Rodrigo Hernandez
Muzamil Syed
Mayra Gamboa
CS178

Group Project</b>

In [6]:
```
##########Libraries Used Throughout The Code:##########

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import mltools as ml
import mltools.dtree as dtree
import mltools.logistic2 as lcs2
import sklearn
from sklearn import svm
from sklearn import tree
%matplotlib inline
#Imported Library for Neural Network
#Imported Library for Third Learner
```

In [9]:
```
##########Imported Data:##########
import numpy as np
'''
We will use the provided Class Kaggle Data in our class Kaggle Competi
tion: CS178 Project 2016
https://inclass.kaggle.com/c/cs178-project-2016
'''
# Get Kaggle training data
X = np.genfromtxt("data/kaggle.X1.train.txt",delimiter=",")
Y = np.genfromtxt("data/kaggle.Y.train.txt",delimiter=",")

# also load features of the test data (to be predicted)
Xe1 = np.genfromtxt("data/kaggle.X1.test.txt",delimiter=",")

perSplit = 0.8 # Percent at which to split the training data
               # (e.g 0.8 = 80/20 split)

Xtr,Xte,Ytr,Yte = ml.splitData(X,Y,0.8)

print(X.shape)
```

```
(60000, 91)
```

In [3]:
```
##########Initialization of the Ensemble:##########
'''
We will start with an Ensemble of size 20...
'''
# Ensemble Variables
size = 20  # the amount of learners in the ensemble
features = 55 # the number of features to select from when bagging

# Create the ensemble
final_ensemble = [None] * size
```

Here we will be learning on decision trees. We try to tackle the problem of overfitting, which occurs with overcomplex trees. These trees can become unstable because small variations in the data might result in a completely different tree being generated. Luckily, we can avoid this problem by either setting the maximum depth of the tree or setting the minimum number of samples required at a leaf node. We proceed to learning a decision tree regressor on the data, and specifying a maximum depth of 20. We predict on the training data and the testing data and obtain the mean squared averages for both data sets. The mean squared averages are displayed as follows. Next, we specify a variety of maximum depths for the decision trees, ranging anywhere from 1 - 19. We want to find out how adjusting the maximum depths changes the complexities of the trees, and when they begin to overfit. Which maximum depth best handles the problem of overfitting? For each maximum depth, we learn a decision tree regressor and calculate the mean squared average of the training data we predicted on, and also calculate the mean squared average of the testing data we predicted on.

```
In [10]: ##########Code for the Decision Tree:##########
         #Imported Library for Decision Trees
         from sklearn import tree

         #Decision Tree Learner INFO...
         # Decision Tree Variables
         depth = 20 # the maxth depth of the decision tree
         nodes = 8 # the minimum number of data to split node

         #learn a decision tree regressor on data, specify max depth of 20
         learner = tree.DecisionTreeRegressor(max_depth=20)
         learner.fit(Xtr, Ytr)
         YhatTrain = learner.predict(Xtr)
         YhatTest = learner.predict(Xte)
         MSETrain = np.mean((Ytr - YhatTrain)**2)
         MSETest = np.mean((Yte - YhatTest)**2)
         print('{}{}'.format("MSE for training data: ", MSETrain))
         print('{}{}'.format("MSE for testing data: ", MSETest))

         #try different ranges of maximum depths
         for depth in range(20):
             learner = tree.DecisionTreeRegressor(max_depth = depth+1)
             learner.fit(Xtr, Ytr)
             Yhat_train = learner.predict(Xtr)
             Yhat_test = learner.predict(Xte)
             mseTrain = np.mean((Ytr - Yhat_train)**2)
             mseTest = np.mean((Yte - Yhat_test)**2)
             print("Depth {:02d} --> mse train: {}, mse validation: {}".format(
         depth+1, mseTrain, mseTest))
```

```
MSE for training data: 0.0357911570162
MSE for testing data: 0.712112171593
Depth 01 --> mse train: 0.557937953351, mse validation: 0.57414551383
Depth 02 --> mse train: 0.505146872922, mse validation: 0.519822936078
Depth 03 --> mse train: 0.472694791396, mse validation: 0.483718589446
Depth 04 --> mse train: 0.451999080531, mse validation: 0.465591656337
Depth 05 --> mse train: 0.434261655119, mse validation: 0.455494004267
Depth 06 --> mse train: 0.418033072227, mse validation: 0.446586295059
Depth 07 --> mse train: 0.397270867629, mse validation: 0.434863566073
Depth 08 --> mse train: 0.377611804684, mse validation: 0.438300262859
Depth 09 --> mse train: 0.353281895038, mse validation: 0.443817028614
Depth 10 --> mse train: 0.32512500598, mse validation: 0.467904685154
Depth 11 --> mse train: 0.2920177674, mse validation: 0.487800548445
Depth 12 --> mse train: 0.25566965481, mse validation: 0.525525238568
Depth 13 --> mse train: 0.216916242863, mse validation: 0.555434048757
Depth 14 --> mse train: 0.178732828402, mse validation: 0.586412749513
Depth 15 --> mse train: 0.143080936773, mse validation: 0.61639887708
Depth 16 --> mse train: 0.11355919281, mse validation: 0.638016511422
Depth 17 --> mse train: 0.0871032692009, mse validation: 0.67195780753
7
Depth 18 --> mse train: 0.0657788229549, mse validation: 0.68580835288
5
Depth 19 --> mse train: 0.0492075353514, mse validation: 0.70996072750
5
Depth 20 --> mse train: 0.0357897739192, mse validation: 0.71208041707
4
```

We stick to specifying a maximum depth of 20, and proceed to learning on decision trees with that fixed depth, and we adjust the next parameter vital to creating the decision trees we want. The next parameter we adjust is the minimum number of samples required at a leaf node, otherwise known as the min_samples_leaf parameter. We learn on a range from 2^3, up to 2^12 minimum number of samples required at a leaf node. We predict on the training data and on the testing data, and we calculate their mean squared averages. The mean squared averages of the training and testing data are displayed as follows.

In [5]:
```
for nodes in range(3, 13):
    learner = tree.DecisionTreeRegressor(max_depth = 20, min_samples_l
eaf = 2**nodes)
    learner.fit(Xtr, Ytr)
    Yhat_train = learner.predict(Xtr)
    Yhat_test = learner.predict(Xte)
    mseTrain = np.mean((Ytr - Yhat_train)**2)
    mseTest = np.mean((Yte - Yhat_test)**2)
    print("2^{} data at leaf node --> mse train: {}, mse validation: {
}".format(nodes, mseTrain, mseTest))
```

```
2^3 data at leaf node --> mse train: 0.159069620689, mse validation: 0
.59296293253
2^4 data at leaf node --> mse train: 0.238927974161, mse validation: 0
.515825198467
2^5 data at leaf node --> mse train: 0.303901057014, mse validation: 0
.457419046287
2^6 data at leaf node --> mse train: 0.348795257147, mse validation: 0
.42866571325
2^7 data at leaf node --> mse train: 0.376680650869, mse validation: 0
.424888119888
2^8 data at leaf node --> mse train: 0.398765237031, mse validation: 0
.429117852099
2^9 data at leaf node --> mse train: 0.414565160586, mse validation: 0
.438279933896
2^10 data at leaf node --> mse train: 0.431358990435, mse validation:
0.450114946475
2^11 data at leaf node --> mse train: 0.452946540527, mse validation:
0.46886867678
2^12 data at leaf node --> mse train: 0.47046962598, mse validation: 0
.483359063336
```

We decide to choose 2^8 minimum number of samples required at each leaf node, based on the results of the mean squared averages from the training and testing data.

Now we proceed to storing our individual learners in an ensemble. We set the size of the ensemble to 20 because we will be storing 20 different learners within the ensemble. We will store the KNN and the decision tree learners in the ensemble. Unfortunately, we will not be able to store the Support Vector learners in this ensemble. We create 10 different decision tree learners. Using a maximum depth of 20 and 8 minimum samples required at each leaf node as our favorite parameters, we add each of the learners to the ensemble of different learners. Finally, we proceed to storing our individual learners in an ensemble. We set the size of the ensemble to 20 because we will be storing 20 different learners within the ensemble. We will store the KNN and the decision tree learners in the ensemble. Unfortunately, we will not be able to store the Support Vector learners in this ensemble.

In [ ]:

```
##########Code to store each learner in the ensemble:##########
from sklearn.ensemble import BaggingRegressor

ensemble = [None]*10 #for now store 10 learners in ensemble before com
bining all learners
# Create learners and add to dtree ensemble
#dtLearners = BaggingRegressor(tree.DecisionTreeRegressor(max_depth =
20, min_samples_leaf = 2**8))
M = Xtr.shape[0]
Me = Xte.shape[0]
YtrHat = np.zeros((M,10))
YteHat = np.zeros((Me,10))
for l in range(10): #add 10 decision tree learners to the ensemble
    Xi, Yi = ml.bootstrapData(Xtr, Ytr, M)
    ensemble[l] = BaggingRegressor(tree.DecisionTreeRegressor(max_dept
h=20,min_samples_leaf = 2**8))
    ensemble[l].fit(Xi, Yi)
    Ythat = ensemble[l].predict(Xtr)
    Yehat = ensemble[l].predict(Xte)
```