Bank Marketing Data Set

A portugese bank wants to sell a product, called Term Deposit, to its customers. The bank has previously established contacts with 4, 521 of these prospective customers through telephone calls, to sell this product. It turned out that only about 12% of these customers actually purchased or subscribed.

The bank understands that not everyone of the customers will subscribe the Term Deposit, and does not want to invest campaign resources on those who are not likely to subscribe.

The goal of this project is to develop a model that can predict those customers who are likely to subscribe the Term Deposit.

The best model will be evaluated using Acuracy Score, Explained Variance, F Measure and Mean Error.

—

We are importing the following packages: Pandas and Numpy to manage the data, Seaborn and Matplot for visualizations, and the Sklearn for machine leaning processes and models

In [1]:
```
%matplotlib inline
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn import model_selection
```

Importing data, called "bank.csv", indicating that the header is the first row, separator is ";", and dropping all "Missing values".
We also need to designate the target variable ("y")

In [2]:
```
data = pd.read_table('bank.csv', header=0, sep=';')
data = data.dropna()


# designate target variable name (y)

targetName = 'y'
targetSeries = data[targetName]
#remove target from current location and insert in collum 0
del data[targetName]
data.insert(0, targetName, targetSeries)


#print dataframe and see target is in position 0
data.head()
```

Out[2]:

	y	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome
0	no	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	79	1	-1	0	unknown
1	no	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	220	1	339	4	failure
2	no	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	185	1	330	1	failure
3	no	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	199	4	-1	0	unknown
4	no	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	226	1	-1	0	unknown

Preprocessing Data And Data Vizualization

The Data processing will take three (3) steps:

Encoding to transform categorical variables into numerical variables
Scaling to normalize the data and ensuring that variables with high values do not exert undue influence
Upscaling to turn the imbalanced data into a balanced data.

Encoding Data with LabelEncoder, and Dummy variables

In [3]:
```
# This code turns a text target into numeric to some scikit learn alogrythms can process it
from sklearn import preprocessing
le_dep = preprocessing.LabelEncoder()
#to convert into numbers
data['y'] = le_dep.fit_transform(data['y'])

# perform data transformation. Creates dummies of any categorical feature
for col in data.columns[1:]:
        attName = col
        dType = data[col].dtype
        missing = pd.isnull(data[col]).any()
        uniqueCount = len(data[attName].value_counts(normalize=False))
        # discretize (create dummies)
        if dType == object:
                data = pd.concat([data, pd.get_dummies(data[col], prefix=col)], axis=1)
                del data[attName]

data.shape
data.describe()
```

Out[3]:

	y	age	balance	day	duration	campaign	pdays	previous	job_admin.	job_blue-collar	...	month_jun	month_mar	month_may	month_nov	month_oct	month_sep	poutcome_failure
count	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	...	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000	4521.000000
mean	0.115240	41.170095	1422.657819	15.915284	263.961292	2.793630	39.766645	0.542579	0.105729	0.209246	...	0.117452	0.010838	0.309224	0.086043	0.017695	0.011502	0.108383
std	0.319347	10.576211	3009.638142	8.247667	259.856633	3.109807	100.121124	1.693562	0.307524	0.406815	...	0.321994	0.103553	0.462225	0.280458	0.131856	0.106640	0.310898
min	0.000000	19.000000	-3313.000000	1.000000	4.000000	1.000000	-1.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	33.000000	69.000000	9.000000	104.000000	1.000000	-1.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	39.000000	444.000000	16.000000	185.000000	2.000000	-1.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	49.000000	1480.000000	21.000000	329.000000	3.000000	-1.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	87.000000	71188.000000	31.000000	3025.000000	50.000000	871.000000	25.000000	1.000000	1.000000	...	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 52 columns

Scaling Data with RobustScaler

In [4]:
```
from sklearn.preprocessing import RobustScaler

data_robust = pd.DataFrame(RobustScaler().fit_transform(data), columns=data.columns)
data_robust.describe()

# Redesinating the targe variable after scaling the original data
targetName = 'y'
targetSeries = data_robust[targetName]

#remove target from current location and insert in collum 0
del data_robust[targetName]
data_robust.insert(0, targetName, targetSeries)

#reprint dataframe and see target is in position 0
data_robust.head()
```

Out[4]:

	y	age	balance	day	duration	campaign	pdays	previous	job_admin.	job_blue-collar	...	month_jun	month_mar	month_may	month_nov	month_oct	month_sep	poutcome_failure	poutcome_other	poutcome_suc
0	0.0	-0.5625	0.951807	0.250000	-0.471111	-0.5	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1	0.0	-0.3750	3.079376	-0.416667	0.155556	-0.5	340.0	4.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0
2	0.0	-0.2500	0.642098	0.000000	0.000000	-0.5	331.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
3	0.0	-0.5625	0.731396	-1.083333	0.062222	1.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	1.2500	-0.314670	-0.916667	0.182222	-0.5	0.0	0.0	0.0	1.0	...	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 52 columns

Splitting Data into Features (X) and Target(y), then Train and Test.

