178 Project

Team Hippo Time

Yiqiao Zhao 38332452

Justin Fu 80167602

Data

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

X = np.genfromtxt("X_train.txt",delimiter=None)
Y = np.genfromtxt("Y_train.txt",delimiter=None)
Xtest = np.genfromtxt("X_test.txt",delimiter=None)

Xval, Yval = ml.shuffleData( X, Y );
Xte, Yte = ml.shuffleData( X, Y);

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
sXte = scaler.fit_transform(Xte)
sXtest = scaler.transform(Xtest)
sXval= scaler.fit_transform(Xval)
```

Neural Network

In [2]:
```
from sklearn.neural_network import MLPClassifier

neuralNet1 = MLPClassifier(solver = 'adam', alpha=1e-5, hidden_layer_sizes=(25
0,3), random_state=1, early_stopping = True)
neuralNet1.fit(sXte, Yte)
```

Out[2]:
```
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=True, epsilon=1e-08,
       hidden_layer_sizes=(250, 3), 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='adam', tol=0.0001, validation_fraction=0.1, verbose=False,
       warm_start=False)
```

In [135]:
```
Xval, Yval = ml.shuffleData( X, Y );
Xte, Yte = ml.shuffleData( X, Y);
print("Training: {}".format(neuralNet1.score(sXte[:50000], Yte[:50000])))
print("Validation: {}".format(neuralNet1.score(sXval[50000:], Yval[50000:])))
```

```
Training: 0.60732
Validation: 0.61092
```

In [61]:
```
nnetYpred = neuralNet1.predict_proba(sXtest) #Using scaled to zero mean data
```

In [62]:
```
np.savetxt('Yhat_kaggle.txt',
          np.vstack( (np.arange(len(nnetYpred)), nnetYpred[:,1])).T,
          '%d, %.2f', header='ID,Prob1', comments='', delimiter=',');
```

Gradient Boosted Learner

In [5]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.preprocessing import StandardScaler

gbc = GradientBoostingClassifier(loss= 'exponential',learning_rate = 1.0, n_es
timators = 100, max_depth = 4)
gbc.fit(sXte, Yte)
```

Out[5]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=1.0, loss='exponential', max_depth=4,
              max_features=None, max_leaf_nodes=None,
              min_impurity_split=1e-07, min_samples_leaf=1,
              min_samples_split=2, min_weight_fraction_leaf=0.0,
              n_estimators=100, presort='auto', random_state=None,
              subsample=1.0, verbose=0, warm_start=False)
```

In [138]:
```
Xval, Yval = ml.shuffleData( X, Y );
Xte, Yte = ml.shuffleData( X, Y);
print("Training: {}".format(gbc.score(sXte[:50000], Yte[:50000])))
print("Validation: {}".format(gbc.score(sXval[50000:], Yval[50000:])))
```

```
Training: 0.59122
Validation: 0.59188
```

In [66]:
```
GradBoostYpred = gbc.predict_proba(sXtest) #scaled to mean zero
```

In [67]:
```
np.savetxt('Yhat_kaggle.txt',
          np.vstack( (np.arange(len(GradBoostYpred)), GradBoostYpred[:,1])).T,
          '%d, %.2f', header='ID,Prob1', comments='', delimiter=',');
```

Ada Boosted Learner

In [8]:
```
from sklearn.ensemble import AdaBoostClassifier

abc = AdaBoostClassifier(n_estimators=500)
abc.fit(sXte, Yte)
```

Out[8]:
```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=500, random_state=None)
```

In [139]:
```
Xval, Yval = ml.shuffleData( X, Y );
Xte, Yte = ml.shuffleData( X, Y);
print("Training: {}".format(abc.score(sXte[:50000], Yte[:50000])))
print("Validation: {}".format(abc.score(sXval[50000:], Yval[50000:])))
```

