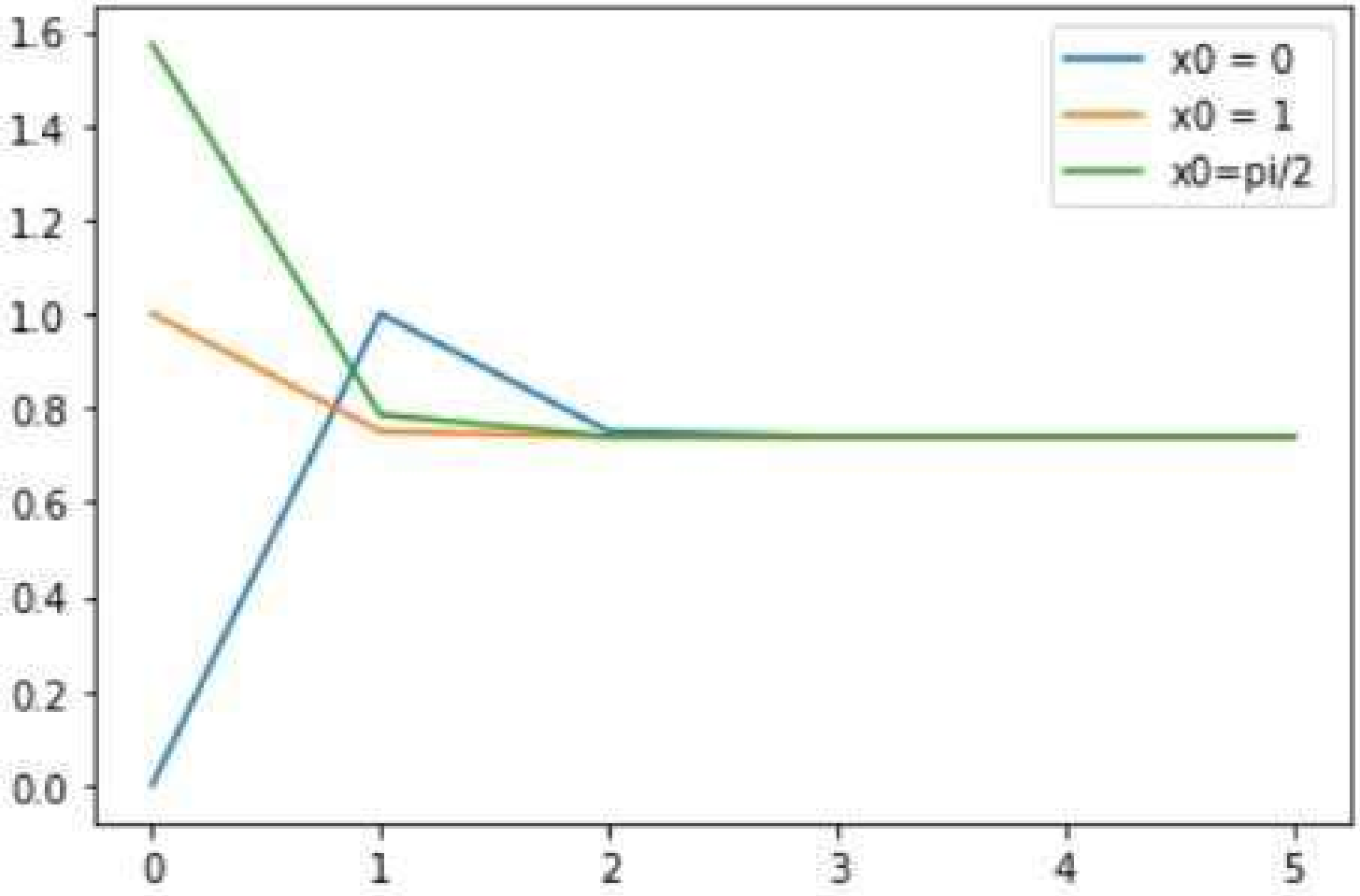
