Machine Learning Assignment 3 Project Report

Saksham Sinha, Srubin Sethu Madhavan, Rahul Agrahari

December 15, 2017

1 Introduction

As a part of machine learning coursework requirement, this report is based on an assignment 3 project which is predicts ratings for various wines. Wine Rating prediction task has two wine data sets-

1. Red Wine Dataset
2. White Wine Dataset

We chose white wine dataset, made and donated for public use by PauloCortez et al.[1], as it has 4898 observations which is around 4 times larger than the red wine dataset. Hence this will help us train on more data making our algorithm more robust and less susceptible to overfitting. Also its not too large that will make algorithms to take huge time to train. The peculiarity of the white wine dataset is that it has both regression and classification properties. Hence both regression and classification algorithms can be used to predict the quality depending which property we chose. We chose classification as the wine quality is quantized from 1 to 10. We decided to club the 10 classes into groups of 3 using following convention -

1. Classes 0 to 4 are mapped to new class 0
2. Classes 5 to 6 are mapped to new class 1
3. Classes 7 to 9 are mapped to new class 2

2 Data and Preprocessing

Since we have used white wine dataset, before going forward to train our algorithms, it's necessary to analyze the data for its quality. Noisy data can actually train algorithm to learn to classify noise into classes hence reducing the accuracy. Also we have to do some analysis on features and figure out there co-relations so that we can do some feature selection to get faster training of algorithms. And also we need to normalize the data if required. The following description of features are taken from the original paper of dataset by PauloCortez et al.[1] and wikipedia[2].

1. Fixed acidity: Amount of non-volatile or fixed amount of acid in the wine.
2. Volatile acidity: Amount of volatile acid present in wine
3. Citric acid: Citric acid adds freshness to wine.
4. Residual sugar: It adds sweetness to wine.
5. Chlorides: The amount of sodium chloride in wine.
6. Free sulfur dioxide: it prevents microbial growth and the oxidation of wine.[3]
7. Total sulfuric dioxide: Total amount of SO2 in bounded and free form. In low concentrations, SO2 is undetectable in wine below 50 ppm but above this level SO2 changes smell and taste of wine.
8. Density: Density of wine.

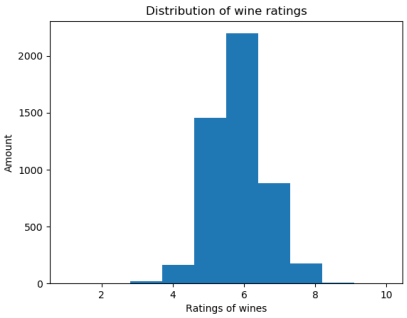


Figure 1
Wine quality distribution.

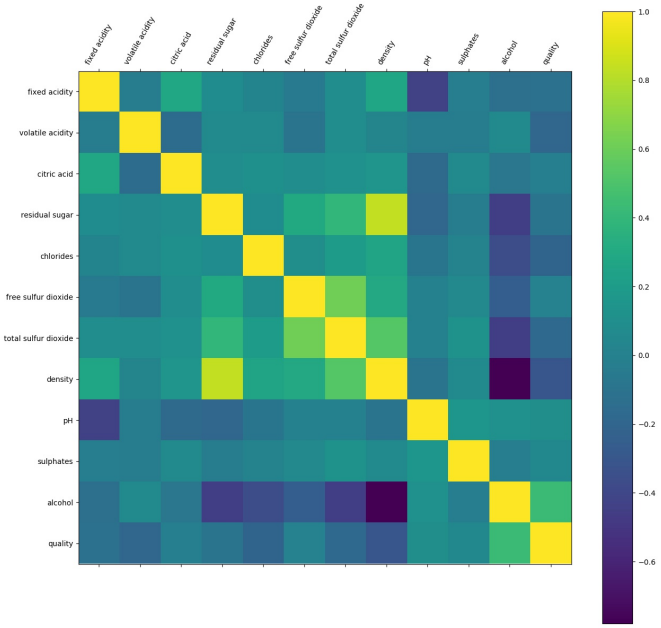


Figure 2
Correlation Matrix between features.

9. pH: pH value of wine.
10. Sulfates: Amount of sulfates in wine. This is how SO2 gets into wine.
11. Alcohol: Amount of alcohol present in wine.
12. Quality: Wine quality between 1 to 10.