In [5]:
```
# Define Dependent variables(features) as X

X = data_robust.iloc[:,1:]
y = data_robust.iloc[:,0]

X_train, X_test, y_train, y_test = train_test_split(
    data_robust.iloc[:,1:].values, data_robust.iloc[:,0].values, test_size=0.70, random_state=0)

print(X_test.shape)
print(X_train.shape)
print(y_test.shape)
print(y_train.shape)
```

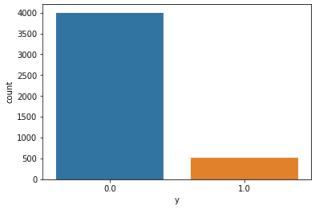
```
(3165, 51)
(1356, 51)
(3165,)
(1356,)
```

Data Visualization

Out of the 4, 521 customers of the Bank, about 88.5% did not subscribe the Term Deposit. Only 11.5% did.

In [6]:
```
sns.countplot(x="y", data=data_robust)
pd.value_counts(y)
```

Out[6]:
```
0.0    4000
1.0     521
Name: y, dtype: int64
```



A profile of the Bank's customers based of age

Using Pew's generational groups we can classify the Bank's custoemers as Millennials (18 and 34), Gen X (35 and 50), Baby Boomers(51 to 69) and the Silent generation (70 and 87).

Majority of the Bank's Customers are between the ages of 35 and 50. This group does not take loans or default loan payments. They own houses and are actively employed. 42% of this group responded positive to the campaign and subscribed the Term Deposit.

The Silent Generation (70 and 87) form the minority customers. Most of them are retired. They do not take loans or default loan payments. Half of members of this group responded positively to the campaigned, and also subscribed the Term Deposit.

A Correlation matrix showing how the Features relate to each other

In [7]:
```
cor = X.corr()
cor.head(7)
```

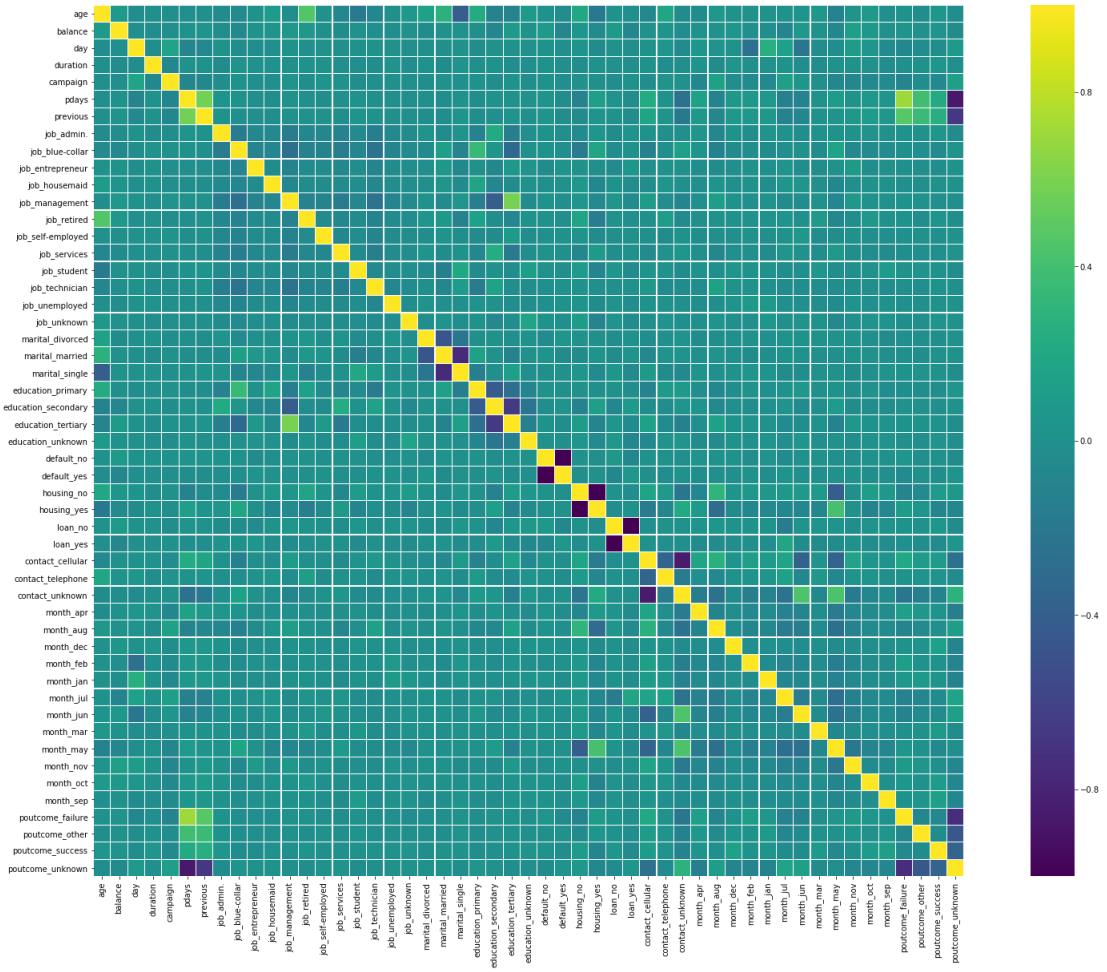
Out[7]:

