Final Course Project

(a) Data

The data are satellite-based measurements of cloud temperature (infrared imaging), being used to predict the presence or absence of rainfall at a particular location. The data are courtesy of the UC Irvine Center for Hydrometeorology and Remote Sensing, and have been pre-processed to extract features corresponding to a model they use actively for predicting rainfall across the globe. Each data point corresponds to a particular lat-long location where the model thinks there might be rain; the extracted features include information such as IR temperature at that location, and information about the corresponding cloud (area, average temperature, etc.). The target value is a binary indicator of whether there was rain (measured by radar) at that location; you will notice that the data are slightly imbalanced (positives make up about 30% of the training data).

The training data (X_train, Y_train) and the test data features (X_test) are provided on the Kaggle link: https://inclass.kaggle.com/c/uc-irvine-2017w-cs178

Train the models using the former, make predictions using the test data features, and upload them to Kaggle. Kaggle will then score your predictions, and report your performance on a subset used for placing your team on the current leaderboard (the "leaderboard" data). After the competition, the score on the remainder of the data will be used to determine your final standing; this ensures that your scores are not affected by overfitting to the leaderboard data.

(b) Scoring

Scoring of predictions is done using AUC, the area under the ROC (receiver-operator characteristic) curve. This gives an average of your learner's performance at various levels of sensitivity to positive data. This means that you will likely do better if, instead of simply predicting the target class, you also include your confidence level of that class value, so that the ROC curve can be evaluated at different levels of specificity. To do so, you can report your confidence in class +1 (as a real number); the predictions will then be sorted in order of confidence, and the ROC curve evaluated.

(c) Project requirements

Project will consist of learning several predictors for the Kaggle data, as well as an ensemble "blend" of them, to try to do as well as possible at the prediction task. Specifically, learn *at least three* (more is good) different types of models; suggestions include: **K-Nearest neighbor, Linear models, Kernel methods, Random forest, Boosted learners, Neural network**, or **Other.**

Then, take your models and combine them using a blending or stacking technique -- this could be as simple as a straight average / vote, a weighted vote, or a stacked predictor (linear or other model type). Feel free to experiment and see what performance you can get.

(d) Project report

1. Neural Network

Step 1: Select the best value for number of hidden layers (h)
Split the data into (Xtr, Ytr) and (Xte, Yte) with a fraction of 0.75. We then trained the data with MLPRegressor from sklearn package, and calculated different mean square errors with a list of c values (i.e. refer to Table 1.1). We used ADAM as it generally works better on large data sets than LBFGS. We chose the best value of h = 4 because the predictor would be underfitting when h is smaller than 4 and overfitting when h is larger than 4.

Step 2: Pack the 2 predictors into an ensemble predictor and train it
We built an ensemble predictor that included MLPRegressor (with h=4 and ADAM algorithm) and GradientBoostingRegressor. We predicted the data (Xtr, Ytr) and (Xte, Yte) separately in it. The data set from X testing set was also trained for future prediction.

Step 3: Choose the best loss function for Gradient Boosting
We applied all 4 functions available in the predictor (i.e. refer to Table 1.2). Huber function, which is the combination of least squares regression and last absolute deviation, got the highest AUC score on validation data, which is 0.70687.

Step 4: Choose the best number of estimators for Gradient Boosting
Because Gradient Boosting used to robust to overfitting so a larger number of estimators usually results in better performance. Therefore, We tested the GradientBoostingRegressor (GBR) with different number of estimators to find the one with the highest AUC score (refer to Table 1.3). We chose 4000 estimators since the AUC score on validation started to decrease after 4000.

Step 5: Conclusion
AUC score on training data = 0.82025, AUC score on validation data = 0.72466, Kaggle score on test data = 0.73012

2. Random Forest

Step 1: Pack the 3 predictors into an ensemble predictor and train it
We built the “stacked” ensemble predictor that included RandomForestRegressor (assume number of trees=100), ExtraTreesRegressor, GradientBoostingRegressor from sklearn package. Because there are only 14 features in the X training set, we decided to add new features to it by multiplying the number of trees by 2. The data is then trained in the GBR. We split the data into (Xtr, Ytr) and (Xte, Yte) with a fraction of 0.75. The data sets (Xtr, Ytr) and (Xte, Yte) are predicted separately in the predictor. The data set from X testing set was also trained for future prediction.

