ESE 417 Final Project

1st Myan Sudharsanan
Computer Science and Engineering Dept.
Washington University in St. Louis
St. Louis, MO
m.s.sudharsanan@wustl.edu

2nd Reedham Kalariya
Computer Science and Engineering Dept.
Washington University in St. Louis
St. Louis, MO
j.kalariya@wustl.edu

3rd Jeremy Stiava
Computer Science and Engineering Dept.
Washington University in St. Louis
St. Louis, MO
email address or ORCID

***Abstract*—Your abstract should consist of a single paragraph up to 250 words, with correct grammar and unambiguous terminology. Provide a concise summary of the project. Include the conclusions reached and the potential implications of those conclusions. It should be self-contained – no abbreviations, footnotes, references, or mathematical equations. It also should highlight what is unique in your work.**

I. Introduction

We were given the red wine dataset from the UCI Machine Learning Library to explore. According to the library's website, the red wine set is actually a subset of data that also consists of white wine. Both the red and white variants consist of Vinho Verde wine. Features of this dataset include fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, and quality (scored between 0 and 10). As the site had suggested, we treated the quality column as the "y" or output while the remaining features are regarded as part of our "x" or input set. However, before we could pass in that entire input set, we wanted to attempt some data wrangling. Upon initial observations, we identified very low classification scores for each model, hence we wanted to see if we could manipulate the dataset in some sense. In fact, the site even suggests we utilize outlier detection algorithms to sort out the good wines from the bad ones. We also came to the realization that certain columns may not be very relevant for our classifications which we dealt with either by dropping or combining. After this data manipulation however, we came to the conclusion that…

II. Methods

Our first target was to understand the nature of the given data. First, as starter code, we employed the tactic of loading the dataset into a given dataframe, and separating the aforementioned features from the output. We then plugged in our data into the ANN model, the SVM model, and the Random Forest model, to get a rough idea of the weaknesses of this dataset. We found very low classification scores between 0.5 and 0.6, so we backtracked to our dataset to figure what may be causing these low scores. As a general rule of thumb it is best practice to retain columns with high variance and drop the columns with relatively low variance in the context of the given statistic. For the former case, columns with high variance indicate the true ranges for each statistic in a dataset, so they are vital for establishing a strong classification score. However, they do present the problem of dealing with outliers, since outliers can either heavily inflate or deflate a classification score. Before we can deal with outliers, however, we need to work with the extraneous data. For the latter case of dropping columns, we noticed columns such as pH and sulphates have very low variance, hence can be dropped as it will not contribute significantly to the classification score. Additionally, a column with low variance probably indicates that it is not an essential ingredient of wine in terms of affecting the quality, given all of the wine had similar values. In order to drop these columns, we use the pandas dataframe function "drop" and convert the returned multi-dimensional array back to a pandas dataframe. This process is described…

The curse of dimensionality is the problem regarding the use of several dimensions of data to identify a certain classification. Hence, we believed this necessitated the use of a dimensionality reduction algorithm to alleviate this issue. One such algorithm is PCA, or principal component analysis. This is quite similar to

Having dealt with extraneous data, we wanted to integrate the high variance columns into our models without the outliers greatly affecting the classification score. We went about doing so by standardizing the data. Instead of selecting min and max values to scale by, we set the mean for each column to be zero with a standard deviation of one, such that the data in each column is normally distributed. By centering the data in such a manner, we are minimizing the negative impact of outliers, while also retaining a normal distribution within each column. We achieved this by using the sklearn.preprocessing.StandardScaler package, then fitting and transforming to our modified dataframe (which we have dropped columns from). However, this yielded poor results, as we will discuss in the following section. The implementation follows as below…

The second model we tackled was KNearestNeighborsClassifier, or KNN for brevity. Essentially, in KNN, you calculate a distance value relative to test points, which serves as your predicted value. Following that, we sort the predicted values in ascending order. Then, with a given value of K, we pick the first K elements of the set. Within this subset, we then select the mode classification of the subset, or in other words, the most frequent classification within the subset, which will subsequently become the classification for the test point.

We then proceeded with the process of cross-validation and hyperparameter tuning. Starting with the former, cross-validation is the process used to avoid overfitting of the data, with our specific cross-validator being KFold cross-validation. KFold cross-validation involves splitting the dataset into K equally sized subsets of the testing set and using each fold as the test set once while training the model on the remaining folds. Note that our final implementation does not include KFolds, due to massive runtimes for when we used KFold. Once we have completed cross-validation, we move on to hyperparameter tuning, which is essentially the process of modifying several different parameters of the classifiers to find the optimal parameters to give us the best accuracy. For example, for our random forest model, we modify the number of estimators, the criterion, the *max_depth*, *max_features*, and *class_weight*. In order to run through repeated iterations of changing parameters, we used the *GridSearchCV* function within *sklearn.model_selection*.