Classification of Wine based on quality and derivation of results on various Machine Learning concepts

Abstract

This report comprises of a machine learning algorithm used to classify wines into ‘Good wines’ and ‘Bad wines’. The method comprises of modelling the data using various machine learning models and using the best model of the bunch to implement the final model. Through this method, we will arrive at a conclusion for the best performing model for the required task and then see the difference in validation scores for different validation models. The proposed method uses certain amounts of data manipulation to classify the data into a binary model and then the respective feature importance is also found out. We use the Red Wine dataset from the UCA Machine learning repository and is extensively tested on using the classifiers Random Forest Classifier, XGBoost Classifier, K-Nearest Neighbours, Logistic regression, Support Vector Machine classifier, Decision Tree classifier and GaussianNB, with the Random Forest Classifier performing best and thus being used for further result extraction.

1.Main Objectives’

•Testing the Red Wine dataset to classify a random wine into good and bad quality.

•Manipulating the data to make the classification binary

•Studying and Implementing different classification models to conclude on the best performing model and the reason behind it.

•Perform a detailed study on the feature importance

•Perform analysis of tuning of Hyperparameters using RandomSearch and Cross validation technique

•Study the effect of Oversampling and how it balances the output classes.

•Understand how different features affect training differently and removing certain features results in better performance.

•Confusion matrix to be put out and cross validation to be done for accurate results.

2.Status and Other Details

•Completed

•Total time spent on the project – 2 Weeks

3.Major Stumbling Blocks

•Manipulation of the available dataset to suit the objective and code.

•Studying 7 different classification models.

•Tuning the hyperparameters for the best result.

•Implementing CrossValidation due to time constraints

4.Introduction

Wine has been made in Portugal since atleast 2000 BC when the Tartessians planted vines in the Southern Sado and Tagus valleys. By the 10th century BC, the Phoenicians had arrived and introduced new grape varieties and winemaking techniques to the area. Up until this point, viticulture was mostly centered on the southern coastal areas of Portugal. In later centuries, the Ancient Greeks, Celts and Romans would do much to spread viticulture and winemaking further north[1].

Portuguese wines were first shipped to England in the 12th century from the Entre Douro e Minho region (which today includes modern Portuguese wine regions such as the douro and vinho verde). In 1386, Portugal and England signed the Treaty of Windsor which fostered close diplomatic relations between the two countries and opened the door for extensive trade opportunities. Portuguese wine production increased fivefold between the late 17th century and early 18th century due to a boom in demand within Portugal, its overseas possessions, and Britain[2]

Vinho Verde refers to Portuguese wine that originated in the historic Minho province in the far north of the country. The modern-day 'Vinho Verde' region, originally designated in 1908, includes the old Minho province plus adjacent areas to the south. In 1976, the old province was dissolved.

Vinho Verde is not a grape variety, it is a DOC for the production of wine. The name means "green wine," but translates as "young wine", with wine being released three to six months after the grapes are harvested[3]

The idea of this entire model is to classify the Vinho Verde wines as ‘Good’ and ‘Bad’ depending on various properties of a given wine. Various physiochemical tests are to be conducted on a wine to derive the features that we use in training the model[4].

The dataset we have consists of 11 features and a final ‘quality’ feature rating the wines quality on a scale of 3-8. We start by conducting a series of exploratory data analysis by plotting distribution plots and Histograms of each feature and creating a correlation matrix.

The distribution plots for all features tells us that each feature has a normal distribution which is ideal for training.

While plotting the correlation matrix, we see that the feature ‘alcohol’ is highly positively correlated to the result ‘quality’. The positive correlation tells us about the influence the corresponding feature has on the result. Thus alcohol level plays a good role in determining the quality of a wine. Similarly negative correlation tells us that either one of the feature is necessary.

One of the pair of features having high negative correlation can be eliminated to improve accuracy which we will be performing after fixing on the model.

The traditional train test split function is imported from the Sci kit learn library. We are splitting the data with a train-test ratio of 0.3. The effect of this ratio on the accuracy score will be discussed later on.

We then begin to train the model by using Seven different classifiers. The output accuracy is calculated by using the accuracy_score method from sci-kit learn.metrics.

The accuracy scores of each classifier is as follows:

Score	**Model**
0.8930	**Random Forest**
0.8916	**XGBoost**
0.8720	**KNN**
0.8700	**Logistic Regression**
0.8680	**SVM**
0.8640	**Decision Tree**
0.8330	**GaussianNB**

Hence we confirm that the Random Forest Classifier tests the model best. We then proceed to perform comparative analysis for the K-fold Validation techniques and effect of oversampling on this model.

Rest of the Paper is organized as:

Section 5 containing the background theory

Section 6 containing the Algorithm used

Section 7 containing Experimental Details and Results and comparison using various classifiers. It also contains the various studies conducted on the Random Forest model.

We conclude this report in Section 8 with the results of this entire project

5 . Background Theory

- **Random Forest**

Random Forests were introduced by Leo Breiman [5]who was inspired by earlier works of Amit and Geman.[6]Random Forest is a machine learning algorithm that use supervised learning technique to solve classification and regression problems. It is based on the concept of ensemble learning. Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. [7]Here ,the random forest instead of relying on a single decision tree, takes prediction from each tree and based on the majority votes of predictions ,it predicts the final output. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. One of the most important features of the Random Forest Algorithm is that it can handle the data set containing continuous variables as in the case of regression and categorical variables as in the case of classification. It performs better results for classification problems. Ensemble learning uses 2 type of models : Bagging and Boosting . The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Bagging is composed of two parts: aggregation and bootstrapping. Bootstrapping is a sampling method, where a sample is chosen out of a set, using the replacement method. The learning algorithm is then run on the samples selected.The bootstrapping technique uses sampling with replacements to make the selection procedure completely random. When a sample is selected without replacement, the subsequent selections of variables are always dependent on the previous selections, making the criteria non-random. Model predictions undergo aggregation to combine them for the final prediction to consider all the possible outcomes. The aggregation can be done based on the total number of outcomes or the probability of predictions derived from the bootstrapping of every model in the procedure.[8]

Consider the regression or classification setting. The data is given as in section 1: we have pairs (X_i,Y_i) $(i=1, \dots, n)$, where $X_i \in IRd$ denotes the d-dimensional predictor variable and the response $Y_i \in$ IR (regression) or $Y_i \in \{0, 1, \dots, J - 1\}$ (classification with J classes). The target function of interest is usually $IE[Y | X = x]$ for regression or the multivariate function $IP[Y = j|X = x]$ $(j = 0, \dots, J - 1)$ for classification. The function estimator, which is the result from a given base procedure, is

$$\hat{g}(\bullet) = h_n((X_1, Y_1), \dots, (X_n, Y_n))(\bullet) : IR^d \to IR,$$

where the function $h_n(\bullet)$ defines the estimator as a function of the data.

Bagging is defined as follows.

Step 1. Construct a bootstrap sample $(X_1^*, Y_1^*), \dots, (X_n^*, Y_n^*)$ by randomly drawing n times with replacement from the data $(X_1, Y_1), \dots, (Xn, Yn)$.