Step 2: Find the best number of estimators for Random Forest
Because we needed to find the best estimator (i.e. number of tree) for the ensemble predictor, we randomly chose several estimator values and calculated the AUC scores on training data and validation data (i.e. refer to Table 2.1). We chose the best estimator to be 275 since it had the highest AUC score on validation data.

Step 3: Choose the best number of estimators for Gradient Boosting
Because Gradient Boosting used to robust to overfitting so a larger number of estimators usually results in better performance. Therefore, We tested the GradientBoostingRegressor (GBR) with different number of estimators to find the one with the highest AUC score (refer to Table 2.2). We chose 8000 estimators since the AUC score on validation data tended to grow very slow starting at 4000.

Step 4: Conclusion
AUC score on training data = 0.96943, AUC score on validation data = 0.76335, Kaggle score on test data = 0.76631

3. KNN

Step 1: Select the best k value
Split the data into (Xtr, Ytr) and (Xte, Yte) with a fraction of 0.75. We then trained the data with KNeighborsRegressor from sklearn package, and calculated different mean square errors with a list of k values (i.e. refer to Table 3.1). We chose the best k value to be 200 as the predictor would be underfitting when K is smaller than 200 and overfitting when K is larger than 200.

Step 2: Pack the 3 predictors into an ensemble predictor and train it
we built the "stacked" ensemble predictor that included KMeans, KNeighborsRegressor, GradientBoostingRegressor from sklearn package. First, the data sets were put in KMeans for clustering with the number of clusters=8, and then trained in KNeighborsRegressor with k=200. The sets were boosted by using Gradient Boosting. The data sets (Xtr, Ytr) and (Xte, Yte) are predicted separately in the predictor. The data set from X testing set was also trained for future prediction.

Step 4: Choose the best number of estimators for Gradient Boosting
Because Gradient Boosting used to robust to overfitting so a larger number of estimators usually results in better performance. Therefore, We tested the GradientBoostingRegressor (GBR) with different number of estimators to find the one with the highest AUC score (refer to Table 3.2). We chose 16000 estimators since the AUC score on validation data tended to grow very slow starting at 8000.

Step 5: Conclusion
AUC score on training data = 0.93615, AUC score on validation data = 0.75676, Kaggle score on test data = 0.76155

4. SVM

Step 1: Select the best c value
The X training set and Y training set into (Xtr, Ytr) and (Xte, Yte) with the training fraction of 0.75. We then trained the data with SVC from sklearn package, and calculated different mean square errors with a list of c values (i.e. refer to Table 4.1). Note that the kernel was set to RBF. We chose the best c value to be 0.8 because the predictor would be underfitting when C is smaller than 0.8 and overfitting when C is larger than 0.8. We use the same predictor again to calculate different mean square errors with the c value of 0.8 and a list of gamma values (i.e. refer to Table 4.2). We chose the best gamma value to be 0.1 as the predictor would be overfitting when gamma is larger than 0.1.

Step 2: Select the best kernel for SVC
We then calculated the validation MSE to determine which kernel is better to be used in SVC with C value of 0.8 (i.e. refer to Table 4.3). It turned out that RBF generally had a lower MSE than Sigmoid, so we used RBF as our kernel.

Step 3: Pack the 3 predictors into 1
We built the "stacked" ensemble predictor that included KMeans, SVC, GradientBoostingRegressor from sklearn package are used. We used the default values 8 and 100 to be the number of clusters in KMeans and number of estimators in GradientBoostingRegressor respectively. First, the data sets were trained in KMeans, and then in SVC

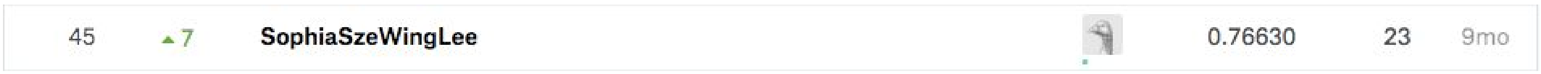
with C=0.8, gamma=0.1 and RBF kernel. The sets were boosted by using Gradient Boosting to have a better performance.

Step 4: Train the ensemble predictor
The data sets (Xtr, Ytr) and (Xte, Yte) are predicted separately in the predictor. The data set from X testing set was also trained for future prediction.