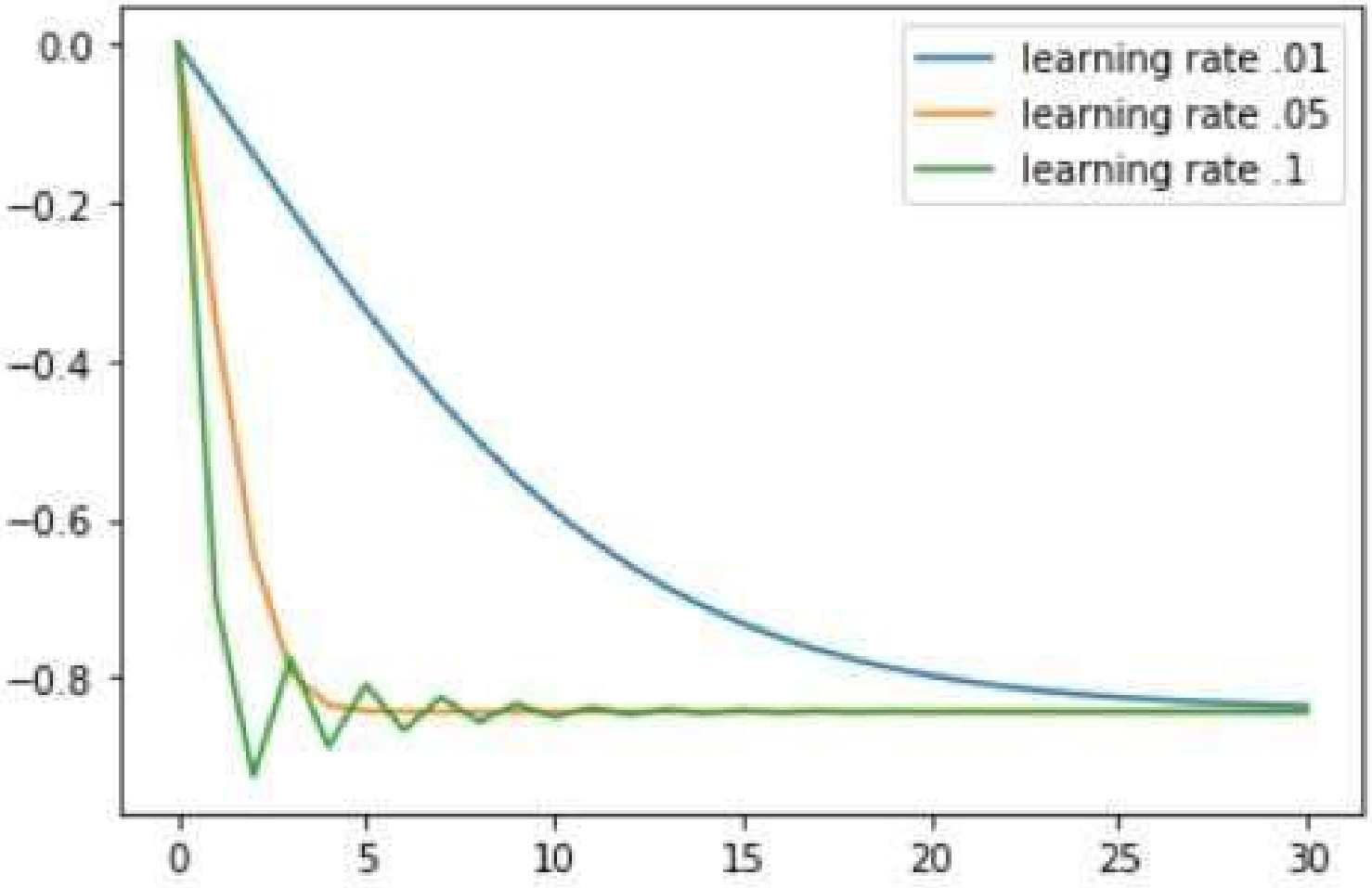
Cory Sweet
Math 251
Homework 4

Q1 Newton's Method



All 3 starting points converge quickly, with x0=1 being the quickest. The closer you start to the root/intersection, the quicker this method converges.

Q2 Gradient Descent



The learning rate of 0.1 gets close to the solution very quickly, but takes longer to converge because it takes larger jumps. A learning rate of 0.05 seems the best learning rate, it approaches and converges quickly. A learning rate of 0.01 is far too slow to approach and converge.

Q3 Logistic Regression

Method	Accuracy	Time (seconds)
One vs. One	0.8441	23.4
One vs. Rest	0.841	162.0
Multinomial	0.8393	65.2
Multiclass LDA	0.8087	3.4

The logistic regression methods are slower than the multiclass LDA, but are better than Multiclass LDA by about 4%.

Q4 L1 Regularized Multinomial Logistic Regression

Method	Accuracy	Time (seconds)
L1 regularized	0.8376	2190.8
One vs. One	0.8441	23.4
One vs. Rest	0.841	162.0
Multinomial	0.8393	65.2

L1 regularized Logistic Regression performs almost as well as the other methods, but is much slower. For this data set, I would prefer one of the other methods from question 3.

```
#Imports and Data

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
from sklearn.metrics import accuracy_score
import sklearn.decomposition
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
from time import time
import math as math

fashion_mnist = keras.datasets.fashion_mnist;
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data();
"""
0       T-shirt/top
1       Trouser
2       Pullover
3       Dress
4       Coat
5       Sandal
6       Shirt
7       Sneaker
8       Bag
9       Ankle boot
"""

label_names=['T-shirt/top','Trouser','Pullover','Dress','Coat','Sandal',
        'Shirt','Sneaker','Bag','Ankle boot']

X_train = np.zeros([60000,784])
for i in range(60000):
   img=train_images[i,:,:]
   X_train[i,:] = img.reshape([784])

X_test = np.zeros([10000,784])
for i in range(10000):
   img=test_images[i,:,:]
   X_test[i,:] = img.reshape([784])

X_sub = X_train[:1000,:]
sub_labels = train_labels[:1000]

#1

def f(x):
```

```
    return (math.cos(x) - x)

def df(x):
    return (-1*math.sin(x) - 1)

def newtons_method(x0):
    xs=[x0]
    for i in range(5):
        xn=xs[-1]
        xs.append(xn - f(xn)/df(xn))
    return(xs)

#xs = newtons_method(math.pi)
plt.plot(newtons_method(0),label = 'x0 = 0')
plt.plot(newtons_method(1),label = 'x0 = 1')
plt.plot(newtons_method(math.pi/2),label = 'x0=pi/2')
plt.legend()

#2

def f(x):
    return (x**4/4 - 3*x**3 + 7*x - 5)

def df(x):
    return (x**3 - 9*x**2 + 7)

def newtons_method(x0):
    xs=[x0]
    for i in range(5):
        xn=xs[-1]
        xs.append(xn - f(xn)/df(xn))
    return(xs)

def grad_descent(x,nu):
    xs = [x]
    for i in range(30):
        xn=xs[-1]
        xs.append(xn - nu*df(xn))
    return(xs)

plt.plot(grad_descent(0,.01),label = 'learning rate .01')
plt.plot(grad_descent(0,.05),label = 'learning rate .05')
plt.plot(grad_descent(0,.1),label = 'learning rate .1')
plt.legend()

#3

col_means = np.mean(X_train, axis = 0)
```

```
X_tilda = X_train - col_means
X_test_centered = X_test - col_means

k=190
PCA = sklearn.decomposition.PCA(n_components = k)
PCA.fit(X_tilda)
Y_train = PCA.transform(X_tilda)
Y_test = PCA.transform(X_test_centered)

start1 = time()
#fits models
for i in range(10):
   for j in range(i+1,10):
      globals()['ovo'+str(i)+str(j)] = LogisticRegression(penalty='none')
      globals()['ovo'+str(i)+str(j)].fit(Y_train[(train_labels==i) | (train_labels==j),],
                                 train_labels[(train_labels==i) | (train_labels==j)]);

#predicts
pred_mat = np.zeros([Y_test.shape[0],45])
col = 0
for i in range(10):
   for j in range(i+1,10):

      pred_mat[:,col] = globals()['ovo'+str(i)+str(j)].predict(Y_test)
      col+=1
pred_mat = pred_mat.astype('int')

#print(pred_mat[0,:])
preds1=np.zeros([pred_mat.shape[0]])
for i in range(pred_mat.shape[0]):

   preds1[i] = np.argmax(np.bincount(pred_mat[i,:]))

score1 = accuracy_score(y_true = test_labels, y_pred = preds1)
score1
end1 = time()

#1v1, 1 vs rest, multinomial
start2 = time()
logreg2 = LogisticRegression(penalty='none', multi_class = 'ovr',max_iter = 500)
logreg2.fit(Y_train, train_labels)
preds2 = logreg2.predict(Y_test)
score2 = accuracy_score(y_true = test_labels, y_pred = preds2)
end2 = time()

start3=time()
logreg3 = LogisticRegression(penalty='none', multi_class = 'multinomial',max_iter=500).fit(Y_train,
train_labels)
```

```
preds3 = logreg3.predict(Y_test)
score3 = accuracy_score(y_true = test_labels, y_pred = preds3)
end3 = time()

start4 = time()
lda = LinearDiscriminantAnalysis()
lda.fit(Y_train,train_labels)
preds4 = lda.predict(Y_test)
score4=accuracy_score(y_true=test_labels, y_pred=preds4)
end4 = time()

print(score1)
print(score2)
print(score3)
print(score4)
print(end1-start1)
print(end2-start2)
print(end3-start3)
print(end4-start4)

#4

start =time()
logreg = LogisticRegression(penalty='l1', multi_class =
'multinomial',max_iter=500,solver='saga').fit(X_train, train_labels)
preds = logreg.predict(X_test)
score = accuracy_score(y_true = test_labels, y_pred = preds)
end = time()

print(score)
print(end-start)
```