	age	balance	day	duration	campaign	pdays	previous	job_admin.	job_blue-collar	job_entrepreneur	...	month_jun	month_mar	month_may	month_nov	month_oct	month_sep	poutcome_failure	poutco
age	1.000000	0.083820	-0.017853	-0.002367	-0.005148	-0.008894	-0.003511	-0.048385	-0.049307	0.015638	...	0.034671	0.057909	-0.119784	0.027659	0.081766	-0.015074	0.012693	-0.02617
balance	0.083820	1.000000	-0.008677	-0.015950	-0.009976	0.009437	0.026196	-0.022386	-0.057691	0.014523	...	0.056023	0.021805	-0.070809	0.120363	0.058694	0.005975	0.025719	0.00012
day	-0.017853	-0.008677	1.000000	-0.024629	0.160706	-0.094352	-0.059114	0.017052	-0.027025	-0.015707	...	-0.217517	-0.024570	-0.028992	0.095832	0.040235	-0.043666	-0.064235	-0.02106
duration	-0.002367	-0.015950	-0.024629	1.000000	-0.068382	0.010380	0.018080	-0.038763	0.028114	0.016267	...	-0.016196	-0.026212	0.008639	0.009572	0.004566	-0.020023	-0.012852	0.00810
campaign	-0.005148	-0.009976	0.160706	-0.068382	1.000000	-0.093137	-0.067833	-0.017895	0.008783	-0.012910	...	0.044317	-0.004045	-0.076263	-0.083385	-0.058536	-0.040207	-0.094021	-0.03043
pdays	-0.008894	0.009437	-0.094352	0.010380	-0.093137	1.000000	0.577562	0.035127	0.009374	-0.014704	...	-0.110324	0.008673	0.090216	0.012549	0.059521	0.047890	0.708380	0.38297
previous	-0.003511	0.026196	-0.059114	0.018080	-0.067833	0.577562	1.000000	0.020665	-0.014861	-0.013226	...	-0.084432	0.019445	0.027549	0.055400	0.088764	0.059763	0.475289	0.35838

7 rows × 51 columns

In [8]:
```
colormap = plt.cm.viridis
plt.figure(figsize=(30,20))
plt.title('Pearson Correlation of Features', y=1.05, size=25)
sns.heatmap(X.corr(),linewidths=0.2,vmax=1.0, square=True, cmap=colormap, linecolor='white', annot=False)
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x2029bb356a0>



We can see from the Correlation Matrix and the Colored Heatmap Matrix that Age positively correlates with balance, eventhough this correlation is weak. It suggests that older customers are more likely to have unused cash (balance) in their accounts- but this suggestion is weak.

There is also a negative correlation between Age and Day, Duration, Campaign Success. This implies that it takes less time of the day and duration of contact with older customers to respond positively and subscribe the Term Deposit.

Preliminary Data Modeling to determine the quality of the data

Since the data is imbalanced (88.5% -No and 11.5% Yes), it important to determine how much this imbalance will impact the metrics: Accuracy, Explained Variance, F1 score, and Mean Squared Error.

In [9]:
```
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, explained_variance_score, f1_score, mean_squared_error


# Train model
lr = LogisticRegression().fit(X_train, y_train)
predicted = lr.predict(X_test)


accuracy = accuracy_score(y_test, predicted)
explained_var = explained_variance_score(y_test, predicted)
f1 = f1_score(y_test, predicted)
mse = mean_squared_error(y_test, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.894786729858
Explained Variance is  -0.0160982372747
F1 Score is  0.397830018083
Mean Squared Error is  0.105213270142
```

Accuracy score for the imbalanced data is 89%. But, this model is so weak that it is able to explain less than 1% of the variations in the target response. With the low F1 score, the model is misclassifying the target response, and where it accurately predicts, it does so with less precision. The Mean Squared Error tells the difference between the actual and predicted response, but this needs to be as low as possible relative to other models.

Balancing the data with Resample (Up-sampling the minority class to be the same as the majority)

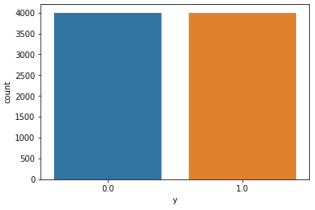
In [10]:
```
from sklearn.utils import resample

# First, we'll separate observations from each class into different DataFrames.
# Next, we'll resample the minority class with replacement, setting the number of samples to match that of the majority class.
# Finally, we'll combine the up-sampled minority class DataFrame with the original majority class DataFrame.

# Separate majority and minority classes
df_majority = data_robust[data_robust.y==0]
df_minority = data_robust[data_robust.y==1]


# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # sample with replacement
                                 n_samples=4000,    # to match majority class
                                 random_state=123) # reproducible results

# Combine majority class with upsampled minority class
data_upsampled = pd.concat([df_majority, df_minority_upsampled])


# Display new class counts
data_upsampled.y.value_counts()
```

Out[10]:
```
1.0    4000
0.0    4000
Name: y, dtype: int64
```

Proof of Balaced Data

In [11]:
```
sns.countplot(x="y", data=data_upsampled)
```

Out[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2029bca3390>
```