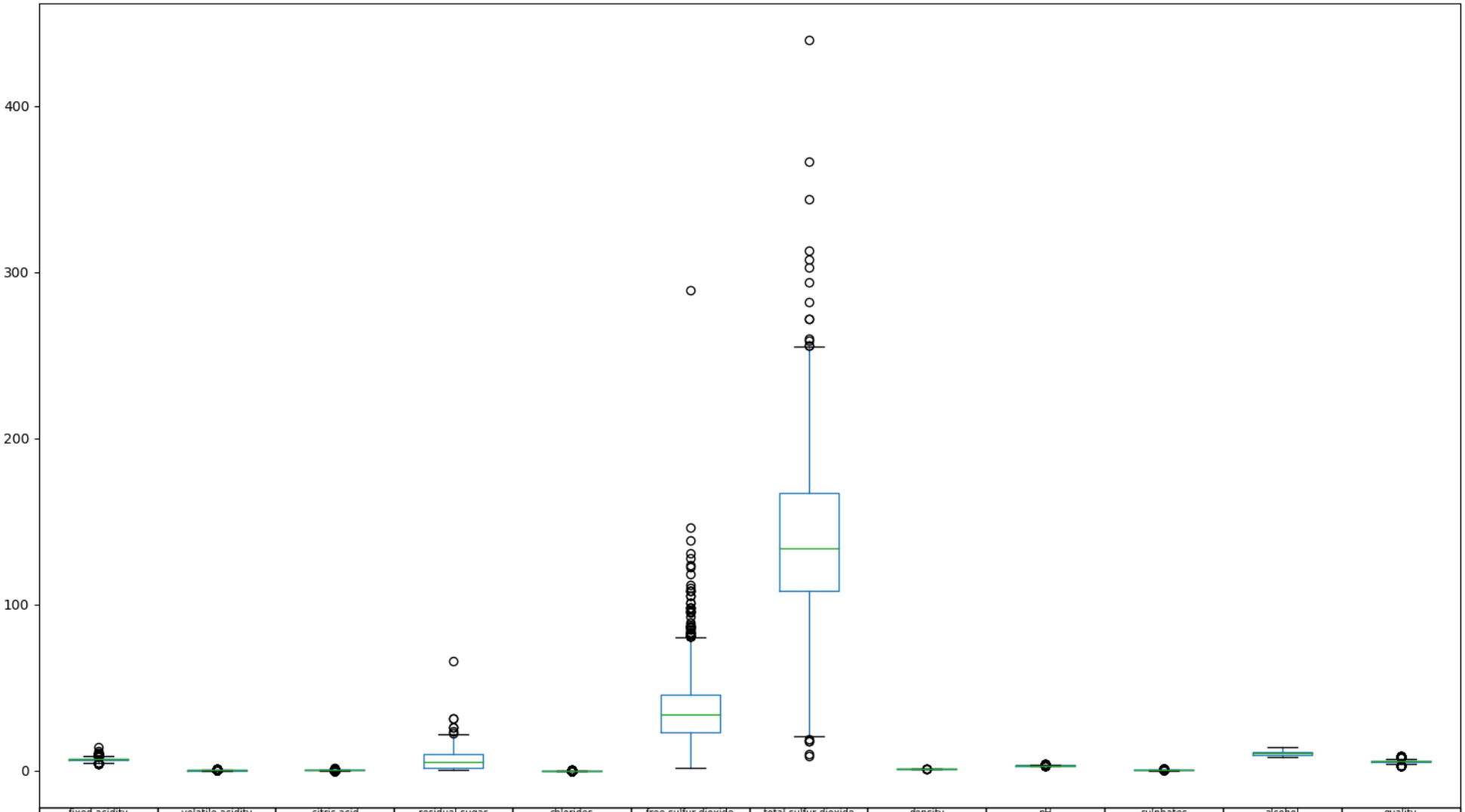
Figure 3 shows the statistics and box plot of each of the feature in white wine dataset and Figure 2 shows the correlation matrix between features. Statistics covers minimum value, maximum value, 25 percentile, 50 percentile, 75 percentile, standard deviation mean and total count. Correlation matrix gives the linear relationship between the features i.e. how one feature is dependent on another. For e.g from the figure 2, we can see that density has high correlation with SO2. This means, higher the amount of SO2 in wine, denser will be the wine. On the contrary, alcohol has very low correlation with density. Hence, even with higher volume of alcohol, the change in the density will be very minimal. Correlation matrix is useful for regression algorithms, however in order to analyze the features, we have calculated this correlation matrix. Figure 1 show the skew nature of data. We can clearly see from the figure 1 that the data is unbalanced. We have high amount of data of average quality wines and very low data of poor and excellent quality wines.