Step 2. Compute the bootstrapped estimator $\hat{g}*(\bullet)$ by the plug-in principle:

$$\hat{g}^*(\bullet) = h_n((X_1^*,Y_1^*), \dots, (X_n^*, Y_n^*))(\bullet). \quad (1)$$

Step3. Repeat steps 1 and 2 M times, where M is often chosen as 50 or 100, yielding $\hat{g}*k(\bullet)$ $(k = 1, \dots, M)$. The bagged estimator is $\hat{g}\ Bag(\bullet) = M^{-1}\sum_{k=1}^{M}\hat{g}^{*k}(\cdot).$. In theory, the bagged estimator is

$$\hat{g}\ Bag(\bullet) = IE*[\hat{g}^*(\bullet)]. \qquad (2)$$

The theoretical quantity in (2) corresponds to $M = \infty$: the finite number M in practice governs the accuracy of the Monte Carlo approximation but otherwise, it shouldn’t be viewed as a tuning parameter for bagging. Whenever we discuss properties of bagging, we think about the theoretical version in (2).[9]

Working of Random forest :

The first step is to make the subsets of the original data. This is done by row sampling and feature sampling by selecting rows and columns with replacement and creating subsets of the training dataset and then individual decision trees are constructed for each sample. Each decision tree will generate an output. Final output is considered based on Majority Voting for classification and averaging for respectively.(Fig 1)[10]

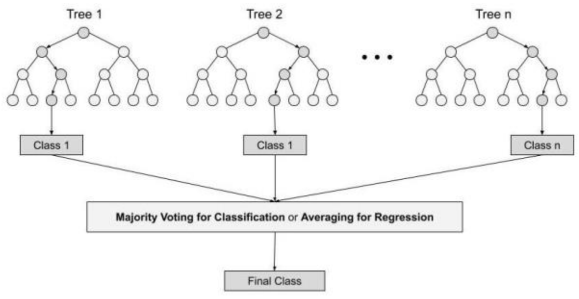

Tree 1
Tree 2
Tree n
Class 1
Class 1
Class n
Majority Voting for Classification or Averaging for Regression
Final Class

Fig 1

The decision tree starts with a root node and ends with a decision made by leaves. It consists of 3 components which are the root node, decision node, leaf node. The node from where the population starts dividing is called a root node. The nodes we get after splitting a root node are called decision nodes and the node where further splitting is not possible is called a leaf node. The question comes how do we know which feature will be the root node? In a dataset there can be 100's of features so how do we decide which feature will be our root node. To answer this question, we need to understand something called the "Gini Index". To select a feature to split further we need to know how impure or pure that split will be. A pure sub-split means that either you should be getting "yes" or "no". We basically need to know the impurity of our dataset and we'll take that feature as the root node which gives the lowest impurity or say which has the lowest Gini index.[10] Mathematically Gini index can be written as:

$$Gini\ Index = 1 - \sum_{i=1}^{n} \left(P_i\right)^2$$
$$= 1 - [(P_+)^2 + (P_-)^2]$$

Hyperparameters of RandomForest:

Hyperparameters are parameters whose values control the learning process and determine the values of model parameters that a learning algorithm ends up learning. The prefix 'hyper_' suggests that they are 'top-level' parameters that control the learning process and the model parameters that result from it. Hyperparameters are used in random forests to either enhance the performance and predictive power of models or to make the model faster.

Following hyperparameters increases the predictive power:

1. n_estimators– number of trees the algorithm builds before averaging the predictions.

2. max_features– maximum number of features random forest considers splitting a node.

3. mini_sample_leaf– determines the minimum number of leaves required to split an internal node.

Following hyperparameters increases the speed:

1. n_jobs– it tells the engine how many processors it is allowed to use. If the value is 1, it can use only one processor but if the value is -1 there is no limit.

2. random_state– controls randomness of the sample. The model will always produce the same results if it has a definite value of random state and if it has been given the same

hyperparameters and the same training data.

3. oob_score – OOB means out of the bag. It is a random forest cross-validation method. In this one-third of the sample is not used to train the data instead used to evaluate its performance. These samples are called out of bag samples

Important Features of Random Forest

1. Diversity- Not all attributes/variables/features are considered while making an individual tree, each tree is different.

2. Immune to the curse of dimensionality- Since each tree does not consider all the features, the feature space is reduced.

3. Parallelization-Each tree is created independently out of different data and attributes. This means that we can make full use of the CPU to build random forests.

4. Train-Test split- In a random forest we don’t have to segregate the data for train and test as there will always be 30% of the data which is not seen by the decision tree.

5. Stability- Stability arises because the result is based on majority voting/ averaging.[10]

- **Support vector machines**

Support vector machines (support vector networks),developed by Vladimir Vapnik[11], are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. It is one of the most robust prediction methods which is based on statistical learning frameworks. Here , we plot each data item as a point in n-dimensional space ,where n is the number of features, with the value of each feature being the value of a particular coordinate, and SVMs build a decision boundry which divides the data such that data of one class are all on one side of the hyperplane, and datas of the other class are all on the other side. This is called a linear classifier and the best line or decision boundary is known as the hyperplanes. There are many possible separating planes that correctly classify the training data . Fig 1 illustrates two classifying planes. Intuitively one prefers the solid plane since small perturbations of any point would not introduce misclassification errors. Without any additional information, the solid line is more likely to generalize better on future data. Geometrically, we can characterize the solid plane as being ‘furthest’ from both classes.

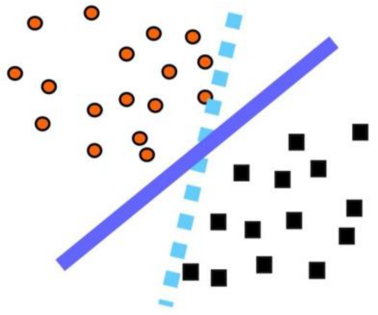


Fig 1:Two possible linear discriminant planes

To construct the plane ‘furthest’ from both the plane, we can examine the convex hull of each class’ training data and then find the closest points in the two convex hulls. The convex hull of a set of points is the smallest convex set containing the points. If we construct the plane that bisects these two points (w=d-c), the resulting classifier should be robust in some sense. In fig 2, the convex hull is indicated by the dotted line and the closest point in the two convex hull is labeled c and d.[12]

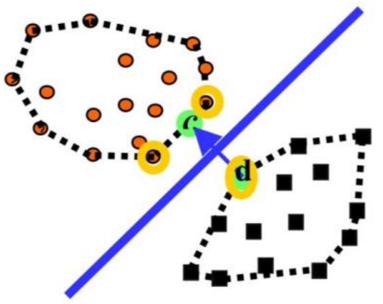


Fig 3.Best plane bisects closest points in the convex hull

One reasonable choice as the best hyperplane is the one that represents the largest separation, or margin, between the two classes. So we choose the hyperplane so that the distance from it to the nearest data point on each side is maximized. If such a hyperplane exists, it is known as the maximum-margin hyperplane and the linear classifier is known as a maximum-margin classifier; or equivalently, the perceptron of optimal . A plane supports a class if all points in that class are on one side of that plane. For the points with the class label +1 we would like there to exist w and b such that $w.x_i > b$ or $w.x_i-b>0$ depending on the class label. Let us suppose the smallest value of I$w.x_i$-b] is k, then $w.x_i-b \geq k$. The argument inside the decision function is invariant under a positive rescaling so we will implicitly fix a scale by requiring $w.x_i-b\geq 1$. For the points with the class label -1 we similarly require $w.x_i-b\leq -1$. To find the plane furthest from both sets, we can simply maximize the distance or margin between the support planes for each class as illustrated in Figure 3. The support planes are "pushed" apart until they "bump" into a small number of data points (the support vectors) from each class. The support vectors in Figure 3 are outlined in bold circles.

