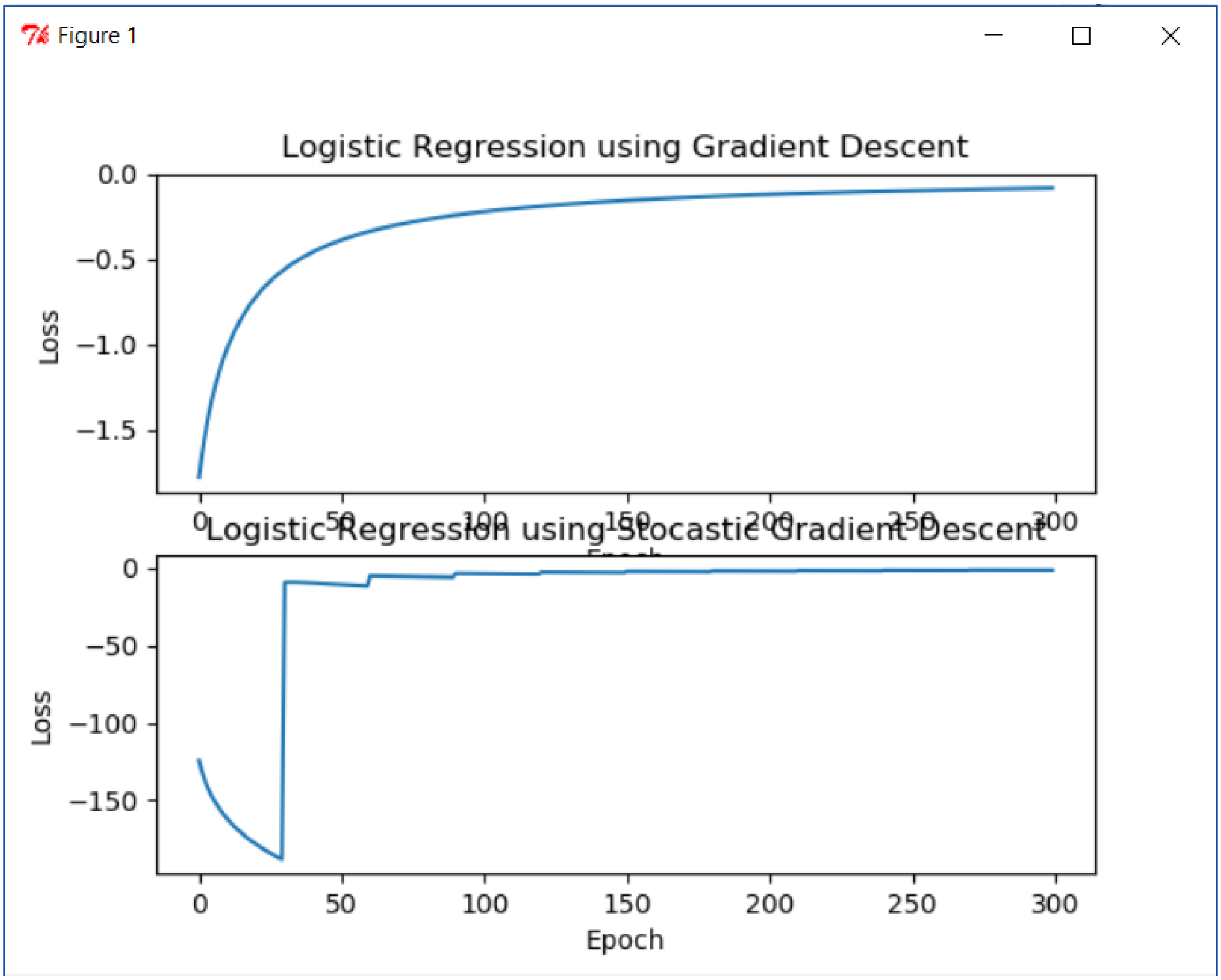
Tanmay Gore
Tgore03@iastate.edu

EE 525 – Assignment 3

03/02/2018

Q1.