```
Training: 0.60576
Validation: 0.60762
```

In [70]:
```
AdaBoostYpred = abc.predict_proba(Xtest)
```

In [71]:
```
np.savetxt('Yhat_kaggle.txt',
           np.vstack( (np.arange(len(AdaBoostYpred)), AdaBoostYpred[:,1])).T,
           '%d, %.2f', header='ID,Prob1', comments='', delimiter=',');
```

KNN Classifier

In [11]:
```
class EnsembleKNNFra(ml.knn.knnClassify):
    def __init__(self, X, Y, **kwargs):
        self.classes = list(np.unique(Y))

        self.ensemble = []
        weights = []

        for i in range(X.shape[1]):
            for j in range(X.shape[1]):
                if (i>=j): continue
                if (j-i == 1):
                    x = X[:, i:(j+1)]
                else:
                    x = X[:, i:(j+1):(j-i-1)]
                tlearner = ml.knn.knnClassify(x,Y, **kwargs)
                w = tlearner.auc(x, Y)
                weights.append(w)
                self.ensemble.append((  tlearner, i, j, w ))

        weights, self.wscale = ml.transforms.rescale(np.array(weights))

    def predictSoft(self, X):

        yhatMtrx = []
        sumW = 0
        for elearner, i,j,w in self.ensemble:
            w = (w-self.wscale[0])*self.wscale[1]*10
            if (w<=10):
                continue
```

```
            sumW += w
            if (j-i == 1):
                x = X[:, i:(j+1)]
            else:
                x = X[:, i:(j+1):(j-i-1)]
            yhatMtrx.append(w * elearner.predict(x))

        yhatM = np.matrix(yhatMtrx)
        p = np.array([(1-m/sumW, m/sumW) for m in np.nditer(yhatM.mean(0)) ])

        return np.ndarray(shape=(X.shape[0], 2), buffer=p)

    def predict(self, X):

        psoft = np.array([float(m>0.5) for m in np.nditer(self.predictSoft(X))
 ])
        return np.ndarray(shape=psoft.shape, buffer=psoft)

class EnsembleKNNAvg(EnsembleKNNFra):
    def predictSoft(self, X):

        yhatMtrx = []
        sumW = 0
        for elearner, i,j,w in self.ensemble:
            w = (w-self.wscale[0])*self.wscale[1]*10
            if (w<=10):
                continue
            if (j-i == 1):
                x = X[:, i:(j+1)]
            else:
                x = X[:, i:(j+1):(j-i-1)]
            sumW += w
            yhatMtrx.append(elearner.predictSoft(x)[:, 1])

        yhatM = np.matrix(yhatMtrx)
        p = np.array([(1-m, m) for m in np.nditer(yhatM.mean(0)) ])

        return np.ndarray(shape=(X.shape[0],2), buffer=p)
```

In [12]:

```
def reduce(X):

    selection = [0, 1, 2, 8]
    return ml.transforms.fsubset(X, len(selection) ,feat = selection)
```

In [13]:
```
ks = (25, 50, 100)
tempXte = Xte[:1000]; tempYte = Yte[:1000]
tempXval = Xval; tempYval = Yval
for k in ks:
    print("K={}, trL={}, vaL={}".format(k, tempXte.shape[0],
tempXval.shape[0]))
    EknnF = EnsembleKNNFra(reduce(tempXte), tempYte, K=k)
    print("\tWith Fraction: Estimated AUC: {}".format(EknnF.auc(reduce(tempXva
l), tempYval)))
    EknnA = EnsembleKNNAvg(reduce(tempXte), tempYte, K=k)
    print("\tWith Average: Estimated AUC:
{}".format(EknnA.auc(reduce(tempXval), tempYval)))
```

```
K=25, trL=1000, vaL=100000
        With Fraction: Estimated AUC: 0.593516889024
        With Average: Estimated AUC: 0.625648239937
K=50, trL=1000, vaL=100000
        With Fraction: Estimated AUC: 0.590695291217
        With Average: Estimated AUC: 0.633037623774
K=100, trL=1000, vaL=100000
        With Fraction: Estimated AUC: 0.592130105318
        With Average: Estimated AUC: 0.632815484703
```

In [117]:
```
print(EknnA.auc(reduce(tempXte), tempYte))
```