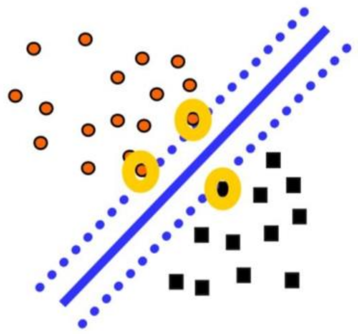


Fig 3 :Best plane maximizes the margin

The distance or margin between these supporting planes $w.x$=b+1 and $w.x$=b-1 is $\gamma= 2/||\mathbf{w}||_2$. Thus maximizing the margin is equal to minimizing 2/||**w**||2.

So far the discussion has been restricted to the case where the training data is linearly separable. To generalize the OSH (optimal separating hyperplane) to the non separable case, slack variable ε_i are introduced.[13] Hence the constraints are modified as

$$\boldsymbol{y_i[(w.x_i)}+b) \geq 1- \varepsilon_i,\ \varepsilon_i\geq 0, i=1.....l$$

The generalized OSH is determined by minimizing,

$$\Phi(\mathbf{w}, \varepsilon) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l} \varepsilon_i$$

When the samples are not linearly separable, a kernel function is used to transforms the data to a higher dimensional space where it is linearly separable . In the nonlinear case, resultant decision function has the following form

$$y(x) = \operatorname{sgn} \sum_{i=1}^{l} (\alpha_i y_i K(x_i \cdot x) + b)$$

Two of the most commonly used kernel functions are polynomial functions and Gaussian radial basis functions.[14] Basic theory of SVM was developed for two class classification, However A multi-class pattern recognition system can be obtained by combining two class SVMs. Usually there are two schemes for this purpose. One is the oneagainst-all strategy to classify between each class and all the remaining. The other is the one-against-one strategy to classify between each pair. While the former often leads to ambiguous classification region , later is preferred.

- **XGBoost**

XGBoost(Extreme Gradient boosting) is an end-to-end tree boosting system, which is used widely by data scientists to achieve state-of-the-art results on many machine learning challenges. XGBoost is a parallelized and carefully optimized version of the gradient boosting algorithm.

Boosting is an ensemble modeling technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models are added. Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor's error.[15]

In XG boost, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.

Some important features of XGboost are:

- Parallelization: The model is implemented to train with multiple CPU cores.
- Regularization: XGBoost includes different regularization penalties to avoid overfitting. Penalty regularizations produce

successful training so the model can generalize adequately.

- Non-linearity: XGBoost can detect and learn from non-linear data patterns.
- Cross-validation: Built-in and comes out-of-the-box.
- Scalability: XGBoost can run distributed thanks to distributed servers and clusters like Hadoop and Spark, so you can process enormous amounts of data. It's also available for many programming languages like C++, JAVA, Python, and Julia.[16]

Regularized Learning Objective:

For a given data set with n examples and m features D = {($\boldsymbol{x}_i$, y_i)} (|D|= n, $\boldsymbol{x}_i \in R^m$, $y_i \in R$), a tree ensemble model (shown in Figure 1) uses K additive functions to predict the output.

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i), \quad f_k \in \mathcal{F},$$

where $F = \{f(\boldsymbol{x}) = w_{q(x)}\}(q : R^m \rightarrow T, w \in R^T)$ is the space of regression trees (also known as CART). Here q represents the structure of each tree that maps an example to the corresponding leaf index. T is the number of leaves in the tree. Each f_k corresponds to an independent tree structure q and leaf weights w. Unlike decision trees, each regression tree contains a continuous score on each of the leaf, we use w_i to represent score on i-th leaf. For a given example, we will use the decision rules in the trees (given by q) to classify it into the leaves and calculate the final prediction by summing up the score in the corresponding leaves (given by w). To learn the set of functions used in the model, we minimize the following *regularized* objective.

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda\|w\|^2$$

Here l is a differentiable convex loss function that measures the difference between the prediction $\hat{y}_i$ and the target y_i. The second term **Ω** penalizes the complexity of the model (i.e., the regression tree functions). The additional regularization term helps to smooth the final learnt weights to avoid over-fitting. Intuitively, the regularized objective will tend to select a model employing simple and predictive functions. A similar regularization technique has been used in Regularized greedy forest (RGF) [17] model. Our objective and the corresponding learning algorithm is simpler than RGF and easier to parallelize. When the regularization parameter is set to zero, the objective falls back to the traditional gradient tree boosting

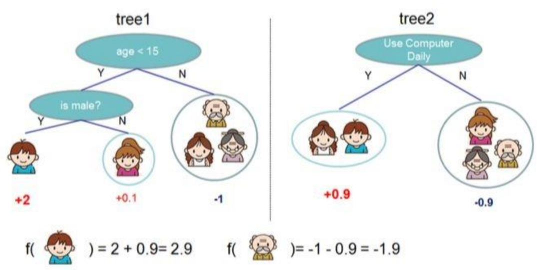


Fig 1

The tree ensemble in the above model includes functions as parameters and cannot be optimized using traditional optimization methods in Euclidean space. Instead, the model is trained in an additive manner. Formally, let be the $\hat{y}_i^{(t)}$ be prediction of the i-th instance at the t-th iteration, we will need to add f_t to minimize the following objective.

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

This means we greedily add the f_t that most improves our model .Second-order approximation can be used to quickly optimize the objective in the general setting .[18]

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i)] + \Omega(f_t)$$

where $g_i = \partial_{\hat{y}\,(t-1)}{}^{l}(y_i\,, \hat{y}^{\,(t-1)})$ and $h_i = \partial_{\hat{y}}{}^{2}{}_{(t-1)})l(y_i, \hat{y}^{\,(t-1)})$ are first and second order gradient statistics on the loss function. We can remove the constant terms to obtain the following simplified objective at step t.

- **GAUSSIAN NAIVE BAYES**

Gaussian Naive Bayes is an extension of Naive Bayes that follows Gaussian normal distribution and supports continuous data. Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without accepting Bayesian probability or using any Bayesian methods. An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification.

In Gaussian Nayes Bayes, when dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution. For example, suppose the training data contains a continuous attribute,x. The data is first segmented by the class, and then the mean and variance of x is computed in each class. Let μ_k be the mean of the values in x associated with class C_k, and let σ_k^2 be the Bessel corrected variance of the values in x associated with class C_k. Suppose one has collected some observation value v. Then, the probability density of v given a class C_k,$p(x=v|C_k)$, can be computed by plugging v into the equation for a normal distribution parameterized by μ_k and σ_k^2. That is,

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

An approach to create a simple model is to assume that the data is described by a Gaussian distribution with no co-variance (independent dimensions) between dimensions. This model can be fit by simply finding the mean and standard deviation of the points within each label, which is all what is needed to define such a distribution. Fig 1 indicates how a Gaussian Naive Bayes (GNB) classifier works. At every data point, the z-score distance between that point and each class-mean is calculated, namely the distance from the class mean divided by the standard deviation of that class. Thus, we see that the Gaussian Naive Bayes has a slightly different approach and can be used efficiently.[19]

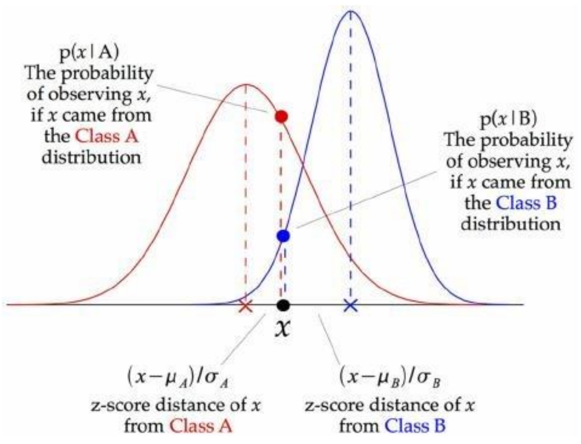

p(x | A)
The probability of observing x, if x came from the Class A distribution
p(x | B)
The probability of observing x, if x came from the Class B distribution
x
(x−μ_A)/σ_A
z-score distance of x from Class A
(x−μ_B)/σ_B
z-score distance of x from Class B

