Jeret Mccoy, Abelardo Riojas, Gustavo Flores

Algorithm	Train Error	Test Error	Train Time
Decision Tree	0%	16%	0.0557 sec
Random Forest (100 trees)	0%	9.1%	0.7881 sec
Random Forest (300 trees)	0%	8.9%	2.3988 sec
Logistic Regression	19.49%	21.65%	0.1850 sec
Naive Bayes (Gaussian)	20.23%	20.35%	0.0093 sec
AdaBoost (30 trees)	0.0%	14.9%	0.0842 sec
AdaBoost (100 trees)	0.0%	15.15%	0.0685 sec
GradientBoost (30 stumps)	8.92%	14.4%	1.0435 sec
GradientBoost (100 stumps)	6.00%	13.1%	3.548 sec
GradientBoost (30 trees)	4.6%	16%	2.7098 sec

SVM

C	gamma	Train Error	Test Error	Train Time
0.01	auto	75.83%	23.05%	2.919 sec
0.01	scale	18.941%	27.75%	1.1835 sec
0.1	auto	75.83%	76.95%	2.9393 sec
0.1	scale	13.04%	15.05%	0.5961 sec
1	auto	0%	76.15%	3.2692 sec
1	scale	9.66%	11.4%	0.5974 sec
10	auto	0%	75.45%	4.9145 sec
10	scale	6.75%	9.6%	0.5725 sec
100	auto	0%	75.45%	3.7333 sec
100	scale	3.82%	9.45%	0.4929 sec

1000	auto	0%	75.45%	6.6823 sec
1000	scale	1.18%	10.5%	0.6015 sec
10000	auto	0%	75.45%	4.8888 sec
10000	scale	0%	10.95%	1.3588 sec
100000	auto	0%	75.45%	4.4996 sec
100000	scale	0%	11%	0.8204 sec