```
0.711762369521
```

In [76]:
```
KNnYpred = EknnA.predictSoft(reduce(Xtest))
```

In [88]:
```
np.savetxt('Yhat_kaggle.txt',
          np.vstack( (np.arange(len(KNnYpred)), KNnYpred[:,1])).T,
          '%d, %.2f', header='ID,Prob1', comments='', delimiter=',');
```

SVM Kernel Methods

In [14]:
```
from sklearn import svm
SVMclf = svm.SVC(kernel="sigmoid", probability=True)
# XtrB, YtrB = balance(Xtr, Ytr)
# XtrB, YtrB = ml.shuffleData(XtrB, YtrB)
# XtrS, (mu,scale) = ml.rescale(Xtr)
# XvaS, (mu,scale) = ml.rescale(Xva, (mu,scale))
# , probability=True

SVMclf.fit(reduce(Xte), Yte)

print(SVMclf.score(reduce(Xte), Yte))
print(SVMclf.score(reduce(Xval), Yval))
```

```
0.65878
0.65878
```

In [140]:
```
Xval, Yval = ml.shuffleData( X, Y );
Xte, Yte = ml.shuffleData( X, Y);
print("Training: {}".format(SVMclf.score(reduce(sXte[:50000]), Yte[:50000])))
print("Validation: {}".format(SVMclf.score(reduce(sXval[50000:]),
Yval[50000:])))
```

```
Training: 0.65236
Validation: 0.65368
```

In [91]:
```
SVMclfYpred = SVMclf.predict_proba(reduce(Xtest))
```

In [92]:
```
np.savetxt('Yhat_kaggle.txt',
          np.vstack( (np.arange(len(SVMclfYpred)), SVMclfYpred[:,1])).T,
          '%d, %.2f', header='ID,Prob1', comments='', delimiter=',');
```

Random Forests

In [20]:
```
class RandomForestFra(ml.base.classifier):
    # soft prediction: return the fraction of members of ensemble that predict
ed class 1,
    def __init__(self, X, Y, **kwargs):
        self.classes = list(np.unique(Y))

        self.ensemble = []
        s = kwargs['size']
        del kwargs['size']
        t = kwargs['type']
        del kwargs['type']
#         print(t+"(x,y, **kwargs)")

        for i in range(s):
            x, y = ml.bootstrapData(X, Y)
            tlearner = eval(t+"(x,y, **kwargs)")
            self.ensemble.append(tlearner)

    def predictSoft(self, X):

        yhatMtrx = []
        for elearner in self.ensemble:
            yhatMtrx.append(elearner.predict(X))
        yhatM = np.matrix(yhatMtrx)
        p = np.array([(1-m, m) for m in np.nditer(yhatM.mean(0)) ])

        return np.ndarray(shape=(X.shape[0], 2), buffer=np.array(p))

    def predict(self, X):

        psoft = np.array([float(m>0.5) for m in np.nditer(self.predictSoft(X))
 ])
        return np.ndarray(shape=psoft.shape, buffer=psoft)

class RandomForestAvg(RandomForestFra):
    # soft prediction: return the average of your ensemble members' soft predi
ction scores
    def predictSoft(self, X):

        yhatMtrx = []
        for learner in self.ensemble:

            yhatMtrx.append(learner.predictSoft(X)[:, 1])
        yhatM = np.matrix(yhatMtrx)
        p = np.array([(1-m, m) for m in np.nditer(yhatM.mean(0)) ])

        return np.ndarray(shape=(X.shape[0],2), buffer=np.array(p))
```

In [25]:
```
t ="ml.dtree.treeClassify"
s =30
nF =3
print("ClassifierType:{}, size:{}, other Parameter: nF={}".format(t, s, nF))
RFF_DecsTree = RandomForestFra(Xte, Yte, type=t, size=s, maxDepth=15, nFeature
s= nF)
print("\tWith Fraction: Estimated AUC: {}".format(RFF_DecsTree.auc(Xval,
Yval)))
RFA_DecsTree = RandomForestAvg(Xte, Yte, type=t, size=s, maxDepth=15, nFeature
s= nF)
print("\tWith Average:  Estimated AUC: {}".format(RFA_DecsTree.auc(Xval,
Yval)))
```