Step 5: Choose the best number of estimators for Gradient Boosting
Because Gradient Boosting used to robust to overfitting so a larger number of estimators usually results in better performance. Therefore, We tested the GradientBoostingRegressor (GBR) with different number of estimators to find the one with the highest AUC score (refer to Table 4.4). We chose 16000 estimators since the AUC score on validation data tended to grow very slow starting at 8000.

Step 6: Conclusion
AUC score on training data = 0.94748, AUC score on validation data = 0.75698, Kaggle score on test data = 0.76318

(e) Final Kaggle standing



45 ▲7 SophiaSzeWingLee 0.76630 23 9mo

Ranked #45 over 212 teams in in-class Kaggle Competition with a final score of 0.76630.

(f) Data Page

h value	MSE (Training data)	MSE (Validation data)
1	0.22478	0.27537
2	0.22480	0.22546
3	0.27368	0.22481
4	0.22548	0.22482
5	0.51241	0.51205
6	1.25718	1.25763
10	12.35137	11.92280

Table 1.1: table of MSE of training data and validation data for different h values

\	Least squares regression (LS)	Least absolute deviation (LAD)	Huber (LS & LAD)	Quantile regression
Validation AUC	0.70350	0.60408	0.70687	0.52816

Table 1.2: table of AUC score on validation data for different loss functions

Number of estimators	1000	2000	4000	8000
Validation AUC	0.72429	0.72434	0.72466	0.72390

Table 1.3: table of AUC score on validation data for different number of estimators in GBR

Estimator	AUC (Training data)	AUC (Validation data)
100	0.97684	0.69563
500	0.99353	0.69320
300	0.98511	0.69703
275	0.98133	0.69769

250	0.98089	0.69723

Table 2.1: table of AUC of training data and validation data for different estimators

Number of estimators	1000	2000	4000	8000
Validation AUC	0.75375	0.75674	0.76015	0.76335

Table 2.2: table of AUC score on validation data for different number of estimators in GBR

K value	MSE (Training data)	MSE (Validation data)
1	0.00467	0.40080
2	0.10193	0.30330
5	0.16406	0.24702
10	0.18345	0.22703
50	0.20062	0.21120
100	0.20439	0.21089
200	0.20642	0.21035
500	0.20862	0.21043
1000	0.21018	0.21114
5000	0.21910	0.21972
7500	0.22478	0.22555

Table 3.1: table of MSE of training data and validation data for different k values

Number of estimators	1000	2000	4000	8000	16000
Validation AUC	0.72377	0.73670	0.74631	0.75465	0.75676

Table 3.2: table of AUC score on validation data for different number of estimators in GBR

C value	MSE (Training data)	MSE (Validation data)
0.1	0.34627	0.32880
0.5	0.32653	0.32840
0.8	0.03373	0.32600
1	0.02373	0.33000
1.3	0.01680	0.33640
10	0.00600	0.34120
100	0.00427	0.34240

Table 4.1: table of MSE of training data and validation data for different c values

gamma value	MSE (Training data)	MSE (Validation data)
0.01	0.09413	0.32600
0.1	0.02880	0.32400
0.5	0.01200	0.32560
0.8	0.01107	0.32600
1	0.01093	0.32600
1.3	0.00533	0.32620
10	0.00387	0.32640

Table 4.2: table of MSE of training data and validation data for different gamma values

/	Sigmoid	RBF
MSE (Validation)	0.34080	0.32600

Table 4.3: table of MSE of validation data for different types of kernel.