Fig 1

Another common technique for handling continuous values is to use binning to discretize the feature values, to obtain a new set of Bernoulli-distributed features; some literature in fact suggests that this is necessary to apply naive Bayes, but it is not, and the discretization may throw away discriminative information.[20]Sometimes the distribution of class-conditional marginal densities is far from normal. In these cases, kernel density estimation can be used for a more realistic estimate of the marginal densities of each class. This method, which was introduced by John and Langley, can boost the accuracy of the classifier considerably.[21]

- **The k-Nearest-Neighbours**

The *k*-Nearest-Neighbours (kNN) is a non-parametric classification method, which is simple but effective in many cases.[22] For a data record ***t*** to be classified, its *k* nearest neighbours are retrieved, and this forms a neighborhood of ***t***. Majority voting among the data records in the neighbourhood is usually used to decide the classification for ***t*** with or without consideration of distance-based weighting. However, to apply *K*NN we need to choose an appropriate value for *k*, and the success of classification is very much dependent on this value. In a sense, the *k*NN method is biased by *k*. There are many ways of choosing the *k* value, but a simple one is to run the algorithm many times with different *k* values and choose the one with the best performance. In order for *k*NN to be less dependent on the choice of *k*, Wang[23] proposed to look at multiple sets of nearest neighbours rather than just one set of *k*-nearest neighbours.

The proposed formalism is based on contextual probability, and the idea is to aggregate the support of multiple sets of nearest neighbours for various classes to give a more reliable support value, which better reveals the true class of ***t***. However, in its basic form the method is relatively slow, which needs O(n^2) to classify a new instance, though it is indeed less dependent on *k* and is able to achieve classification performance close to that for the best *k*NN.

*k*NN is a case-based learning method, which keeps all the training data for classification. One way to improve its efficiency is to find some representatives to represent the whole training data for classification, viz. building an inductive learning model from the training dataset and using this model(representatives) for classification. There are many existing algorithms such as decision trees or neural networks initially designed to build such a model. One of the evaluation standards for different algorithms is their performance. As *k*NN is a simple but effective method for classification and it is convincing as one of the most effective methods on Reuters corpus of newswire stories in text categorization, it motivates us to build a model for *k*NN to improve its efficiency whilst preserving its classification accuracy as well. Looking at Figure 1, a training dataset including 36 data points with two classes{square, circle} is distributed in 2-dimensional data space.

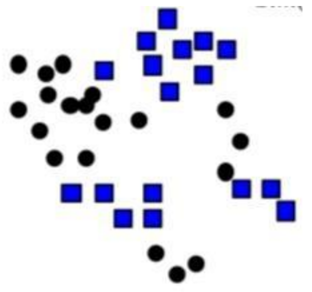


Fig 1: The distribution of data points

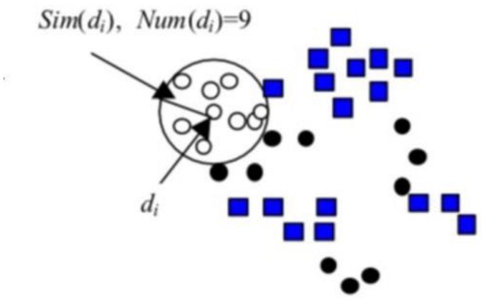


Fig 2: The first obtained representative

If we use Euclidean distance as our similarity measure, it is clear that many data points with the same class label are close to each other according to distance measure in many local areas. In each local region, the central data point d_i looking at Figure 2 Sim(d_i), Num(d_i)=9 di for example, with some extra information such as Num(d_i) - the number of data points inside the local region and Sim(d_i) - the similarity of the most distant data point inside the local region to d_i , might be an ideal representative of this local region. If we take these representatives as a model to represent the whole training dataset, it will significantly reduce the number of data points for classification, thereby to improve its efficiency.

Obviously, if a new data point is covered by a representative it will be classified by the class label of this representative. If not, we calculate the distance of the new data point to each representative’s nearest boundary and take each representative’s nearest boundary as a data point, then classify the new data point in the spirit of KNN. In model construction process, each data point has its largest local neighbourhood which covers the maximal number of data points with the same class label. Based on these local neighbourhoods, the largest local neighbourhood (called largest global neighbourhood) can be obtained in each cycle. This largest global neighbourhood can be seen as a representative to represent all the data points covered by it. For data points not covered by any representatives, we repeat the above operation until all the data points have been covered by chosen representatives. Obviously, we needn’t choose a specific k for our method in the model construction process, the number of data points covered by a representative can be seen as an optimal k but it is different in different representatives. The k is generated automatically in the model construction process. Further, using a list of chosen representatives as a model for classification not only reduces the number of data for classification, also significantly improves its efficiency. From this point of view, our proposed method overcomes the two shortcomings inherited in the kNN method.[24]

- **LOGISTIC REGRESSION**

Logistic Regression (LR) is one of the most important statistical and data mining techniques employed by statisticians and researchers for the analysis and classification of binary and proportional response data sets.[25]Some of the main advantages of LR are that it can naturally provide probabilities and extend to multi-class classification problems.[26] Another advantage is that most of the methods used in LR model analysis follow the same principles used in linear regression. What’s more, most of the unconstrained optimization techniques can be applied to LR. Recently, there has been a revival of LR importance through the implementation of methods such as the truncated Newton. Truncated Newton methods have been effectively applied to solve large scale optimization problems. Komarek and Moore[25] were the first to show that the Truncated-Regularized Iteratively Re-weighted Least Squares (TR-IRLS) can be effectively implemented on LR to classify large data sets, and that it can outperform the Support Vector Machines (SVM), which is considered a state of-the-art algorithm. Later on, trust region Newton method, which is a type of truncated Newton, and truncated Newton interior-point methods were applied for large scale LR problems. With regard to imbalanced and rare events data, and/or small samples as well as certain sampling strategies (such as choice-based sampling), however, the standard binary methods, including LR, are inconsistent unless certain corrections are applied. The most common correction techniques are prior correction and weighting. King and Zeng[25] applied these corrections to the LR model, and showed that they can make a difference when the population probability of interest is low.

Now let $\mathbf{X} \in R^{n\times d}$ be a data matrix where n is the number of instances (examples) and $\boldsymbol{d}$ is the number of features (parameters or attributes), and y be a binary outcomes vector. For every instance $xi \in R^d$(a row vector in $\mathbf{X}$), where $i = 1 \ldots n$, the outcome is either $y_i = 1$ or $y_i = 0$. Let the instances with outcomes of $y_i = 1$ belong to the positive class (occurrence of an event), and the instances with outcomes $y_i = 0$ belong to the negative class (non-occurrence of an event). The goal is to classify the instance x_i as positive or negative. An instance can be thought of as a Bernoulli trial (the random component) with an expected value $E[y_i]$ or probability p_i. A linear model to describe such a problem would have the matrix form

$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \varepsilon$, (1)

where ε is the error vector, and where

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \mathbf{X} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1d} \\ 1 & x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}, \beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_d \end{pmatrix}, \text{and } \varepsilon$$

The vector $\boldsymbol{\beta}$ is the vector of unknown parameters such that $x_i \leftarrow [1, x_i]$ and $\beta \leftarrow [\beta_0, \beta^T]$. From now on, the assumption is that the intercept is included in the vector $\boldsymbol{\beta}$. Now, since y is a Bernoulli random variable with a probability distribution

$$P(y_i) = \begin{cases} p_i, & \text{if } y_i = 1; \\ 1 - p_i, & \text{if } y_i = 0; \end{cases}$$

Then the expected value of the response is

$E[y_i] = 1(p_i) + 0(1 - p_i) = p_i = x_i\boldsymbol{\beta}$,

With a variance

$V(y_i) = p_i(1 - p_i)$.