From the data collected the best method is Random Forest with 100 trees.

```
#load the data
import pandas as pd
import numpy as np
import time
X = pd.read_csv('X.dat', sep=' ', header=None)
Xtest = pd.read_csv('Xtest.dat', sep=' ', header=None)
Y = pd.read_csv('Y.dat', header=None)
Ytest = pd.read_csv('Ytest.dat', header=None)
X = np.array(X)
Xtest = np.array(Xtest)
Y = np.array(Y)
Ytest = np.array(Ytest)

print(X.shape, Xtest.shape, Y.shape, Ytest.shape)

#start with decision tree classifier

from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier()

start = time.time()
tree.fit(X,Y)
end = time.time()


preds = tree.predict(X)
count = 0
for pred, target in zip(preds,Y):
    if pred==target:
        count+=1
```

```
train_accuracy = count/len(X)


preds = tree.predict(Xtest)
count = 0
for pred, target in zip(preds,Ytest):
    if pred==target:
        count+=1
test_accuracy = count/len(Xtest)

print('Decision Tree Train Accuracy: {}%'.format(train_accuracy))
print('Decision Tree Test Accuracy: {}%'.format(test_accuracy))
print('Decision Tree Training Time: {:.4f} seconds'.format(end-start))
#now we do random forest (100 trees)

from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(n_estimators=100)

start = time.time()
forest.fit(X,Y)
end = time.time()

preds = forest.predict(X)
count = 0
for pred, target in zip(preds,Y):
    if pred==target:
        count+=1
train_accuracy = count/len(X)


preds = forest.predict(Xtest)
count = 0
for pred, target in zip(preds,Ytest):
    if pred==target:
        count+=1
test_accuracy = count/len(Xtest)

print('Random Forest (100 trees) Train Accuracy: {}%'.format(train_accuracy))
print('Random Forest (100 trees) Test Accuracy: {}%'.format(test_accuracy))
print('Random Forest (100 trees) Training Time: {:.4f} seconds'.format(end-start))
#now we do random forest (300 trees)

from sklearn.ensemble import RandomForestClassifier
```

```
forest = RandomForestClassifier(n_estimators=300)

start = time.time()
forest.fit(X,Y)
end = time.time()

preds = forest.predict(X)
count = 0
for pred, target in zip(preds,Y):
    if pred==target:
        count+=1
train_accuracy = count/len(X)


preds = forest.predict(Xtest)
count = 0
for pred, target in zip(preds,Ytest):
    if pred==target:
        count+=1
test_accuracy = count/len(Xtest)

print('Random Forest (300 trees) Train Accuracy: {}%'.format(train_accuracy))
print('Random Forest (300 trees) Test Accuracy: {}%'.format(test_accuracy))
print('Random Forest (300 trees) Training Time: {:.4f} seconds'.format(end-start))
#moving on to logistic regression

from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()

start = time.time()
logreg.fit(X,Y)
end = time.time()

preds = logreg.predict(X)
count = 0
for pred, target in zip(preds,Y):
    if pred==target:
        count+=1
train_accuracy = count/len(X)


preds = logreg.predict(Xtest)
```

```
count = 0
for pred, target in zip(preds,Ytest):
    if pred==target:
        count+=1
test_accuracy = count/len(Xtest)

print('Logistic Regression Train Accuracy: {}'.format(train_accuracy))
print('Logistic Regression Test Accuracy: {}'.format(test_accuracy))
print('Logistic Regression Training Time: {:.4f} seconds'.format(end-start))
#let's try gaussian naive bayes now

from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()

start = time.time()
gnb.fit(X,Y)
end = time.time()

preds = gnb.predict(X)
count = 0
for pred, target in zip(preds,Y):
    if pred==target:
        count+=1
train_accuracy = count/len(X)

preds = gnb.predict(Xtest)
count = 0
for pred, target in zip(preds,Ytest):
    if pred==target:
        count+=1
test_accuracy = count/len(Xtest)

print('Gaussian NB Train Accuracy: {}%'.format(train_accuracy))
print('Gaussian NB Test Accuracy: {}%'.format(test_accuracy))
print('Gaussian NB Training Time: {:.4f} seconds'.format(end-start))
#let's try multinomial naive bayes now

from sklearn.naive_bayes import MultinomialNB

mnb = MultinomialNB()

start = time.time()
mnb.fit(X,Y)
```

```
end = time.time()

preds = mnb.predict(X)
count = 0
for pred, target in zip(preds,Y):
    if pred==target:
        count+=1
train_accuracy = count/len(X)

preds = mnb.predict(Xtest)
count = 0
for pred, target in zip(preds,Ytest):
    if pred==target:
        count+=1
test_accuracy = count/len(Xtest)

print('Multinomial NB Train Accuracy: {}%'.format(train_accuracy))
print('Multinomial NB Test Accuracy: {}%'.format(test_accuracy))
print('Multinomial NB Training Time: {:.4f} seconds'.format(end-start))
#let's try complement naive bayes now

from sklearn.naive_bayes import ComplementNB

cnb = ComplementNB()

start = time.time()
cnb.fit(X,Y)
end = time.time()

preds = cnb.predict(X)
count = 0
for pred, target in zip(preds,Y):
    if pred==target:
        count+=1
train_accuracy = count/len(X)

preds = cnb.predict(Xtest)
count = 0
for pred, target in zip(preds,Ytest):
    if pred==target:
        count+=1
test_accuracy = count/len(Xtest)

print('Complement NB Train Accuracy: {}%'.format(train_accuracy))
```

```
print('Complement NB Test Accuracy: {}%'.format(test_accuracy))
print('Complement NB Training Time: {:.4f} seconds'.format(end-start))
#let's try bernoulli naive bayes now

from sklearn.naive_bayes import BernoulliNB