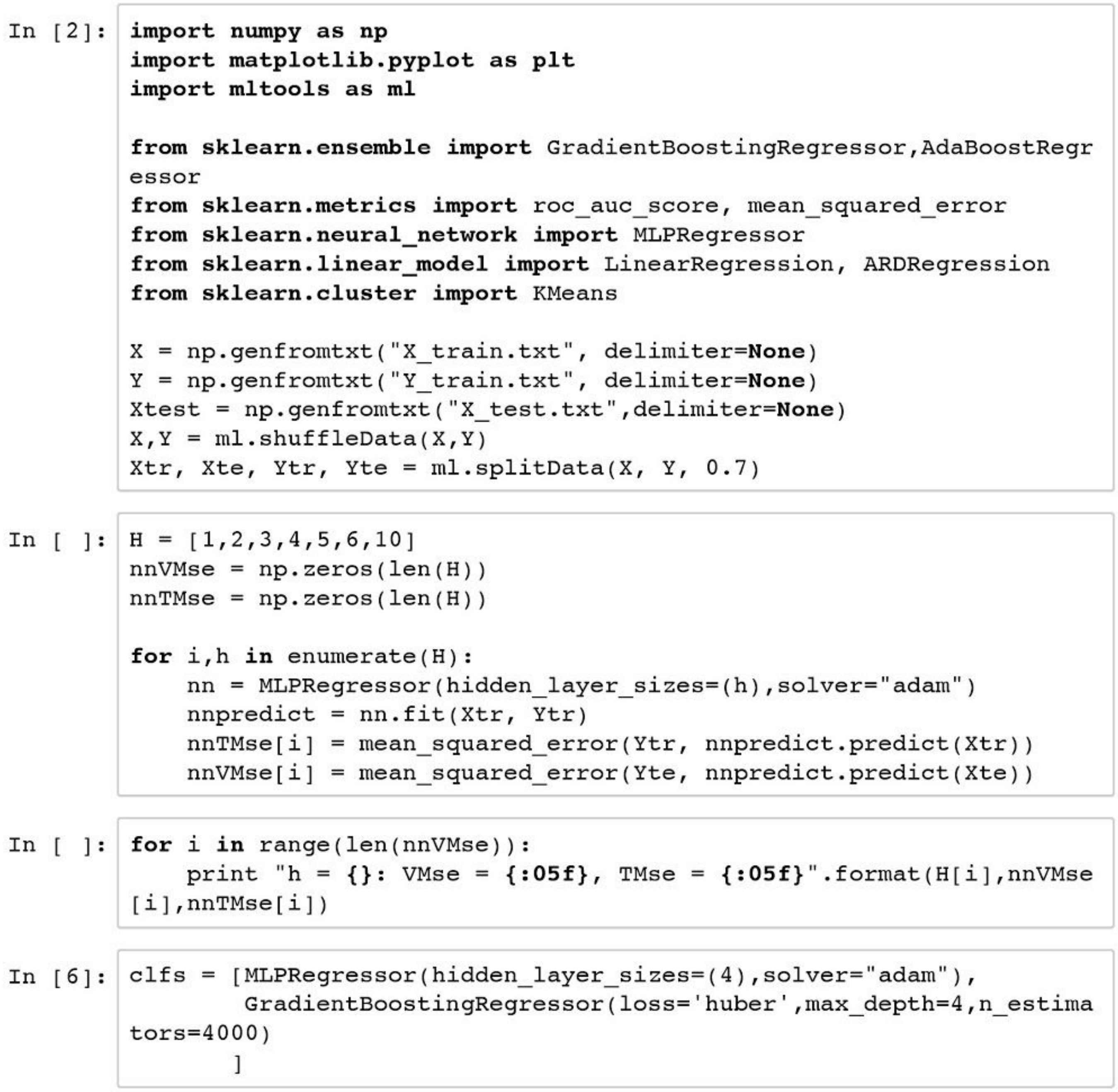
Number of estimators	1000	2000	4000	8000	16000

Validation AUC	0.73099	0.74096	0.74850	0.75544	0.75698

Table 4.4: table of AUC score on validation data for different number of estimators in GBR

(g) Code Page

1. Neural Network



```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        import mltools as ml

        from sklearn.ensemble import GradientBoostingRegressor,AdaBoostRegr
        essor
        from sklearn.metrics import roc_auc_score, mean_squared_error
        from sklearn.neural_network import MLPRegressor
        from sklearn.linear_model import LinearRegression, ARDRegression
        from sklearn.cluster import KMeans

        X = np.genfromtxt("X_train.txt", delimiter=None)
        Y = np.genfromtxt("Y_train.txt", delimiter=None)
        Xtest = np.genfromtxt("X_test.txt",delimiter=None)
        X,Y = ml.shuffleData(X,Y)
        Xtr, Xte, Ytr, Yte = ml.splitData(X, Y, 0.7)
```

```
In [ ]: H = [1,2,3,4,5,6,10]
        nnVMse = np.zeros(len(H))
        nnTMse = np.zeros(len(H))

        for i,h in enumerate(H):
            nn = MLPRegressor(hidden_layer_sizes=(h),solver="adam")
            nnpredict = nn.fit(Xtr, Ytr)
            nnTMse[i] = mean_squared_error(Ytr, nnpredict.predict(Xtr))
            nnVMse[i] = mean_squared_error(Yte, nnpredict.predict(Xte))
```

```
In [ ]: for i in range(len(nnVMse)):
            print "h = {}: VMse = {:05f}, TMse = {:05f}".format(H[i],nnVMse
        [i],nnTMse[i])
```

```
In [6]: clfs = [MLPRegressor(hidden_layer_sizes=(4),solver="adam"),
                GradientBoostingRegressor(loss='huber',max_depth=4,n_estima
        tors=4000)
               ]
```

