**Pooja Narayan**

**1001508253**

**Programming Assignment**

**Step 1.**

**Reading the data and scaling it**

data = pd.read\_csv('wine.csv')

original\_headers = list(data.columns.values)

print(original\_headers)

y = data.quality

X = data.drop('quality', axis=1)

# print(X.describe)

X = preprocessing.scale(X)

print(X)

**Step2. Set training data to 75% and test data to 25%**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, stratify=y)

**Step3. Using Random Forest Classifier and printing accuracy**

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n\_estimators=40, warm\_start=True)

kc = rf.fit(X\_train, y\_train)

print(kc)

predictions = rf.predict(X\_test)

print(predictions)

s = y\_test.values

print(y\_test.values)

length = len(predictions)

print("Length of predictions".format(length))

print(len(predictions))

count = 0

for i in range(length):

if predictions[i] == s[i]:

count = count + 1

# print("Length of count")

# print(count)

accuracy1 = count / length

# Obtaining the confidence score

print("Initial accuracy in model1 is:\n")

print(accuracy1)

**Step4. Print the confusion matrix**

# Confusion Matrix

from sklearn.metrics import confusion\_matrix

print(pd.crosstab(y\_test, predictions, rownames=['True'], colnames=['Predicted'], margins=True))

from sklearn.metrics import classification\_report

classifyrep = classification\_report(y\_test, predictions)

print(classification\_report(y\_test, predictions))

**Step5. 10 Fold Stratified Cross Validation and print the improved accuracy**

Stratified Cross-validation violates the principal that the test labels should never have been looked at before the statistics are calculated, but this is generally thought to be innocuous as the only effect is to balance the folds, but it does lead to loss of diversity (an unwanted loss of variance). It moves even further from the Boostrap idea of constructing a sample similar to what you'd draw naturally from the whole population. Arguably the main reason stratification is important is to address defects in the classification algorithms, as they are too easily biased by over- or under-representation of classes. An algorithm that uses balancing techniques (either by selection or weighting) or optimizes a chance-correct measure (Kappa or preferably Informedness) is less impacted by this, although even such algorithms can't learn or test a class that isn't there.

Forcing each fold to have at least m instances of each class, for some small m, is an alternative to stratification that works for both Bootstrapping and CV. It does have a smoothing bias, making folds tend to be more balanced than they would otherwise be expected to be.

# k-fold Stratified Cross Validation

sratifiedkf = StratifiedKFold(n\_splits=10, shuffle=True)

average = 0.0

value = 0

for train\_index, test\_index in sratifiedkf.split(X, y):

X\_train, X\_test = X[train\_index], X[test\_index]

y\_train, y\_test = y[train\_index], y[test\_index]

rf = RandomForestClassifier(n\_estimators=80, warm\_start=True)

kc = rf.fit(X\_train, y\_train)

print(kc)

predictions = rf.predict(X\_test)

print(predictions)

s = y\_test.values

print(y\_test.values)

length = len(predictions)

print("Length of predictions".format(length))

print(len(predictions))

count = 0

for i in range(length):

if predictions[i] == s[i]:

count = count + 1

print("Length of count")

print(count)

accuracy = count / length

value = value + (accuracy - accuracy1)

print("Test data accuracy in fold :\n {:.2f}".format(accuracy))

average = average + accuracy

average = average / 10

print("Average cross-validation is: {:.2f}".format(average))

print("Improvised accuracy in model1 is: \n {:.2f}".format(abs(value / 10)))

**Step6.** **LinearRegression is what i used to improve the accuracy**

**It considers the top 8 features and eliminates the 3 least valuable features**

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable

*class*sklearn.linear\_model.LogisticRegression(*penalty=’l2’*, *dual=False*, *tol=0.0001*, *C=1.0*, *fit\_intercept=True*, *intercept\_scaling=1*, *class\_weight=None*, *random\_state=None*, *solver=’liblinear’*, *max\_iter=100*, *multi\_class=’ovr’*, *verbose=0*, *warm\_start=False*, *n\_jobs=1*)

*In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the ‘multi\_class’ option is set to ‘ovr’, and uses the cross- entropy loss if the ‘multi\_class’ option is set to ‘multinomial’. (Currently the ‘multinomial’ option is supported only by the ‘lbfgs’, ‘sag’ and ‘newton-cg’ solvers.)*

This class implements L1 and L2 regularized logistic regression using the liblinear library. It can handle both dense and sparse input.

#linear regression -feature selection

#important features

from sklearn.feature\_selection import VarianceThreshold

sel = VarianceThreshold(threshold=(.8 \* (1 - .8)))

X=sel.fit\_transform(X)

reg = linear\_model.LinearRegression(normalize=True)

print(reg.fit(X, y))

a = reg.predict(X)

print(y)

# mean square error

print(np.mean((a - y) \*\* 2))

from sklearn.feature\_selection import RFE

rfe = RFE(reg, 8)

fit = rfe.fit(X, y)

print(fit)

print(("Num Features: %d") % fit.n\_features\_)

print(("Selected Features: %s") % fit.support\_)

print(("Feature Ranking: %s") % fit.ranking\_)

X = data.drop(['citric acid', 'total sulfur dioxide', 'sulphates', 'quality'], axis=1)

X = preprocessing.scale(X)

**Step7.Again perform k-fold Stratified and provide the improved accuracy**

**it is increasing diffrently at different time  
sometimes its 1% sometimes its 5%**

**References**

https://machinelearningmastery.com/feature-selection-machine-learning-python/

http://queirozf.com/entries/pandas-dataframe-by-example

https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/tutorial-random-forest-parameter-tuning-r/tutorial/

https://www.kaggle.com/hadend/tuning-random-forest-parameters

http://scikit-learn.org/stable/modules/preprocessing.html

http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html#sklearn.pipeline.Pipeline

http://blog.datadive.net/random-forest-interpretation-with-scikit-learn/

http://scikit-learn.org/0.15/modules/generated/sklearn.linear\_model.LogisticRegression.html