```
ClassifierType:ml.dtree.treeClassify, size:30, other Parameter: nF=3
        With Fraction: Estimated AUC: 0.833704859666
        With Average:  Estimated AUC: 0.857241041733
```

In [141]:
```
Xval, Yval = ml.shuffleData( X, Y );
Xte, Yte = ml.shuffleData( X, Y);
print("Training: {}".format(RFA_DecsTree.auc(Xte[:50000], Yte[:50000])))
print("Validation: {}".format(RFA_DecsTree.auc(Xval[50000:], Yval[50000:])))
```

```
Training: 0.859770193488
Validation: 0.855987161841
```

In [80]:
```
RFADtYpred = RFA_DecsTree.predictSoft(Xtest)
```

In [89]:
```
np.savetxt('Yhat_kaggle.txt',
          np.vstack( (np.arange(len(RFADtYpred)), RFADtYpred[:,1])).T,
          '%d, %.2f', header='ID,Prob1', comments='', delimiter=',');
```

In [26]:
```
t ="ml.knn.knnClassify"
s =30
k = 25
print("ClassifierType:{}, size:{}, other Parameter: k={}".format(t, s, k))
RFF_KNN = RandomForestFra(reduce(Xte[:1000]), Yte[:1000], type=t, size=s, K=k)
print("\tWith Fraction: Estimated AUC: {}".format(RFF_KNN.auc(reduce(Xval[:100
0]), Yval[:1000])))
RFA_KNN = RandomForestAvg(reduce(Xte[:1000]), Yte[:1000], type=t, size=s, K=k)
print("\tWith Average:  Estimated AUC: {}".format(RFA_KNN.auc(reduce(Xval[:100
0]), Yval[:1000])))
```

```
ClassifierType:ml.knn.knnClassify, size:30, other Parameter: k=25
        With Fraction: Estimated AUC: 0.630819846978
        With Average:  Estimated AUC: 0.639121006981
```

In [120]:
```
print(RFA_KNN.auc(reduce(Xte[:1000]), Yte[:1000]))
```

```
0.726098831216
```

In [81]:
```
RFAKNnYpred = RFA_KNN.predictSoft(reduce(Xtest))
```

In [90]:
```
np.savetxt('Yhat_kaggle.txt',
          np.vstack( (np.arange(len(RFAKNnYpred)), RFAKNnYpred[:,1])).T,
          '%d, %.2f', header='ID,Prob1', comments='', delimiter=',');
```

In [144]:
```
print("finished")
```

```
finished
```

Ensemble of all Classifiers

In [106]:
```
colYpred = [ nnetYpred, GradBoostYpred, RFADtYpred ]
beforefinal = []
for x in range(100000):
    zero = (colYpred[0][x][0] + colYpred[1][x][0] + colYpred[2][x][0])/3.0
    one = (colYpred[0][x][1] + colYpred[1][x][1] + colYpred[2][x][1])/3.0
    temp = np.ndarray( (1,2), buffer = np.array([zero, one]))
    beforefinal.append(temp)
final = np.ndarray((100000,2), buffer = np.array(beforefinal))
```

In [107]:
```
np.savetxt('Yhat_kaggle.txt',
          np.vstack( (np.arange(len(final)), final[:,1])).T,
          '%d, %.2f', header='ID,Prob1', comments='', delimiter=',');
```

Kaggle scores

Classifier	Training Data	Validation Data	Kaggle Score
Neural Net	0.60732	0.61092	0.69068
Gradient Boost	0.59122	0.59188	0.72516
Ada Boost	0.60576	0.60762	0.50535
SVM Kernel	0.65236	0.65368	0.61
KNN	0.71176	0.63303	0.63627
Random Forest: Decision Tree	0.85977	0.85598	0.74408
Random Forest: KNN	0.72609	0.63912	0.62976
Ensemble (Bolded)	n/a	n/a	0.74096

Neural Network

```
We used the MLP Classifier provided by SK learn. We set early stopping to true beca
use the size of the data set was so large. We also gave it hidden layers = (250,3).
 At first we started with (1,1,1,1....1) with 14 hidden layers each with one node,
 but with more layers, the slower the classifier became. We shrunk the layers down,
 increasing the amount of nodes until we decided that less layers and more nodes wa
s the way to go. We also noticed that if the first layer had a large amount of node
s, the layers after it would take exponentially longer with more nodes. We also tra
ined the neural network with a data set that is scaled toe mean zero. The scaled da
ta allowed the values to have equal meaning since each feature had a different mean
ing.
```

Gradient Boosted Learner

