CSC 529 ADVANCED DATA MINING

CASE STUDY – 1 – ENSEMBLE CLASSIFICATION

BY

Sasidhar Mukthinuthalapati

1. INTRODUCTION:

Data mining and Machine Learning techniques are paving the way for doctors to help them understand and synthesize the information present in the huge database of patients. These algorithms can be used on a sample of this huge population and help the doctors in better understanding of a disease in a human or the symptoms and their underlying relation.

Diabetes Mellitus also commonly known as Diabetes has affected around 400 million people in the world, which impacts the body's ability to normalize blood sugar levels. It is caused due to either the pancreas not producing enough amount of insulin or the body cells not reacting to the insulin produced. Untreated diabetes can lead to many complications such as stroke, chronic kidney failure, foot ulcers, eye damage or even death.

For the better analysis of this kind of disease we have taken a sample of the Pima Indian Heritage population who reside near Phoenix, Arizona, USA. Since females are more susceptible to this disease we have just taken the Female records and would be analyzing them for better understanding of the factors which give rise to this disease in females and try to identify the root cause of the problem. World Health Organization, based on the research it had done, it came up with a basic metric in which we measure the 2 Hour post-load glucose and check whether it is at least 200 mg/dl. If so then the patient is more vulnerable to diabetes.

While studying these kinds of datasets the researcher must be very careful because any false interpretations of data can lead to severe consequences. Thus, with the help of machine learning and data mining algorithms we can try to forecast the whether the female would be likely to get diabetes or not. National Institute of Diabetes and Digestive and Kidney Disease (NIDDKD) had provided significant amount of data and the various kinds of algorithms and the findings from the output generated by those algorithms for use in future research. NIDDKD had also provided the list of algorithms they had used like Linear Regression, LDA, SVM, KNN, CART and so on and hence we would not be using those but instead we would be using Ensemble Methods like Boosting, Bagging and Stacking which would or should come up with a model which improves on the initial accuracy and Robustness.

Various Researchers from various research centers have introduced new concepts that are worth experimenting such as Neural Networks, ADAP which can be used to forecast the Diabetes in Pima Indians (Females Only).

2. DATA DESCRIPTION:

The dataset which is being used is a sample of the population of women who are at least 21 years old and belong to the Pima Indian Heritage and who are lining in and around Phoenix, Arizona, USA. They were initially tested with the criteria set by the World Health Organization. The data was collected by the National Institute of Diabetes and Digestive and Kidney Diseases. Each record in the dataset belongs to each patient and the task is to predict whether the patient is vulnerable to diabetes in future.

Data Source: UCI Machine Learning Repository
Data URL: https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes
Number of Observations: 768
Number of Features: 8
Attribute Information:

NumberOfTimePregnant: The number of Times a woman was Pregnant

```
count    768.000000
mean       3.845052
std        3.369578
min        0.000000
25%        1.000000
50%        3.000000
75%        6.000000
max       17.000000
Name: NumberOfTimesPregnant, dtype: float64
```

PlasmaGlucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test

```
count    768.000000
mean     120.894531
std       31.972618
min        0.000000
25%       99.000000
50%      117.000000
75%      140.250000
max      199.000000
Name: PlasmaGlucose, dtype: float64
```

DiastolicBloodPressure:

```
count    768.000000
mean      69.105469
std       19.355807
min        0.000000
25%       62.000000
50%       72.000000
75%       80.000000
max      122.000000
Name: DiastolicBloodPressure, dtype: float64
```

DiabetesPedigreeFunction:

```
count    768.000000
mean       0.471876
std        0.331329
min        0.078000
25%        0.243750
50%        0.372500
75%        0.626250
max        2.420000
Name: DiabetesPedigreeFunction, dtype: float64
```

2HrSerumInsulin:

```
count    768.000000
mean      79.799479
std      115.244002
min        0.000000
25%        0.000000
50%       30.500000
75%      127.250000
max      846.000000
Name: 2HrSerumInsulin, dtype: float64
```