Redesignate Target, Features, Test and Train Variables

In [12]:
```
# designate target variable name (y)

targetName2 = 'y'
targetSeries = data_upsampled[targetName]
#remove target from current location and insert in collum 0
del data_upsampled[targetName]
data_upsampled.insert(0, targetName2, targetSeries)


#reprint dataframe and see target is in position 0
data_upsampled.head()

# Define Dependent variables(features) as X2
# split dataset into testing set and training.

X2 = data_upsampled.iloc[:,1:]
y2 = data_upsampled.iloc[:,0]

X_train2, X_test2, y_train2, y_test2 = train_test_split(
    data_upsampled.iloc[:,1:].values, data_upsampled.iloc[:,0].values, test_size=0.70, random_state=0)

print(X_test.shape)
print(X_train.shape)
print(y_test.shape)
print(y_train.shape)
```

```
(3165, 51)
(1356, 51)
(3165,)
(1356,)
```

Logistic Regression with Balanced Data

In [13]:
```
# Train model
lr2 = LogisticRegression().fit(X_train2, y_train2)
predicted = lr2.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.821428571429
Explained Variance is  0.286262523886
F1 Score is  0.818709209572
Mean Squared Error is  0.178571428571
```

The balanced data has improved the model's performance. Accuracy is down 7% but the model is able to explain 29% of the variations in the target response. Also, the model is not misclassifying the the target response as it previously did, and it is doing so with much precision.

–

Modelling

- Decision Tree
- KNN
- Random Forest
- Liner Support Vector Classification
- Redial Basis Function
- Artificial Neuron Network (ANA)
- Stochastic Gradient Desent
- Adaboost
- Bagging Classifier
- Gradient Boosting
- Extra Trees Classifier
- Stacking

–

In the following section, 12 Machine Learning Methods will be implemented to create at least 36 different models and iterations. Each method will follow FIVE (5) steps: Importing a package, Defining a function, Fitting a model, Making a prediction, and Quality evaluation.

–

Decision Tree Model

A decision tree is a map of the possible outcomes of a series of related choices. It allows an individual or organization to weigh possible actions against one another based on their costs, probabilities, and benefits.

A decision tree typically starts with a single node, which branches into possible outcomes. Each of those outcomes leads to additional nodes, which branch off into other possibilities. This gives it a treelike shape.

In [14]:
```
# Train model
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1,
                            min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)

dt = dt.fit(X_train2, y_train2)
predicted = dt.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.906428571429
Explained Variance is  0.637810543706
F1 Score is  0.91103565365
Mean Squared Error is  0.0935714285714
```

In [15]:
```
# Decision Tree Interation 2 where split is 3

dt = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=3, min_samples_leaf=1,
                            min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)

dt = dt.fit(X_train2, y_train2)
predicted = dt.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.905892857143
Explained Variance is  0.63266061805
F1 Score is  0.909899128056
Mean Squared Error is  0.0941071428571
```

In [16]:
```
# Decision Tree Interation 3 with min samples leaf is 2

dt = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=3, min_samples_leaf=2,
                            min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)

dt = dt.fit(X_train2, y_train2)
predicted = dt.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.879464285714
Explained Variance is  0.518296171172
F1 Score is  0.880340365183
Mean Squared Error is  0.120535714286
```

KNN Model

KNN estimates how likely a data point is to be a member of one group or the other depending on what group the data points nearest to it are in. The k-nearest-neighbor is a classification (or regression) algorithm that in order to determine the classification of a point, combines the classification of the K nearest points. It is supervised because you are trying to classify a point based on the known classification of other points.

In [17]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto',
                           leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=1)

knn = knn.fit(X_train2, y_train2)
predicted = knn.predict(X_test2)


accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.818035714286
Explained Variance is  0.277922259146
F1 Score is  0.82415875755
Mean Squared Error is  0.181964285714
```

In [18]:
```
# Interation 2 where neighbors is 8

knn = KNeighborsClassifier(n_neighbors=8, weights='uniform', algorithm='auto',
                           leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=1)

knn = knn.fit(X_train2, y_train2)
predicted = knn.predict(X_test2)


accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.805714285714
Explained Variance is  0.222979727214
F1 Score is  0.803963963964
Mean Squared Error is  0.194285714286
```

In [19]:
```
# Interation 3 where neighbors is 3

knn = KNeighborsClassifier(n_neighbors=3, weights='uniform', algorithm='auto',
                           leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=1)

knn = knn.fit(X_train2, y_train2)
predicted = knn.predict(X_test2)


accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.8525
Explained Variance is  0.423383838532
F1 Score is  0.860142228242
Mean Squared Error is  0.1475
```

In [20]:
```
# Interation 4 where neighbors is 1

knn = KNeighborsClassifier(n_neighbors=1, weights='uniform', algorithm='auto',
                           leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=1)

knn = knn.fit(X_train2, y_train2)
predicted = knn.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.897678571429
Explained Variance is  0.596939840833
F1 Score is  0.901257970016
Mean Squared Error is  0.102321428571
```

Random Forest

Random forest algorithm is a supervised classification algorithm. This algorithm creates the forest with a number of trees. In general, the more trees in the forest the more robust the forest looks like. In the same way in the random forest classifier, the higher the number of trees in the forest gives the high accuracy results.

In [21]:
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
                            min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,
                            random_state=None, verbose=0, warm_start=False, class_weight=None)

rf = rf.fit(X_train2, y_train2)
predicted = rf.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.925892857143
Explained Variance is  0.706632810875
F1 Score is  0.927662541398
Mean Squared Error is  0.0741071428571
```