Code:

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar  1 12:13:14 2018

@author: tgore03
"""

import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
import random
from sklearn.metrics import log_loss

#Generate Data
mu, sigma = 0.5, 0.3
```

Tanmay Gore
Tgore03@iastate.edu

EE 525 – Assignment 3

03/02/2018

```
s1 = (np.random.randn(100, 2) / 10) + 0.5
s2 = (np.random.randn(100, 2) / 10) - 0.5
y1 = np.ones(100)
y2 = np.zeros(100)
x = np.vstack((s1, s2))
y = np.hstack((y1, y2))

#Train Logistic Regression Model
def sigmoid(scores):
  return 1 / (1 + np.exp(-scores))

def log_likelihood(features, target, weights):
  z = np.dot(features, weights)
  ll = np.sum( target*z - np.log(1 + np.exp(z)) )
  return ll

def logistic_regression(features, target, num_steps=30000, learning_rate=0.001, add_intercept = False):
  #Preprocess data
  if add_intercept:
    intercept = np.ones((features.shape[0], 1))
    features = np.hstack((intercept, features))
  weights = np.zeros(features.shape[1])

  #Initilize variables
  i=0
  loss = [None]*(num_steps)
  epoch = [None]*(num_steps)

  #Train Model
  for step in xrange(num_steps):
    #Predict based on current weights
    z = np.dot(features, weights)
    predictions = sigmoid(z)

    # Update weights with gradient
    output_error_signal = target - predictions
    gradient = np.dot(features.T, output_error_signal)
    weights += learning_rate * gradient

    # Print log-likelihood every so often
    loss[i] = log_likelihood(features, target, weights)
    epoch[i] = i;
    i+=1

  #Plot Loss w.r.t. iteration
  global plt
  plt.subplot(211)
```

```
    plt.plot(epoch, loss)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Logistic Regression using Gradient Descent')
    return weights

def stocastic_logistic_regression(features, target, num_steps=30000, batch_size=10, learning_rate=0.1,
add_intercept=False):
    #Preprocess data
    if add_intercept:
        intercept = np.ones((features.shape[0], 1))
        features = np.hstack((intercept, features))
    weights = np.zeros(features.shape[1])

    #Initilize variables
    i=0
    data_size=len(features[:,0])
    steps_per_epoch = data_size/batch_size
    no_of_epoch = num_steps/steps_per_epoch
    print no_of_epoch
    loss = [None]*(no_of_epoch)
    epoch = [None]*(no_of_epoch)

    #Train Model
    index=0;
    for step in xrange(num_steps):
        x = features[index:index+batch_size]
        y = target[index:index+batch_size]
        index = index+batch_size

        #Predict based on current weights
        z = np.dot(x, weights)
        predictions = sigmoid(z)

        # Update weights with gradient
        output_error_signal = y - predictions
        gradient = np.dot(x.T, output_error_signal)
        weights += learning_rate * gradient

        # Print log-likelihood after each epoch (Obtained by len(features)/batch_size)
        if step % data_size/batch_size == 0:
            loss[i] = log_likelihood(features, target, weights)
            epoch[i] = i;
            i+=1
            index=0

    #Plot Loss w.r.t. iteration
```

Tanmay Gore
Tgore03@iastate.edu

EE 525 – Assignment 3


```
    global plt
    plt.subplot(212)
    plt.plot(epoch, loss)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Logistic Regression using Stocastic Gradient Descent')
    plt.show()
    return weights

#Define figure for plot
global plt
plt.figure(1)

#Train using Gradient Descent
print "Training using Gradient Descent"
weights = logistic_regression(x, y, num_steps = 300, learning_rate = 0.1, add_intercept=True)