TricepsSkinFoldThickness

```
count    768.000000
mean      20.536458
std       15.952218
min        0.000000
25%        0.000000
50%       23.000000
75%       32.000000
max       99.000000
Name: TricepsSkinFoldThickness, dtype: float64
```

BMI: Body Mass Index

```
count    768.000000
mean      31.992578
std        7.884160
min        0.000000
25%       27.300000
50%       32.000000
75%       36.600000
max       67.100000
Name: BMI, dtype: float64
```

Age:

```
count    768.000000
mean      33.240885
std       11.760232
min       21.000000
25%       24.000000
50%       29.000000
75%       41.000000
max       81.000000
Name: Age, dtype: float64
```

The Class/Target variable is labeled as Class in this Dataset and it is a Binary variable and 0 means the woman is not vulnerable to Diabetes in Future and 1 means the woman is vulnerable to diabetes in the future and necessary steps and medications must be taken to keep it in control. The class distribution is given below:

```
0    500
1    268
Name: Class, dtype: int64
```

Based on the initial research I feel that Diabetes Pedigree Function might be a driving factor in determining whether a person would have or is vulnerable to diabetes in the future because it gives us insight into the diabetes mellitus history in relatives and the genetic relationship of those relatives to the patient.

So, based on the above data we can also say that all the attributes are integers and that the population is mostly under the age of 50. There are few missing values which have been written down as 0 which might be erroneous.

3. DATA CLEANING:

The dataset which has been obtained from the UCI Machine Learning Repository and for our analysis we will split the data into 80 20 ratios with 80% of the data as training and the rest as Testing.

Further we will also perform 10-fold cross validation on the training and then apply the model on test set and on satisfactory results being obtained we will run the model on the whole set. In 10-fold CV initially 9 folds will be taken as training set and the remaining fold will be used as test set. This process runs till the all the folds have been used as test set at least once.

After going through the dataset, I have found out that there are quite few cases in which there are missing values and these values are written down as 0. There are few possible ways of handling the missing data which are as follows:

- Replace missing values with the sensible values like mean or median given the distribution of the data.
- Replace the missing data with prediction – Multiple Imputation.
- Remove the records with missing values but this is also not feasible because we only have 768 records and if we remove we might be left with small dataset.

Since with the number of records is very less it's advisable to replace the missing values with the mean and by doing so we are not reducing the number of records and not decreasing the amount of learning that the algorithm can learn.

Since the number of attributes in the dataset are very less, all the 8 features will be considered for the learning process. Hence there is no need to perform any kind of Dimensionality reduction. In the preliminary analysis of each attribute we can see that there aren't many influential outliers present in the dataset and the once which are present will not affect the model by much.

4. DATA ANLAYSIS:

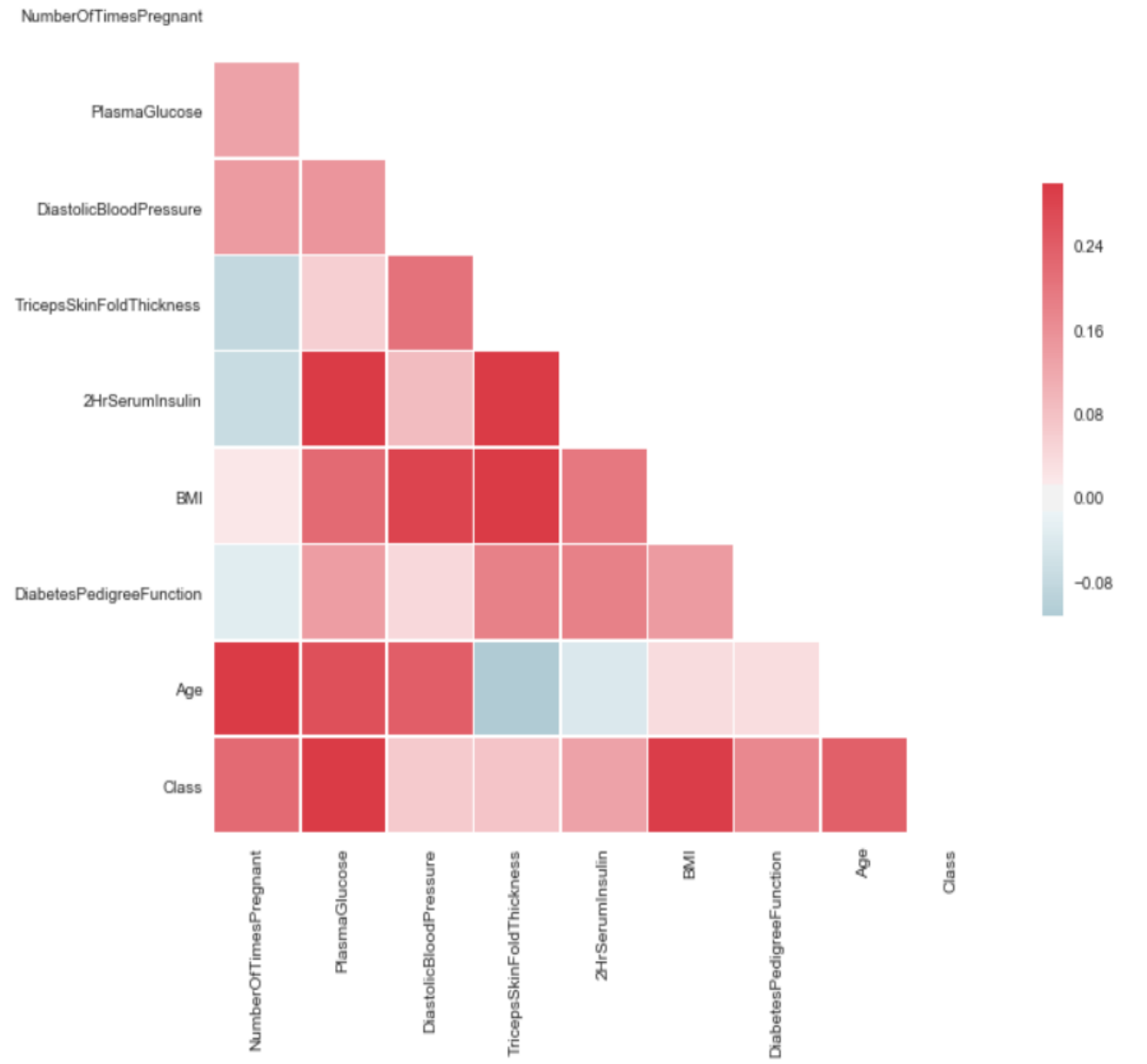
To see whether the dataset has multicollinearity present we must analyze the correlation values and the pair plot and heatmap these are given below:

Based on the Correlation values we can see that there doesn't exist any kind of multi collinearity in the data set and the other interesting fact is that the no attribute is very strongly correlated to the Class variable and hence strengthens our decision to not remove any attribute from our analysis.

We can also see that except Age and Number of Time Pregnant, Insulin and Glucose, BMI and Skin thickness are the pairs with correlation values higher than 0.5. Apart from that all others pairs do not have much strong correlation.

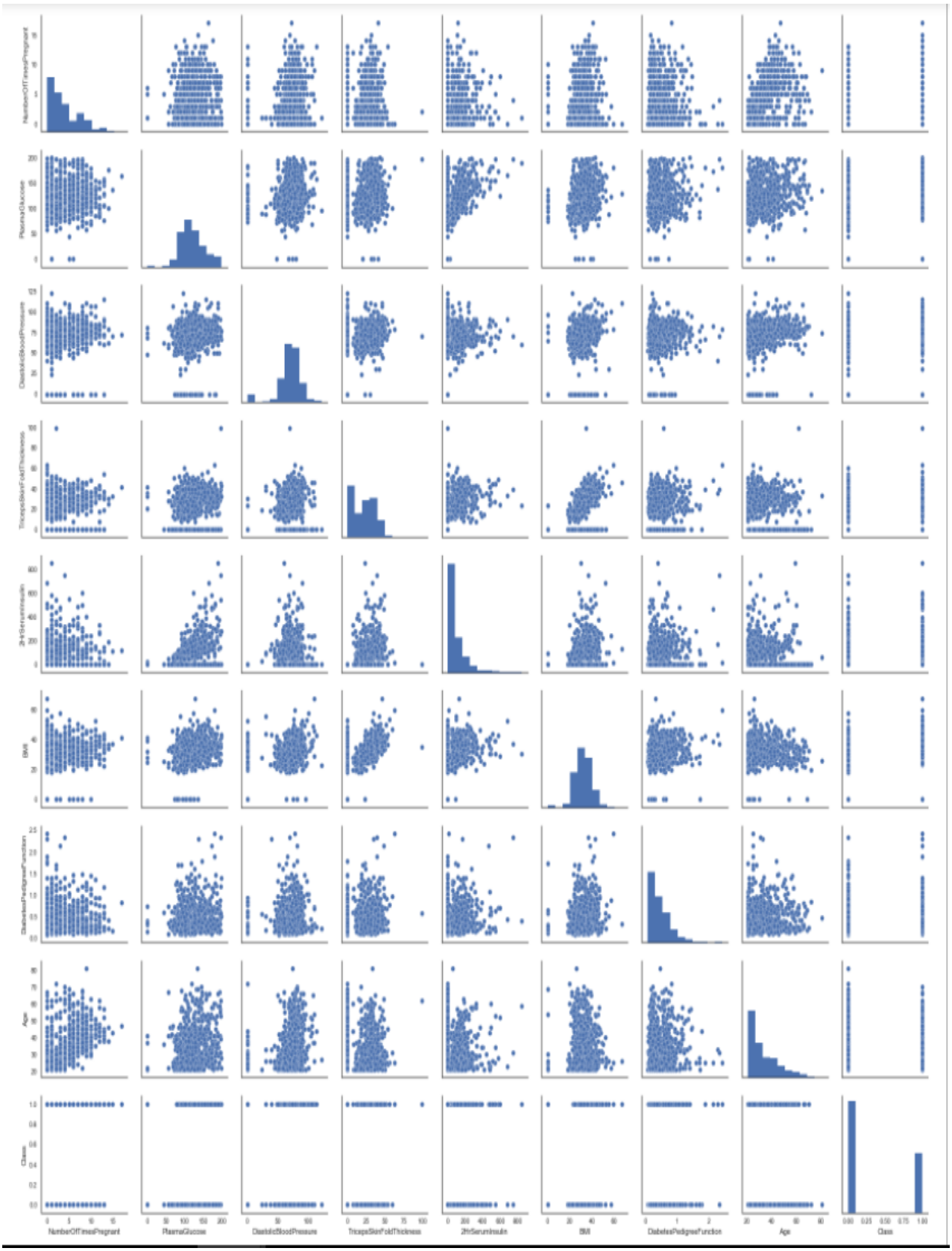
	NumberOfTimesPregnant	PlasmaGlucose	DiastolicBloodPressure	TricepsSkinFoldThickness	2HrSerumInsulin	BMI	DiabetesPedigreeFunction	Age	Class
NumberOfTimesPregnant	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
PlasmaGlucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
DiastolicBloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
TricepsSkinFoldThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
2HrSerumInsulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Class	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

The heatmap given below also provides the same information but in a more visually appealing manner.



The Pair plot given below gives us the scatterplot matrix with histograms as the diagonal elements which show the distribution of that variable. Few of the histograms aren't normally distributed but we wouldn't be applying any kind of transformation to those variables mainly because this is a binary classification problem.

The other things which can be noticed from the scatterplot (Pair Plot) matrix is that there isn't any high correlation between independent variables especially the correlation between Age, BMI And Diabetes Pedigree function.