It follows from the linear model

$y_i = x_i\boldsymbol{\beta} + \varepsilon_i$

that

$$\varepsilon_i = \begin{cases} 1 - p_i, & \text{if } y_i = 1 \text{ with probability } p_i; \\ -p_i, & \text{if } y_i = 0 \text{ with probability } 1 - p_i; \end{cases}$$

Therefore , ε_i has a distribution with an expected value

$E[\varepsilon_i] = (1 - p_i)(p_i + (-p_i)(1 - p_i) = 0$,

and a variance

$V(\varepsilon_i) = E[\varepsilon_i^2] - E[\varepsilon_i] = (1 - p_i)^2(p_i) + (-p_i)^2(1 - p_i) - (0)$

$= p_i(1 - p_i)$.

Since the expected value and variance of both the response and the error are not constant), and the errors are not normally distributed, the least squares approach cannot be applied. In addition, since $y_i \in \{0, 1\}$, linear regression would lead to values above one or below zero. Thus, when the response vector is binary, the logistic response function, as shown in Figure 1, is the appropriate one.[25]

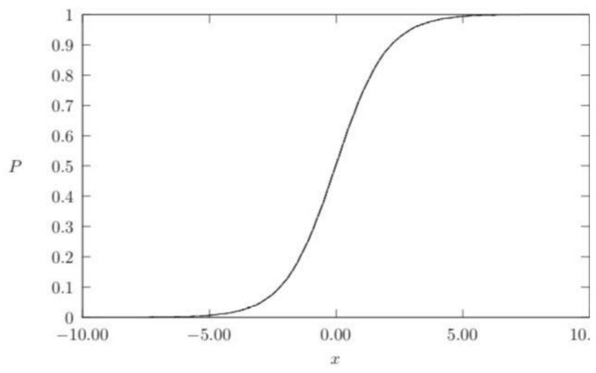


Fig 1

- **DECISION TREE**

Decision Trees are a type of Supervised Machine Learning where the data is continuously split according to a certain parameter. A normal tree includes root, branches and leaves. The same structure is followed in Decision Tree. It contains root node, branches, and leaf nodes. Testing an attribute is on every internal node, the outcome of the test is on branch and class label as a result is on leaf node.[27] A root node is parent of all nodes and as the name suggests it is the topmost node in Tree. A decision tree is a tree where each node shows a feature (attribute), each link (branch) shows a decision (rule) and each leaf shows an outcome (categorical or continues value) .[28] As decision trees mimic the human level thinking so it's so simple to grab the data and make some good interpretations. The whole idea is to create a tree like this for the entire data and process a single outcome at every leaf. Decision Tree is similar to the human decision-making process and so that it is easy to understand. It can solve in both situations whether one has discrete or continuous data as input. The example of Decision Tree is as follow [29]

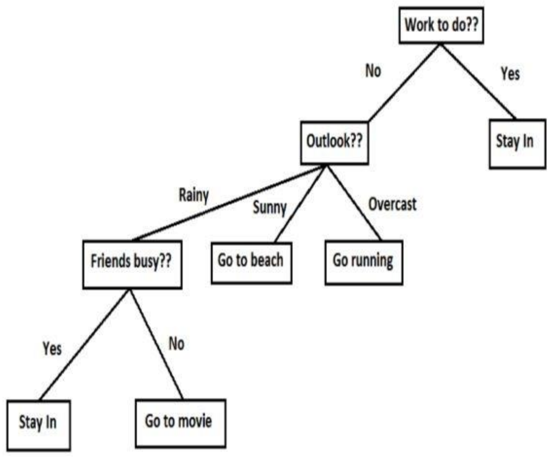


Fig 1

The most typical decision tree learning system is ID3, which originated in the concept learning system CLS, and finally evolved into C4.5 (C5.0), which can deal with continuous attributes. It is a learning guide, based on a decision tree composes of training subsets. If the tree fails to classify correctly all the given training subset, choose other training subset adding to the original subset, repeat it until the correct decision set. To train a number of training instance classification, a decision tree which can classify an unknown instance classification based on specific occurrence of attribute value sets. Using decision tree to classify instances, you can test gradually the value of the objects' properties starting at roots, and then going down the branch until reach a leaf node, in which class is the class of the object. Decision tree is a widely used method of classification. There are multiple decision tree methods, such as ID3, C4.5, PUBLIC, CART, CN2, SLIQ, SPRINT etc. Most developed decision tree is a variant of the core algorithm. The following will introduce firstly the basic idea of the decision tree classification, the construction and pruning of the decision tree, and then describe in detail the algorithm of ID3 and C4.5, and the analysis and improvement of the decision tree algorithm.

Construction and Pruning of decision tree:

Decision tree classification algorithm is usually divided into two steps: to construct and prune Decision Trees.

Construction of decision tree:

The input of decision tree construction algorithm is a set of classic labeled examples. The result of structure is a binary tree or ternary tree. The internal nodes of a binary tree (non-leaf nodes) are usually represented as a logical judgment, such as in the form of (AI=VI) logical judgment, AI is the attribute, of which VI is a value. Tree’s edge is the branch outcome of logic. The internal node of ternary tree is an attribute, of which edges are all values. Where there are several attribute values, there are several sides. Tree leaf nodes are category tag. Method of constructing decision tree is a top-down recursive structure. With the ternary tree as an example, its structural idea is starting to establish decision tree with single node represented the training sample. If the samples are all in the same class, it can be leaf nodes, and contents of nodes are the category tags. Otherwise, select an attribute based on certain strategy, divide example collections into several subsets In accordance with the attribute and values, and make all the examples in each subset has the same attribute value. Then deal with recursive process of each subset one by one. This idea is actually "divide and rule". So does the binary tree, and the only difference is how to choose better logical judgment.

The key to construct decision tree is how to choose better logical judgment or attribute. There can be many choices to the same set of examples. Research shows that, in general, the smaller the tree, the stronger forecasting ability. The key of constructing decision tree as small as possible, is to choose the proper attribute caused branch. Attribute selection depends on Impurity measurement method on the various examples of subset. Impurity measurement method includes Information Gain, Gain Ra-tio, Gini-index, distance measurement, X2 statistics, the weight of evidence, the minimum description length etc. Different measurement brings different effects, especially for multi valued attributes, select appropriate measurement method is greatly affected the result. ID3, C4.5, C5.0 algorithm use the concept of information gain to construct decision tree, while Gini-in-dex is used for CART x, where each classification decisions is relative to previously selected target classification.

Pruning of Decision tree

The object of Data mining is real world data, which are generally not perfect. Maybe there are some missing values in attribute field; or lack essential data resulting to incomplete data; or data are inaccurate even wrongly, or containing noise, so it is necessary to discuss the problem of noise.

The basic decision tree construction algorithm does not consider the noise, so the generated decision tree fits completely with training examples, which will lead to excessive fitting and will destruct predictive performance. Pruning is a technique to overcome noise, at the same time it also can make the tree simplified and easy to understand.[28,29]

6. Experimentation And Results

- *Database used*

The data base we are using to train and validate our algorithm is the Red Wine dataset from the UCI Machine learning repository. This database consists of 1599 rows and 12 columns. The 12th column in the dataset is the quality column or the result column.

