ST563 Final Project

Mana Azizsoltani

10 March, 2021

Contents

Introduction

There has been a push towards the implementation of different analytics solutions within the agriculture industry, especially in the realms of crop health and yield. These sort of analyses can help farmers to reduce waste and improve profits. In particular, the viticulture industry stands to benefit from such methods, being that the production of wine is multifaceted. Winemakers can work to optimize revenue by analyzing their input and output (crops and wines). Specifically, costs may be cut out if machine learning methods were able to predict the quality of wine based on different metrics, since wine producers would no longer have to hire expert wine tasters to determine wine quality.

In this study, our objective is to try and predict the quality of wine using a variety of statistical learning techniques. We will look at this problem from a regression standpoint as well as a classification standpoint. Using regression, we will try and predict the exact rating (0 to 10) of a particular wine. On the other hand, when we look at classification, we will try to classify a particular wine as either good or bad, taking wines rated from 0-5 as low quality wines and wines rated from 6-10 as high quality wines. We will be assessing the prediction accuracy of the following methods:

Table 1: ML Methods

Classification	Regression
Classification Tree	Regression Tree
Random Forest	Boosted Tree
Support Vector Machine (SVM)	Multiple Linear Regression (MLR)
KNN Classification	KNN Regression
Logistic Regression	LASSO Regression

Required Libraries

To run the code for the project, the following libraries are required:

- `caret`: to do the heavy lifting of training and tuning the models
- `tidyverse`: for all the data reading and wrangling
- `knitr`: for rmarkdown table outputs
- `rmarkdown`: for output documents
- `corrplot`: for correlation structure visualization
- `leaps`: for subset selection
- `rpart`: fitting the basic classification tree
- `rpart.plot`: visualization of the classification tree
- `randomForest`: fitting random forest models
- `kernlab`: fitting the support vector machine
- `class`: fitting the k-nearest neighbor model

Data

The data set used is the `Wine Quality` dataset from the UCI Machine Learning Repository. The data set is composed of 1600 observations of different variants of the Portuguese "Vinho Verde" red wine. For each wine, various features of the wine were measured, including a measurement of quality, given by an expert wine taster. Our objective it to use the different features to predict the quality of the wine.

Variable Descriptions

Each entry in the dataset represents the different metrics of the following attributes of a single type of wine.

- **fixed acidity**: the quantity of fixed acids found in the wines. The predominant fixed acids found in wines are tartaric, malic, citric, and succinic, all of which come from the grapes with the exception of the succinic acid, which comes from the yeast in fermentation.
- **volatile acidity**: the steam distillable acids present in wine, primarily acetic acid but also lactic, formic, butyric, and propionic acids. These acids generally come from the fermentation process.
- **citric acid**: added to the wine as a natural preservative or for acidity and tartness.
- **residual sugar**: the quantity of sugar left in the wine after the fermentation process.
- **chlorides**: the amount of salt in a wine.
- **free sulfur dioxide**: the amount of SO_2 that is not bound to other molecules.
- **total sulfur dioxide**: total amount of SO_2 in the wine. Sulfur Dioxide is used throughout all stages of the winemaking process to prevent oxidation and bacteria growth.
- **density**: density of the wine.
- **pH**: pH of the wine.
- **sulphates**: quantity of sulphates in the wine. Sulphates are used as a preservative.
- **alcohol**: alcohol content by volume.
- **quality**: score of quality of the wine given by expert tasters (score between 0 and 10).

Methods

Data Processing & Partitioning

Fortunately, the `wine` data set that we worked with was very tidy; there was no missing values. We didn't standardize the data initially, but when training the support vector machine as well as the k-nearest neighbors models we centered and scaled the data.

As mentioned in the introduction, our focus is on the classification accuracy of machine learning methods. In particular, we will be using traditional cross-validation methods to assess the accuracy, sensitivity, and specificity of each of the models. We split the data into a training and test data set in order to later evaluate the model's prediction accuracy. For this project we used a 75/25 split, training the data on the 75% and testing the trained models on the withheld 25%. We will then repeat this process over 5 folds of the data, averaging the results.

The tree-based, LASSO, and K-nearest neighbors models require parameters to be tuned (more on those later). Since we used the `caret` package in R to fit all of our models, we used the tunes of the parameters that were deemed "best" by the `train()` function.