In [22]:
```
# Iteration 2 where estimators is 12

rf = RandomForestClassifier(n_estimators=12, criterion='gini', max_depth=None, min_samples_split=2,
                            min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,
                            random_state=None, verbose=0, warm_start=False, class_weight=None)

rf = rf.fit(X_train2, y_train2)
predicted = rf.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.918214285714
Explained Variance is  0.675800732125
F1 Score is  0.920125566795
Mean Squared Error is  0.0817857142857
```

In [23]:
```
# Iteration 3 where estimators is 10

rf = RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
                            min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1,
                            random_state=None, verbose=0, warm_start=False, class_weight=None)

rf = rf.fit(X_train2, y_train2)
predicted = rf.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.921071428571
Explained Variance is  0.68571563048
F1 Score is  0.92229254571
Mean Squared Error is  0.0789285714286
```

Linear Support Vector Classification

The Linear Support Vector is similar to SVC with parameter kernel='linear', but implemented in terms of "liblinear" rather than "libsvm", so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

In [24]:
```
from sklearn.svm import LinearSVC

lsv = LinearSVC(penalty='l2', loss='squared_hinge', dual=True, tol=0.0001, C=1.0,
          multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0,
          random_state=None, max_iter=1000)

lsv = lsv.fit(X_train2, y_train2)
predicted = lsv.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.776428571429
Explained Variance is  0.138363036659
F1 Score is  0.753349093775
Mean Squared Error is  0.223571428571
```

In [25]:
```
# Iteration2 were penalty is l1

lsv = LinearSVC(penalty='l1', loss='squared_hinge', dual=False, tol=0.0001, C=1.0,
          multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0,
          random_state=None, max_iter=1000)

lsv = lsv.fit(X_train2, y_train2)
predicted = lsv.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.817321428571
Explained Variance is  0.271400763197
F1 Score is  0.812396845773
Mean Squared Error is  0.182678571429
```

In [26]:
```
# Iteration2 were penalty is l2, and hinge

lsv = LinearSVC(penalty='l2', loss='hinge', dual=True, tol=0.0001, C=1.0,
          multi_class='ovr', fit_intercept=True, intercept_scaling=1, class_weight=None, verbose=0,
          random_state=None, max_iter=1000)

lsv = lsv.fit(X_train2, y_train2)
predicted = lsv.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.805178571429
Explained Variance is  0.222334950144
F1 Score is  0.808428446005
Mean Squared Error is  0.194821428571
```

Radial basis function kernel

In machine learning, the (Gaussian) radial basis function kernel, or RBF kernel, is a popular kernel function used in various kernelized learning algorithms. In particular, it is commonly used in support vector machine classification

In [27]:
```
from sklearn.svm import SVC

rbf = SVC(C=1.0, kernel='rbf', degree=3, gamma='auto',
    coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None,
    verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

rbf = rbf.fit(X_train2, y_train2)
predicted = rbf.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.823035714286
Explained Variance is  0.293541808276
F1 Score is  0.825742922455
Mean Squared Error is  0.176964285714
```

In [28]:
```
# Iteration and C-2

rbf = SVC(C=2.0, kernel='rbf', degree=4, gamma='auto',
    coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None,
    verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

rbf = rbf.fit(X_train2, y_train2)
predicted = rbf.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.837321428571
Explained Variance is  0.35127199656
F1 Score is  0.840371473629
Mean Squared Error is  0.162678571429
```

In [29]:
```
# Iteration 3 where C-50

rbf = SVC(C=50, kernel='rbf', degree=4, gamma='auto',
    coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None,
    verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)

rbf = rbf.fit(X_train2, y_train2)
predicted = rbf.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.8825
Explained Variance is  0.54102025799
F1 Score is  0.888019060585
Mean Squared Error is  0.1175
```

Artificial Neuron Network (ANN)

An Artificial Neurol Network (ANN) is a computational model. It is based on the structure and functions of biological neural networks. It works like the way human brain processes information. It includes a large number of connected processing units that work together to process information.

In [30]:
```
from sklearn.neural_network import MLPClassifier


ann = MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=False,
       epsilon=1e-08, hidden_layer_sizes=(5, 2), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True,
       solver='lbfgs', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)

ann = ann.fit(X_train2, y_train2)
predicted = ann.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.760892857143
Explained Variance is  0.0501341914374
F1 Score is  0.749553314659
Mean Squared Error is  0.239107142857
```

In [31]:
```
# Iteration 2- where hidden layer 5, 1

ann = MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=False,
       epsilon=1e-08, hidden_layer_sizes=(5, 1), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True,
       solver='lbfgs', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)

ann = ann.fit(X_train2, y_train2)
predicted = ann.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.835
Explained Variance is  0.343097294755
F1 Score is  0.838968281631
Mean Squared Error is  0.165
```

In [32]:
```
# Iteration 3- where hidden layer 3, 1

ann = MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=False,
       epsilon=1e-08, hidden_layer_sizes=(3, 1), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=1, shuffle=True,
       solver='lbfgs', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)

ann = ann.fit(X_train2, y_train2)
predicted = ann.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.826785714286
Explained Variance is  0.335072722052
F1 Score is  0.839669421488
Mean Squared Error is  0.173214285714
```

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to discriminative learning of linear classifiers under convex loss functions such as (linear) Support Vector Machines and Logistic Regression.