```
The gradient Boosted Learner was taken from SK learn. We tried both deviant and exp
ontial loss and decided to go with exponential. We predicted that it was more suite
d for the large data set. We looped through 0.5, 1.0, 1.5, 2.0 and decided to go wi
th learning_rate because it consistently had higher AUC scores. With n_esitmators a
nd max_depth, we found that with more estimators the less amount of depth we needed
 or the Learner would be extremely slow. In addition, we sclaed the data set to mea
n zero as well.
```

Ada Boosted Learner

The Ada Boosted Learner was taken from SK learn. We can this learner a n_estimator of 500 by looping through a collection of numbers and deciding on the one with the best score.We scaled the data set to mean zero as well. This learner however did not perform very well and was not included in the final ensemble.

KNN Classifier

The problem with KNN learner is mainly the high dimension of the data and unbalanced classes, especially the high dimension which cause the Euclidean distance formula to perform poorly.

My solution for the problem with high dimension is to split the data into sections with lower dimensions and combine them back together and I wrote an ensemble K-nearest neighbor classifier, which based on the KNN classifier in mltool. In summary, it creates several classifiers with each pair of two columns (column 0 and 1, column 0 and 2, column 1 and 2, etc). I also manually chose four features (column 0, 1, 2 and 8) to improve the computational performance. The prediction of ensemble KNN classifier is the weighted average of the predictions from all individual classifiers and the weight of each classifier is determined by its scaled AUC score based on training data. The classifiers with weight less than a threshold will be ignored whereas the predictions others will be multiplied by the weight and summed up together. Regarding to unbalanced classes, I removed some of the data point to make the proportion of each class equal, which increases the performance a little (from 0.59 to 0.61).

The critical parameter setting is K, which is determined by testing out several possible settings. Starting from a number, the possible parameter setting doubles until the model is overfitting. For example, the possible settings are 25, 50, 100, 200 for 1000 data points and the model is overfitting when K=200.

SVM Kernel4

For Support Vector Machine, I used the Scikit-learn library (sklearn.svm). The support vector machine doesn’t work well with high dimension data since it also depends on distance between data points. I manually reduced the number of features to 6 (0, 1, 2, 8, 10 and 11) and train the model with it. Without the process of reducing features, it will require a huge amount of time to execute the code. Scaling the data doesn’t affect the result much. The validation score only increased by 0.001. On the other hand, balancing data actually has negative on the result, causes the score to decrease by 0.03.
Sklearn.svm supports different choices of kernels: RBF, linear, polynomial and sigmoid. Polynomial kernel takes too long to even produce a result. The results of RBF and linear are rather similar while the result of sigmoid kernel is about 0.02 lower. In the end, I chose RBF kernel to train on reduced, not balanced, not scaled data.

Random Forest Tree & Random Forest KNN

```
I generalized the Random forest to take any classifier and I tried it on decision t
ree and KNN classifier. For random forest on KNN classifier, it improves the perfor
mance a little bit, but it takes too much computation to improve further. Random fo
rest with decision tree is way better in terms of performance and speed. Parameters
 are size, max depth of each decision tree, and numbers of features. Among them, si
ze of the forest is the most critical one. Larger amount leads to a better predicti
on by reducing the overfitting caused by the relatively large maximum depth of each
 tree. The parameter I chose is maximum depth = 15, number of feature = 3 and size
 of forest = 30. The final result is the average of soft prediction
```

Ensemble

```
For the Emsemble, we wanted to include every learner that we studied. However, we n
oticed that some of our learners were very weak and in turn were excluded. In the e
nd, we only used the neural Network, the Gradient Boosted Learner, and the Random F
orest Tree. We took the predictSoft or predict_prob of each learner and averaged th
e values with their corresponding indexes. We chose this method because our three l
earners already had very high scores and we did not want to run the risk of corrupt
ing it. The average of the three learners gave us a reasonably high score on Kaggl
e, 0.74.
```

In [ ]: