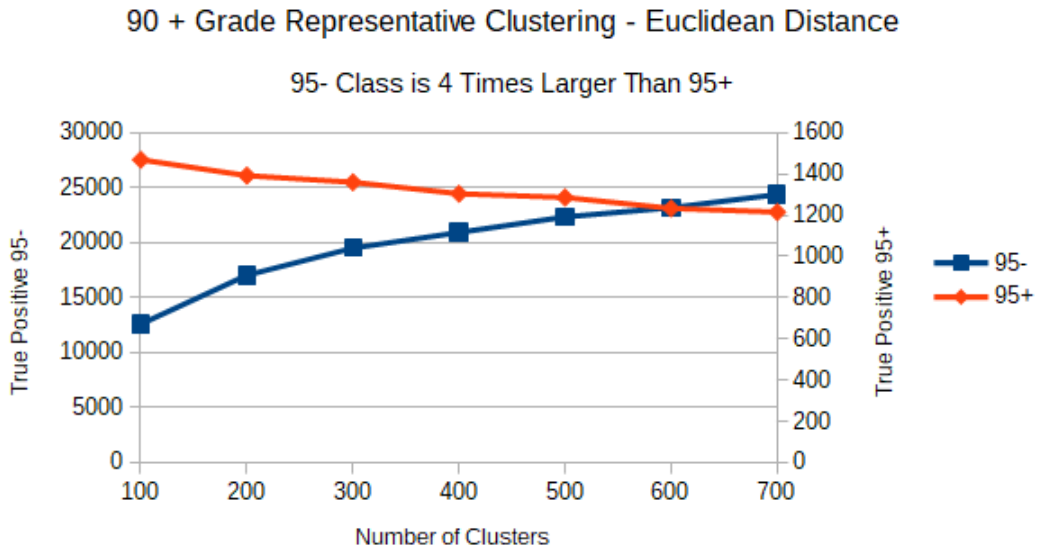
6.4.3 Representative Clustering

In representative clustering, we varied the number of clusters from 10 to 90 with a step size of 20 and from 100 to 700 with a step size of 100. In addition, the number of points picked from the representative clusters was varied so that the majority class was the same size, 2 times the size, 4 times the size, 8 times the size, and 16 times the size of the minority class. In all cases, the quadratic polynomial kernel performed best from an accuracy standpoint, however the cubic polynomial kernel performed better in terms of the number of minority instances that were correctly classified. Unfortunately, the graphs of the number of correctly classified instances are mirror images of each other. A results of a subset of the experiments is shown in Figure 6.4, clearly demonstrating this mirror image. If the number of correctly classified minority instances increases, the number of correctly classified majority classes decreases, and vice versa.

When Jaccard distance was used as the distance metric, the mirror image problem remained, although the linear kernel became the best kernel again. In fact, on several occasions, the classifier could predict all of the minority instances right at the cost of predicting none of the majority instances correctly. Therefore, the choice of distance metric had no effect on the overall success of the method: the method always failed to classify a good proportion of both classes correctly simultaneously, just like with oversampling and undersampling.



(a)



(b)

Figure 6.4: The number of correctly classified instances for each class as the number of clusters is varied using the representative clustering method with Euclidean distance. Note the axis scales.

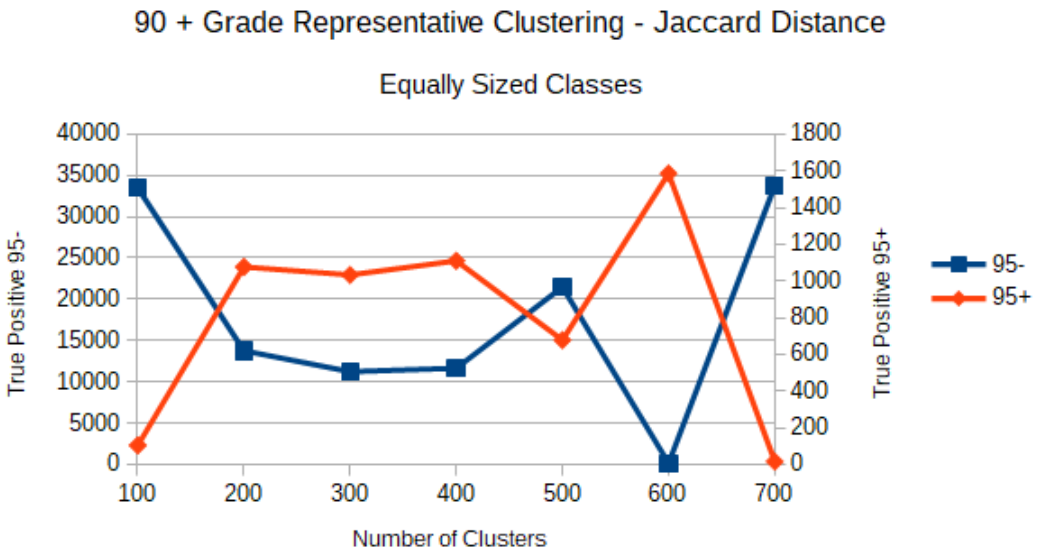
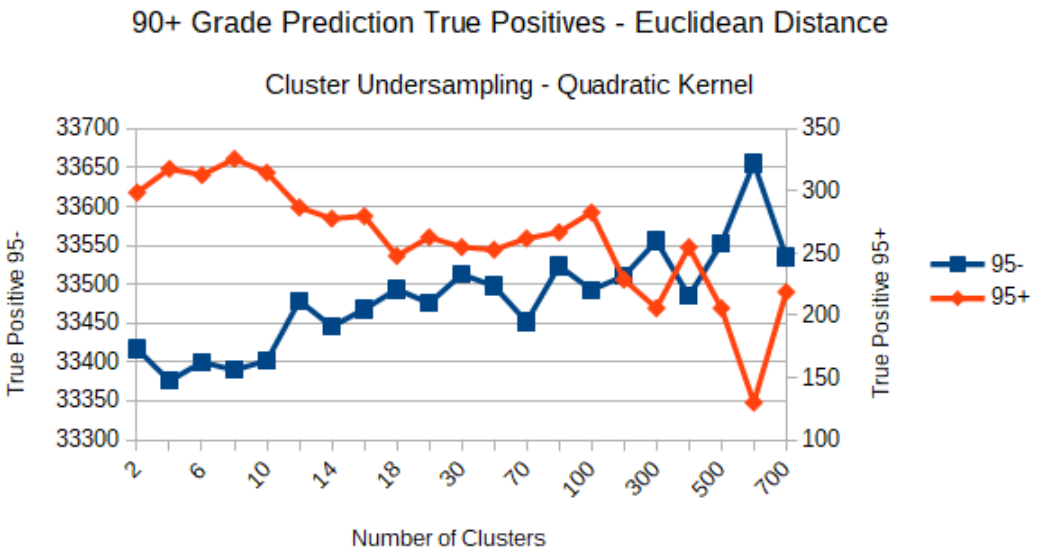


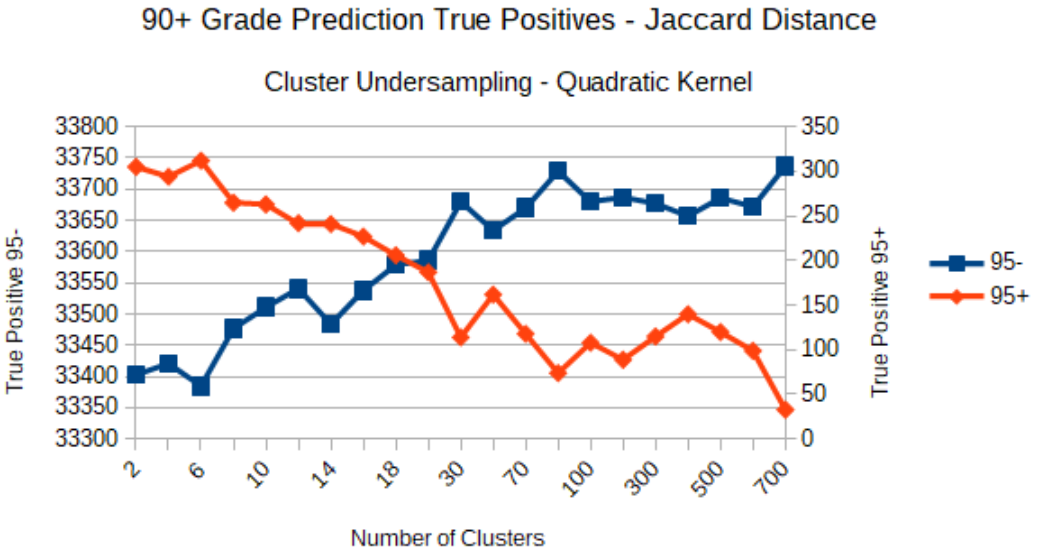
Figure 6.5: The number of correctly classified instances for each class as the number of clusters is varied using the representative clustering method with Jaccard distance. Note the axis scales.

6.4.4 Cluster Based Undersampling

In cluster based undersampling, the clusters made from the entire dataset are undersampled to balance out intraclass and interclass imbalances. In this method, the quadratic kernel performed the best. Unfortunately, just like with representative clustering, the mirror image problem remains, as shown in Figures 6.6 and 6.7. Neither the distance metric or the use of the class information to make the granules changed this problem in any way. Therefore, this method is also a failure.



(a)



(b)

Figure 6.6: The number of correctly classified instances for each class as the number of clusters is varied using the cluster undersample method with Euclidean and Jaccard distance. Note the axis scales.

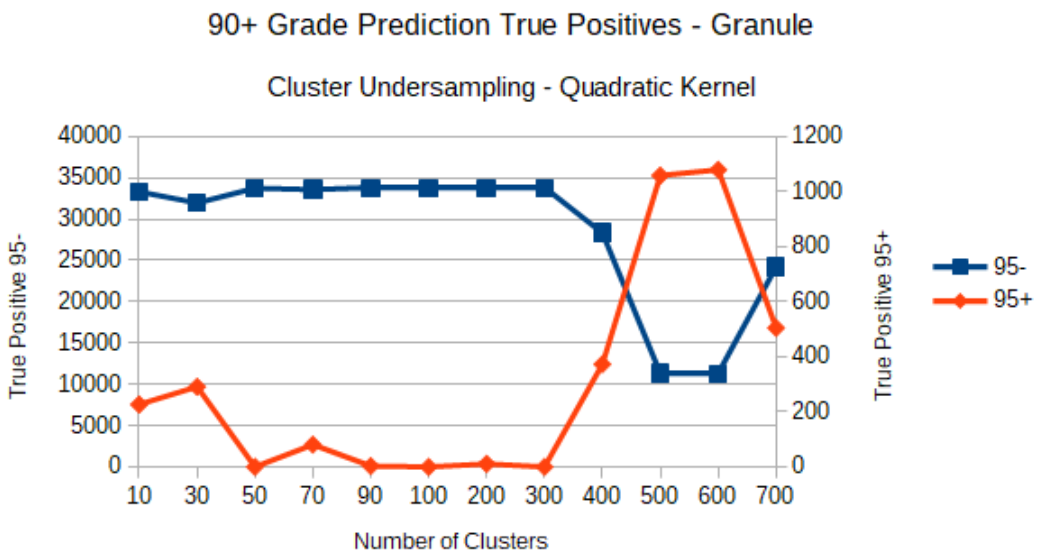


Figure 6.7: The number of correctly classified instances for each class as the number of clusters is varied using the cluster undersample method with Euclidean distance and granules. Note the axis scales.