In [33]:
```
from sklearn.linear_model import SGDClassifier

sg = SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=50, n_iter=None,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=0.21, verbose=0, warm_start=False)


sdg = sdg.fit(X_train2, y_train2)
predicted = sdg.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.666785714286
Explained Variance is  -0.0983358373983
F1 Score is  0.558447704685
Mean Squared Error is  0.333214285714
```

In [34]:
```
# Iteration 2- where penalty is L1

sdg = SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.1,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=50, n_iter=None,
       n_jobs=1, penalty='l1', power_t=0.5, random_state=None,
       shuffle=True, tol=0.21, verbose=0, warm_start=False)


sdg = sdg.fit(X_train2, y_train2)
predicted = sdg.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.710357142857
Explained Variance is  -0.0723983245235
F1 Score is  0.659243697479
Mean Squared Error is  0.289642857143
```

In [35]:
```
# Iteration 3- where epsilon is .2

sdg = SGDClassifier(alpha=0.0001, average=False, class_weight=None, epsilon=0.2,
       eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=50, n_iter=None,
       n_jobs=1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=0.21, verbose=0, warm_start=False)


sdg = sdg.fit(X_train2, y_train2)
predicted = sdg.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.761785714286
Explained Variance is  0.0777583288807
F1 Score is  0.780303030303
Mean Squared Error is  0.238214285714
```

Adaboost

An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

In [36]:
```
from sklearn.ensemble import AdaBoostClassifier

ada = AdaBoostClassifier(base_estimator=None, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None)

ada = ada.fit(X_train2, y_train2)
predicted = ada.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.831428571429
Explained Variance is  0.325872627
F1 Score is  0.831968672125
Mean Squared Error is  0.168571428571
```

In [37]:
```
# Interation 2 estimators is 30

ada = AdaBoostClassifier(base_estimator=None, n_estimators=30, learning_rate=1.0, algorithm='SAMME.R', random_state=None)

ada = ada.fit(X_train2, y_train2)
predicted = ada.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.831964285714
Explained Variance is  0.327887061078
F1 Score is  0.831934273977
Mean Squared Error is  0.168035714286
```

In [38]:
```
# Interation 3 learning rate is 2

ada = AdaBoostClassifier(base_estimator=None, n_estimators=30, learning_rate=2.0, algorithm='SAMME.R', random_state=None)

ada = ada.fit(X_train2, y_train2)
predicted = ada.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.278571428571
Explained Variance is  -1.81168160028
F1 Score is  0.161825726141
Mean Squared Error is  0.721428571429
```

Bagging Classifier

Bootstrap aggregating, also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach.

In [39]:
```
from sklearn.ensemble import BaggingClassifier

bag = BaggingClassifier(base_estimator=None, n_estimators=10, max_samples=1.0, max_features=1.0,
                        bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=1,
                        random_state=None, verbose=0)

bag = bag.fit(X_train2, y_train2)
predicted = bag.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.919285714286
Explained Variance is  0.684984500478
F1 Score is  0.922469982847
Mean Squared Error is  0.0807142857143
```

In [40]:
```
# Iteration 2 where estimators is 15

bag = BaggingClassifier(base_estimator=None, n_estimators=15, max_samples=1.0, max_features=1.0,
                        bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=1,
                        random_state=None, verbose=0)

bag = bag.fit(X_train2, y_train2)
predicted = bag.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.914821428571
Explained Variance is  0.673088586757
F1 Score is  0.919302994417
Mean Squared Error is  0.0851785714286
```

In [41]:
```
# Iteration 3 where estimators is 30

bag = BaggingClassifier(base_estimator=None, n_estimators=30, max_samples=1.0, max_features=1.0,
                        bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=1,
                        random_state=None, verbose=0)

bag = bag.fit(X_train2, y_train2)
predicted = bag.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.915535714286
Explained Variance is  0.675945759133
F1 Score is  0.919979698867
Mean Squared Error is  0.0844642857143
```

Gradient Boosting

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

In [42]:
```
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier(loss='deviance', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse',
                                min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3,
                                min_impurity_decrease=0.0, min_impurity_split=None, init=None, random_state=None,
                                max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto')

gb = gb.fit(X_train2, y_train2)
predicted = gb.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.868571428571
Explained Variance is  0.481145659796
F1 Score is  0.873409012728
Mean Squared Error is  0.131428571429
```

In [43]:
```
# Iteration 2 where is .2

gb = GradientBoostingClassifier(loss='deviance', learning_rate=0.2, n_estimators=100, subsample=1.0, criterion='friedman_mse',
                                min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3,
                                min_impurity_decrease=0.0, min_impurity_split=None, init=None, random_state=None,
                                max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto')

gb = gb.fit(X_train2, y_train2)
predicted = gb.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.888214285714
Explained Variance is  0.562289355285
F1 Score is  0.893064571233
Mean Squared Error is  0.111785714286
```

In [44]:
```
# Iteration 3 where is .8

gb = GradientBoostingClassifier(loss='deviance', learning_rate=0.8, n_estimators=100, subsample=1.0, criterion='friedman_mse',
                                min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3,
                                min_impurity_decrease=0.0, min_impurity_split=None, init=None, random_state=None,
                                max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, presort='auto')

gb = gb.fit(X_train2, y_train2)
predicted = gb.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.918035714286
Explained Variance is  0.681926433169
F1 Score is  0.921658986175
Mean Squared Error is  0.0819642857143
```

Extra Trees Classifier

The Extra-Tree method (standing for extremely randomized trees) with the main objective of further randomizing tree building in the context of numerical input features, where the choice of the optimal cut-point is responsible for a large proportion of the variance of the induced tree.