3 Algorithm and feature selection

We chose three classification algorithms to predict the classes for our dataset-

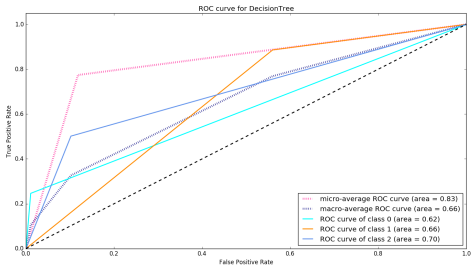
1. K-Nearest Neighbor
2. Logistic Regression
3. Decision Tree

We have tried doing feature selection by finding out feature importance for each of the algorithm. However, while finding out the feature importance for logistic regression and KNN on scikit learn,

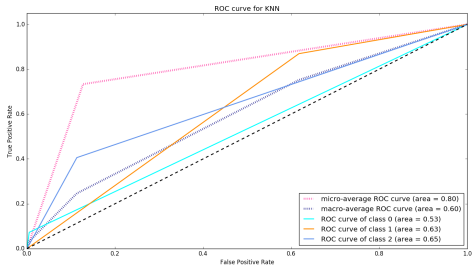


	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	4898.0	4898.0	4898.0	4898.0	4898.0	4898.0	4898.0	4898.0	4898.0	4898.0	4898.0	4898.0
mean	6.85478766844	0.278241118824	0.334191506737	6.39141486321	0.0457723560637	35.3080849326	138.360657411	0.99402737648	3.18826663944	0.489846876276	10.5142670478	5.87790935076
std	0.843868227688	0.100794548425	0.121019804203	5.07205778401	0.0218479680937	17.0071373252	42.4980645541	0.00299090691694	0.151000599615	0.114125833949	1.23062056776	0.885638574968
min	3.8	0.08	0.0	0.6	0.009	2.0	9.0	0.98711	2.72	0.22	8.0	3.0
25%	6.3	0.21	0.27	1.7	0.036	23.0	108.0	0.9917225	3.09	0.41	9.5	5.0
50%	6.8	0.26	0.32	5.2	0.043	34.0	134.0	0.99374	3.18	0.47	10.4	6.0
75%	7.3	0.32	0.39	9.9	0.05	46.0	167.0	0.9961	3.28	0.55	11.4	6.0
max	14.2	1.1	1.66	65.8	0.346	289.0	440.0	1.03898	3.82	1.08	14.2	9.0

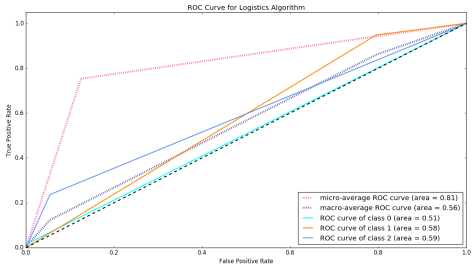
Figure 3: Boxplot and statistics of each features



(a)
Area Under Curve for Decision Tree



(b)
Area Under Curve for K-nearest neighbor



(c) Area Under Curve for K-nearest neighbor

Figure 4: Area Under Curve for a)Decision tree, b)K-Nearest, and c)Logistic Algorithms

we found out that its not possible as these implementation don't expose there feature importance variable[4, 5]. Although one recommendation was to use area under the curve for each feature to perform KNN feature selection[6]. However, we lost quite a lot of time finding the solution to do feature selection using ROC and hence decided to skip the feature selection. Figure 4 shows ROCs for all the three algorithms used. For evaluation we use K-fold evaluation with 10 folds with f1 score and accuracy score metrics to compare the results.

3.1 KNN

We iteratively ran KNN using 10-fold validation with increasing neighbors with f1 score metric. The results showed that the best score is achieved with 3 neighbors. Hence we decided to go ahead with 3 neighbors for comparison with other algorithms.