#Train using Stocastic Gradient Descent
print "Training using Stocastic Gradient Descent"
weights = stocastic_logistic_regression(x, y, num_steps = 2000, batch_size=30, learning_rate = 0.1, add_intercept=True)
```

Q2)

Q.2)

a) For linear models

$L(\omega) = \frac{1}{2}(y - \langle \omega, x \rangle)^2$

Given the higher dimensional mapping

$x \Rightarrow \phi(x)$

$L(\omega) = \frac{1}{2}(y - \langle \omega, \phi(x) \rangle)^2$

b) $\nabla L(\omega) = 0$

The ~~optimal~~ closed form expression for optimal linear predictor ω is.

$\omega = (X^T X)^{-1} X^T y$

Given ϕ where $\phi(i) = \phi(x_i)$

the closed form expression becomes

$\omega = (\phi^T \phi)^{-1} \phi^T y$

c) Given $f(z) = \langle \omega, \phi(z) \rangle$

from b, we get

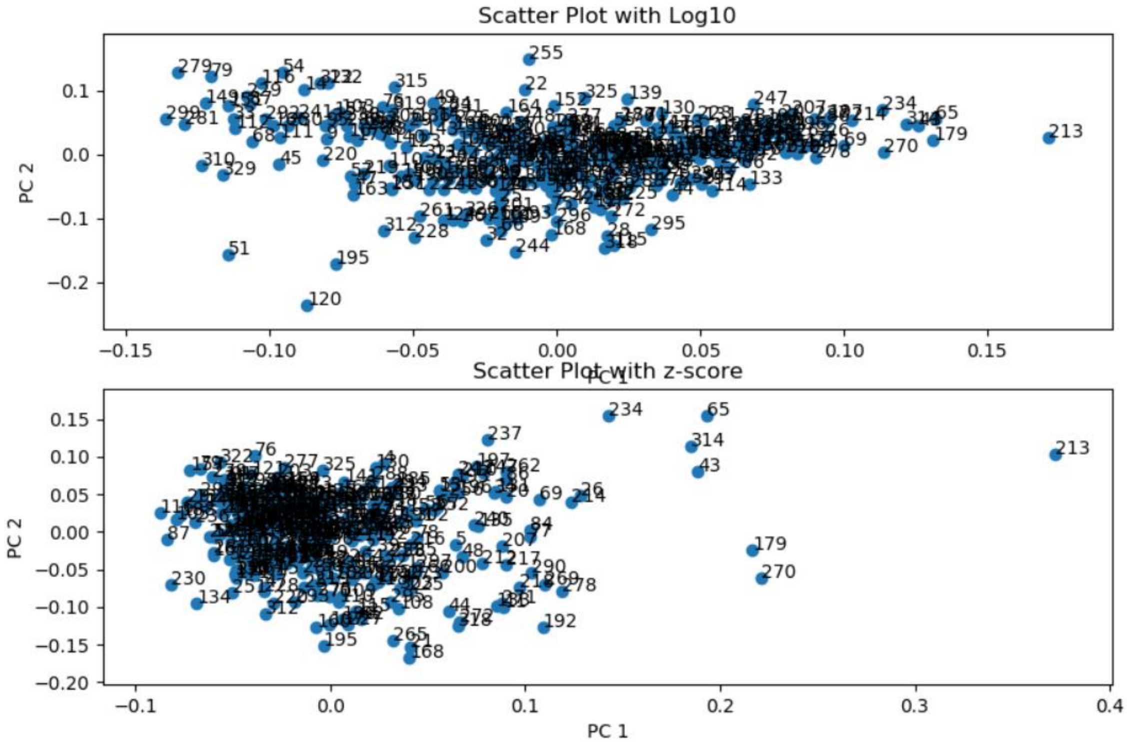
$f(z) = \langle (\phi^T \phi)^{-1} \phi^T y, \phi(z) \rangle$

Tanmay Gore
Tgore03@iastate.edu

EE 525 – Assignment 3

03/02/2018

Q3)



With Log10 Normalization:

Principle Directions:

1. [0.03507288 0.09335159 0.40776448 0.10044536 0.15009714 0.03215319 0.87434057 0.15899622 0.01949418]
2. [0.0088782   0.00923057 -0.85853187 0.22042372 0.05920111 -0.06058858 0.30380632 0.33399255 0.0561011]

The two components appear to correlate most with HealthCare and Arts

Variance of each features is :

[8.40161907 1.85948255 0.68742394 0.83689849 0.61121211 0.40852812 0.26982829 0.13831459 0.06133763]

Outlier Cities:

1. 213 - New-Orleans,LA
2. 120 – Gary-Hammond,IN
3. 51 – Brockton,MA
4. 195 – Middletown,CT

With Z-score Normalization:

Principle Directions:

1. [0.20641395 0.35652161 0.46021465 0.28129838 0.35115078 0.27529264 0.46305449 0.32788791 0.13541225]
2. [0.21783531 0.250624   -0.29946528 0.35534227 -0.17960448 -0.48338209 -0.19478992 0.38447464 0.47128328]

Tanmay Gore
Tgore03@iastate.edu

EE 525 – Assignment 3

03/02/2018

Variance of each feature is:

[11.57517228 43.0323617 41.5675071 33.35398373 27.25678957 22.21711637 18.02308595 11.5894271 4.40125211]

Since the variance of each feature does not vary much projection on 2d causes lot of data to be lost. Hence the 2d plot cannot be trusted to accurately represent the data.

Outlier Cities:

1. 213 – New-Orleans,LA
2. 270 – San-Diego,CA
3. 179 – Lorain-Elyria,OH
4. 43 – Boise-City,ID
5. 314 – Waco,TX
6. 65 – Chattanooga,TN-GA
7. 234 – Peoria,IL

Code:

```
import numpy as np
import math
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.decomposition import TruncatedSVD

f = open("places.txt", "r")
no_features = 9
no_records = 329
features = np.empty([no_records, no_features])
target = ["" for x in range(no_records)]

#Read the labels
f.readline()

#Read the file
lineno=-1
colno=-1
i=0
for line in f:
  lineno+=1
  for word in line.split():
    #Store label column
    if colno == -1:
      colno+=1
      target[lineno]=str(word)
      continue
    colno+=1
    #Skip last 5 columns
    if colno > no_features:
      break;
```

```
        #Store word in matrix
        features[lineno,colno-1] = word
    colno=-1
    i+=1
f.close()

print "PCA using Log10"
#Taking log of matrix
x = np.log10(features)

#Taking mean of features and subtracting it from the features matrix
means = np.mean(x, axis=0)
std = np.std(x, axis=0)

for i in range(no_records):
    x[i]=(x[i] - means)

#SVD
u,s,v = np.linalg.svd(x, full_matrices=True)
d = np.diag(s[0:2])
scores = np.dot(u[:,0:2],d)
print "Principle Directions"
print v[0]
print v[1]

svd = TruncatedSVD(n_components=9)
svd.fit(x.T)
print "Variance of indivial features"
print svd.explained_variance_
print "Total Variance of Principle Components =", svd.explained_variance_ratio_.sum()

#Plot the 2 principle components
components = svd.components_.T

plt.subplot(211)
plt.scatter(components[:,0], components[:,1])
for row in range(no_records):
    plt.annotate(str(row+1), (components[row,0], components[row,1]))
plt.xlabel("PC 1")
plt.ylabel("PC 2")
plt.title("Scatter Plot with Log10")

print "\n\n PCA using z-score"
#using z-scores normalize data
means = np.mean(features, axis=0)
std = np.std(features, axis=0)
```

Tanmay Gore
Tgore03@iastate.edu

EE 525 – Assignment 3


```
for i in range(no_records):
    for j in range(no_features):
        x[i][j]=(features[i][j] - means[j])/std[j]

u,s,v = np.linalg.svd(x, full_matrices=True)
print "Principle Directions:"
print v[0]
print v[1]

svd = TruncatedSVD(n_components=9)
svd.fit(x.T)
components = svd.components_.T
print "Variance of each features: \n",svd.explained_variance_
print "Total variance of Principle components =",svd.explained_variance_ratio_.sum()

plt.subplot(212)
plt.scatter(components[:,0], components[:,1])
for row in range(no_records):
    plt.annotate(str(row+1), (components[row,0], components[row,1]))
plt.xlabel("PC 1")
plt.ylabel("PC 2")
plt.title("Scatter Plot with z-score")
plt.show()
```

Q4)

I spent about 20hrs on this assignment.

References:

Discussed with Nitesh Gupta however I completed my assignment on my own.