The quality column consisted of values ranging from 3-8. For making the classification binary, the quality column was manipulated into Good wine and Bad wine (Wines having a quality above 6 as Good wine).

The new quality column was taken as the Y (Result) data and the other 11 columns made up the X (Inputs). These were then split into train and test data in the ratio 0.7:0.3 using the corresponding sklearn function.

- *Data Analysis*

Analyzing the data gives us insights into how we can efficiently train our model. Learning about each feature and attribute thoroughly is essential for Neural Network to work efficiently.

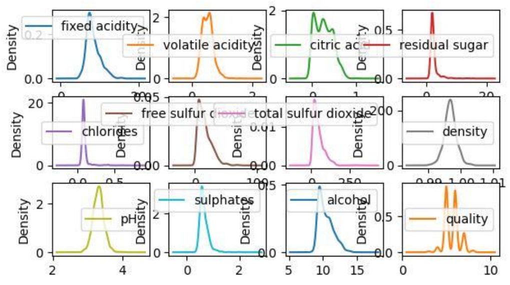
We start by printing out the shape of the dataset to work on. We get that it has 1599 rows and 12 columns. This information is essential to work on the data later on.

Next, we need to check if any of the features has null values present in it. These null values will badly affect the training and thus will have to be dealt with by filling in with the mean of that specific feature. This trick helps us to maximize the usability of the dataset. The number of null entries for each feature is given below.

fixed acidity	0
volatile acidity	0
citric acid	0
residual sugar	0
chlorides	0
free sulfur dioxide	0
total sulfur dioxide	0
density	0
pH	0
sulphates	0
alcohol	0

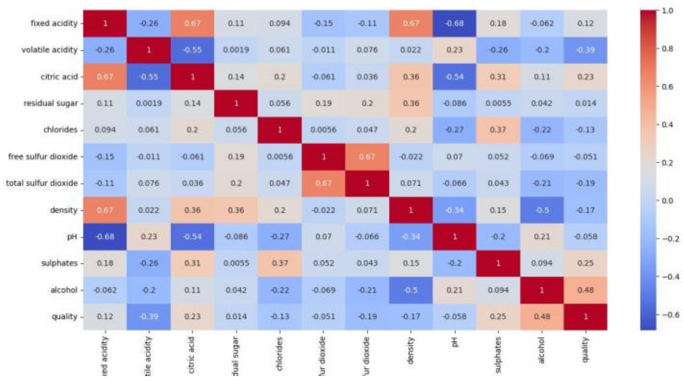
quality	0

This tells us that the dataset is complete and no further modification is needed on the dataset. Now let us plot some graphs and try to derive some results.First we are plotting a Distribution Plot of each feature to verify normal distribution. Normal distribution is necessary for any feature to give proper results. If not, we can improve the data by normalizing a feature.

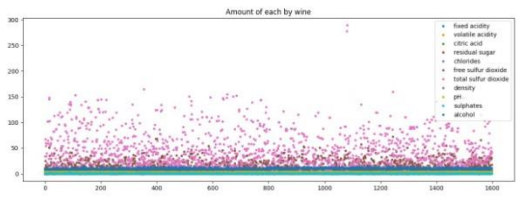


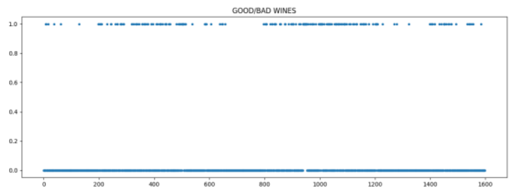
From the dist plot we can derive that all of the features follows a normal distribution and no manipulation is necessary.

Next we plot the correlation matrix. From the matrix, multiple results can be derived. The feature highly positively correlating with the quality tells us that it is important in determining the result. And the features highly negatively correlated with each other will not affect and removing one of those features is ideal



Next, after manipulating the data, we plot the amount of each feature in each wine and whether each wine is good or bad.





We then proceed to do train_test split on the data to train with the ratio 0.7:0.3. The test data was used to test for about 450 different physiochemical properties of wines thus giving us a pretty good accuracy score for all the classifiers (SVM, DTree, LogReg, XGB, RanFor, GaussianNB, KNN). A comparative analysis of each of these classifiers is given in the sections below.

- *Gaussian Naïve Bayes Classifier*

First we try to classify the train data using the Gaussian Naïve Bayes classifier. All the hyperparameters are kept as default and the X_test and Y_test are passed into the fit function. The GNB trains on a dataset of around 1150 datas of various wines and the results are stored in a variable called model3.

Accuracy of this fit is measured and found out to be 0.833. Tuning of the parameters is not conducted.

- *Logistic Regression*

  We then tried a hand at using the Logistic Regression method to model our sample ideally. Logistic Regression is not the ideal way to go for such a classification of a large dataset and we are hoping to get a result supporting that statement. In Logistic regression, A linear equation (z) is given to a sigmoidal activation function (σ) to predict the output (ŷ). Here the inputs and the weights form the linear equation which is activated by a sigmoidal function. These altogether gives out a model which is stored in the variable ‘model’.

  Accuracy of this classifier is around 0.8700, performing better than Gaussian NB.

- *K Nearest Neighbours*

  The KNN Classifier works on the assumption that similar data points are situated in close proximity with each other. i.e, In our data set Wines with similar features are grouped together with similar quality.

  For the KNN model, we design it with number of nearest neighbours initialized to 3 (n_neighbours = 3). This means that after making an ordered set of the closest data points to the query data point in ascending order it will take the first 3 data points only into account. And as it is a classifier it will output the mode of these 3 data points thus getting a result.

  Accuracy of the KNN model is 0.8720, giving us a better result than the Linear regression. This is because the assumption of similar data points situated closer is very close to the truth in our Red Wines dataset.

- *Support Vector Machine Classifier*

  Here, we use the SVC from Support Vector Machines and not NuSVC. SVC will not be practical for classifying large datasets as it will take a lot of process time but for a small dataset like the Red-Wine set, it is not a problem.

  The SVC has a kernel function and as we have called the SVC without specifying the kernel, the default kernel of RBF function will be used. Also the Gamma function to be used is in default ‘scale’ mode. So the function will be 1/(number of features * X.Var())

  The SVC generates an accuracy score of 0.86875 which is lesser than the models we saw before.

- *Extra Gradient Boosted Tree (XGBoost)*

XGBoost is basically a model consisting of many trees just like the Random forest model. But unlike the trees which usually have decision statements in their leaf nodes, XGBoost trees have scores of whether the instance belongs to a group. This is then converted into different classification categories when the tree reaches maximum depth.

XGBoost has many parameters but the most important ones include learning rate and the maximum depth of a single tree. We have called the XGBoost function with default parameters entirely. Tuning of these parameters can be conducted using GridSearch CV.

The accuracy of this model is around 0.8916 which is the highest we have seen. Thus it makes sense why XGB is one of the worlds most sought after algorithms.

- *Decision Tree*

  Decision tree takes our dataset features and starts making yes/no questions to continually split the dataset to isolate all the data points into different classes. Here we give 'entropy' as the loss function. Thus a split in node is performed only if the entropy of each of the resulting nodes is lower than the entropy of the parent node. Here we give the max_depth as the default value which is None. This is to ensure maximum accuracy as the Tree goes on until all nodes are pure. We give the random state parameter as 7 to increase the randomness in splitting and to obtain a deterministic behavior.

  Accuracy of the Decision Tree model is found out to be 0.8640 which is one of the lowest scores we arrived at. This is because decision trees are prone to overfitting. If we take the mean set accuracy of the training set it will be real close to 1.0. This is because as we build a tall tree, splitting the feature set until we get pure leaf nodes, we are likely overfitting the training set. Thus it results in a complex tree which will not behave well with other data sets. If we decide to make the tree small then it will underfit the data resulting in high bias. It is hard to find a balance between the two.

- *Random Forest Classifier*

  From the previous test we understand that a single decision tree has subpar accuracy, but when we combine multiple trees to control for bias and variance the results are much better. Random Forest works on having a large number of individual decision trees working as an ensemble. The random forest makes n number of trees from our dataset and it picks the most voted classification as the result classification. As these trees are very loss correlated with each other thus can together predict very accurately.

  We give the hyperparameter n_estimators = 1000 to increase the accuracy of the Classifier. This gives us 1000 trees for our dataset which is very less correlated with each other. Max_depth is given as the usual None to build each tree until we get pure leaf nodes. Random state is similar to the decision tree parameter and we initialize it to get a deterministic value.

The accuracy of our Random Forest Classifier is a good 0.89583 which is the best score we arrived at after testing it with all the other classifiers. The apparent high score with Random Forest Classifier is also explained with the less correlated features and result in our dataset.

Thus we choose the Random Forest Classifier to train the data.

Next we use a random Wine property set to see if the model classifies the wine.

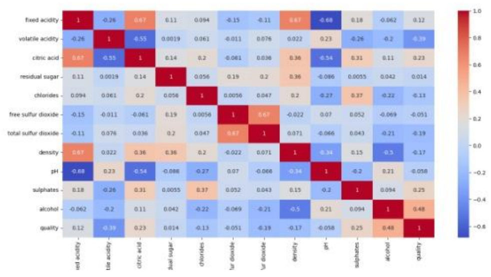
```
X =
[[8.0,0.9,0.01,2.4,0.082,15,50,0.9873,
3.8,5,50]]
```

When we predict using this input we are getting the output as ‘1’ or Good Wine.

This ensures us that our model is set and ready to go. We now proceed with our project by analyzing this trained model and then trying to maximize the accuracy even more.

- *Feature Importance extraction*

Features are what our model trains by using them to predict the result. But with each feature the role it plays in determining the result is different. This is seen clearly in the correlation matrix plotted with certain features like ‘Alcohol level’ highly influencing the ‘quality’.



We are trying to find the importance of each feature of our trained model. We use the feature_importances_ method to get each features importance in an array. Thus we get the result:

feature fixed acidity	7.44 %.
feature volatile acidity	11.59 %.
feature citric acid	7.92 %.
feature residual sugar	6.44 %.
feature chlorides	7.36 %.
feature free sulfur dioxide	6.51 %.
feature total sulfur dioxide	8.75 %.
feature density	9.77 %.
feature pH	6.06 %.
feature sulphates	11.02 %.
feature alcohol	17.15 %.

From this we understand that the most important feature is ‘alcohol’ and it is followed by ‘volatile acidity’. This result can be seen by observing the correlation matrix too.

The least imtant feature is pH which is negatively correlated with other features too. Thus removing pH as a feature might help in improving the accuracy score.

- *Balancing the Data classes using Oversampling*

  Imbalanced classes pose a serious issue when it comes to classification problems. Presence of minority classes leads to the model training the major class which will lead to less accurate models. This problem is tackled using a technique called as oversampling.

  Different types of oversampling are :

a. Random Over sampling
b. SMOTE
c. Borderline SMOTE
d. K-Means SMOTE
e. SVM SMOTE
f. Adaptive Synthetic Sampling (ADASYN)
g. SMOTE – Nominal and Continuous

Here we are using an oversampling technique called SMOTE (Synthetic Minoriy Oversampling Technique). SMOTE uses K-Nearest Neighbours algoritghm to create synthetic data for the minority classes. The synthetic data created will balance the Good and Bad wine datas (0 and 1) which will improve the performance considerably.

RandomForest accuracy before SMOTE : 89.583%

RandomForest accuracy after SMOTE : 93.132% and above

Better results can be generated by varying the oversampling technique used.

- *Tuning the hyperparameters for the best result using Random Search Cross Validation*

  The best way to think about hyperparameters is like the settings of an algorithm that can be adjusted to optimize performance. Hyperparameter tuning relies more on experimental results than theory and thus the best method to determine the optimal settings is to try many different combinations and evaluate the performance of each model. For tuning, we perform many iterations of K-Fold Cross Validation process but each time with different settings. We then compare the models, select the best one and train it on the full training set.

  First we create a hyperparameter grid to sample the parameters. On each iteration the algorithm will choose a different combination of these parameters. Random search is used to that we don't need to iterate across all possible settings but random ones.

  Here we give the number of K folds as cv = 3 and number of iterations as n_iters = 10. Increasing these will increase the accuracy score but will exponentially increase the training time which is not in the scope of this project.

Next we can evaluate this model to find out that we have had a good improvement over the previous accuracy scores.

- *Effect of removing negatively correlated features*

As mentioned earlier, eliminating certain features which negatively correlate with many other features plays an important part in training the model better.

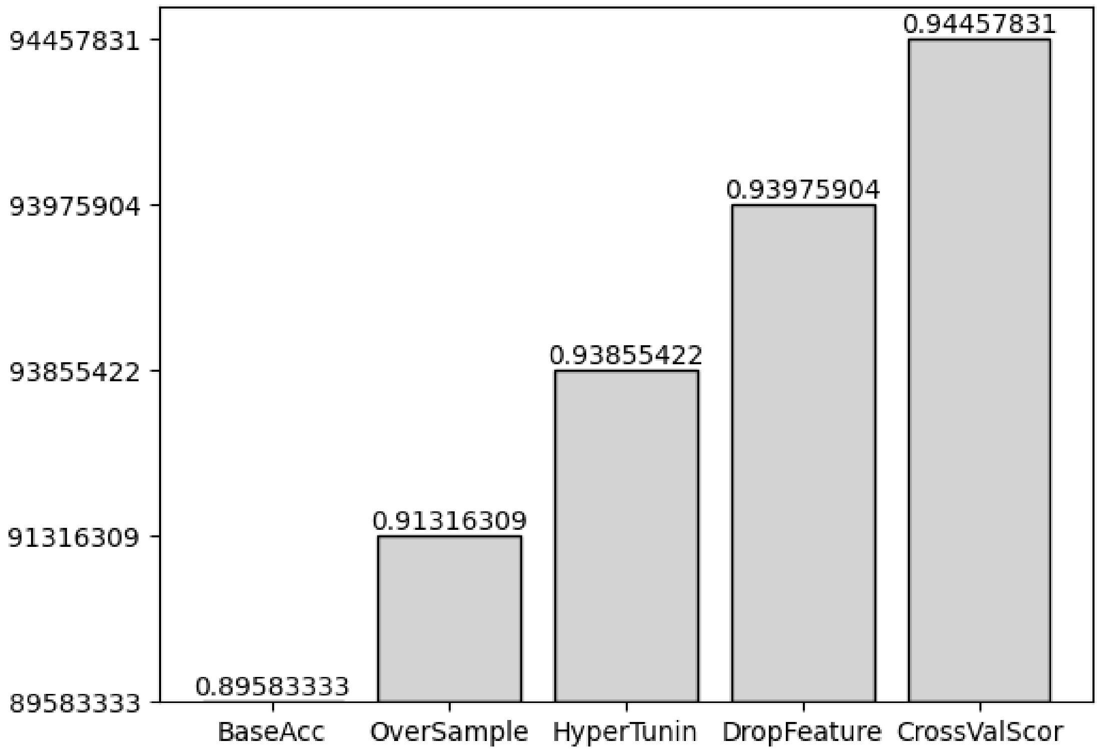
From the correlation matrix we observe that the feature ‘pH’ is negatively correlated with many other features including the ‘quality’.

We opt to remove this feature and then use oversampling on it to notice the difference in accuracy.

The output we get clearly tells us that this chucking of a bad feature helps in increasing the result significantly.

We now compare the accuracy after each step : from Base random forest classifier to accuracy after oversampling, tuning and removing features not necessary.

Finally we conduct cross validation using cv= 7 folds to arrive at the best accuracy achieved.



Comparison of Accuracy scores after each step

7.Results and Discussion

The project was completed successfully and all the expected outcomes were generated.

i.Red Wine dataset was studied and necessary data manipulation was made to make the classification binary. This is done for ease of analysis later on because of the sample space.

ii.Data was analysed and checked for null values. The quality of the dataset was determined using the DistPlot and Normal distribution was confirmed. Further, we analysed the features using a correlation matrix and came to the conclusion that ‘alcohol’ feature is highly positively correlated with ‘quality’ label whereas ‘pH’ feature is negatively correlated with many features. Such a feature will improve the accuracy upon dropping from the dataset.

iii.Train-Test split is used to split the dataset and then it is modelled using different classifiers.

iv.In this project, we have used 7 different classifiers. Out of the 7, RandomForest classifier performs the best with an accuracy score of 0.8953. Thus we conclude that RandomForestClassifier works best and then proceed to increase the accuracy.

v. Feature importance was extracted from the model and studied which showed us similar results which we pre-determined from the correlation matrix. We find that ‘alcohol’ feature affects the ‘quality’ the most and ‘pH’ affecting it the least.

vi. We have two classes of results; Good Wine(1) and Bad Wine(0). But the number of these two are not the same. Imbalanced classes in classification problems will lead to inaccurate models. To prevent this, Oversampling methods are used. SMOTE is being used in this and it increases the accuracy drastically from the base RandomForest accuracy.

vii.Hyperparameter tuning of RandomForestClassifier can be conducted to further improve upon the efficiency. Here, we use a random search grid and then use RandomSearchCV to take random values from these parameters and through cross-validation take the scores. The best score-giving parameters will be returned which we can use to train later. The score is higher than previous score and it can be made even better butdue to time constraints number of iterations of the RandomSearch was limited to 10. Increasing this will lead to better parameters.

viii. Effect of removing negatively correlated parameters will help the accuracy score. From the correlation matrix, we find ‘pH’ affects everything negatively. Thus we drop ‘pH’ and then do the oversampling and tuning. This leads to a higher score.

ix. Finally, cross validation was performed on the dataset to get the best accuracy score.

i. x. Project Wine Classification was concluded and it classified wines into the required classes. Necessary results were also extracted. Study of accuracy and ways to improve it was discussed.

8. Future Work

Will do regression problem on a bigger dataset which requires data manipulation at a larger extent. Looking to execute a live Cryptocurrency price predictor using the previous data available.

9. References

[1] - Gourmet Girl "The wines of Portugal" Gourmet Girl Magazine, Accessed: 6 December 2009

[2] - Disney, A. R. (2009). A history of Portugal and the Portuguese empire : from beginnings to 1807. Volume 1, Portugal. Cambridge University Press. p. 259

[3] - "Vinho Verde: 101 | Vinho Verde". winesofvinhoverde.com.

[4] - P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

[5] -5. Breiman, L.: Random Forests. Machine Learning 45 (1) pp. 5–32 (2001).

[6]. .[Amit, Y., Geman, D.: Shape quantization and recognition with randomized trees. Neural

Computation 9(7) pp. 1545-1588 (1997)

[7]. Opitz, D.; Maclin, R. (1999). "Popular ensemble methods: An empirical study". Journal of Artificial Intelligence Research. 11: 169–198. doi:10.1613/jair.614

[8.] "What is Bagging (Bootstrap Aggregation)?". CFI. Corporate Finance Institute. Retrieved December 5, 2020

[9] Breiman, L.: Bagging predictors. Machine Learning, 24, 123–140 (1996)

[10].https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/#:~:text=RandomforestisaSupervised,averageincaseofregression.

[11]. . Vapnik, V. N. The nature of statistical learning theory. New York: Springer, 1995

[12]. Bennett K. and Bredensteiner E. Geometry in Learning, in Geometry at Work, C. Gorini Editor, Mathematical Association of America, Washington D.C., 132-145, 2000.

[13.] Daniel L. Swets, John (Juyang) Weng, Using Discriminant Eigenfeatures for image retrieval,IEEE trans. pattern anal. mach. intell. Vol. 18, no. 8, 1996,pp.831-836

[14]. Simon Hayekins, Neural Networks: A comprehensive foundation, published by Pearson education (Singapore) Pte. Ltd

[15.]Leo Breiman (1996). "BIAS, VARIANCE, AND ARCING CLASSIFIERS" (PDF). TECHNICAL REPORT. Archived from the original (PDF) on 2015-01-[19.] Retrieved 19 January 2015. Arcing [Boosting] is more successful than bagging in variance reduction

[16.] J. Ye, J.-H. Chow, J. Chen, and Z. Zheng. Stochastic gradient boosted distributed decision trees. In Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM '09.

[17].T. Zhang and R. Johnson. Learning nonlinear functions using regularized greedy forest. IEEE Transactions on Pattern Analysis and Machine Intelligence, 36(5), 2014.

[18.] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. Annals of Statistics, 28(2):337–407, 2000.

[19.] https://iq.opengenus.org/gaussian-naive-bayes/

[20.] Hand, D. J.; Yu, K. (2001). "Idiot's Bayes — not so stupid after all?". International Statistical Review. 69 (3)

[21]. John, George H.; Langley, Pat (1995). Estimating Continuous Distributions in Bayesian Classifiers. Proc. Eleventh Conf. on Uncertainty in Artificial Intelligence. Morgan Kaufmann. pp. 338–345. arXiv:1302.4964

[22.]D. Hand, H. Mannila, P. Smyth.: Principles of Data Mining. The MIT Press. (2001)

[23]. H. Wang.: Nearest Neighbours without k: A Classification Formalism based on Probability, technical report, Faculty of Informatics, University of Ulster,

N.Ireland, UK (2002)

[24] T. Mitchell.: Machine Learning. MITPress and McGraw-Hill (1997)

[25] Maalouf,M. (xxxx) 'Logistic Regression in Data Analysis: An Overview',International Journal of Data Analysis Techniques and Strategy(IJDATS),

[26.] T. Hastie, R. Tibshirani, and J. Friedman. The Elements of Statistical Learning. Springer Verlag, 2 edition, 2009.

[27]. Gershman A, Meisels A, Lüke KH, Rokach L, Schclar A, Sturm A. A Decision Tree Based Recommender System. InIICS 2010

Jun 3 (pp. 170-179).

[28]. Jadhav SD, Channe HP. Efficient recommendation system using decision tree classifier and collaborative filtering. Int. Res. J. Eng.

Technol. 2016;3:2113-8

[29]. Banu GR. A Role of decision Tree classification data Mining Technique in Diagnosing Thyroid disease. International Journal of

Computer Sciences and Engineering. 2016;4(11):111-5.