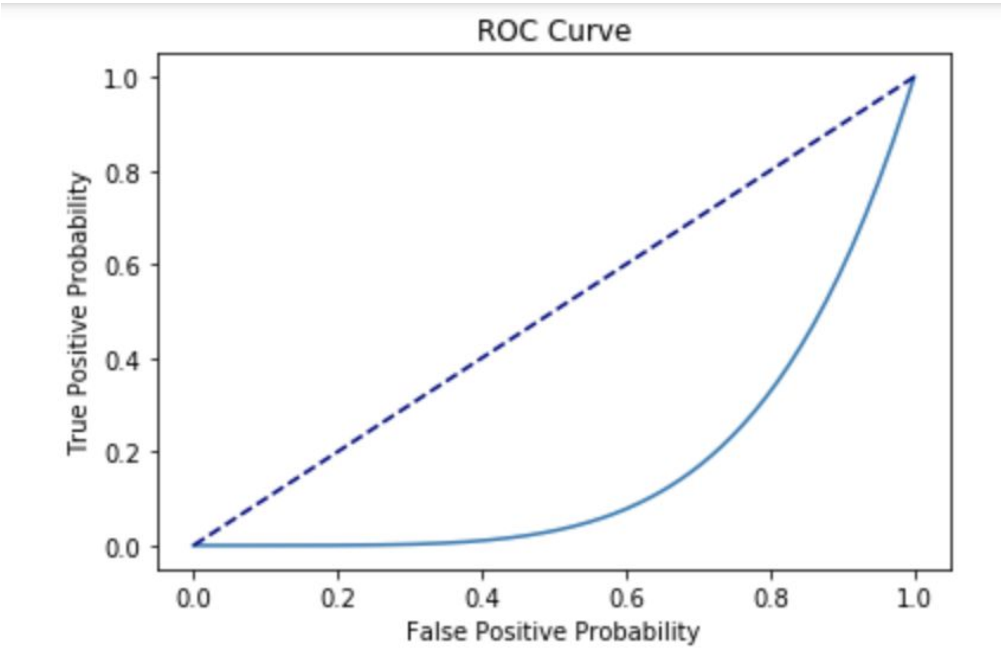
2b)

```
import matplotlib.pyplot as plt
import numpy as np

#intialize t values
t_range = np.linspace(0,10,1000)

#probability of miss detection and false alarm with respect to t
Pmd = 1- np.exp(-5*t_range)
Pfa = np.exp(-t_range)

#plot probability of false alarm on x axis and and probability of true positie on y axis
plt.plot(Pfa,1-Pmd)
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel('False Positive Probability')
plt.ylabel('True Positive Probability')
plt.title('ROC Curve')
plt.show()
```



3) (worked with Kevin)

```
#import necessary libraries
from mnist import MNIST
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
import math
mndata = MNIST('Samples')
mndata.gz = True


#load testing and training data from mnist data
testimages, testlabels = mndata.load_testing()
trainimages, trainlabels = mndata.load_training()

#randomly sample about 5% of the training set. In the present result, 2914 samples from
#60000 data were randomly chosen
msk = np.random.rand(len(trainimages)) < 0.05
testimages = np.array(testimages)
trainimages = np.array(trainimages)
testlabels = np.array(testlabels)
trainlabels = np.array(trainlabels)
trainimages = np.array(trainimages[msk])
trainlabels = np.array(trainlabels[msk])

#import algorithms to use
from sklearn.linear_model import LinearRegression,LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.metrics import accuracy_score, zero_one_loss
import time
```