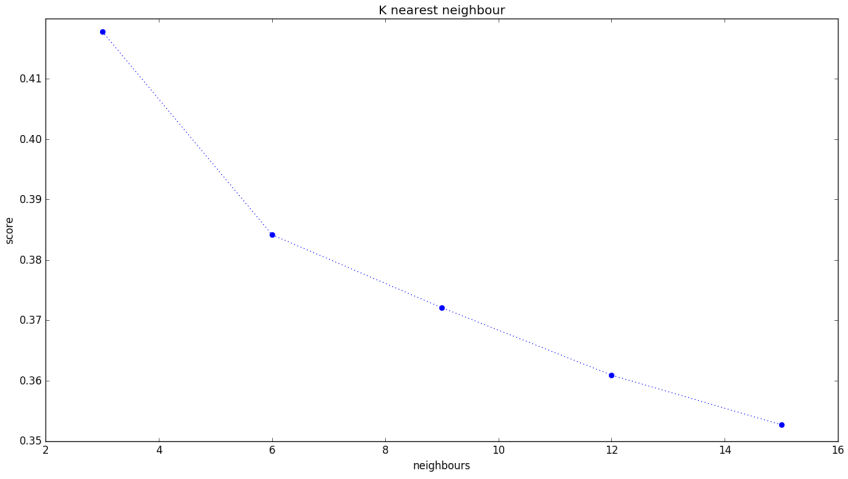


Figure 5: K-Nearest neighbor with 10 fold validation.

3.2 Decision Tree

We iteratively ran decision tree algorithm using 10-fold cross validation with increasing depth with f1 score metric. The results showed that the best score is achieved with the depth of 9. Hence we decided to go ahead with depth 9 for comparison with other algorithms.

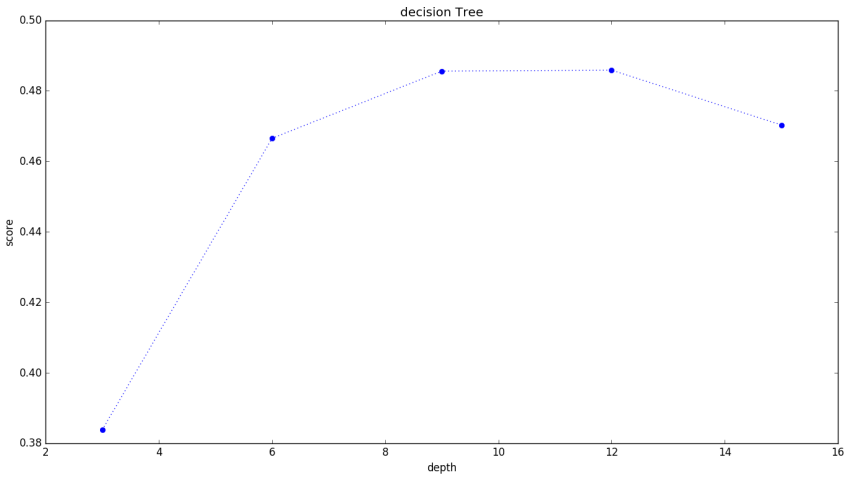


Figure 6: Decision tree with 10 fold validation.

3.3 Logistic Regression

For logistic, we tried using different solvers and class weights as none and balanced. We found that with class weight as balanced, we got poor results on all solvers as the weights were more biased for the higher frequency classes. For default class weight i.e. none, all results were similar with every solver we used. Hence, we decided to take 'newton-cg' as a solver for final comparison.

3.4 Comparison

We chose two metrics to compare the performances. First we selected accuracy score and second as f1 score. We selected f1 score as it is immune to unbalanced data. We tested all three algorithms on these two metrics and found that logistic regression works the best and k-nearest neighbor algorithm performs the worst on the accuracy score while for f1 score, decision tree performs best while logistic and KNN performs nearly the same. The comparison is shown in figure 7.

4 Conclusion

We conclude that for the white wine dataset, decision tree should be used as it had best score on f1 metric. It's also to be noted that quality of wine also depends on other factors which the dataset doesn't focus on i.e. age of wine, packaging and storage. Adding these parameters might increase the efficiency of prediction of wine rating.

5 Appendix

Following are the details of the contribution of the individual members-

5.1 Rahul Agrahari

1. Trinity ID : 17316735
2. Contributions : Worked on Logistic regression algorithm and created chart of comparison. Provided equal contribution in report making.