In [45]:
```
from sklearn.ensemble import ExtraTreesClassifier

etc = ExtraTreesClassifier(n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2,
                           min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=False, oob_score=False, n_jobs=1,
                           random_state=None, verbose=0, warm_start=False, class_weight=None)

etc = etc.fit(X_train2, y_train2)
predicted = etc.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.930357142857
Explained Variance is  0.721981311287
F1 Score is  0.930949008499
Mean Squared Error is  0.0696428571429
```

In [46]:
```
# Iteration 2 where estimators is 12

etc = ExtraTreesClassifier(n_estimators=12, criterion='gini', max_depth=None, min_samples_split=2,
                           min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=False, oob_score=False, n_jobs=1,
                           random_state=None, verbose=0, warm_start=False, class_weight=None)

etc = etc.fit(X_train2, y_train2)
predicted = etc.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.934107142857
Explained Variance is  0.736932103508
F1 Score is  0.934632418069
Mean Squared Error is  0.0658928571429
```

In [47]:
```
# Iteration 3 where estimators is 13

etc = ExtraTreesClassifier(n_estimators=13, criterion='gini', max_depth=None, min_samples_split=2,
                           min_samples_leaf=1, min_weight_fraction_leaf=0.5, max_features='auto', max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=False, oob_score=False, n_jobs=1,
                           random_state=None, verbose=0, warm_start=False, class_weight=None)

etc = etc.fit(X_train2, y_train2)
predicted = etc.predict(X_test2)

accuracy = accuracy_score(y_test2, predicted)
explained_var = explained_variance_score(y_test2, predicted)
f1 = f1_score(y_test2, predicted)
mse = mean_squared_error(y_test2, predicted)

print("Accuracy score is: ", accuracy)
print("Explained Variance is ", explained_var)
print ("F1 Score is ", f1)
print ("Mean Squared Error is ", mse)
```

```
Accuracy score is:  0.498392857143
Explained Variance is  3.33066907388e-16
F1 Score is  0.665236562984
Mean Squared Error is  0.501607142857
```

Stacking- Voting Classifier

Voting Classifier is a way of combining the predictions from multiple machine learning algorithms.It works by first creating two or more standalone models from your training dataset. A Voting Classifier can then be used to wrap your models and average the predictions of the sub-models when asked to make predictions for new data.

In [48]:
```
# Soft Voting

from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import cross_val_score


clf1 = LogisticRegression()
clf2 = RandomForestClassifier()
clf3 = DecisionTreeClassifier()
clf4 = KNeighborsClassifier()
clf5 = ExtraTreesClassifier()

eclf1 = VotingClassifier(estimators =[
         ('lr', clf1), ('rf', clf2), ('dt', clf3), ('knn', clf4), ('extree', clf5)],
                         voting='soft', weights=None, n_jobs=1, flatten_transform=None)


eclf1 = eclf1.fit(X, y)


print('5-fold cross validation:\n')
for clf, label in zip([clf1, clf2, clf3, clf4, clf5], ['Logistic Regression', 'Random Forest',
                                                       'Decision Tree', 'KNN', 'Extra Trees']):

    scores = cross_val_score(clf, X2, y2, cv=5, scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

```
5-fold cross validation:

Accuracy: 0.82 (+/- 0.01) [Logistic Regression]
Accuracy: 0.98 (+/- 0.00) [Random Forest]
Accuracy: 0.96 (+/- 0.01) [Decision Tree]
Accuracy: 0.90 (+/- 0.01) [KNN]
Accuracy: 0.99 (+/- 0.00) [Extra Trees]
```

In [49]:
```
clf1 = LogisticRegression()
clf2 = RandomForestClassifier()
clf3 = DecisionTreeClassifier()
clf4 = KNeighborsClassifier()
clf5 = ExtraTreesClassifier()


eclf1 = VotingClassifier(estimators=[
     ('logreg', clf1), ('randforest', clf2), ('decisiontree', clf3), ('KNN', clf4), ('extratrees', clf5)], voting='hard')
eclf1 = eclf1.fit(X2, y2)


print('5-fold cross validation:\n')
for clf, label in zip([clf1, clf2, clf3, clf4, clf5], ['Logistic Regression', 'Random Forest',
                                                       'Decision Tree', 'KNN', 'Extra Trees']):

    scores = cross_val_score(clf, X2, y2, cv=5, scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

```
5-fold cross validation:

Accuracy: 0.82 (+/- 0.01) [Logistic Regression]
Accuracy: 0.98 (+/- 0.00) [Random Forest]
Accuracy: 0.96 (+/- 0.01) [Decision Tree]
Accuracy: 0.90 (+/- 0.01) [KNN]
Accuracy: 0.99 (+/- 0.00) [Extra Trees]
```