In [7]:
```
temp_results_t = np.zeros((  len(Ytr)        ,len(clfs) ))   #x1,x2
temp_results_v = np.zeros((  len(Yte)        ,len(clfs) ))   #y1,y2
test1        = np.zeros((  Xtest.shape[0] ,len(clfs) ))
for i, clf in enumerate(clfs):
    clf.fit(Xtr,Ytr)
    temp_results_v[:,i] = clf.predict(Xte)
    temp_results_t[:,i] = clf.predict(Xtr)
    test1[:,i] = clf.predict(Xtest)

lr = LinearRegression(normalize=True)
lr.fit(temp_results_v, Yte)
```

Out[7]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normal
ize=True)
```

In [8]:
```
print "AUC on Train data: ",roc_auc_score(Ytr, lr.predict(temp_resu
lts_t))
print "AUC on Valid data: ",roc_auc_score(Yte, lr.predict(temp_resu
lts_v))
```

```
AUC on Train data:  0.820249024437
AUC on Valid data:  0.72466409965
```

In [20]:
```
Ypred_lr = lr.predict(test1)
np.savetxt('Yhat_nn_lr_3.txt', np.vstack( (np.arange(len(Ypred_lr))
, Ypred_lr) ).T, '%d, %.2f',header='ID,Prob1',comments='',delimiter
=',');
```

2. Random Forest

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import mltools as ml
        from mltools import dtree
        from mltools import cluster
        #import mltools.logistic2 as lc2

        import pandas as pd
        import numpy as np
        from sklearn.cross_validation import KFold
        from sklearn import metrics
        from sklearn.ensemble import RandomForestRegressor, ExtraTreesRegre
        ssor, GradientBoostingRegressor
        from sklearn.linear_model import RidgeCV
        from sklearn.metrics import roc_auc_score
```

```
sklearn/cross_validation.py:44: DeprecationWarning: This module wa
s deprecated in version 0.18 in favor of the model_selection modul
e into which all the refactored classes and functions are moved. A
lso note that the interface of the new CV iterators are different
from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
In [2]: X = np.genfromtxt("X_train.txt", delimiter=None)
        Y = np.genfromtxt("Y_train.txt", delimiter=None)
        Xtest = np.genfromtxt("X_test.txt",delimiter=None)
```

```
In [27]: n_trees = 275
         clfs = [
                 ExtraTreesRegressor(n_estimators = n_trees *2, bootstrap=Tr
         ue),
                 RandomForestRegressor(n_estimators = n_trees),
                 GradientBoostingRegressor(n_estimators = 30000)
              ]
```

```
In [28]: X1,X2,Y1,Y2 = ml.splitData(X,Y,0.75)
         temp_results_t = np.zeros((  len(Y1)          ,len(clfs) ))   #x1,x2
         temp_results_v = np.zeros((  len(Y2)          ,len(clfs) ))   #y1,y2
         test1         = np.zeros((  Xtest.shape[0] ,len(clfs) ))
         for i, clf in enumerate(clfs):
             clf.fit(X1,Y1)
             temp_results_v[:,i] = clf.predict(X2)
             temp_results_t[:,i] = clf.predict(X1)
             test1[:,i] = clf.predict(Xtest)
```

```
In [29]: #alphas = [0.0001, 0.005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0]
         alphas = [0.0001, 0.005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0, 5
         .0, 10.0, 50.0, 100.0, 500.0, 1000.0]

         from sklearn.metrics import roc_auc_score
         from sklearn.linear_model import LinearRegression

         lr = LinearRegression(normalize=True)
         lr.fit(temp_results_v, Y2)

         print "AUC on Train data: ",roc_auc_score(Y1, lr.predict(temp_resul
         ts_t))
         print "AUC on Valid data: ",roc_auc_score(Y2, lr.predict(temp_resul
         ts_v))

         AUC on Train data:  0.991269247971
         AUC on Valid data:  0.76047274566

In [30]: Ypred = bclf.predict(test1)

         Ypred_lr = lr.predict(test1)
         np.savetxt('Yhat_rf_3.txt', np.vstack( (np.arange(len(Ypred_lr)) ,
         Ypred_lr) ).T, '%d, %.2f',header='ID,Prob1',comments='',delimiter='
         ,');

In [27]: print type(temp_results_v)

         <type 'numpy.ndarray'>
```