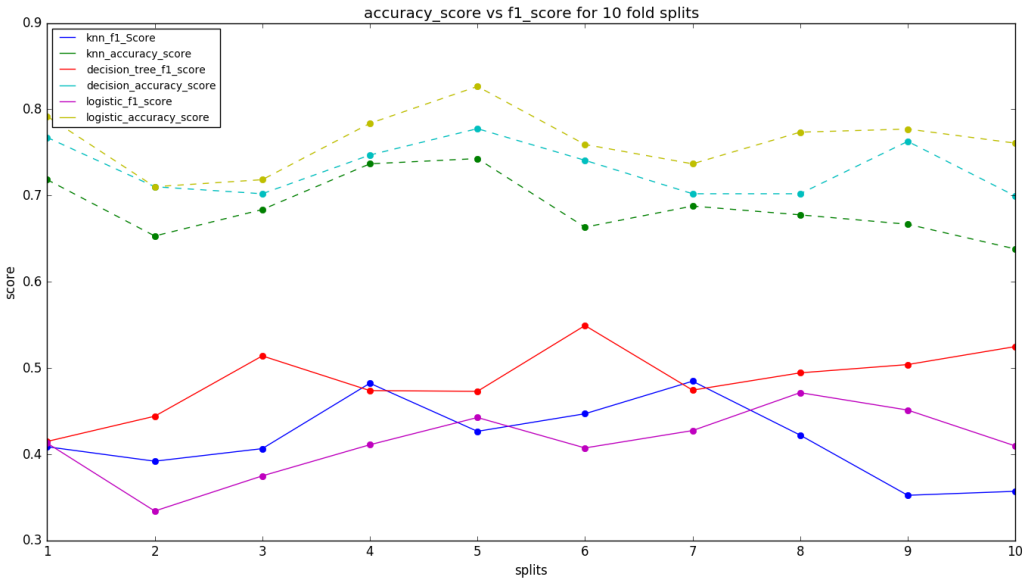


Figure 7: Comparison of algorithms on accuracy and f1 scores

3. Time spent : 15 hours

5.2 Saksham Sinha

1. Trinity ID : 17311349
2. Contributions : Worked on K-nearest neighbor algorithm and created chart of KNN. Provided equal contribution in report making.
3. Time spent : 15 hours

5.3 Srubin Sethu Madhavan

1. Trinity ID : 17311213
2. Contributions : Worked on Decision tree algorithm and created chart of Decision tree algorithm. Provided equal contribution in report making.
3. Time spent : 15 hours

For each member, equal contribution in report involves figuring out the issues, creating charts of box plots, calculating statistics of each feature, finding out the dataset information, figuring out a way on how can we combine box plot and statistics table in a single plot and also figuring out a way to show this plot sideways on latex.

6 Source code

1. Classifier.py

```
from sklearn.feature_selection import RFE
class classifier:

    def __init__(self, X, y):
        self.X = X
        self.y = y

    def decision_tree(self, n):
```

```
        from sklearn.tree import DecisionTreeClassifier
        simpleTree = DecisionTreeClassifier(max_depth=9)
        return simpleTree.fit(self.X, self.y)

    def random_forest(self):
        from sklearn.ensemble import RandomForestClassifier
        simpleTree = RandomForestClassifier(max_depth=5)
        return simpleTree.fit(self.X, self.y)

    def gradient_boosting(self):
        from sklearn.ensemble import GradientBoostingClassifier
        simpleTree = GradientBoostingClassifier(max_depth=5)
        return simpleTree.fit(self.X, self.y)

    def support_vector(self):
        from sklearn.svm import SVC
        supportVector = SVC(kernel='linear', random_state=0, probability=True)
        return supportVector.fit(self.X, self.y)

    def linear_support_vector(self):
        from sklearn.svm import LinearSVC
        supportVector = LinearSVC(random_state=0)
        return supportVector.fit(self.X, self.y)

    def logistic(self):
        from sklearn.linear_model import LogisticRegression
        linear = LogisticRegression(multi_class='multinomial', solver='newton-cg'
        return linear.fit(self.X, self.y)

    def KNN(self, n):
        from sklearn.neighbors import KNeighborsClassifier
        knn = KNeighborsClassifier(3)
        return knn.fit(self.X, self.y)

    def sgd(self):
        from sklearn.linear_model import SGDRegressor
        sgd = SGDRegressor()
        return sgd.fit(self.X, self.y)
```

2. configuration.py

```
def isTasty(quality):
    if quality >= 7:
        return 1
    else:
        return 0


def levelOfTaste(quality):
    if quality >= 7:
        return 1
    elif quality < 7 and quality >=5:
        return 0
    else:
        return -1


class configure:
```

```
    def __init__(self, dataseetName):
        self.datasetName = dataseetName

    def getdataset(self):
        import pandas as pd
        dataset = pd.read_csv(self.datasetName, ';')
        return dataset

    def binaryClassConversion(self, featureName, split=2):

        dataset = self.getdataset()

        if split == 2:
            dataset['taste'] = dataset[featureName].apply(isTasty)
        else:
            dataset['taste'] = dataset[featureName].apply(levelOfTaste)
        return dataset
```

3. DatasetTraining.py

```
from classifiers import classifier


class training:

    def __init__(self, dataset=None, X=None, y=None):

        self.dataset = dataset
        self.X = X
        self.y = y

    def k_fold_cross_validation(self, n_split=2, classifierName=None, n=0):

        from sklearn.model_selection import KFold
        import numpy as np
        from evalMetrics import evalMetric

        kf = KFold(n_splits=n_split)
        X = np.array(self.X)
        y = np.array(self.y)
        dataset = np.array(self.dataset)
        score = 0
        feature_imp = 0
        scores = []
        scores1 = []
        score1 = []
        score2 = []
        for train_indices, test_indices in kf.split(dataset):
            x_train = X[train_indices]
            x_test = X[test_indices]
            y_train = y[train_indices]
            y_test = y[test_indices]
            clf = classifier(x_train, y_train)
            if classifierName == "randomforest":
                c = clf.random_forest()
                predict = c.predict(x_test)
            elif classifierName == "svm":
                c = clf.support_vector()
                predict = c.predict(x_test)
```

```
            elif classifierName == "gradientboosting":
                c = clf.gradient_boosting()
                predict = c.predict(x_test)
            elif classifierName == "decisiontree":
                c = clf.decision_tree(n)
                predict = c.predict(x_test)
            elif classifierName == "linearsvm":
                c = clf.linear_support_vector()
                predict = c.predict(x_test)
            elif classifierName == "logistic":
                c = clf.logistic()
                predict = c.predict(x_test)
            elif classifierName == "knn":
                c = clf.KNN(n)
                predict = c.predict(x_test)
            elif classifierName == "sgd":
                c = clf.sgd()
                predict = c.predict(x_test)

            e1 = evalMetric(y_test, predict).F1_score()
            score1.append(e1)
            e = evalMetric(y_test, predict).accuracy_skore()
            score2.append(e)
            # scores1.append(e1)
            score += e1

            if classifierName == 'randomforest':
                feature_imp += c.feature_importances_
            # print("Score: ", score)

        print("Average Score: ", score / n_split)
        scores.append(score1)
        scores.append(score2)
        return score / n_split, feature_imp / n_split, scores

    def normal_training(self, classifierName=None, featureScaling = False, test_s

        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(self.X, self.y, test_
        trainedClassifier = None
        y_pred = None
        if featureScaling:
            from sklearn.preprocessing import StandardScaler
            sc_X = StandardScaler()
            X_train = sc_X.fit_transform(X_train)
            X_test = sc_X.transform(X_test)
        clf = classifier(X_train, y_train)
        if classifierName == "randomforest":
            trainedClassifier, x_f = clf.random_forest()
            y_pred = trainedClassifier.predict(X_test)

        elif classifierName == "svm":
            trainedClassifier = clf.support_vector()
            y_pred = trainedClassifier.predict(X_test)

        elif classifierName == "gradientboosting":
            trainedClassifier = clf.gradient_boosting()
            y_pred = trainedClassifier.predict(X_test)
```

```
        elif classifierName == "decisiontree":
            trainedClassifier, x_f = clf.decision_tree()
            y_pred = trainedClassifier.predict(X_test)

        elif classifierName == "linearsvm":
            trainedClassifier = clf.linear_support_vector()
            y_pred = trainedClassifier.predict(X_test)

        elif classifierName == "logistic":
            trainedClassifier = clf.logistic()
            y_pred = trainedClassifier.predict(X_test)

        elif classifierName == "knn":
            trainedClassifier = clf.KNN()
            y_pred = trainedClassifier.predict(X_test)

        elif classifierName == "sgd":
            trainedClassifier = clf.sgd()
            y_pred = trainedClassifier.predict(X_test)

        else:
            print ("classifier not found")
        dict = {"X_train": X_train, "X_test": X_test, "y_train": y_train, "y_test
                "y_pred": y_pred, "x_f":x_f}

        return dict
```

4. evalMetrics.py

```
from sklearn.metrics import precision_recall_fscore_support, accuracy_score, confu
class evalMetric:

    def __init__(self, y_test=None, y_pred=None):

        self.y_test = y_test
        self.y_pred = y_pred

    def precision_recall_fscore_supports(self):
        return precision_recall_fscore_support(self.y_test, self.y_pred, average=

    def confusion_metric(self):
        return confusion_matrix(self.y_test, self.y_pred)

    def area_under_the_curve(self):
        return roc_auc_score(self.y_test, self.y_pred)

    def accuracy_skore(self):
        return accuracy_score(self.y_test, self.y_pred)

    def F1_score(self):
        return f1_score(self.y_test, self.y_pred, average='macro')

    def roc(self):
        return roc_auc_score(self.y_test, self.y_pred)
```