bnb = BernoulliNB()

start = time.time()
bnb.fit(X,Y)
end = time.time()

preds = bnb.predict(X)
count = 0
for pred, target in zip(preds,Y):
    if pred==target:
        count+=1
train_accuracy = count/len(X)

preds = bnb.predict(Xtest)
count = 0
for pred, target in zip(preds,Ytest):
    if pred==target:
        count+=1
test_accuracy = count/len(Xtest)

print('Bernoulli NB Train Accuracy: {}%'.format(train_accuracy))
print('Bernoulli NB Test Accuracy: {}%'.format(test_accuracy))
print('Bernoulli NB Training Time: {:.4f} seconds'.format(end-start))
#moving on to adaboost (30 trees)

from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

abc = AdaBoostClassifier(DecisionTreeClassifier(), n_estimators=30)

start = time.time()
abc.fit(X,Y)
end = time.time()

preds = abc.predict(X)
count = 0
for pred, target in zip(preds,Y):
    if pred==target:
```

```
        count+=1
train_accuracy = count/len(X)

preds = abc.predict(Xtest)
count = 0
for pred, target in zip(preds,Ytest):
    if pred==target:
        count+=1
test_accuracy = count/len(Xtest)

print('Adaboost (30 trees) Train Accuracy: {}%'.format(train_accuracy))
print('Adaboost (30 trees) Test Accuracy: {}%'.format(test_accuracy))
print('Adaboost (30 trees) Training Time: {:.4f} seconds'.format(end-start))
#continuing onto adaboost (100 trees)

from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

abc = AdaBoostClassifier(DecisionTreeClassifier(), n_estimators=100)

start = time.time()
abc.fit(X,Y)
end = time.time()

preds = abc.predict(X)
count = 0
for pred, target in zip(preds,Y):
    if pred==target:
        count+=1
train_accuracy = count/len(X)

preds = abc.predict(Xtest)
count = 0
for pred, target in zip(preds,Ytest):
    if pred==target:
        count+=1
test_accuracy = count/len(Xtest)

print('Adaboost (100 trees) Train Accuracy: {}%'.format(train_accuracy))
print('Adaboost (100 trees) Test Accuracy: {}%'.format(test_accuracy))
print('Adaboost (100 trees) Training Time: {:.4f} seconds'.format(end-start))
#now gradientboost (30 stumps)

from sklearn.ensemble import GradientBoostingClassifier
```

```
gbc = GradientBoostingClassifier(n_estimators=30, learning_rate=1.0, max_depth=1) #depth=1
makes it a stump

start = time.time()
gbc.fit(X,Y)
end = time.time()

preds = gbc.predict(X)
count = 0
for pred, target in zip(preds,Y):
    if pred==target:
        count+=1
train_accuracy = count/len(X)

preds = gbc.predict(Xtest)
count = 0
for pred, target in zip(preds,Ytest):
    if pred==target:
        count+=1
test_accuracy = count/len(Xtest)

print('GradientBoost (30 stumps) Train Accuracy: {}%'.format(train_accuracy))
print('GradientBoost (30 stumps) Test Accuracy: {}%'.format(test_accuracy))
print('GradientBoost (30 stumps) Training Time: {:.4f} seconds'.format(end-start))


#now gradientboost (100 stumps)

from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1)
#depth=1 makes it a stump

start = time.time()
gbc.fit(X,Y)
end = time.time()

preds = gbc.predict(X)
count = 0
for pred, target in zip(preds,Y):
    if pred==target:
        count+=1
train_accuracy = count/len(X)
```

```
preds = gbc.predict(Xtest)
count = 0
for pred, target in zip(preds,Ytest):
    if pred==target:
        count+=1
test_accuracy = count/len(Xtest)

print('GradientBoost (100 stumps) Train Accuracy: {}%'.format(train_accuracy))
print('GradientBoost (100 stumps) Test Accuracy: {}%'.format(test_accuracy))
print('GradientBoost (100 stumps) Training Time: {:.4f} seconds'.format(end-start))


#now gradientboost (30 trees)

from sklearn.ensemble import GradientBoostingClassifier

gbc = GradientBoostingClassifier(n_estimators=30, learning_rate=1) #depth makes it a tree

start = time.time()
gbc.fit(X,Y)
end = time.time()

preds = gbc.predict(X)
count = 0
for pred, target in zip(preds,Y):
    if pred==target:
        count+=1
train_accuracy = count/len(X)

preds = gbc.predict(Xtest)
count = 0
for pred, target in zip(preds,Ytest):
    if pred==target:
        count+=1
test_accuracy = count/len(Xtest)

print('GradientBoost (30 trees) Train Accuracy: {}%'.format(train_accuracy))
print('GradientBoost (30 trees) Test Accuracy: {}%'.format(test_accuracy))
print('GradientBoost (30 trees) Training Time: {:.4f} seconds'.format(end-start))
```

```
#LAST ONE!! Support Vector Machines

from sklearn.svm import SVC
Cs = np.logspace(-2, 5, num =8)
gammas = ['auto', 'scale']

for C in Cs:
    for gamma in gammas:
        svc = SVC(C=C, gamma=gamma)

        start = time.time()
        svc.fit(X,Y)
        end = time.time()

        preds = svc.predict(X)
        count = 0
        for pred, target in zip(preds,Y):
            if pred==target:
                count+=1
        train_accuracy = count/len(X)

        preds = svc.predict(Xtest)
        count = 0
        for pred, target in zip(preds,Ytest):
            if pred==target:
                count+=1
        test_accuracy = count/len(Xtest)

        print('SVM Classification (C = {}, gamma = {}) Train Accuracy: {}%'.format(C, gamma,
train_accuracy))
        print('SVM Classification (C = {}, gamma = {}) Test Accuracy: {}%'.format(C, gamma,
test_accuracy))
        print('SVM Classification (C = {}, gamma = {}) Training Time: {:.4f} seconds'.format(C,
gamma, end-start))
```