NumberOfTimesPregnant
PlasmaGlucose
DiastolicBloodPressure
TricepsSkinFoldThickness
2HrSerumInsulin
BMI
DiabetesPedigreeFunction
Age
Class

5. EXPERIMENTAL RESULTS:

Recently there has been lot of discussion about “Ensemble Learning” in which we combine models which do better than random and come up with an ensemble of such models which does way better than any single model alone. One more important thing is that the small size of the dataset will limit the performance of some algorithms

5.1. Boosting Algorithms:

Boosting is an iterative technique which adjust the weight of the observations based on the last classification. The incorrectly classified records will be given more weight in the next pass. Initially all the records will be given the weight of 1/N. In general, the boosting algorithm can be used to build strong predictive models. However, on giving extra weights the model also becomes susceptible to overfitting.

There are multiple boosting algorithms like Gradient Boosting, AdaBoost etc. the algorithm explored in this model is AdaBoost and the output is given below:
NOTE: The number of estimators has been set to 100

```
#ensemble method: Boosting (AdaBoost)
from sklearn.ensemble import AdaBoostClassifier
clfAda = AdaBoostClassifier(n_estimators=100)
scoresAda = cross_val_score(clfAda, x_train, y_train)
scoresAda.mean()
```

```
0.75080503746213934
```

```
#predicting on test set and then comparing it with original label
clfAda.fit(x_train,y_train)
yhatada = clfAda.predict(x_test)
accuracy_score(y_test, yhatada)
```

```
0.69480519480519476
```

```
#on the complete data set accuracy
scoresAda1 = cross_val_score(clfAda,X,Y)
scoresAda1.mean()
```

```
0.74480942912184334
```

As seen from the above results we can see that the overall mean accuracy of the models on the whole dataset is around 74.48%.

5.2. **Bagging Algorithm:** In bagging we use the same learner on various samples of the dataset and then compute the mean of all the predictions. In generalized bagging technique, we can use varied learners on different population. In this I've used the base estimator as Knn with 10-fold cross validation and all the features.

The output from the Bagging Algorithm is as follows:

```
# Ensemble method: Bagging(Bootstrap Aggregating) bootstrap set to default as T
from sklearn.ensemble import BaggingClassifier
bagging1 = BaggingClassifier(knn_cv, max_samples=0.5, max_features=1)
```

```
bagging1.fit(x_train,y_train)
```

```
BaggingClassifier(base_estimator=GridSearchCV(cv=10, error_score='raise',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric
='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid=...       pre_dispatch='2*n_jobs', refit=True, return_train
_score=True,
       scoring=None, verbose=0),
         bootstrap=True, bootstrap_features=False, max_features=1,
         max_samples=0.5, n_estimators=10, n_jobs=1, oob_score=False,
         random_state=None, verbose=0, warm_start=False)
```

```
yhatensemble = bagging1.predict(x_test)
```

```
#Accuracy on the test set
accuracy_score(y_test,yhatensemble)
```

```
0.67532467532467533
```

```
yhaten = bagging1.predict(X)
```

```
#Accuracy on the whole set
accuracy_score(Y,yhaten)
```

```
0.7734375
```

As we can see this Ensemble has worked better than the Boosting Algorithm.

5.3. Stacking Algorithm: Stacking is very interesting way of combining models. Here a learner is used to combine outputs from various other learners. Main disadvantage by using this technique is either the bias or variance can decrease based on the combining learner that is used. The output from running the stacking algorithm is given below. The KNN, Naïve Bayes, Random Forest were the three algorithms which were used to

```
#ensemble method: Stacking
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from mlxtend.classifier import StackingClassifier
```

```
clf1 = KNeighborsClassifier(n_neighbors=30) # As we got 30 to be the best number of neighbors in the above statements
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()
lr = LogisticRegression()
sclf = StackingClassifier(classifiers=[clf1, clf2, clf3],meta_classifier=lr)

print('10-fold cross validation:\n')

for clf, label in zip([clf1, clf2, clf3, sclf],['KNN','Random Forest','Naive Bayes','StackingClassifier']):

    scores = model_selection.cross_val_score(clf, x_train, y_train, cv=10, scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

```
10-fold cross validation:

Accuracy: 0.75 (+/- 0.04) [KNN]
Accuracy: 0.74 (+/- 0.05) [Random Forest]
Accuracy: 0.77 (+/- 0.04) [Naive Bayes]
Accuracy: 0.74 (+/- 0.05) [StackingClassifier]
```

As we can see the average accuracy is around 74~75 %.

6. EXPERIMENTAL ANALYSIS:

The basic models like Decision Tree, KNN and Naïve Bayes all had performed poorly as the accuracy for them was only around the range of 65~68% accuracy but whereas the Ensemble methods have done considerably well. This can be because the ensemble methods combine and make use of various models to come up with an ensemble of models which perform better than the individual one. The 10-fold cross validation was used and it has helped the models to learn better because rather than learning once and testing it this method will repeat the process 10 times which in return gives the model to learn more and capture more variance when compared to usual single pass.

The Summarized table of Accuracy is as follows:

ENSEMBLE ALGORITHMS	ACCURACY
Boosting	74.48%
Bagging	77.34%
Stacking	74.5%

As per the above table we can see that the accuracy for the Bagging model is 77.34% which is better than the other Ensemble methods. As we can see in Stacking the Naïve Bayes had performed well with 77% accuracy and a standard deviation of around 0.04.

For some fitting methods, there are tuning parameters built into the model that compensate for the number of predictors added to the model during fitting. These predictors compensations prevent the model from suffering over-fitting and optimize the bias-variance trade-off. These parameters can be altered using cross validation to let the model to better fit to the dataset. Thus, there are a lot more work needs to be done than just running through basic statistical learning method to obtain high performance model.

Since most of the models had nearly the same accuracy it's better to select the model based on the Ease of interpretation, scalability, time etc.

7. CONCLUSION:

In this study, we have exploited the models which performed good and created an ensemble from those good models which performed better than initial one. This experimentation has given us a better insight into the machine learning applications in medical diagnosis. The only drawback being the dataset was too small to experiment with it. We also didn't have much choice with the number of features/attributes which may be because of privacy concerns. However, this study was a good start for building models that aids diagnose patients, and bridge the gap between the doctors and large datasets in predicting accurate data with high performance.

This method of analyzing these kinds of datasets can be applied on other datasets to solve or understand other medical datasets and provide doctors and researchers with new things and theory to ponder about.

PYTHON CODE:

```
# coding: utf-8

# In[9]:


import scipy
import csv
import pandas as p
from sklearn.naive_bayes import GaussianNB
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree
import matplotlib.pyplot as plt
get_ipython().magic('matplotlib inline')
import seaborn as sns
from sklearn.model_selection import cross_val_score
from mlxtend.plotting import plot_decision_regions
import matplotlib.gridspec as gridspec
import itertools


# In[2]:


sns.set(style="white")


# In[3]:


data = p.read_csv('C:\\Users\\msasi\\Desktop\\CS1\\pima-indians-diabetes.csv', delimiter=',')


# In[4]:


data.head()


# In[5]:


data['Class'].value_counts()   #Gives us the Class Distibution of the DataSet
```

In[26]:

```
#Basic Statistics for Various Columns
notp1 = data[data['Class'] == 1]['NumberOfTimesPregnant']
notp1.describe()
```

In[27]:

```
bi_nopt1 = data[data['Class'] == 1]
bi_nopt1['NumberOfTimesPregnant'].hist()
```

In[24]:

```
notp0 = data[data['Class'] == 0]['NumberOfTimesPregnant']
notp0.describe()
```

In[28]:

```
bi_nopt0 = data[data['Class'] == 0]
bi_nopt0['NumberOfTimesPregnant'].hist()
```

In[53]:

```
#Scatterplot
plt.scatter(data['Class'],data['NumberOfTimesPregnant'])
```

In[21]:

```
#Description of the attributes:
data['NumberOfTimesPregnant'].describe()
```

In[22]:

```
data['PlasmaGlucose'].describe()

# In[15]:

data['DiastolicBloodPressure'].describe()

# In[16]:

data['DiabetesPedigreeFunction'].describe()

# In[17]:

data['2HrSerumInsulin'].describe()

# In[18]:

data['TricepsSkinFoldThickness'].describe()

# In[19]:

data['BMI'].describe()

# In[20]:

data['Age'].describe()

# In[50]:

#corr = data.corr()
data.corr()

# In[47]:
```

```
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})


# In[63]:


#p.scatter_matrix(data, alpha=0.2, diagonal='kde')


# In[62]:


sns.pairplot(data)


# In[6]:


X = data.values[:, 0:7]
Y = data.values[:,8]
x_train, x_test, y_train, y_test = train_test_split( X, Y, test_size = 0.2, random_state = 100)


# In[23]:


# Decision Tree Algorithm
clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100)
clf_gini.fit(x_train, y_train)


# In[8]:


from sklearn import metrics
print(clf_gini.feature_importances_)
```

```
# In[11]:


scipy.stats.chisquare(X)


# In[13]:


trainingtest = clf_gini.predict(x_test)


# In[14]:


accuracy_score(y_test, trainingtest)


# In[15]:


clf_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100)
clf_entropy.fit(x_train, y_train)


# In[16]:


trainingtestentropy = clf_entropy.predict(x_test)


# In[17]:


accuracy_score(y_test, trainingtestentropy)


# In[18]:


#Decision tree with k fold cross validation
cv_results = cross_val_score(clf_gini, x_train, y_train, cv=10)


# In[19]:


print(cv_results)
```

In[20]:

```
np.mean(cv_results)
```

In[33]:

```
#Naive Bayes Algorithm
gnb = GaussianNB()
x_testing = gnb.fit(x_train, y_train).predict(x_test)
```

In[34]:

```
print("Number of mislabeled points out of a total %d points : %d" % (x_test.shape[0],(y_test !=
x_testing).sum()))
```

In[35]:

```
accuracy_score(y_test,x_testing)
```

In[27]:

```
#knearest neighbors with 10 fold cross validation
from sklearn.model_selection import GridSearchCV
param_grid = {'n_neighbors':np.arange(1,50)}
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn,param_grid,cv=10)
knn_cv.fit(x_train,y_train)
```

In[25]:

```
knn_cv.best_params_
```

In[26]:

```
knn_cv.best_score_

# In[27]:

yhat = knn_cv.predict(x_test)

# In[28]:

yhat.std()

# In[29]:

accuracy_score(y_test,yhat)

# In[28]:

# Ensemble method: Bagging(Bootstrap Aggregating) bootstrap set to default as True
from sklearn.ensemble import BaggingClassifier
bagging1 = BaggingClassifier(knn_cv, max_samples=0.5, max_features=1)

# In[29]:

bagging1.fit(x_train,y_train)

# In[55]:

yhatensemble = bagging1.predict(x_test)

# In[56]:

#Accuracy on the test set
accuracy_score(y_test,yhatensemble)
```

In[30]:

```
yhaten = bagging1.predict(X)
```

In[31]:

```
#Accuracy on the whole set
accuracy_score(Y,yhaten)
```

In[7]:

```
#ensemble method: Stacking
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from mlxtend.classifier import StackingClassifier
```

In[10]:

```
clf1 = KNeighborsClassifier(n_neighbors=30) # As we got 30 to be the best number of neighbors
in the above statements
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()
lr = LogisticRegression()
sclf = StackingClassifier(classifiers=[clf1, clf2, clf3],meta_classifier=lr)

print('10-fold cross validation:\n')

for clf, label in zip([clf1, clf2, clf3, sclf],['KNN','Random Forest','Naive Bayes','StackingClassifier']):

    scores = model_selection.cross_val_score(clf, x_train, y_train, cv=10, scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

In[24]:

```
#ensemble method: Boosting (AdaBoost)
```

```
from sklearn.ensemble import AdaBoostClassifier
clfAda = AdaBoostClassifier(n_estimators=100)
scoresAda = cross_val_score(clfAda, x_train, y_train)
scoresAda.mean()


# In[54]:


#predicting on test set and then comparing it with original label
clfAda.fit(x_train,y_train)
yhatada = clfAda.predict(x_test)
accuracy_score(y_test, yhatada)


# In[51]:


#on the complete data set accuracy
scoresAda1 = cross_val_score(clfAda,X,Y)
scoresAda1.mean()
```