3. KNN

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import mltools as ml

        from sklearn.feature_selection import f_classif, SelectKBest
        from sklearn.neighbors import KNeighborsClassifier,KNeighborsRegres
        sor
        from sklearn.metrics import accuracy_score, roc_auc_score,mean_squa
        red_error
        from sklearn.cluster import KMeans
        from sklearn.linear_model import LinearRegression, ARDRegression
        from sklearn.ensemble import GradientBoostingRegressor,AdaBoostRegr
        essor
```

```
In [2]: X = np.genfromtxt("X_train.txt", delimiter=None)
        Y = np.genfromtxt("Y_train.txt", delimiter=None)
        Xtest = np.genfromtxt("X_test.txt",delimiter=None)
        X,Y = ml.shuffleData(X,Y)
        Xtr, Xte, Ytr, Yte = ml.splitData(X, Y, 0.75)
```

```
In [ ]: X = np.genfromtxt("X_train.txt", delimiter=None)[:10000]
        Y = np.genfromtxt("Y_train.txt", delimiter=None)[:10000]
        Xtest = np.genfromtxt("X_test.txt",delimiter=None)
        X,Y = ml.shuffleData(X,Y)
        Xtr, Xte, Ytr, Yte = ml.splitData(X, Y, 0.75)

        K = [1,2,5,10,50,100,200,500,1000,1500,2000,5000,7500]
        knnpredict = []
        knnpredict1 = []
        knnVMse = []
        knnTMse = []
        for i,k in enumerate(K):
            knn = KNeighborsRegressor(n_neighbors = k)
            knn.fit(Xtr[:, 0:4], Ytr)
            knnpredict.append(knn.predict(Xte[:, 0:4]))
            knnpredict1.append(knn.predict(Xtr[:, 0:4]))

        for p in knnpredict:
            knnVMse.append(mean_squared_error(Yte, p))

        for p in knnpredict1:
            knnTMse.append(mean_squared_error(Ytr,p))

        for i in range(len(knnVMse)):
            print "k = {:04d}: VMse = {:05f}, TMse = {:05f}".format(K[i],kn
        nVMse[i],knnTMse[i])
```

```
In [ ]: bestK = K[knnVMse.index(min(knnVMse))]
        print bestK

        print len(knnpredict)
```

```
In [13]: clfs = [KMeans(n_clusters=8, init='random'),
                 KNeighborsRegressor(n_neighbors = 200),
                 GradientBoostingRegressor(n_estimators=16000)
                ]

         temp_results_t = np.zeros((  len(Ytr)        ,len(clfs) ))   #x1,x2
         temp_results_v = np.zeros((  len(Yte)        ,len(clfs) ))   #y1,y2
         test1          = np.zeros((  Xtest.shape[0] ,len(clfs) ))
         for i, clf in enumerate(clfs):
             clf.fit(Xtr,Ytr)
             temp_results_v[:,i] = clf.predict(Xte)
             temp_results_t[:,i] = clf.predict(Xtr)
             test1[:,i] = clf.predict(Xtest)

         lr = LinearRegression(normalize=True)
         lr.fit(temp_results_v, Yte)
```

```
Out[13]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normal
         ize=True)
```

```
In [14]: print "AUC on Train data: ",roc_auc_score(Ytr, lr.predict(temp_resu
         lts_t))
         print "AUC on Valid data: ",roc_auc_score(Yte, lr.predict(temp_resu
         lts_v))
```

```
         AUC on Train data:  0.936152812476
         AUC on Valid data:  0.756755744071
```

```
In [16]: Ypred_lr = lr.predict(test1)
         np.savetxt('Yhat_knn_lr_3_15.txt', np.vstack( (np.arange(len(Ypred_
         lr)) , Ypred_lr) ).T, '%d, %.2f',header='ID,Prob1',comments='',deli
         miter=',');
```

4. SVM

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import mltools as ml

        from sklearn.metrics import accuracy_score, roc_auc_score,mean_squa
        red_error
        from sklearn.ensemble import GradientBoostingRegressor,AdaBoostRegr
        essor
        from sklearn.svm import SVC,LinearSVC

        X = np.genfromtxt("X_train.txt", delimiter=None)
        Y = np.genfromtxt("Y_train.txt", delimiter=None)
        Xtest = np.genfromtxt("X_test.txt",delimiter=None)
        X,Y = ml.shuffleData(X,Y)
        Xtr, Xte, Ytr, Yte = ml.splitData(X, Y, 0.75)
```

```
In [ ]: C = [0.01, 0.1, 0.5, 0.8, 1,1.3,10,100]
        svcVMse = np.zeros(len(C))
        svcTMse = np.zeros(len(C))
        for i,c in enumerate(C):
            svr_c = SVC(C=c, kernel='rbf')
            predict_svr_c = svr_c.fit(Xtr, Ytr)
            svcVMse[i] = mean_squared_error(Yte, predict_svr_c.predict(Xte)
        )
            svcTMse[i] = mean_squared_error(Ytr, predict_svr_c.predict(Xtr)
        )
```

```
In [ ]: for i in range(len(svcVMse)):
            print "c = {:}: VMse = {:05f}, TMse = {:05f}".format(C[i],svcVM
        se[i],svcTMse[i])
```

```
In [ ]: G = [0.01, 0.1, 0.5, 0.8, 1,1.3,10,100]
        svcVMse2 = np.zeros(len(G))
        svcTMse2 = np.zeros(len(G))
        for i,g in enumerate(G):
            svr_c = SVC(C=0.8, kernel='rbf', gamma=g)
            predict_svr_c = svr_c.fit(Xtr, Ytr)
            svcVMse2[i] = mean_squared_error(Yte, predict_svr_c.predict(Xte
        ))
            svcTMse2[i] = mean_squared_error(Ytr, predict_svr_c.predict(Xtr
        ))
```

```
In [ ]: for i in range(len(svcVMse2)):
            print "g = {:}: VMse = {:05f}, TMse = {:05f}".format(G[i],svcVM
        se2[i],svcTMse2[i])
```

In [15]:
```
from sklearn.cluster import KMeans,MiniBatchKMeans,AffinityPropagat
ion
from sklearn.linear_model import LinearRegression, ARDRegression

clfs = [KMeans(n_clusters=8),
        SVC(C=0.8, kernel='rbf', gamma=0.1),
        GradientBoostingRegressor(n_estimators=16000)
       ]
```

In [16]:
```
temp_results_t = np.zeros((  len(Ytr)        ,len(clfs) ))   #x1,x2
temp_results_v = np.zeros((  len(Yte)        ,len(clfs) ))   #y1,y2
test1          = np.zeros((  Xtest.shape[0] ,len(clfs) ))
for i, clf in enumerate(clfs):
    clf.fit(Xtr,Ytr)
    temp_results_v[:,i] = clf.predict(Xte)
    temp_results_t[:,i] = clf.predict(Xtr)
    test1[:,i] = clf.predict(Xtest)

lr = LinearRegression(normalize=True)
lr.fit(temp_results_v, Yte)
```

Out[16]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normal
ize=True)
```

In [17]:
```
print "AUC on Train data: ",roc_auc_score(Ytr, lr.predict(temp_resu
lts_t))
print "AUC on Valid data: ",roc_auc_score(Yte, lr.predict(temp_resu
lts_v))
```

```
AUC on Train data:  0.947482712235
AUC on Valid data:  0.756948110858
```

In [18]:
```
Ypred_lr = lr.predict(test1)
np.savetxt('Yhat_svm_lr_6.txt', np.vstack( (np.arange(len(Ypred_lr)
) , Ypred_lr) ).T, '%d, %.2f',header='ID,Prob1',comments='',delimit
er=',');
```