In [50]:
```
# Iteration 3- Hard voting where cv is 3

clf1 = LogisticRegression()
clf2 = RandomForestClassifier()
clf3 = DecisionTreeClassifier()
clf4 = KNeighborsClassifier()
clf5 = ExtraTreesClassifier()


eclf1 = VotingClassifier(estimators=[
     ('logreg', clf1), ('randforest', clf2), ('decisiontree', clf3), ('KNN', clf4), ('extratrees', clf5)], voting='hard')
eclf1 = eclf1.fit(X2, y2)


print('5-fold cross validation:\n')
for clf, label in zip([clf1, clf2, clf3, clf4, clf5], ['Logistic Regression', 'Random Forest',
                                                       'Decision Tree', 'KNN', 'Extra Trees']):

    scores = cross_val_score(clf, X2, y2, cv=3, scoring='accuracy')
    print("Accuracy: %0.2f (+/- %0.2f) [%s]" % (scores.mean(), scores.std(), label))
```

```
5-fold cross validation:

Accuracy: 0.82 (+/- 0.01) [Logistic Regression]
Accuracy: 0.97 (+/- 0.00) [Random Forest]
Accuracy: 0.95 (+/- 0.00) [Decision Tree]
Accuracy: 0.89 (+/- 0.01) [KNN]
Accuracy: 0.98 (+/- 0.00) [Extra Trees]
```

Summary Of Models

In [58]:
```
summary = pd.read_excel("model summary.xlsx")
summary
```

Out[58]:

	Model	Interation	Accuracy %	Explained Variable %	F Score %	Mean Error
0	Extra Trees	Model 3	50	333.0	67.0	0.50
1	Extra Trees	Model 2	93	73.0	93.0	0.07
2	Extra Trees	Model 1	93	72.0	93.0	0.07
3	Random Forest	Model 2	92	69.0	92.0	0.08
4	Bagging	Model 3	92	69.0	92.0	0.08
5	Random Forest	Model 1	92	68.0	92.0	0.08
6	Random Forest	Model 3	92	68.0	92.0	0.08
7	Boosting	Model 3	92	68.0	92.0	0.08
8	Decision Tree	Model 1	91	66.0	92.0	0.09
9	Bagging	Model 1	91	66.0	92.0	0.09
10	Bagging	Model 2	91	65.0	91.0	0.10
11	Decision Tree	Model 2	90	61.0	91.0	0.10
12	Boosting	Model 2	89	56.0	89.0	0.11
13	Redial Basis	Model 3	88	54.0	88.0	0.12
14	Decision Tree	Model 3	88	52.0	88.0	0.12
15	Boosting	Model 1	87	48.0	87.0	0.13
16	KNN	Model 3	85	42.0	86.0	0.15
17	Redial Basis	Model 2	84	35.0	84.0	0.16
18	ANN	Model 2	84	34.0	84.0	0.17
19	ANN	Model 3	83	34.0	84.0	0.17
20	Adaboost	Model 1	83	33.0	83.0	0.17
21	Adaboost	Model 2	83	33.0	83.0	0.17
22	Redial Basis	Model 1	82	29.0	83.0	0.18
23	KNN	Model 1	82	28.0	82.0	0.18
24	Liner SVC	Model 2	82	27.0	81.0	0.18
25	Liner SVC	Model 3	81	23.0	80.0	0.19
26	KNN	Model 2	81	22.0	80.0	0.19
27	Stochastic Gradient	Model 2	79	21.0	81.0	0.21
28	Stochastic Gradient	Model 1	79	17.0	80.0	0.21
29	Liner SVC	Model 1	78	15.0	80.0	0.21
30	Stochastic Gradient	Model 3	79	15.0	80.0	0.23
31	ANN	Model 1	76	5.0	75.0	0.24
32	Adaboost	Model 3	28	-1.8	16.0	0.72
33	Stacking- Extra T Votes	Model 1	99	NaN	NaN	NaN
34	Stacking- Extra T Votes	Model 2	99	NaN	NaN	NaN
35	Stacking- Extra T Votes	Model 3	98	NaN	NaN	NaN

Decision

The most important metrics for measuring the quality of the models is the Explained Variable Score. It measures how much of the variations in the target response ("y) that can be explained by a model. The higher the Explained Variable Score, the better.

From the model summary, there is a positive correlation between Explained Variable and Accuracy Score, and Explained Variable and the F Score. Meaning, where the Explained Variable Score is high, the Accuracy Score and the F Score of that model will also be high.

Also, there is a negative correlation between Explained Variable Score and Mean Squared Error (MSE). Meaning, where the Explained Variable is high, the MSE reduces.

Therefore, based on the metrics, the Extra Trees Classifier Model- (Iteration 2) is the best fit. This model has the highest Explained Variable score of 73%, Accuracy of 93%, F score of 93% and MSE of 0.067

In addition, the Stacking Voting Classifier voted 99%, 99% and 98% on all three iterations for the Extra Trees Classifier Model.