6.4.5 SMOTE

The last two attempts are both SMOTE based, and we first try the vanilla SMOTE algorithm. The number of nearest neighbors was varied from 3 to 9 with a step size of 2. All of them performed similarly: between 1,060 and 1,070 minority instances were correctly classified and between 28,150 and 28,230 majority instances were correctly classified, with the SMOTE set to 3 nearest neighbors performing the best of all of them. These results were achieved with the linear kernel. These results show promise compared to the other methods, the precision and recall is 0.161 and 0.669, respectively. While two thirds of the minority instances were correctly classified, 84% of the instances that were predicted as minority instances were actually majority instances. Unfortunately, the other kernels continued to perform poorly.

Parameters	Majority Right	Minority Right	Precision	Recall
3, 21	30,335	832	0.195	0.525
7, 23	30,226	842	0.192	0.531

Table 6.1: The comparison of the two best borderline SMOTE methods.

6.4.6 Borderline SMOTE

The results for borderline SMOTE are reported last. Borderline SMOTE requires two parameters: the number of nearest neighbors to determine the border and the number of nearest on-the-border neighbors to generate synthetic examples between. These parameters were searched, although not exhaustively, with values ranging from between 3 and 23. The best results were attained when the number of nearest neighbors used to define the border remained low: between 3 and 7. Additionally, the number of neighbors to use to generate synthetic examples with remained high, around 21 and 23. The two best results used parameters of 3 with 21 and 7 with 23. Of the two, the first one performed slightly better on the minority class while the second one performed slightly worse, improving on the majority class. Neither classified as many minority examples correctly as vanilla SMOTE, but both classified many more majority instances right while still classifying many minority instances. The results are in Table 6.1. The linear kernel again performed the best.

6.4.7 Quartiles

All of these techniques attempt to deal with the class imbalance problem via resizing and balancing out the classes. Rather than balance out already imbalanced classes, it may be easier to solve the imbalance problem by simply changing the classifications themselves. Before, the classes of the wines were divided based on Wine Spectator's 100-point scale. Instead of using the groupings on this scale to classify wines, we can create our own groupings based on quartiles. This is the same method that was used to classify price. By using quartiles, the median of the wine grades is used to

Class	Grade Range
1	≤ 86
2	86-88
3	88-90
4	> 90

Table 6.2: The multiclass grade classification when using quartiles to create the groups instead of Wine Spectator's groupings.

Kernel	Accuracy
Linear	50.97%
Quadratic	43.58%
Cubic	44.86%
RBF	50.91%
Laplace	50.91%

Table 6.3: The results when using quartiles to define grade classifications.

form two groups, and the medians of those groups are used to create a total of four groups. In this way, four roughly equally sized classes are created. These new groups are shown in Table 6.2.

By changing the class definitions with quartiles, the classes become balanced. However, we can no longer compare between the 95- and 95+ classes since they no longer exist. Instead, we can compare the multiclass classification on the old classification scheme with the new classification scheme. These results are shown in Table 6.3. Clearly, these results are much worse than before with around 20% loss in accuracy compared to the old classification scheme. However, the classifiers are never suffering under the class imbalance problem: the loss in accuracy is purely due to the difficulty in distinguishing between the quartiles. Indeed, these results suggest that, when using quartiles, classifying grade is almost as hard as classifying price.

6.5 Remarks

Overall, the attempt to deal with the imbalance problem proved frustrating. No matter the method, a slight increase in performance on the 95+ class would result

in a major decrease in performance for the 95- class, forcing accuracy downwards by several percentage points. It was expected that some accuracy loss would ensue, but the magnitude of the loss was less than palatable. Of all the methods, borderline SMOTE seems like the best compromise, with SMOTE not far behind. Interestingly, the linear kernel tended to perform better than the RBF kernel, even though the RBF kernel had better performance on the regular dataset. This is probably due to the increased power of the model: the more flexibility the model had, the easier it was to ignore the minority class to optimize accuracy. While the methods tried here tended not to perform well, there are other methods that could be tried. One example is an algorithm specifically designed for SVMs [57]. Experimenting with these algorithms is left to future work.

While there are not too many conclusions that can be drawn from this chapter, other than that this class imbalance problem is difficult, there is evidence to suggest that it is difficult to tell the difference between the 95- and 95+ classes based only on the review attributes. It may very well be that wine ratings between the two classes are inconsistent: attributes that could lead to a 95+ wine may not always yield a 95+ rating.

Even so, the results when classifying grade based on quartiles seems to lean against this interpretation in part. The classifiers had great difficulty distinguishing between quartiles. This may be because the wine critics, by knowing the classes Wine Spectator uses, are influenced to adjust their scores to fit on the borders between the classes. In this way, using quartiles actually means that wines which are virtually identical in quality from the perspective of the wine critic are actually in different quartiles. For example, both an 89 and 88 wine are considered "very good" by the critic, yet end up in different classes when using quartiles. This would explain part of the poor performance on the quartile classification scheme. However, it may or may not suggest that there is inconsistency between the 95- and 95+ wines in terms

of their attributes. One would expect that there be a clear distinction between the classes, and yet, an attribute that a wine with a 95 rating may also be given to a 94 wine. But, because the wine critic knows that 95+ wines are considered classic while 95- wines are not, the critic may hesitate to assign a wine a grade higher than 94 even if it the attributes suggest it is deserving of the higher score. This would lead to great difficulties for the classifier in distinguishing between the two classes, with or without a class imbalance problem.

Chapter 7

Multi-Label Classification: Predicting Grade, Price, and Region Simultaneously

In prior chapters, each of the response variables were treated independently. That is, each classifier was given one task at a time: predict region, predict grade, or predict price, but do not predict two or all three of these at once. As one could image, predicting all three simultaneously is a more difficult problem than predicting one at a time. Nevertheless, the aim of this chapter is to do precisely that. This problem is not inherently different from what was done before: it is still supervised machine learning. To make this clear, let us review some of the different kinds of tasks that there are within supervised machine learning.

Normally, a prediction or classification problem focuses on determining a single variable. That variable may be a binary variable, or it may take on multiple values. When the possible values are numeric, the type of problem is regression. Conversely, when the possible values are categorical, the type of problem is classification. As we saw in the prior two chapters, if there are more than two possible values a categorical variable may take on, the problem is a multi-class classification problem. The focus of this chapter is when there are multiple response variables, however, and the type of problem is different entirely. While the problem for each variable on its own may still be considered regression, binary classification, or multi-class classification, the whole problem with multiple response variables is known as multi-target prediction [58]. There is a special case of this problem: when all of the response variables are categorical. If all of the categorical variables are binary, the problem is known as multi-label classification, since each response variable is also known as a label. If the categorical variables are not binary, it may be known as multi-class multi-label

classification, or just multi-target prediction, which is a term that also applies to the regression case [58]. This chapter will do both multi-label classification and multi-class multi-label classification.

7.1 Multi-Label Methodology

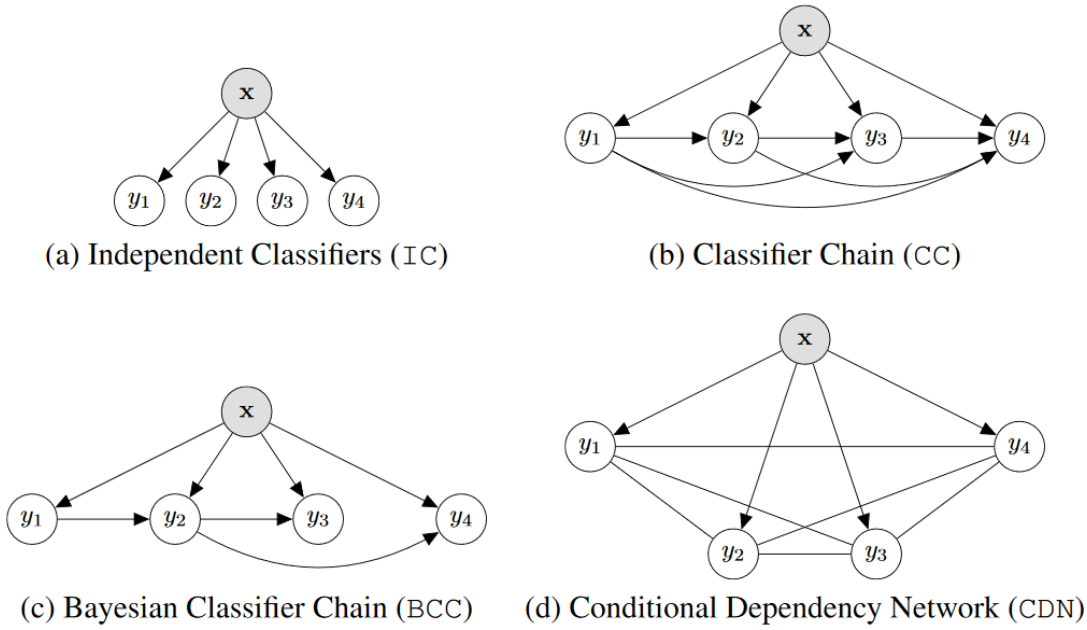
There are two general approaches to solve the multi-label problem: problem transformation and algorithm adaptation. Problem transformation manipulates the data set so that the multi-label problem becomes one or more single-label problems [46]. Algorithm adaptation is where the algorithm itself is able to handle the multi-label problem directly. It turns out that many, though not all, of the algorithm adaptation methods actually use problem transformation within them [59]. Therefore, most of the discussion on multi-label methods will revolve around how the data is transformed.

7.1.1 Naive Approaches and Binary Relevance

There are several problem transformation methods. The simplest, and least useful, is to remove all instances where there are multiple labels, creating a single-label problem with much information loss. Because all wines have multiple labels, this method cannot be used for this problem. The second simplest method is to treat each label independently, which we did as a baseline. If the data is split up correctly, this method can also be called Binary Relevance. This method is effectively the traditional classification problem, although applied multiple times to cover all of the labels.

In the binary relevance (BR) method, the original data set is split up into several data sets [59]. Each data set has all of the instances from the original data set and their attributes. What makes these data sets different from each other is that each

data set is going to correspond to a label. Thus, in addition to the columns needed to represent the instance attributes, there will be one column to hold the information about that particular label. For this case, that means there are 800 columns in each of these split data sets: 799 for the wine review attributes, and 1 for the label. The value in the label column will be a '1' or a '0', where a '1' means that instance has that label, and a '0' means that instance does not have that label. One data set each will correspond to the region, price, and grade label. If we consider region, for example, this data set has the 799 attribute column plus a column for the 'old world' label. Every wine that is 'old world' has a '1' in this column, and every wine that does not has a '0' in this column. Because a wine can either be 'old world' or 'new world', there is no need to have a separate data set for the new world wines. However, there are four classes each for price and grade. Rather than using binary relevance, a generalized approach must be used, called class relevance. Rather than training a binary classifier on each label, a multi-class classifier is trained on the multi-class labels. Because the binary/ class relevance methods essentially transform the data into several single label problems, the performance between the baseline method and binary relevance should be comparable.



(a) Independent Classifiers (IC)

(b) Classifier Chain (CC)

(c) Bayesian Classifier Chain (BCC)

(d) Conditional Dependency Network (CDN)

Figure 7.1: A graphical comparison of different multi-label methods. Here, x are the attributes, and each label is y_n.

7.1.2 Label Powerset

The second problem transformation method is the label powerset (LP) method [46]. In this method, a new label is created. The value of this label is built from a combination of the original labels. An old world wine of outstanding quality that costs $51 would be given a label value that corresponds to 'old world and outstanding and price quartile 4', where a new world wine of outstanding quality that costs $12 would be given a label value that corresponds to 'new world and outstanding and price quartile 1'. The total number of classes in this new label is equal to the number of classes from each dependent variable multiplied together. In the two class problem for grade and price, each label is 2 class, so there would be a total of 8 classes. Having four classes each for grade and price makes the problem more difficult, with 32 possible combinations. A multi-class classifier, such as an SVM using the one versus one approach, is then trained on this new label.

7.1.3 Classifier Chains

The third problem transformation method, classifier chains (CC), attempts to preserve dependencies between the dependent variables while still using binary relevance [60]. In the first stage of a classifier chain, a regular classifier is built for one dependent variable, like in a normal single-label problem. This classifier will make predictions for what the dependent variable should be for any given instance. For the second stage of the classifier chain, the first stage predictions are made an independent variable, an input, to be used to predict the second dependent variable [60]. Thus, the second stage classifier uses both the attributes and the first dependent variable to make predictions about the second dependent variable. The third stage classifier would use the attributes, the first dependent variable, and the second dependent variable to make predictions about the third dependent variable, and so on. A visual of the classifier chain is shown in Figure 7.1 (b) [61].

How this operates depends on if one is training or testing. In both training and testing, the first stage of the classifier chain trains as you would expect for a normal single label problem. In training but not in testing, the second stage classifier uses the known, correct values of the first label. In testing, the second stage classifier takes the testing output from the first stage classifier to fill in the values of the first label. Thus, while in training it would appear not to matter what order the classifier chain is in, in testing it becomes obvious that the order does matter. While there are ensemble methods to determine the correct order, the data set is too large for this to be computationally feasible for the amount of resources we have to work on this problem. Thus, we will simply choose a descending order of prediction accuracy for the classifier chain. That is, the label we can independently predict most accurately in a single-label setup will be in stage 1, the second most accurate in stage 2, and so forth. To compare the performance of this order, we will also choose one other order at random.

7.1.4 Other Chain-Like Methods

Bayesian Classifier Chains (BCC) are another classifier chain method [62]. The basic difference between the BCCs and CCs is that BCCs are not fully parameterized while CCs are [61]. That is, BCCs do not necessarily use all the labels in the previous stages to classify the next staged label. This is shown in Figure 7.1 (c) [61]. The sparsity in the connections between labels leads to a faster classifier, and potentially a more accurate one, as labels which do not have dependencies between them need not be linked [61].

Another approach uses the Classifier Trellis (CT) [61]. A CT forms a directed graph between the labels in a trellis structure, where no directed loops can exist [61]. The structure is defined and fixed at the beginning of the problem, so that the computational complexity involved with finding the right structure in CCs does not occur with CTs. In the trellis, shown in the figure, label dependence goes from the top left to the bottom right. Since this structure is fixed, one only needs to decide which labels to put where in the trellis. This is accomplished using a label-frequency based pairwise heuristic [61]. A random label is initially placed in the upper left corner of the trellis [61]. After that, the heuristic is used to place the remaining labels. Classification then takes place as it does with classifier chains, except that the dependency structure is based on the trellis and not a chain. An example of a classifier trellis on nine labels is shown in Figure 7.2 [61]. The classifier trellis is designed to be computationally efficient even when handling large numbers of labels. Because it is designed for performance at a large scale, we may not see performance increases on our data set versus other methods, since there is only three labels.

The last approach used is the Conditional Dependency Network (CDN) [63]. A CDN is made by forming an undirected, fully connected graph between the labels, shown in Figure 7.1 (d) [61]. Unlike classifier chains, CDNs do not try to approximate the dependence of one label on another by using several probabilistically correct

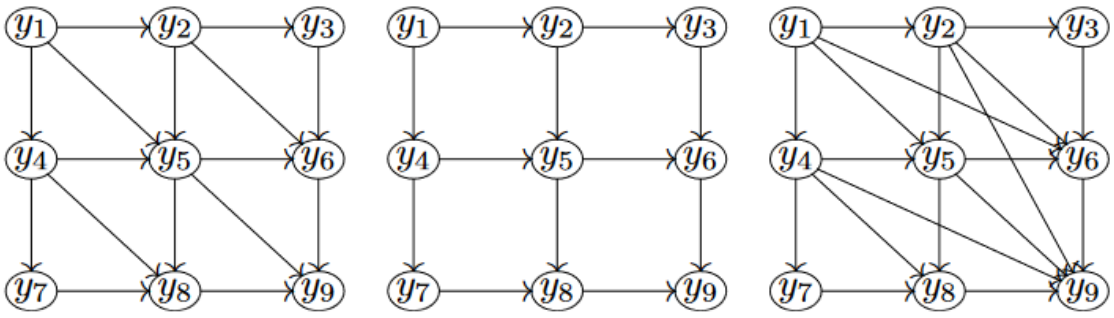


Figure 7.2: Three possible trellis structures for nine labels.

classifiers chained together [61]. Instead, CDNs encode the dependencies between the different labels. This is done by training as many binary classifiers as there are labels, where each classifier defines a conditional probabilist distribution on one label given all the other labels and the input attributes [63]. Thus, there is a conditional probability distribution built for each label [63].

7.2 Evaluation Metrics

Unlike single-label data, multi-label data may have different degrees of how many labels are contained within them [59]. The two metrics for determining this are label cardinality and label density. The first metric is the average number of labels in the instances. The label density is the label cardinality divided by the total number of labels. Data sets with the same cardinality but different density may behave differently. For example, a dataset with higher density suggests that there are more instances assigned to more items than a dataset with lower density, and it may therefore be harder to predict every single label that an instance has. It could also mean that there is less variance within the data, which could in turn lead to easier predictions. Regardless, not all multi-label data sets are created equal, and care should be taken when comparing the performance of classifiers accordingly.

Since every instance has a constant number of labels, there is no need to worry

about predicting the correct number of labels for an instance. Instead, predicting the right labels for that instance is the only thing to this problem is concerned with.

With an understanding of how data may have different degrees of how multi-label they are, it makes sense to now consider the model evaluation metrics. The first model evaluation metric we use is hamming loss, which is the analog to error in a single-label problem [46]. It is defined as

$$HammingLoss = \frac{1}{|D|}\sum_{i=1}^{|D|}\frac{|Y_i \Delta Z_i|}{|L|}$$

where D is the multi-label data set, Y_i is the true label set of the i^{th} observation in D, and Z_i is the predicted label set of the i^{th} observation in D, and L is the set of possible labels [59]. For each instance, hamming loss measures the symmetric difference between the predicted label set is from the true label set, normalized based on the number of labels. The lower this difference, the better the performance [46].

Another loss function is zero-one loss, which is a measure based on exact matches between the true and predicted label set. For each observation, this measure compares both the predicted label set and true label set. If they are not an exact match, the loss for that observation is one. If they are an exact match, the loss is zero. Specifically, zero-one loss is defined as

$$ZeroOneLoss = \frac{1}{|D|}\sum_{i=1}^{|D|}\begin{cases} 1, & Y_i = Z_i \\ 0, & Y_i \neq Z_i \end{cases}$$

where D is the multi-label data set, Y_i is the true label set of the i^{th} observation in D, and Z_i is the predicted label set of the i^{th} observation in D. Zero-one loss is clearly a stricter metric than Hamming Loss. It is suitable for when a model models dependence between labels, where Hamming Loss is not directly improved by a model which incorporate labels dependence [64].

Accuracy can also measure model performance. There are two versions of accuracy: overall accuracy and per-label accuracy. Overall accuracy, also known as Jaccard index, is defined as

$$Accuracy = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|}$$

where D is the multi-label data set, Y_i is the true label set of the i^{th} observation in D, and Z_i is the predicted label set of the i^{th} observation in D [59]. The Jaccard index calculates the average of the number of times the true labels and predicted labels for an instance intersect divided by the number of true and predicted labels for that instance. In other words, for each observation, this measures how many labels are common between what is predicted and what is true, then divides that by how many different truths and predictions there are. Overall accuracy may also be given a forgiveness factor by raising this ratio to some exponent other than 1. This allows for small errors to not be penalized as much and may be useful if there are a large number of labels. It should be noted that this measure of accuracy is not directly related to loss, unlike in a single-label problem where accuracy and error are directly related.

Precision and recall can also be used to evaluate a multi-label problem. Precision measures, on average, how much is in common between the predicted set and true label set divided by the size of the predicted set, where recall measures, on average, how much is in common between the predicted set and true label set divided by the size of the true label set. The numerator in both of these metrics, like accuracy, counts the number of correctly classified labels for an instance [59]. In precision, the denominator counts the number of predicted labels. Thus, by dividing the number of correctly classified labels by the number of predicted labels, precision measures how many of the predicted labels were relevant or useful. In recall, the denominator

counts the number of true labels for an instance. By dividing the number of correctly classified labels by the number of true labels, recall measures how many of the correct labels the classifier actually selected. The F1 score combines both precision and recall into a single metric by taking the harmonic mean of both, shown by this formula:

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Other metrics, such as coverage and ranking loss, are useful where labels are ranked or the classifier outputs confidence for a specific label [46]. This problem has neither ranked labels nor will the SVMs used output confidence, so these metrics are not used here.

7.3 Implementation

The single-label results can serve as the baseline to compare the multi-label results to. In Chapter 5, we used the LibSVM implementation from within WEKA (the Waikato Environment for Knowledge Analysis) [51] [48]. For the multi-label approach in this chapter, we used LibSVM from within MEKA (a Multi-Label Extension for WEKA) [65]. In all cases, C-classification was used, the cost of the constraints violation parameter was set to one, and five fold cross validation was employed. Based on the results in Chapter 5, the linear and radial basis function (RBF) kernels were used again as they provided the best results. Using the polynomial kernels with degrees 2 and 3 (quadratic and cubic kernels) performed worse, so we will not report them in the results. A comparison between the kernels for one multilabel approach is shown in Figure 7.3 Top, while the same comparison for the single label approach is shown in Figure 7.3 Bottom.

The multi-label evaluation metrics were provided by MEKA. However, MEKA did

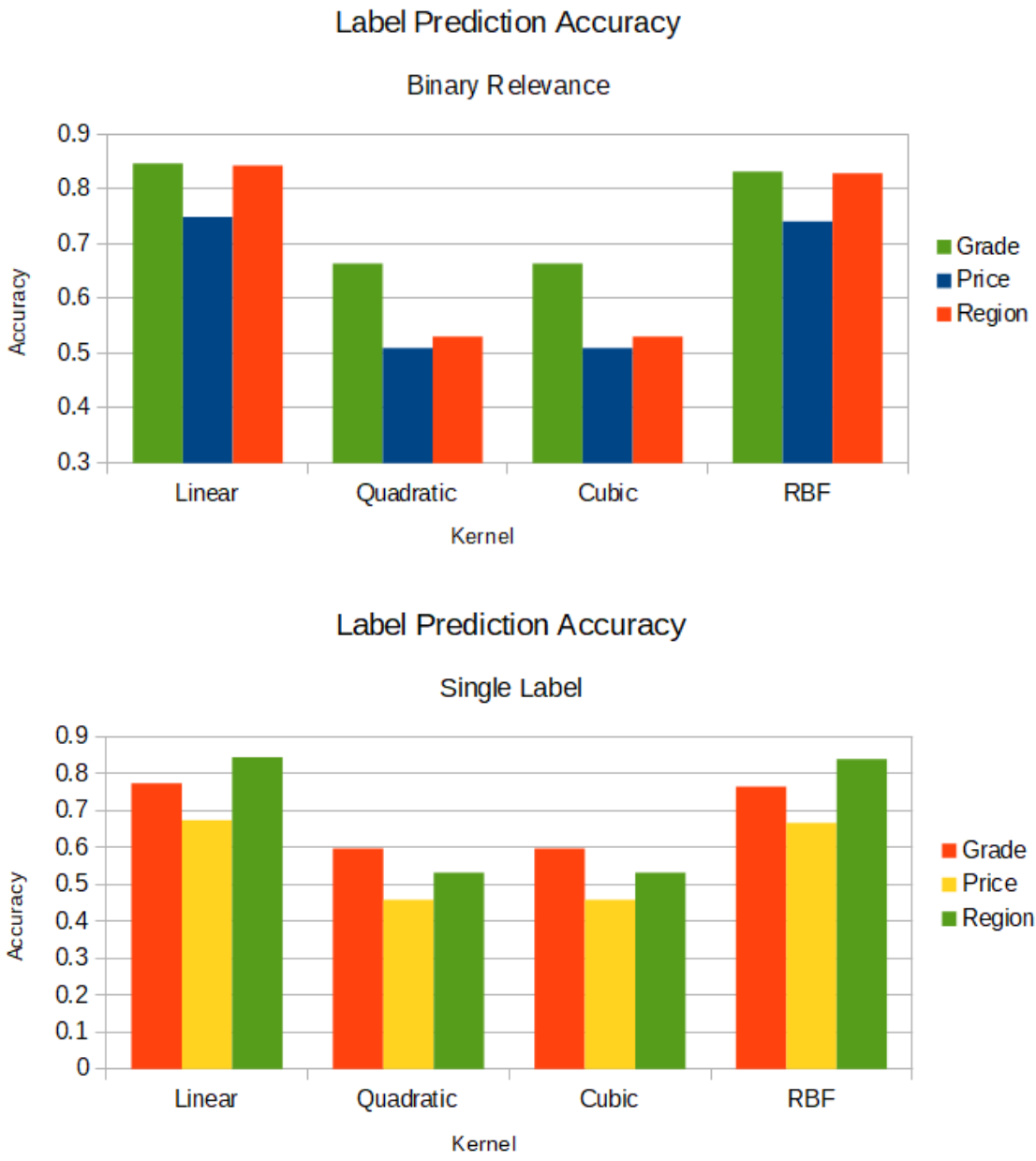


Figure 7.3: Top: The multi-label accuracies for binary relevance using the 2 class dataset. The Quadratic and Cubic kernels perform poorly. Bottom: The results for single label accuracy using the 2 class dataset.

not provide Jaccard index or F1 score for the multi-target problem. Therefore, these metrics are omitted in our results for these experiments. Additionally, WEKA could not provide the multi-label evaluation metrics because it is inherently single label.

A final limitation to note is that MEKA does not have multi-target extensions for every multi-label algorithm. For this reason, some algorithms cannot be applied to the multi-class multi-label problem without significant performance penalty. Label powerset should not need a multi-target extension, as the additional class values would only result in an increased number of classes that label powerset can form. However, neither the classifier trellis nor the conditional dependency network have multi-target extensions, either. The results for these algorithms will still be reported, with the caveat that any deficiencies in the accuracy may be caused by the implementation and not the method.

Several of the multi-label and multi-target implementations have parameters. Except where noted here, we used the defaults (or, when only one value value was appropriate for this dataset, that value was used instead). For classifier chains, the random seed was used to control the chain order. For conditional dependency networks, we reduced the number of iterations from 1,000 to 100 to reduce the runtime. This last part was necessary even though the performance was possibly impacted: the run time for CDNs was a couple of days with only 100 iterations, and it would have been even longer if the number was 1,000.

7.4 Results

7.4.1 Two Class Results

First, we consider the data set with 2 classes each for the labels grade, price, and region.

Method	Kernel	Grade	Price	Region	Ham Loss	0/1 Loss	Jac Idx	F1 Score
Single Label	Linear	0.859	0.749	0.844				
	RBF	0.849	0.741	0.829				
Binary Relevance	Linear	0.847	0.749	0.843	0.187	0.451	0.676	0.787
	RBF	0.832	0.741	0.829	0.199	0.472	0.655	0.768
Label Powerset	Linear	0.849	0.753	0.851	0.182	0.423	0.691	0.794
	RBF	0.824	0.736	0.824	0.205	0.458	0.654	0.760
Classifier Chain	Linear	0.847	0.732	0.844	0.193	0.452	0.665	0.779
Grade, Price, Region	RBF	0.832	0.722	0.830	0.206	0.474	0.641	0.758
Classifier Chain	Linear	0.847	0.733	0.843	0.197	0.452	0.665	0.779
Region, Grade, Price	RBF	0.832	0.723	0.829	0.205	0.474	0.642	0.759
Bayesian	Linear	0.847	0.749	0.843	0.187	0.451	0.677	0.787
Classifier Chain	RBF	0.832	0.742	0.829	0.199	0.471	0.656	0.769
Classifier Trellis	Linear	0.847	0.732	0.844	0.192	0452	0.665	0.779
	RBF	0.832	0.722	0.829	0.206	0.474	0.641	0.758
Conditional	Linear	0.837	0.725	0.844	0.198	0.458	0.653	0.766
Dependency Network	RBF	0.812	0.710	0.830	0.216	0.486	0.624	0.737

Table 7.1: Results for the two class dataset. The best multi-label method is the linear kernel label powerset, beating all the other linear kernel methods, except for in Grade versus the single-label approach.

Single Label Results

We use the single label approach as a baseline. The results are good: using the linear kernel, accuracies of 84.4%, 85.9%, and 74.9% for region, grade, and price, respectively were achieved. Consider that if random chance were to be used, we would expect per-label accuracies of 50% each. The RBF kernel does not perform as well with an accuracy of 82.9%, 84.9%, and 74.1% for region, grade, and price.

Multi-Label Results

In all of the multi-label methods, the linear kernel performed the best, with the RBF kernel closely following. Occasionally the multi-label linear kernel classifiers would outperform the single label approach, but often only in one of the labels. The results are shown in Table 7.1.

Binary Relevance Because of the way Binary Relevance does the problem transformation, we expect the results to be similar to the single label results. Indeed, the two class results remain competitive for two of the three labels, suffering a 1.2 percent degradation in accuracy for grade. This degradation is shared with all the multi-label methods and not specific to the binary relevance.

Label Powerset Of all the multi-label methods, label powerset performs the best, outperforming the single label approach in price and region when using the linear kernel. The linear kernel label powerset performs the best in all metrics versus all other tried multi-label methods and it is the only classifier to outperform the linear single label method in Price and Region.

Classifier Chains, CT, and CDN Because classifier chains depend on order, we tested two different chain orders. The first chain order is Region, Grade, Price. The other order we compared is Grade, Price, Region. Interestingly, there was not a significant decrease in accuracy between the different orders. This is likely because there is a dependency between grade and price, and another dependency between price and region. In addition, the data has a high dimension in number of attributes, so the influence from a single extra dimension added on at each stage of the chain is dwarfed by the influence of all the other dimensions. Also, it is interesting to note that the performance in Price was less than binary relevance or label powerset, while the other two labels remained competitive.

The Bayesian Classifier Chain improved on the Classifier Chain and was able to maintain performance in all three labels. Because of this, BCC was able to outperform the standard classifier chain in the multi-label metrics. Also the order of the chain did not matter: the algorithm was able to identify the correct chain order regardless of our settings, and the performance was identical no matter the settings we used for chain order.