a) Linear Regression

```
#perform one hot encoding on the labels
onehotlabel = np.zeros((trainlabels.shape[0],10))
onehotlabel[np.arange(trainlabels.shape[0]),trainlabels] = 1

#define function that runs linear regression
def linreg():

    #initialize and train through linear regression. Time the time it takes to train
    linearReg = LinearRegression()
    start = time.time()
    linearReg.fit(trainimages,onehotlabel)
    end = time.time()

    #initialize vectors to store predicted labels from the one-hot encoded predictions
    testPred = np.zeros(len(testimages))
    trainPred = np.zeros(len(trainimages))

    #obtain predicted labels from the one-hot encoded predictions
    for i in range(len(testimages)):
        testPred[i] = np.argmax(linearReg.predict([testimages[i]])[0])

    for i in range(len(trainimages)):
        trainPred[i] = np.argmax(linearReg.predict([trainimages[i]])[0])

    lrtime = end-start
    lrtrainerr = 1 - accuracy_score(trainlabels,trainPred)
    lrtesterr = 1 - accuracy_score(testlabels,testPred)

    #return training time, training error, and testing error
    return lrtime, lrtrainerr, lrtesterr

linRegTime = [None] * 10
linRegTrainEr = [None] * 10
linRegTestEr = [None] * 10

#run linear regression 10 times
for x in range(0, 10):
    linRegTime,linRegTrainEr, linRegTestEr = linreg()
```

Result

- The average training time for linear regression: 0.209075927734375 sec
- The average model training error for linear regression: 0.06966369251887439

- The average model testing error for linear regression: 0.1986

B) Linear Discriminant Analysis (LDA)

```
#define function that runs LDA
def lda():
   lda = LinearDiscriminantAnalysis()
   start = time.time()
   lda.fit(trainimages,trainlabels)
   end = time.time()

   #predictions
   testPredLDA = lda.predict(testimages)
   trainPredLDA = lda.predict(trainimages)

   ldatime = end-start
   ldatrainerr = 1 - accuracy_score(trainlabels,trainPredLDA)
   ldatesterr = 1 - accuracy_score(testlabels,testPredLDA)

    #return training time, training error, and testing error
   return ldatime,ldatrainerr,ldatesterr

ldaTime = [None] * 10
ldaTrainEr = [None] * 10
ldaTestEr = [None] * 10

#run LDA 10 times
for x in range(0, 10):
    ldaTime, ldaTrainEr, ldaTestEr = lda()
```

Result

- The average training time for LDA: 0.5831048488616943 sec
- The average model training error for LDA: 0.06280027453671932
- The average model testing error for LDA: 0.1824

C) **Logistic Regression**

```
#function that runs logistic regression
def logreg():

   #using ‘lbfgs’ solver because it results in fastest training time
   logReg = LogisticRegression(solver = 'lbfgs')
   start = time.time()
   logReg.fit(trainimages,trainlabels)
   end = time.time()
```

```
    #predictions
    logTestPred = logReg.predict(testimages)
    logTrainPred = logReg.predict(trainimages)

    lgtime = end-start
    lgtrainerr = 1 - accuracy_score(trainlabels,logTrainPred)
    lgtesterr = 1 - accuracy_score(testlabels,logTestPred)

    #return training time, training error, and testing error
    return lgtime, lgtrainerr, lgtesterr

logRegTime = [None] * 10
logRegTrainEr = [None] * 10
logRegTestEr = [None] * 10

#run logistic regression 10 times
for x in range(0, 10):
      logRegTime, logRegTrainEr, logRegTestEr = logreg()
```

Result

- The average training time for logistic regression: 0.578467845916748 sec
- The average model training error for logistic regression: 0.0
- The average model testing error for logistic regression: 0.129

d) Random Forest

```
#function that runs random forest and returns training time, training error, and testing error
def rf(tree):

    #tree is the number of trees (or baggins) to be used in random forest
    rf = RandomForestClassifier(n_estimators=tree)
    start = time.time()
    rf.fit(trainimages,trainlabels)
    end = time.time()

    testPredRF = rf.predict(testimages)
    trainPredRF = rf.predict(trainimages)

    rftime = end-start
    rftrainerr = 1 - accuracy_score(trainlabels,trainPredRF)
    rftesterr = 1 - accuracy_score(testlabels,testPredRF)

    return rftime,rftrainerr,rftesterr

rfTime = [None] * 10
rfTrainEr = [None] * 10
```

```
rfTestEr = [None] * 10


# 50 trees (baggins) were used
tree = 50

#run random forest 10 times
for x in range(0, 10):
   rfTime, rfTrainEr, rfTestEr = rf(tree)
```