5. Graph.py

```
import matplotlib.pyplot as plt
```

```
from configuration import configure
import numpy as np

cfg = configure("winequality-white.csv")
ds=cfg.getdataset()

#box plot
ds.plot(kind='box', subplots=False, layout=(5,3), legend = False, figsize = (30,1
ax1 = plt.axes()
x_axis = ax1.axes.get_xaxis()
x_axis.set_visible(False)
x_label = x_axis.get_label()
x_label.set_visible(False)
plt.show()
plt.savefig("Charts/"+ "BoxwithTable" + "_plot_scores.jpg")

#correlation matrix
ds_corr = ds.corr()
fig = plt.figure(figsize=(15,15))
ax = fig.add_subplot(111)
cax = ax.matshow(ds_corr)
fig.colorbar(cax)
ticks = np.arange(0,12,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(list(ds),rotation=60)
ax.set_yticklabels(list(ds))
plt.show()
```

6. roc_curve.py

```
import numpy as np
from itertools import cycle
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
from scipy import interp
from sklearn.model_selection import train_test_split
from configuration import configure
from matplotlib import pyplot as plt
from evalMetrics import evalMetric
from sklearn.multiclass import OneVsRestClassifier

# importing the dataset and pre-procesing it

datasetName = 'winequality-white.csv'

cnfg = configure(datasetName)

dataset = cnfg.binaryClassConversion('quality', 3)

X = dataset[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar'
             'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol']]
y = dataset['taste']
y = label_binarize(y, classes=[-1, 0, 1])
n_classes = y.shape[1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
from sklearn.tree import DecisionTreeClassifier

classifier = OneVsRestClassifier(DecisionTreeClassifier())
```

```
y_score = classifier.fit(X_train, y_train).predict(X_test)
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
roc_auc[2]

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure()
lw = 2
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (area = {1:0.2f})'
                   ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Some extension of Receiver operating characteristic to multi-class')
plt.legend(loc="lower right")
plt.show()
```

7. WineRatingPredictor.py

```
from configuration import configure
from DatasetTraining import training
from matplotlib import pyplot as plt
```

```
# importing the dataset and pre-procesing it

datasetName = 'winequality-white.csv'


# configure constructor takes
#
# datasetName as parameter

cnfg = configure(datasetName)

# binaryClassConversion takes featureName to convert the feature value in two cla
#
# returns the whole dataset with updated binary values of 0 and 1

dataset = cnfg.binaryClassConversion('quality', 3)

#dataset = cnfg.getdataset()

X = dataset[['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar'
             'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol']]
y = dataset['taste']

classifierList = ["knn", "decisiontree", "logistic"]

training = training(dataset, X, y)

# clf = training.normal_training(classifierList[2], True, 0.3)
# print clf['x_f']
# eval = evalMetric(clf['y_test'], clf['y_pred'])
# print "accuracy_Score for ", classifierList[2], ": ", eval.accuracy_skore()
# print clf['y_pred']

# for cl in classifierList:
#     clf = training.normal_training(cl, True, 0.3)
#     eval = evalMetric(clf['y_test'], clf['y_pred'])
#     print "accuracy_Score for ", cl, ": ", eval.accuracy_skore()
#     print "confusion_metric for ", cl, ": \n", eval.confusion_metric()
#     print "precision_recall_fscore_supports for ", cl, ": ", eval.precision_reca
#     # print "area_under_the_curve ", cl, ": ", eval.area_under_the_curve()

# ---------------------K-Fold training -------------------
scores = []
for cl in classifierList:
    score, imp, scoreList = training.k_fold_cross_validation(10, cl, 6)
    scores.append(scoreList)

# ---------------------------------------rating distribution histograph ----------
plt.hist(dataset['quality'], range=(1, 10))
plt.xlabel('Ratings of wines')
plt.ylabel('Amount')
plt.title('Distribution of wine ratings')
plt.show()

# ------------------------performance evaluator-------------------------------------
# clff = training.normal_training("randomforest", True, 0.3)
# eval = evalMetric(clff['y_test'], clff['y_pred'])
```

```
# eval.precision_recall_fscore_supports()
# print clff['trainedmodel'].feature_importances_
# clf = training.k_fold_cross_validation(10, 'randomforest')
# score , fimp = clf
# print sum(fimp)
#
# fig, ax = plt.subplots()
# ax.scatter(clff['y_test'], clff['y_pred'], edgecolors=(0, 0, 0))
# ax.plot([clff['y_test'].min(), clff['y_test'].max()], [clff['y_test'].min(), cl
# plt.plot(clff['y_test'], clff['y_pred'], color='blue', linewidth=1)
# # plt.plot(clff['X_test'], clff['y_test'], color='yellow', linewidth=0.25)
# plt.show()

# ------------------------------Graph generation-----------------------------------

# ------------------------------feature Importance---------------------------------
"""
Bar chart demo with pairs of bars grouped for easy comparison.
"""
import numpy as np
import matplotlib.pyplot as plt



score, imp, s1 = training.k_fold_cross_validation(10, 'randomforest')
n_groups = len(imp)
means_men = (imp)
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.25
opacity = 0.4
error_config = {'ecolor': '0.5'}
rects1 = plt.bar(index, means_men, bar_width,
                 alpha=opacity,
                 color='b',
                 error_kw=error_config,
                 label='Features')
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Feature Importance')
plt.xticks(index + bar_width/2)
plt.legend()
ax.set_xticklabels(['fixed acidity', 'volatile acidity', 'citric acid', 'residual
            'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol'],rota
plt.tight_layout()
plt.show()

# ------------------------------------------------------KNN-----------------------
a = []
b = []
import numpy as np
for i in range(1,6):
    score, imp, s1 = training.k_fold_cross_validation(10, 'knn', i*3)
    a.append(i*3)
    b.append(score)
plt.plot(a, b)
plt.axis([2, 16, 0.35, 0.42])
plt.xlabel('neighbours')
```

```
plt.ylabel('score')
plt.title('K nearest neighbour')
plt.show()
#-----------------------------------------decision tree-----------------------------------

a = []
b = []
for i in range(1,6):
    score, imp, s1 = training.k_fold_cross_validation(10, 'decisiontree', i*3)
    a.append(i*3)
    b.append(score)
plt.plot(a, b)
# plt.axis([2, 16, 0.68, 0.75])
plt.xlabel('depth')
plt.ylabel('score')
plt.title('decision Tree')
plt.show()

# ------------------------------k-fold training map-----------------------------------

plt.plot([i for i in range(1, 11)], scores[0][0], label='knn')
plt.plot([i for i in range(1, 11)], scores[1][0], label='decision tree')
plt.plot([i for i in range(1, 11)], scores[2][0], label='logistic')
plt.xlabel('splits')
plt.ylabel('score')
plt.title('K-fold training map')
plt.legend()
plt.show()
#-------------------------accuracy score vs f1_score-----------------------------

plt.plot([i for i in range(1, 11)], scores[0][0], label='knn_f1_Score')
plt.plot([i for i in range(1, 11)], scores[0][1], label='knn_accuracy_score')
plt.plot([i for i in range(1, 11)], scores[1][0], label='decision_tree_f1_score')
plt.plot([i for i in range(1, 11)], scores[1][1], label='decision_accuracy_score')
plt.plot([i for i in range(1, 11)], scores[2][0], label='logistic_f1_score')
plt.plot([i for i in range(1, 11)], scores[2][1], label='logistic_accuracy_score')
plt.xlabel('splits')
plt.ylabel('score')
plt.title('accuracy_score vs f1_score for 10 fold splits')
plt.legend(loc=2, prop={'size': 10})
#plt.figure(figsize=(15,15))
plt.show()
```

References

[1] 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167923609001377

[2] W. contributors, "Acids in wine — wikipedia, the free encyclopedia," 2017. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Acids_in_wine&oldid=815328944

[3] M. Carel, "Sulfur dioxide (so2) in wine," October 2011. [Online]. Available: https://winobrothers.com/2011/10/11/sulfur-dioxide-so2-in-wine/

[4] "Adding feature selection to knn #6920," 2016, scikit library issue. [Online]. Available: https://github.com/scikit-learn/scikit-learn/issues/6920

[5] "Add forward selection to scikit-learn #6545," 2016, scikit library issue. [Online]. Available: https://github.com/scikit-learn/scikit-learn/issues/6545

[6] M. Kuhn, "Simulated annealing feature selection," 2015. [Online]. Available: https://www.r-bloggers.com/simulated-annealing-feature-selection/