The Classifier Trellis has performance nearly identical to the Classifier Chain, even barely outperforming it in Hamming Loss. while the Conditional Dependency Network performs worst of all the tried methods. Conditional dependency networks remain competitive on the region label, but suffer in the other two labels.

Summary

Multi-label methods tend to be less successful than the single label method for this problem, with the exception of the label powerset method. Region tends to be the easiest label to predict, with accuracies between 82.9% and 85.1%. Wine quality is similar to region, although a bit more challenging for multi-label to predict. Wine price is the hardest to predict, with accuracies ranging from 71% to 74.9%. Despite single label besting most other approaches, the label powerset is able to outperform the single label approach.

7.4.2 Four Class Results

We now compare the results when using the dataset with two classes for region and four classes each for the grade and price labels. When multi-target extensions to the multi-label algorithms exist, the multi-target results are reported instead. Otherwise, the multi-label results are reported, often with reduced accuracy. The results are shown in Table 7.2.

Single Label Results

As with before, the linear kernel performs best: 84.4% accuracy for region, 75.1% accuracy for grade, and 47.1% accuracy for price. For comparison to random chance, we would expect there to be 25% accuracy for grade and price and 50% accuracy in region.

Multi-Target Results

In all of the multi-label methods, the linear kernel still performed the best. This is to be expected, considering the results for the two class dataset are better with a linear kernel.

Class Relevance Class relevance is the multi-target extension to binary relevance. This approach performs identically in per-label accuracies when compared to the single label approach. It has average Hamming Loss, but the second worse Zero-One Loss.

Label Powerset Label powerset unfortunately suffers degraded performance in grade and price, but remains competitive in region. This is due to the increased number of classes that must be formed from the greater number of label combinations found in the four class dataset. When there are more classes, the probability of error increases. While LP performed the best with two classes, it now performs the worst with multiple classes. Even so, it has the highest performance in predicting Region.

Classifier Chains, CT, and CDN Classifier chains continue to perform well when given the multi-target dataset. Like in the two class problem, the order of the classifier chain shows some, but minor, differences, with a chain order of region, grade, price beating the chain order of grade, price, region in zero-one loss only by tenths of a percent.

The Bayesian classifier chain was able to outperform the regular classifier chain, like in the two class problem. The order of the chain set by parameters was also unimportant: the algorithm decided upon the same order that was best regardless. Although the zero-one loss is slightly higher compared to regular classifier chains, the per-label accuracy and Hamming loss are improved.

Both the Classifier Trellis and Conditional Dependency Network have average

results compared to the other methods, with the classifier trellis performing better between the two of them. Although not as good as the classifier chain in per-label accuracies, the Classifier Trellis get very close in Hamming Loss and zero-one loss, especially with the linear kernel. Despite the CDN having a poorer Hamming Loss, it is able to remain competitive in Zero-One Loss.

Summary

The single label approach still remains difficult to beat. While most algorithms are able to tie its performance in Grade and Region, the same is not true for Price. In our case, the Bayesian Classifier Chain is best able to scale up to the multi-target approach compared to the other methods we tried. It achieves the best Hamming Loss, ties the single label approach in Grade and Region, and outperforms the single label approach in Price.

Method	Kernel	Grade	Price	Region	Hamming Loss	0/1 Loss
Single Label	Linear	0.752	0.470	0.844		
	RBF	0.702	0.448	0.829		
Class Relevance	Linear	0.752	0.470	0.844	0.312	0.697
	RBF	0.702	0.449	0.829	0.340	0.737
Label Powerset	Linear	0.583	0.298	0.849	0.423	0.818
	RBF	0.544	0.247	0.821	0.463	0.864
Classifier Chain	Linear	0.752	0.463	0.843	0.314	0.693
Grade, Price, Region	RBF	0.702	0.432	0.829	0.346	0.738
Classifier Chain	Linear	0.752	0.464	0.844	0.314	0.692
Region, Grade, Price	RBF	0.702	0.435	0.829	0.344	0.736
Bayesian	Linear	0.752	0.473	0.844	0.311	0.694
Classifier Chain	RBF	0.702	0.449	0.830	0.340	0.736
Classifier Trellis	Linear	0.752	0.463	0.843	0.314	0.693
	RBF	0.702	0.432	0.829	0.346	0.738
Conditional	Linear	0.739	0.459	0.843	0.319	0.694
Dependency Network	RBF	0.680	0.420	0.830	0.357	0.744

Table 7.2: Results for the four class dataset. The best multi-label method is the linear BCC method, beating or tying all the other multi-label linear kernel methods in per-label accuracy.

7.5 Remarks

Treating each label independently is usually the most successful approach to working with this data when considering per-label accuracy. This is to be expected when each label is important on their own, and no care is given to getting multiple targets correct simultaneously. Although it is difficult for the multi-label/ multi-target approach to tie, let alone outperform, the single label approach in per-label accuracy, the Bayesian Classifier Chain is able to do this for both of our data sets. While Label Powerset performs the best for the two class data, the BCC remains the most reliable overall.

In this chapter, we have extended the results of Chapters 4 and 5 by showing that it is possible to simultaneously classify where a wine originated stylistically, how good it is, and how expensive it is. Although the single label results are usually better, the multi-label approach is not a failure.

Chapter 8

Predicting the Actual Grades and Prices with Regression

Throughout this work, the task has been the classification of wines into different categories. Of course, classification is the only option on the region label. However, in grade and price, we have the real numeric values for each wine. Rather than classifying these wines into different grade and price categories, this chapter aims to predict the actual numeric values for grade and price. In addition, we will map these predictions back to the classes used in the prior chapters in order to directly compare the results of this chapter with the remainder of the work.

8.1 Evaluation Metrics

There are several metrics that can be used to evaluate a regression model. Most of these metrics are based on the concept of error. A regression model predicts an actual number that it thinks an instance has based on its attributes. The difference between the prediction and the actual value is called the error. If the error is positive, that means the model predicted a value higher than the actual value. Similarly, if the error is negative, the model predicted a value lower than the actual value. The average of all of these error is known as the Mean Error (ME). This value shows the average magnitude the model was off over all the instances.

Unfortunately, a ME of zero does not mean the model is perfect. In fact, because positive and negative numbers cancel out when averaging them, the ME can mask significant problems with the model. A better metric is to consider the Mean Absolute Error (MAE). This metric sacrifices the knowledge of how far high or low the model

tends to predict in exchange for information about how much error accumulated from all of the instance predictions. To compute the metric, one simply takes the absolute value of the error before adding it into the average.

Another metric is the Mean Squared Error (MSE). Like the MAE, MSE considers how much error accumulated overall. Unlike MAE, MSE works by squaring the values instead of taking the absolute value. This has the disadvantage of being sensitive to outliers, which will be discussed in the section below. However, MSE is better than MAE in detecting when a few, large errors are made (while MAE is better for the exact opposite problem). Since simply squaring the errors results in enormous values, it is common to take the square root after the average is done, resulting in Root Mean Squared Error (RMSE).

The MAE and RMSE give us measures for absolute error: that is, they give us a real number which reflects the actual value of error. This value is not normalized or compared to the range of possible values that may be taken on. Because the errors are in an absolute sense, it is not clear if any given error is good or bad. To understand this, consider a regression model which performs with an MAE of 0.1 units. This may sound like a small error, but because there is no comparison to the actual value ranges in the data, it is impossible to determine if the error is good or bad. If the possible numeric values range from 1 to 100, then the error is quite good. On the other hand, if the possible numeric values range from 0 to 0.3, then an error of 0.1 is so large that predictions from this model are probably useless. Normalizing the MAE and RMSE allows us to better understand how good or bad the model performs.

There are two ways to normalize the error. The first way is to take the error and divide it by the range of possible values. This can be expressed by the formula

$$\text{Minimax Normalized Error}_y = \frac{\text{Error}_y}{\max(y) - \min(y) \cdot 100},$$

where y is the variable being predicted. Multiplying by 100 transforms the relative error into an easy to interpret percentage value. Normalizing this way makes sense when clear boundaries can be established for the maximum and minimum values. This is the case for grade: the minimum and maximum grades in the dataset is 80 and 100, respectively. For price, however, this method is not as sensical. While we can clearly define a lower price of \$0, there is no limit to how high a wine may cost. Of course, we could use the highest price found in the dataset. However, this may be deceiving for two reasons. First, the price dataset contains outliers (which is described below). Second, because of the presence of outliers, it is likely that a new dataset could contain wines with a price higher than the wines found in the dataset used here. Third, because wines may take on a wide range of prices, with only a relatively few number of wines being expensive, it is more appropriate to consider how the model performs on the majority of the wines: the wines that do not cost near the maximum or minimum price. A more reliable normalization technique, one that is both robust to outliers and can describe the bulk of the data, not just the extremes, is needed. Fortunately, we have such a metric: the standard deviation. Rather than taking the error and dividing it by the range of the data, dividing the error by the standard deviation makes the normalized error more robust to outliers and removes the need for us to know what the range of prices could be. This formula is

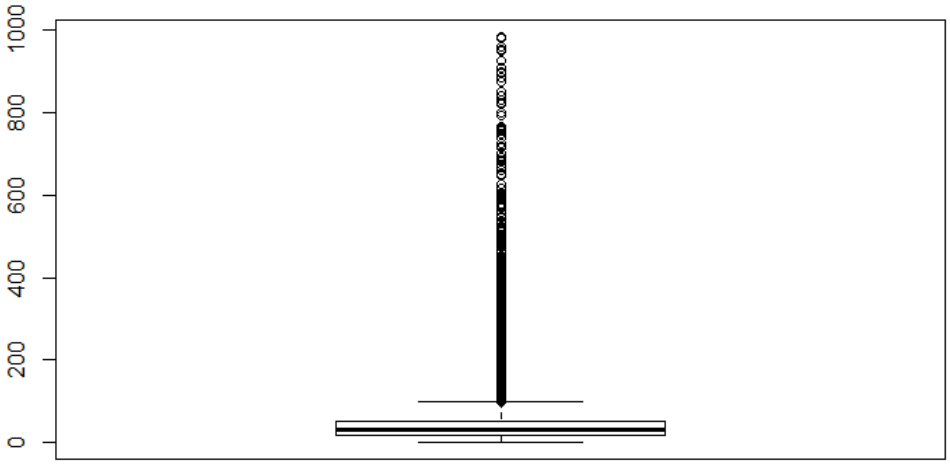
$$\text{St. Dev. Normalized Error}_y = \frac{\text{Error}_y}{\sigma_y} \cdot 100,$$

where y is the variable being predicted and σ_y is the standard deviation of the variable.

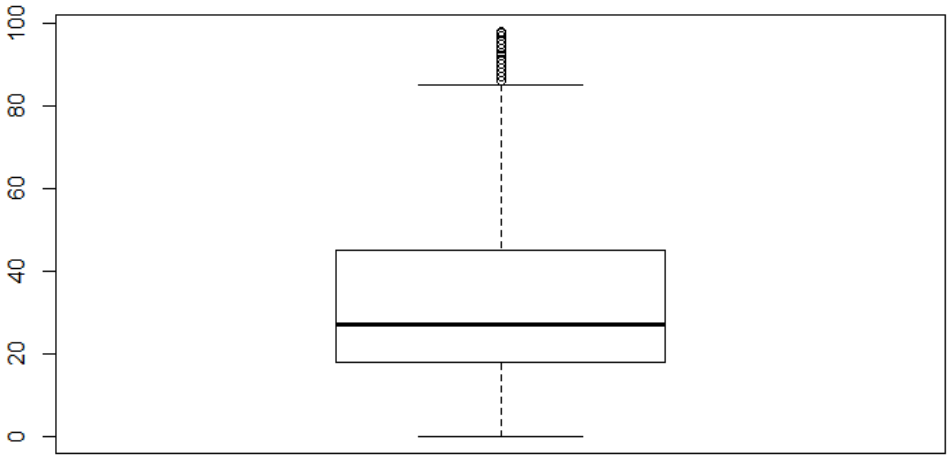
From these considerations, the minimax normalized error will be used for grade and the standard deviation normalized error will be used for price. This normalization will be applied to MAE and RMSE to form NMAE and NRMSE. Note that it does not make sense to normalize MSE in this way due to the difference in units. Although

one could use variance to normalize MSE, we will refrain from doing so.

Another metric is the coefficient of correlation (r), which measures how strong the attributes and response variable is related as well as providing the type of relationship. Values may be between -1 and 1, where a zero means there is no relationship. A value of 1 suggests a perfect positive relationship and a value of -1 suggests a perfect negative relationship. By squaring the coefficient of correlation, we get the coefficient of determination (r^2). The coefficient of determination measures how much variance between the predicted and actual values is explained by the model, where 0 means none of the variance ie explained by the model and a value of 1 means all of the variance is explained by the model.



(a) Boxplot with outliers



(b) Boxplot without outliers

Figure 8.1: Boxplots of price before and after outliers are removed.

8.1.1 Outliers

Considering the distribution of prices, there are many extreme values. This will pose a problem when considering metrics sensitive to outliers, such as mean squared error. To deal with this, we removed all wines with prices higher than $98: the upper whisker in the price boxplot in Chapter 2, shown again in Figure 8.1 (a). It turns out that there are exactly 6,800 wines with prices higher than $98. By removing wines with these prices, the boxplot becomes much less dominated by extreme values, as shown in Figure 8.1. This will allow us to evaluate our regression model without the influence of overly expensive wines. Due note, however, that since the median price of classic quality wines is $112, which is higher than $98, the model will have less than half of the highest quality wines incorporated in the model.

8.2 Implementation

Support vector machines are inherently binary classifiers: they draw a hyperplane between two classes of data. However, the notion of drawing a hyperplane does not need to be restricted to separation of data: one could draw a hyperplane that follows a trend in the data. That is precisely the purpose of Support Vector Regression (SVR), as detailed back in Chapter 3. Fortunately, the Kernlab implementation of SVMs also allows them to operate in regression mode. For this reason, they will be our choice of implementation for this chapter.

Unlike with classification, the costs of constraints violation is not a hyperparameter that is set in regression mode. Instead, an analog of this hyperparameter called epsilon is used, which was set to a value of 1. Otherwise, the hyperparameters remained the same: five fold cross validation and default kernel parameters were used. To speed up the computations, the polynomial kernel was omitted (the performance was found to be significantly worse than the other kernels

after a single of the cross validation folds was run, and the time it took for the polynomial kernel to run was over an order of magnitude longer).

In order to compute the evaluation metrics, the gof function from the R package hydroGOF was used [66].

8.3 Results

Let us first consider the results for running SVR on grade, shown in Table 8.1. All three models report similar levels of error, with the two RBF models tying for first place. The zero value for ME suggests that the RBF/Laplace models are able to find a perfect balance for where to draw the hyperplane. The low value for RMSE suggests there are not many large errors, rather, there are many smaller errors, which is what we expect with a good regression model. The normalized errors are also small, suggesting that the model can predict within about 8% of the actual value of the instance. The coefficients of correlation and determination are also good values, suggesting a strong positive relationship and suggesting the model can explain the majority of the variance between the independent and dependent variables.

Next, consider the results for running SVR on price, shown in Table 8.2. There is clearly evidence of outliers: the MSE values are very large, as such the RMSE is over twice the value of MAE, suggesting that there are many at least several large errors. In addition, the ME is very negative, suggesting that the actual values are higher than the model predictions. This would be the case for large positive outliers. Note

Kernel	ME	MAE	MSE	RMSE	NMAE	NRMSE	r	r^2
Linear	-0.07	1.64	4.27	2.07	8.2%	10.4%	0.73	0.54
RBF	0.00	1.59	4.02	2.00	8.0%	10.0%	0.75	0.56
Laplace	0.00	1.59	4.02	2.00	8.0%	10.0%	0.75	0.56

Table 8.1: Table of results for the regression model on Grade. Normalized error is calculated using the minimax method.

Kernel	ME	MAE	MSE	RMSE	NMAE	NRMSE	r	r^2
Linear	-9.56	20.53	2082.15	45.63	41.8%	93.0%	0.44	0.19
RBF	-9.88	20.52	2133.56	46.19	41.8%	94.2%	0.43	0.18
Laplace	-9.88	20.52	2133.56	46.19	41.8%	94.2%	0.43	0.18

Table 8.2: Table of results for the regression model on Price. The normalized error is calculated using the standard deviation method.

that the normalized errors are not comparable to the normalized errors for grade since the method of calculation is different (indeed, if the method used here were minimax, the normalized errors would be around 2-4%). This interpretation for NMAE is that the predicted price is almost half a standard deviation away from the actual value. Similarly, the NRMSE says that the predicted price (when squaring and rooting the error) is almost a whole standard deviation away from the actual value. Both of these values suggest there is some room for improvement: a large number of wines can be contained within a standard deviation of the mean price point.

Of the three kernels, the linear model, being less sensitive to outliers, is reporting better results than the RBF/Laplace models. By removing the outliers, significantly better results are attained, shown in Table 8.3, allowing the RBF and Laplacian kernels to retake the lead. As for actual performance, the ME decrease to a third of its original size. Although it is still negative, suggesting there may still be some outliers in the new dataset, the RMSE is much smaller than before. In addition, the RMSE is no longer over twice as large as the MAE, which also decreased. Unfortunately, the NMAE increases, while, interestingly, the NRMSE decreases. This can be explained by the standard deviation being cut by nearly half, from around $40 with outliers to around $20 without outliers. The NMAE is not as sensitive to the removal of outliers as NRMSE, but both are equally sensitive to the change in standard deviation, leading to one increasing while the other decreases. Making the transition from a dataset with outliers to a dataset without outliers justifies the use of standard deviation for computing normalized error: the normalized minimax errors on the

Kernel	ME	MAE	MSE	RMSE	NMAE	NMRSE	r	r^2
Linear	-3.80	13.07	325.25	18.03	64.3%	88.7%	0.50	0.25
RBF	-3.75	12.94	318.64	17.85	63.6%	87.8%	0.51	0.27
Laplace	-3.75	12.94	318.62	17.85	63.6%	87.8%	0.51	0.27

Table 8.3: Table of results for the regression model on Price without outliers. The normalized error is calculated using the standard deviation method.

dataset without outliers would be around 13-17%. The large drop in price range drives up the normalized error, even though the model performs better without outliers. Improved performance is clear to observe by considering the coefficients of correlation and determination: they have both increased by a good amount, suggesting a much stronger relationship and a model which explains more of the variance than before.

It would be nice to compare the regression results to the classification results. We can do this by checking to see if the predicted value from the regression model can be converted to the same class value that the actual value would be converted to. For example, if the actual grade is 84 and the predicted grade is 82, both of these can be mapped to the "Good" grade category, and so the regression model would be considered to have classified this instance correctly. Conversely, if the actual grade is 84 and the predicted grade is 85, the regression model is considered incorrect, even though the error is smaller, because an 85 grade wine is a "Very good" wine. This means that errors along borderline wines are going to cost the regression model more than errors along non-borderline wines. The regression models provide more information than the classification models, so the use case for a regression model is different than for classification. For these reasons, a lower classification accuracy from the regression model does not at all imply that the model is any worse. In fact, it is actually better for the regression model to perform worse at the classification task in exchange for better metrics found in the tables above.

The classification accuracy for grade using the RBF kernel is 66.53% while the classification accuracy for price using the linear kernel is 38.1%. When the outliers

are removed, the linear model performs worse with an accuracy of 34.0%. However, this may be explained by the reduced number of outliers: the 6,000 fewer outliers is about 5.7% of the data, a little over the percentage of accuracy lost. Since the outliers are not close to the boundaries between the classes, the loss of these instances means that misclassifications along border instances are a more frequent occurrence. This would outweigh any of the gains made in predicting the other instances by removing the outliers. In other words, the removal of the outliers should be expected to reduce classification accuracy since outliers are easier to classify than instances along class boundaries.

8.4 Remarks

Regression models can successfully predict, within a few points, the correct grade of a wine. On average, the model is only 1.6 points away from the right value if we do not let errors cancel each other out. This is considerably more granular information than that provided by the classification model: a misclassification could mean many more points away than that. Another plus to using the regression model is that the class imbalance problem does not exist since there are not any classes (although there may be leverage points which pulls the model one way or another more than the other points). Regression models can also predict price, although, as we have seen with classification, it is harder to predict. On average, the model is off by about $13 per bottle of wine. This is not exactly stellar results: consumers would easily be swayed by a price difference of $13 in all but the most expensive wines. This gets even worse when outliers are present: the predictions are off by an accumulated average of $20.53. Fortunately, the mean error is negative at -$4 without outliers, suggesting that the model would be more likely to tell the consumer that the price is more expensive than it really is, not less. This may be desireable if sticker shock is to be avoided, however

it may also persuade customers not to consider the wine at all.

By predicting finer information than simple classes, we are able to more accurately judge what a wine really is. This does not mean classification is not valuable, of course, since classes are a good way to segment the wines into groups. But it does give us the ability to make finer predictions, avoid the pitfalls of class imbalances, and reduce the chance of larger errors. This comes at the cost that some data may need to be thrown out to build a good model for the remainder of the data, accepting that the model would likely not perform well on outliers.

Chapter 9

Conclusion and Future Works

This work focused on predicting three key aspects of a wine: how good it is, how expensive and it is, and, from the perspective of its region, what kind it is. Several methods were tried to make these predictions: both classification and regression, single label and multi-label classification, binary classification and multiclass classification. In the end, this work achieved many good results which can potentially be improved upon in the future. In this chapter, we review some of the best results in prior chapters and discuss what could be done in the future to improve these results.

9.1 Results Overview

Many results were gathered throughout Chapters 4 through 8, and while the results are summarized at the end of each chapter, an overview will also be provided here. Please note that, as discussed in the chapters themselves, the results between chapters are not always directly comparable.

In Chapter 4, SVMLight with a linear kernel was used to make classifications, both on the two class and multiclass datasets. For the multiclass problem, the SVMs were constructed into two hierarchies: one for grade and one for price (region remains a two class problem). Both accuracy and ranking (via coverage) were used for evaluation metrics in the multiclass problem. The results are summarized in Table 9.1.

In Chapter 5, both LibSVM from WEKA and Kernlab from R were used to make classifications, both on the two class and multiclass datasets. The one-versus-one approach was used by both implementations for the multiclass problem. Two dimension reduction methods were also attempted: principal component analysis

Variable	Accuracy	Coverage	Misclassified Coverage
Origin	84.23%		
Grade (2 class)	84.09%		
Grade (4 class)	73.07%	0.4294	1.5
Price (2 class)	74.65%		
Price (4 class)	45.61%	0.8698	1.6

Table 9.1: Chapter 4 Results Summary

Implementation	Multiclass?	Kernel	Grade	Price	Region
LibSVM	No	Linear	85.92%	74.85%	84.40%
	Yes	Linear	75.22%	47.05%	
Kernlab	No	Laplace	85.48%	75.89%	88.52%
	Yes	RBF	76.21%	48.71%	

Table 9.2: Chapter 5 Results Summary

and selection via ratio of occurrence between classes. Both of these methods had disappointing results. In Table 9.2, the best results from this chapter is reported, without dimension selection.

In Chapter 6, the class imbalance problem between the Classic and Outstanding grade classes were addressed. Kernlab and R were the SVM implementations used in this chapter. Several different techniques were tried to solve the imbalance problem. While most attempts were failures, SMOTE and borderline SMOTE produced promising, but not excellent, results. In addition to dealing with this single imbalance, quartiles were used to classify grades to avoid the imbalance problem entirely. Unfortunately, this approach led to significantly reduced accuracy compared to the results in Chapter 5. The best results from this chapter are summarized in Table 9.3.

In Chapter 7, several multi-label techniques were compared to the single label

Technique	Parameters	Majority Right	Minority Right	Precision	Recall
SMOTE	3	28,229	1,061	0.161	0.669
B SMOTE	3, 21	30,335	832	0.195	0.525
B SMOTE	7, 23	30,226	842	0.192	0.531

Table 9.3: Chapter 6 Results Summary

Method	Multiclass?	Grade	Price	Region	Ham. Loss	0/1 Loss
Single Label	No	0.859	0.749	0.844		
Label Powerset	No	0.849	0.753	0.851	0.182	0.423
BCC/ BR	No	0.847	0.749	0.843	0.187	0.451
Single Label	Yes	0.752	0.470	0.844		
Class Relevance	Yes	0.752	0.470	0.844	0.312	0.697
BCC	Yes	0.752	0.473	0.844	0.311	0.694

Table 9.4: Chapter 7 Results Summary

Target	ME	MAE	MSE	RMSE	NMAE	NRMSE	r	r^2
Grade	0.00	1.59	4.02	2.00	8.0%	10.0%	0.75	0.56
Price (Outliers)	-9.56	20.53	2082.15	45.63	41.8%	93.0%	0.44	0.19
Price	-3.75	12.94	318.62	17.85	63.6%	87.8%	0.51	0.27

Table 9.5: Chapter 8 Results Summary

methodology. MEKA and LibSVM were used as the implementations in this chapter. The single label method was hard to beat, although label powerset proved competitive in the binary classification problems and Bayesian classifier chains proved competitive in the multiclass classification problems. BCCs and binary relevance also nearly tied in the binary classification problem (with BCC winning by 0.001 in Jaccard Index). The linear kernel proved to be reliably more accurate than the RBF kernel for this implementation. The results are shown in Table 9.4.

In Chapter 8, regression was used instead of classification on grade and price. Kernalb and R were used as the implementation for this chapter. The results for price were improved when outliers were removed (there were not outliers for grade). The RBF and Laplacian kernels tied and performed the best on grade and price without outliers, while the linear kernel performed better on price with outliers. Note that while the minimax method was used for computing normalized error on grade, the standard deviation method was used for normalizing error for price. Thus, the normalized errors are not comparable between grade and price. The results are shown in Table 9.5.

In summary, we are able to make good predictions on all three targets. We can

predict with 88.52% accuracy on origin, 85.92% accuracy on two class grade, 76.21% accuracy on four class grade, 75.89% accuracy on two class price, and 48.71% accuracy on four class price. We can classify all three labels at once with sometimes better performance than predicting each one independently. In addition, we can build good regression models for grade and price. With grade, we can predict within 1.6 points of the actual value, and with price, we can predict within $13 of the actual price, which is a fair value considering the difficulty of the problem all models have had in predicting price.

9.2 Future Works and Advice

While this work examined the data from many different angles and used many different methods to make predictions, there are some improvements that could be made that is left to future work. In addition, there are some pitfalls that were encountered during this work that can hopefully be avoided in the future.

First, as has been mentioned before, the results between Chapters 4 and 5 are not comparable. This is due to two reasons: the value of the hyperparameter C is different and the other kernels in SVMLight are not ever used. In future work, setting the values for C and trying the additional kernels may result in SVMLight outperforming or tying the results attained in Chapter 5.

Second, the imbalanced problem is not really solved yet. Although SMOTE and borderline SMOTE show promise, they cannot improve the classifier without significant cost to classifying the majority class. It may be that it is almost impossible to distinguish the two classes based on their attributes, or it may be that other techniques are required. In addition, this work did not explore balancing the 90- and 90+ classes. The work did briefly address balancing all four classes using quartiles to form the grade groups instead of Wine Spectator's 100 point scale. The results

from this classification scheme is poor, almost as poor as classifying price. This may be because of the nature of quartiles themselves or it may be that there are human elements influencing how wines are classified in Wine Spectator's scheme. Either way, this method did not yield great results, and improving upon them is left to future work.

Third, the results for classifying price proved disappointing with fewer than half of them classified correctly in the multiclass problem. As just mentioned, the usage of quartiles may be to blame. Since the median values are used as cut points when building quartiles, it may be difficult to distinguish between those that lie just above or just below the median. Although the resulting classes are balanced, it may prove useful to use a different classification scheme, even one that results in imbalance, to improve classification on price (and also grade).

Fourth, it is clear that removing outliers yields improvements when performing regression on price. In this work, outliers were simply decided based on values above the whisker in the box plot. More sophisticated methods are recommended for future works. Additionally, there may still be outliers in the dataset where the instances above the whisker are removed: the regressors have a mean error of -3.75, suggesting they are still guessing consistently lower than the actual price, which may be due to the highly priced wines that remain.

Fifth, additional attention could be paid to the region of origin classification scheme. In this work, North Africa was considered a new world region. However, depending on one's source of information, North Africa could also be considered old world as well. Additionally, it may be interesting to divide parts of Asia into their own category: places such as East Asia were producing wines long before the discovery of the new world, and thus could either be grouped into the old world category or made a separate category. Consideration should be given for having a classification scheme that is not Eurocentric as the one used in this work. For example, classifying origin

by continent could be used, provided the classes are balanced.

Sixth, this work relied exclusively on SVMs and SVRs to make predictions (although, while not reported earlier, an attempt at using Naive Bayes was tried and failed). Dimension selection methods unfortunately failed. It may be worthwhile to try a deep learning approach on wine reviews to make predictions. Consider that convolutional and/or recurrent neural networks could be applied directly to the wine reviews to extract the keywords, rather than using the Computational Wine Wheel. The important keywords would be learned while the order of the words in the reviews could also be considered. This could potentially be a step up from the current bag-of-words model. Fortunately, the raw data used in this work is large enough for deep learning techniques to be applied (although caution must still be taken to avoid overfitting).

Lastly, for those who wish to continue using this dataset and SVMs, to find an SVM implementation that takes better advantage of the modern architecture of computers. While Microsoft R Open has some multithreading capability, it still can take days to run some of the experiments found in this work. Additionally, R is very memory inefficient, and a machine with a minimum of 32 GB is required to run all of the experiments found in this thesis. A less memory intensive implementation is therefore recommended. If these are not options, restricting the experiments to binary classification (and avoiding regression) will greatly speed up the calculations and reduce memory requirements.

9.3 Conclusion

Attributes found in wine reviews are not only intuitive for humans to understand, but meaningful and informative to machine learning models as well. We have seen that the models are capable of making many accurate quantitative predictions from

qualitative descriptions concerning wine price and quality. They are also capable of making qualitative predictions about wines: are wines of the new world or of the old world style? This thesis, by exploring these problems found in Wineinformatics, can therefore serve as a foundational work for others in the field.

Bibliography

[1] (2017) World wine production by country. Wine Institute. [Online]. Available: http://www.wineinstitute.org/files/World_Wine_Production_by_Country_2015.pdf

[2] T. Hastie, R. Tibshirani, and J. Friedman, *Unsupervised Learning*. New York, NY: Springer New York, 2009, pp. 485–585.

[3] J. Dy and C. Bordley, "Feature selection for unsupervised learning," *Journal of Machine Learning Research*, vol. 5, pp. 845–889, 2004.

[4] I. Guyon and A. Elisseef, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[5] J. Blakeman, "On tests for linearity of regression in frequency distributions," *Biometrika*, vol. 4, no. 3, pp. 332–350, 1905. [Online]. Available: http://www.jstor.org/stable/2331617

[6] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 161–168. [Online]. Available: http://doi.acm.org/10.1145/1143844.1143865

[7] R. Caruana, N. Karampatziakis, and A. Yessenalina, "An empirical evaluation of supervised learning in high dimensions," in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08. New York, NY, USA: ACM, 2008, pp. 96–103. [Online]. Available: http://doi.acm.org/10.1145/1390156.1390169

[8] A. Niculescu-Mizil and R. Caruana, "Predicting good probabilities with supervised learning," in *Proceedings of the 22Nd International Conference on*

Machine Learning, ser. ICML '05. New York, NY, USA: ACM, 2005, pp. 625–632. [Online]. Available: http://doi.acm.org/10.1145/1102351.1102430

[9] R. Caruana and A. Niculescu-Mizil, "Data mining in metric space: An empirical analysis of supervised learning performance criteria," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '04. New York, NY, USA: ACM, 2004, pp. 69–78. [Online]. Available: http://doi.acm.org/10.1145/1014052.1014063

[10] X. Zhou and M. Belkin, "Chapter 22 - semi-supervised learning," in *Academic Press Library in Signal Processing: Volume 1*, ser. Academic Press Library in Signal Processing, P. S. Diniz, J. A. Suykens, R. Chellappa, and S. Theodoridis, Eds. Elsevier, 2014, vol. 1, pp. 1239 – 1269. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B978012396502800022X

[11] M. F. A. Hady and F. Schwenker, *Semi-supervised Learning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 215–239.

[12] X. Zhu, "Semi-supervised learning literature survey," 2005.

[13] X. Zhu and A. B. Goldberg, "Introduction to semi-supervised learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 3, no. 1, pp. 1–130, 2009.

[14] Y. Er and A. ATASOY, "The classification of white wine and red wine according to their physicochemical qualities," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 4, pp. 23 – 26, 2016.

[15] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, "Modeling wine preferences by data mining from physicochemical properties," *Decision Support Systems*, vol. 47, no. 4, pp. 547 – 553, 2009, smart Business Networks: Concepts and Empirical Evidence.

[16] S. E. Ebeler, *Linking Flavor Chemistry to Sensory Analysis of Wine.* Boston, MA: Springer US, 1999, pp. 409–421.

[17] V. Velchev, C. Rhodes, A. Yu, and B. Chen, "The computational wine wheel 2.0 and the trimax triclustering in wineinformatics," *In Industrial Conference on Data Mining*, pp. 223–238, 2016.

[18] About our tastings. [Online]. Available: http://www.winespectator.com/display/show/id/about-our-tastings

[19] B. Chen, V. Velchev, B. Nicholson, J. Garrison, M. Iwamura, and R. Battisto, "Wineinformatics: Uncork napa's cabernet sauvignon by association rule based classification," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, Dec 2015, pp. 565–569.

[20] B. Chen, H. Le, C. Rhodes, and D. Che, "Understanding the wine judges and evaluating the consistency through white-box classification algorithms," in *Advances in Data Mining. Applications and Theoretical Aspects*, P. Perner, Ed. Cham: Springer International Publishing, 2016, pp. 239–252.

[21] N. Wariishi, B. Flanagan, T. Suzuki, and S. Hirokawa, "Sentiment analysis of wine aroma," in *2015 IIAI 4th International Congress on Advanced Applied Informatics*, July 2015, pp. 207–212.

[22] B. Flanagan, N. Wariishi, T. Suzuki, and S. Hirokawa, "Predicting and visualizing wine characteristics through analysis of tasting notes from viewpoints," in *HCI International 2015 - Posters' Extended Abstracts*, C. Stephanidis, Ed. Cham: Springer International Publishing, 2015, pp. 613–619.

[23] A. C. Noble, R. A. Arnold, J. Buechsenstein, E. J. Leach, J. O. Schmidt, and P. M. Stern, "Progress towards a standardized system of wine aroma

terminology," *American Journal of Enology and Viticulture*, vol. 38, no. 2, pp. 143–146, 1984.

[24] B. Chen, C. Rhodes, A. Crawford, and L. Hambuchen, "Wineinformatics: Applying data mining on wine sensory reviews processed by the computational wine wheel," in *2014 IEEE International Conference on Data Mining Workshop*, Dec 2014, pp. 142–149.

[25] Wine spectator's 100-point scale. [Online]. Available: http://www.winespectator.com/display/show/id/scoring-scale

[26] K. Anderson, *The World's Wine Markets: Globalization at Work*. Edward Elgar Pub., 2004. [Online]. Available: https://books.google.com/books?id=PMKc8Ed8qoIC

[27] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[28] H. Le, "Classification on wine informatics," 2015.

[29] D. Fradkin and I. Muchnik, "Support vector machines for classification," *Mathematics Subject Classification*, 2000.

[30] B. Baesens, T. Van Gestel, S. Viaene, M. Stepanova, J. Suykens, and J. Vanthienen, "Benchmarking state-of-the-art classification algorithms for credit scoring," *Journal of the Operational Research Society*, vol. 54, no. 6, pp. 627–635, Jun 2003. [Online]. Available: https://doi.org/10.1057/palgrave.jors.2601545

[31] E. Byvatov, U. Fechner, J. Sadowski, and G. Schneider, "Comparison of support vector machine and artificial neural network systems for drug/nondrug classification," *Journal of Chemical Information and Computer Sciences*,

vol. 43, no. 6, pp. 1882–1889, 2003, pMID: 14632437. [Online]. Available: http://dx.doi.org/10.1021/ci0341161

[32] H. Yoon, S.-C. Jun, Y. Hyun, G.-O. Bae, and K.-K. Lee, "A comparative study of artificial neural networks and support vector machines for predicting groundwater levels in a coastal aquifer," *Journal of Hydrology*, vol. 396, no. 1, pp. 128 – 138, 2011.

[33] O. Chapelle, P. Haffner, and V. N. Vapnik, "Support vector machines for histogram-based image classification," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1055–1064, Sep 1999.

[34] V. Vapnik and A. Learner, "Pattern recognition using generalized portraits," in *Avtamarika I Telerncкllanjka*, vol. 21, no. 6, 1963, pp. 774–780.

[35] B. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, vol. 1, 1992, pp. 23–34.

[36] T. Poggio and F. Girosi, "Network for approximation and learning," in *Proceedings of the IEEE*, vol. 78, 1990, pp. 1481–1497.

[37] C. Cortes and V. Vapnik, "Support-vector networks," in *Machine Learning*, vol. 3, 1995, pp. 273–297.

[38] M. Law, "A simple introduction to support vector machines," *Lecture for CSE*, vol. 802, 2006.

[39] B. Schölkopf, "The kernel trick for distances," in *Advances in neural information processing systems*, 2001, pp. 301–307.

[40] G. F. Smits and E. M. Jordaan, "Improved svm regression using mixtures of kernels," in *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, vol. 3. IEEE, 2002, pp. 2785–2790.

[41] S.-i. Amari and S. Wu, "Improving support vector machine classifiers by modifying kernel functions," *Neural Networks*, vol. 12, no. 6, pp. 783–789, 1999.

[42] J.-P. Vert, K. Tsuda, and B. Schölkopf, "A primer on kernel methods," *Kernel methods in computational biology*, vol. 47, pp. 35–70, 2004.

[43] A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis, "kernlab – an S4 package for kernel methods in R," *Journal of Statistical Software*, vol. 11, no. 9, pp. 1–20, 2004. [Online]. Available: http://www.jstatsoft.org/v11/i09/

[44] Z.-D. Zhong, X.-J. Zhu, and G.-Y. Cao, "Modeling a pemfc by a support vector machine," *Journal of Power Sources*, vol. 160, no. 1, pp. 293–298, 2006.

[45] M. Sellathurai and S. Haykin, "The separability theory of hyperbolic tangent kernels and support vector machines for pattern classification," vol. 2, pp. 1021–1024, 03 1999.

[46] C. A. Tawiah and V. S. Sheng, "Empirical comparison of multi-label classification algorithms." in *AAAI*, 2013.

[47] T. Joachims, "Making large-scale svm learning practical," Universität Dortmund, LS VIII-Report, LS8-Report 24, 1998.

[48] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[49] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria. [Online]. Available: https://www.R-project.org

[50] Microsoft and R. C. Team, *Microsoft R Open*, Microsoft, Redmond, Washington, 2017. [Online]. Available: https://mran.microsoft.com/

[51] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[52] H. Hotelling, "Analysis of a complex of statistical variables into principal components." *Journal of educational psychology*, vol. 24, no. 6, p. 417, 1933.

[53] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[54] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: a new over-sampling method in imbalanced data sets learning," in *International Conference on Intelligent Computing*. Springer, 2005, pp. 878–887.

[55] F. Leisch, "A toolbox for k-centroids cluster analysis," *Computational Statistics and Data Analysis*, vol. 51, no. 2, pp. 526–544, 2006.

[56] S. Nguyen, J. Quinn, and A. Olinsky, "An oversampling technique for classifying imbalanced datasets," in *Advances in Business and Management Forecasting*. Emerald Publishing Limited, 2017, pp. 63–80.

[57] R. Akbani, S. Kwek, and N. Japkowicz, "Applying support vector machines to imbalanced datasets," in *European conference on machine learning*. Springer, 2004, pp. 39–50.

[58] E. Spyromitros-Xioufis, W. Groves, G. Tsoumakas, and I. Vlahavas, "Multi-label classification methods for multi-target regression," 11 2012.

[59] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *Database Technologies: Concepts, Methodologies, Tools, and Applications*, vol. 4, 2009.

[60] J. Read, B. Pfahringer, G. Holmes, and E. Frank, "Classifier chains for multi-label classification," *Machine Learning*, vol. 85, no. 3, pp. 333–359, 2011.

[61] J.Read, L.Martino, P. Olmos, and D. Luengo, "Scalable multi-output label prediction: From classifier chains to classifier trellises," *arXiv*, 2015.

[62] J. Zaragoza, L. Sucar, E. Morales, C. Bielza, and P. Larranaga, "Bayesian chain classifiers for multidimensional classification." pp. 2192–2197, 01 2011.

[63] Y. Guo and S. Gu, "Multi-label classification using conditional dependency networks," *International Joint Conference on Artificial Intelligence*, 2011.

[64] J. Read. (2015) Multi-label classification. Aalto University. [Online]. Available: https://jmread.github.io/talks/Tutorial-MLC-Porto.pdf

[65] J. Read, P. Reutemann, B. Pfahringer, and G. Holmes, "MEKA: A multi-label/multi-target extension to Weka," *Journal of Machine Learning Research*, vol. 17, no. 21, pp. 1–5, 2016. [Online]. Available: http://jmlr.org/papers/v17/12-164.html

[66] Mauricio Zambrano-Bigiarini, *hydroGOF: Goodness-of-fit functions for comparison of simulated and observed hydrological time series*, 2017, r package version 0.3-10. [Online]. Available: http://hzambran.github.io/hydroGOF/