Result

- The average training time for Random Forest: 0.5060811042785645 sec
- The average model training error for Random Forest: 0.0
- The average model testing error for Random Forest: 0.07499999999999996

e) SVM

- **Linear SVM**

```
#function that runs linear SVM and returns training time, training error, and testing error
def lsvc():
   lsvc = LinearSVC()
   start = time.time()
   lsvc.fit(trainimages,trainlabels)
   end = time.time()

   testPredLSVC = lsvc.predict(testimages)
   trainPredLSVC = lsvc.predict(trainimages)

   lsvctime = end-start
   lsvctrainerr = 1 - accuracy_score(trainlabels,trainPredLSVC)
   lsvctesterr = 1 - accuracy_score(testlabels,testPredLSVC)

   return lsvctime,lsvctrainerr,lsvctesterr

lsvcTime = [None] * 10
lsvcTrainEr = [None] * 10
lsvcTestEr = [None] * 10

#run linear SVM 10 times
for x in range(0, 10):
        lsvcTime,lsvcTrainEr,lsvcTestEr = lsvc()
```

Result

- The average training time for linear SVC: 1.1613731384277344
- The average model training error for linear SVC: 0.00034317089910773646
- The average model testing error for linear SVC: 0.17179999999999995

e) SVM

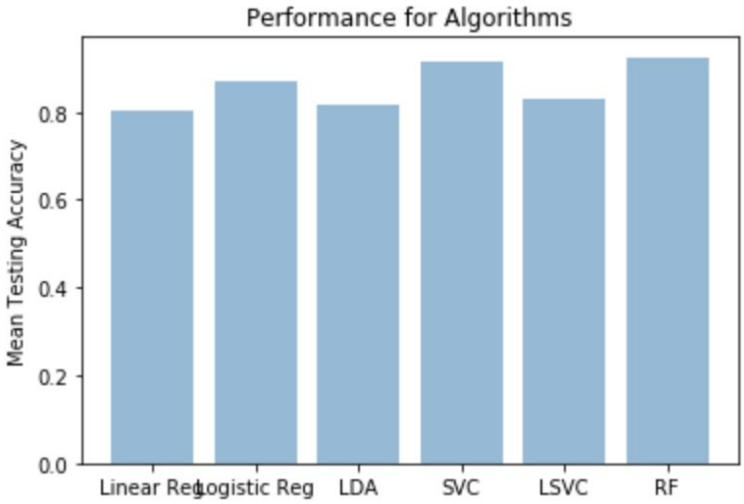
- **SVM with Gaussian radial basis kernel.**

```
def svc(gam):

    #the lower the gamma, the smoother the decision boundary
    svc = SVC(gamma = gam)
    start = time.time()
    svc.fit(trainimages,trainlabels)
    end = time.time()

    testPredSVC = svc.predict(testimages)
    trainPredSVC = svc.predict(trainimages)


    svctime = end-start
    svctrainerr = 1 - accuracy_score(trainlabels,trainPredSVC)
    svctesterr = 1 - accuracy_score(testlabels,testPredSVC)

    return svctime,svctrainerr,svctesterr


svcTime = [None] * 10
svcTrainEr = [None] * 10
svcTestEr = [None] * 10

#only very small gamma can classify correctly, Otherwise, all predictions converge to one
#class (e.g. predicting every image as 1) on the test set.
gamma =  0.000001

#run SVM with radial basis 10 times
for x in range(0, 10):
        svcTime, svcTrainEr, svcTestEr = svc(gamma)
```

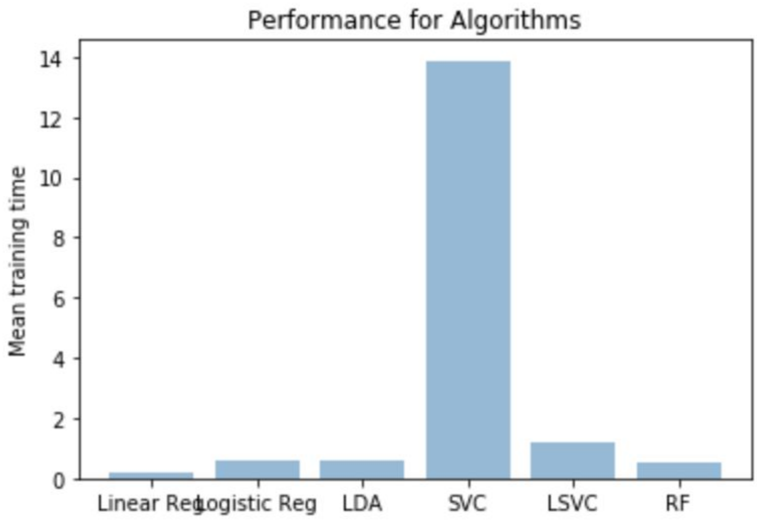
Result

- The average training time for SVC: 13.875026226043701
- The average model training error for SVC: 0.0
- The average model testing error for SVC: 0.08320000000000005

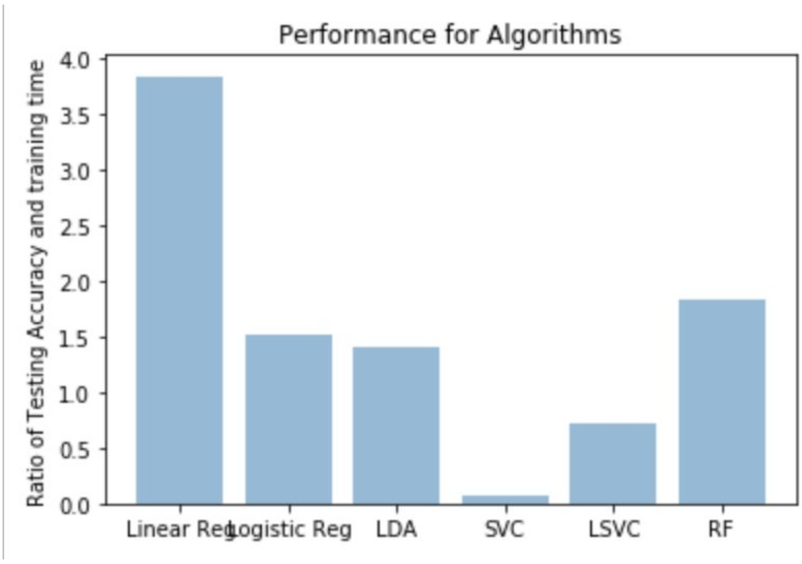
f) **Conclusion**



The above graph shows the average testing accuracy (1 - average testing error) of the algorithms. Random Forest had the highest accuracy, whereas linear regression had the lowest accuracy.



The above graph shows the average training time of the algorithms. SVM with radial basis kernel was significantly slower than all the others, whereas linear regression was fastest.



The above graph shows the ratio of testing accuracy and training time. The higher the ratio, the more "efficient" an algorithm is in terms of taking less time to achieve more accuracy. If we are heavily limited by time and computing powers, then linear regression seems to be the best algorithm to use. However, if we are pursuing only the high accuracy, then the random forest and SVM with radial basis kernel are the best algorithms to use. However, although SVM with radial basis kernel was equally accurate as the random forest, it takes too much time (computing powers) for this task. Thus, SVM with radial basis kernel is not a good algorithm for this task, unless we have unlimited computing power or time. As shown in the ratio graph, random forest is the second most "efficient" algorithm, and random forest can achieve really high accuracy. Thus, random forest is the best algorithm to use for this task in general.