Variable Selection

Basic Tree

The basic classification or regression tree is based on partitioning the data into subgroups using simple binary splitting. Initially, all objects are considered a single group. Then, the group is binarily split into two subgroups based on the criteria of a certain variable. In the classification setting, we classify the observations in a specific region with majority vote, while in the context of regression the mean of the group is used for prediction. We want to grow the tree as big as we can, and then prune it back using cost-complexity pruning. This is done to not overfit the data, but pruning increases the bias. The pruning parameter needs to be tuned, which is done automatically in the `caret` package.

The advantage of using a basic tree is that it is easy to understand and has a good interpretability, which is not something that translates over to ensemble methods like the random forest. Additionally, the basic classification or regression tree is not very computationally expensive, unlike the random forest, boosting, and the support vector machine. We used the `caret` and `rpart` packages in R to fit our trees.

Random Forest

The random forest model is an ensemble tree-based method, which creates multiple trees from bootstrap samples and averages the results. Many bootstrap samples are created with replacement and then a classification tree is fitted on each bootstrap sample with a random subset of the predictors. Once a prediction has been made by all of the bootstrapped trees, the final classification is based on majority vote of the bootstrap predictions. Similarly, the prediction in the regression context is an average of all of the predictions of the bootstrapped trees.

The advantage for using an ensemble method over a regular classification or regression tree is that because there are many bootstrap samples being averaged together, there is less variance. This is similar to how the variance of the sample mean goes down as the sample size increases. Although it will probably increase our prediction accuracy, the random forest loses the interpretability that the basic trees have. Furthermore, the algorithm that is used to fit the random forest is very computationally expensive.

To train our model, we used the `randomForest` and `caret` packages in R. The maximum number of predictors for a bootstrap sample as well as the number of trees in the forest are both parameters that need tuning. Again, we will use the "best" tune, automatically given to us by the `caret` package.

Support Vector Machine

A support vector machine is a type of classification rule where it essentially maximizes the margin between groups by choosing the "line" with the widest "margin", but allowing for error. We put "line" in quotation marks because with higher-dimensional data, the "line" is actually a hyper-plane-thing. Similarly, the "margin" doesn't actually exist the in sense of a clean margin between two things, since we are allowing for some sort of error. The support vectors are the points closest to the middle that carry the most weight when classifying, since they are the most prone to error and are deciding factors of where the classification line could go. We used a radial kernel function to deal with the non-linearity and higher-dimensions.

The support vector machine is pretty effective in higher-dimensional spaces, but doesn't perform very well with very large data sets with lots of noise. If we were to look at hotels with millions of records, a support vector machine may not be suitable. We fit our SVM using the `kernlab` and `caret` packages in R.

K-Nearest Neighbors (KNN)

K-NN is a "model free" approach, meaning that it doesn't assume any probability model on the data. Given an observation, x, we want to find the k training observations that are "closest" to x, and then either

classify the new observation using majority vote among these k neighbors or predict the new observation by averaging the closest k neighbors. We define "closeness" based on Euclidean distance in our model. As this is the case, we need to center and scale the data, so that different scales of measurement are kept constant.

The K-nearest neighbors model is relatively intuitive and simple and has no underlying assumptions, making it a good model to consider. In our classification, we are only using it for a binary classification, but it is also very easy to extend the K-NN model to multiple classes. On the other hand, the K-NN algorithm is very sensitive to outliers, high dimensional data, and imbalanced data.

The number (k) of closest neighbors is a parameter that needs tuning. We used the `class` and `caret` packages in R to fit this model. Based on our cross-validation, we used a k value of 5.

Logistic Regression

The logistic regression model uses the "logit" function $log(\frac{p}{1-p})$, which links the mean to the linear form of the regression model, $\boldsymbol{X\beta}$. Using it for binary classification, we round the fitted values either up or down to 1 or 0. The logistic regression function is defined as

$$\ln\left(\frac{p(x)}{1-p(x)}\right) = x'\beta, \qquad \text{where } p(x) = (1 + e^{-x'\beta})$$

The logistic regression model assumes that each observation is independent, that there is little or no multi-collinearity among the predictors, and that there is a linear relationship between the predictor variables and log odds. This differs from the typical regression model, where the residuals must be normally distributed and have constant variance.

To fit the logistic regression model, we used the `glm()` function in base R. The advantages to using a logistic regression model are similar to those for the basic classification tree: it is not computationally expensive and easy to interpret the results. On the other hand, one of the major drawbacks to using a logistic regression model is that it is subject to the distributional assumptions on the data and errors mentioned above. If our assumptions are broken, our predictions may not be very reliable.

LASSO Regression

Boosting

Multiple Linear Regression