Xi'an Jiaotong-Liverpool University
西交利物浦大学

CSE315 Assignment 2 Report

SVM, PCA and Clustering

Author Zhiyong Liu

Module CSE315 - Machine Learning

Date 28^{th} / Dec / 2018

CSE315 Assignment 2

ID: 1509264 Name: Zhiyong Liu

Task1: Classification

1. **Using MNIST database, Implement two classification algorithms and achieve a high accuracy.**

1. softmax

Code:

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf

# Import the data set.
data = input_data.read_data_sets("MNIST_data/",  one_hot=True)

# Training set features.
x = tf.placeholder("float", shape=[None, 784])

# Training set labels.
y_ = tf.placeholder("float", shape=[None, 10])

# Weight value.
W = tf.Variable(tf.zeros([784, 10]))

# Bias.
b = tf.Variable(tf.zeros([10]))

# Build the model.
y = tf.nn.softmax(tf.matmul(x, W) + b)

# Calculate the cost.
cross_entropy = -tf.reduce_sum(y_*tf.log(y))

# Change variables to make cost moves in a decreasing direction.
train_step =
tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

# Start the session.
sess = tf.InteractiveSession()

# Initialize the variables.
```

```
sess.run(tf.initialize_all_variables())

# Training 1000 times.
for i in range(1000):
    batch = data.train.next_batch(50)
    train_step.run(feed_dict={x: batch[0], y_: batch[1]})

# Test whether the predicted result is equal to the actual result.
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))

# Calculate the prediction accuracy.
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))

print("Accuracy:", accuracy.eval(feed_dict={x: data.test.images, y_:
data.test.labels}))
print("Done!")
```

Output:

```
Accuracy: 0.9174
Done!
```

The output shows that the accuracy achieves 91.74%.

2. KNN algorithm

Code:

```
import numpy as np
import tensorflow as tf

# Import MNIST data
from tensorflow.examples.tutorials.mnist import input_data

K = 5
mnist_data = input_data.read_data_sets("/tmp/data/", one_hot=True)

# Training set.
X_training, Y_training = mnist_data.train.next_batch(5500)

# Test set.
X_testing, Y_testing = mnist_data.test.next_batch(1000)

# The training set features.
x_training = tf.placeholder("float", [None, 784])

# The training set labels.
```

```
y_training = tf.placeholder("float", [None, 10])

# The testing set features.
x_testing = tf.placeholder("float", [784])

# Euclidean Distance.
distance =
tf.negative(tf.sqrt(tf.reduce_sum(tf.square(tf.subtract(x_training,
x_testing)), reduction_indices=1)))

# Prediction: Get min distance neighbors.
values, indices = tf.nn.top_k(distance, k=K, sorted=False)

nearest_neighbors = []
for i in range(K):
    nearest_neighbors.append(tf.argmax(y_training[indices[i]], 0))

neighbors_tensor = tf.stack(nearest_neighbors)
y, idx, count = tf.unique_with_counts(neighbors_tensor)
pred = tf.slice(y, begin=[tf.argmax(count, 0)], size=tf.constant([1],
dtype=tf.int64))[0]

accuracy = 0.

# Initializing the variables.
init = tf.initialize_all_variables()

# Launch the graph.
with tf.Session() as sess:
    sess.run(init)

    # loop over test data
    for i in range(len(X_testing)):
        # Get nearest neighbor.
        nn_index = sess.run(pred, feed_dict={x_training: X_training,
y_training: Y_training, x_testing: X_testing[i, :]})
        # Get nearest neighbor class label and compare it to its true
label.
        print("Test", i, "Prediction:", nn_index,
              "True label:", np.argmax(Y_testing[i]))

        # Calculate accuracy.
        if nn_index == np.argmax(Y_testing[i]):
            accuracy += 1. / len(X_testing)

    print("Done!")
    print("Accuracy:", accuracy)
```

Output:

```
Test 0 Prediction: 6 True label: 6
Test 1 Prediction: 5 True label: 5
Test 2 Prediction: 7 True label: 7
Test 3 Prediction: 3 True label: 5
Test 4 Prediction: 8 True label: 8
Test 5 Prediction: 7 True label: 7
......
......
Test 997 Prediction: 8 True label: 2
Test 998 Prediction: 2 True label: 2
Test 999 Prediction: 1 True label: 1
Done!
Accuracy: 0.9420000000000007
```

The output shows that the accuracy achieves 94.2%.

2. **Describe the techniques, including data preparation, feature reduction and training tricks in your classification algorithms.**

To user MNIST data set, I used tensorflow to import MNIST data set as following:

```
from tensorflow.examples.tutorials.mnist import input_data

# Import the data set.
data = input_data.read_data_sets("MNIST_data/",  one_hot=True)
```

In the Softmax, I used the cross entrop as the cost and use gradient descent technique to obtain the optimal W and bias, which is shown as following:

```
# Calculate the cost.
cross_entropy = -tf.reduce_sum(y_*tf.log(y))

# Change variables to make cost moves in a decreasing direction.
train_step =
tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
```

In the KNN algorithm, I used the Euclidean Distance as distance in KNN, which is shown as following:

```
# Euclidean Distance.
distance =
tf.negative(tf.sqrt(tf.reduce_sum(tf.square(tf.subtract(x_training,
x_testing)), reduction_indices=1)))
```

In the KNN algorithm, because the whole set of MNIST data is extremely large, which will cause algorithm operates slow. So in Task1.2-KNN.py, to reduced the size of samples, I used 5500 observations as training set and 1000 observations as testing set. The result reaches an accuracy about 93% and time used is less than 20 seconds.

3. **Analyse some other techniques can be applied in your classification algorithms to improve your model's performance (accuracy, efficiency and storage).**

   One of the classification algorithm used is KNN algorithm. To imporve this, we can weigh the vote of each neighbor by its distance to the observation, which is known as Distance-weighted KNN. In this case, it can break ties when k is even. Otherwise, it can also helps deal with noisy data and outliers.

Task 2 SVM and PCA

Whole code:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA


iris = pd.read_csv("iris.data.txt", header=None)

x = iris.iloc[:, 0:4]
y = iris.iloc[:, 4]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

# Normalization
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)


# Implement SVM algorithm
clf = svm.SVC()
```

```
clf.fit(x_train, y_train)
y_predict = clf.predict(x_test)
print("Accuracy: ", accuracy_score(y_test, y_predict))

# Task 2.2
pca = PCA(n_components=3)
x = pca.fit_transform(x)
x = pd.DataFrame(x)

# split the principle component
x_fpc = x.iloc[:, 0]
x_spc = x.iloc[:, 1]
x_tpc = x.iloc[:, 2]

# print(x_fpc)
# print(x_spc)
# print(x_tpc)


# Task 2.3
x_train, x_test, y_train, y_test = train_test_split(x_fpc, y,
test_size=0.2)
x_train = np.array(x_train).reshape((-1, 1))
x_test = np.array(x_test).reshape((-1, 1))
clf.fit(x_train, y_train)
y_predict = clf.predict(x_test)
print("First principle accuracy: ", accuracy_score(y_test, y_predict))

x_train, x_test, y_train, y_test = train_test_split(x_spc, y,
test_size=0.2)
x_train = np.array(x_train).reshape((-1, 1))
x_test = np.array(x_test).reshape((-1, 1))
clf.fit(x_train, y_train)
y_predict = clf.predict(x_test)
print("Second principle accuracy: ", accuracy_score(y_test, y_predict))

x_train, x_test, y_train, y_test = train_test_split(x_tpc, y,
test_size=0.2)
x_train = np.array(x_train).reshape((-1, 1))
x_test = np.array(x_test).reshape((-1, 1))
clf.fit(x_train, y_train)
y_predict = clf.predict(x_test)
print("Third principle accuracy: ", accuracy_score(y_test, y_predict))


# Task 2.4
print(x)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

```
clf.fit(x_train, y_train)
y_predict = clf.predict(x_test)
print("Three principle components combined accuracy: ",
accuracy_score(y_test, y_predict))
```

1. **Using iris.data, select training dataset and validation dataset and implement SVM algorithm (based on public packages or libraries) to classify the type of iris (achieving 90% accuracy).**

Code:

```
# Normalization.
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)


# Implement SVM algorithm.
clf = svm.SVC()
clf.fit(x_train, y_train)
y_predict = clf.predict(x_test)
print("Accuracy: ", accuracy_score(y_test, y_predict))
```

Related Output:

```
Accuracy:  0.9666666666666667
```

The output shows that the accuracy achieves 96.7%.

2. **Using iris.data, reduce the dimension of features and extract the first, second and third principal component.**

Code:

```
# Task 2.2
pca = PCA(n_components=3)
x = pca.fit_transform(x)
x = pd.DataFrame(x)

# Split and extract the principle component.
x_fpc = x.iloc[:, 0]
x_spc = x.iloc[:, 1]
x_tpc = x.iloc[:, 2]
```

```
print(x_fpc)
print(x_spc)
print(x_tpc)
```

Related output:

```
0     -2.684207
1     -2.715391
2     -2.889820
3     -2.746437
4     -2.728593
5     -2.279897
6     -2.820891
7     -2.626482
8     -2.887959
9     -2.673845
10    -2.506527
11    -2.613143
12    -2.787434
13    -3.225200
14    -2.643543
15    -2.383869
16    -2.622526
17    -2.648323
18    -2.199078
19    -2.587346
20    -2.310532
21    -2.543235
22    -3.215858
23    -2.303129
24    -2.356171
25    -2.507917
26    -2.469056
27    -2.562391
28    -2.639821
29    -2.632848
         ...
120    2.428167
121    1.198097
122    3.499265
123    1.387668
124    2.275854
125    2.614194
126    1.257625
127    1.290670
128    2.122854
129    2.387564
```

```
130    2.840961
131    3.232343
132    2.158738
133    1.443103
134    1.779640
135    3.076522
136    2.144987
137    1.904863
138    1.168853
139    2.107654
140    2.314303
141    1.922451
142    1.414072
143    2.563323
144    2.419391
145    1.944017
146    1.525664
147    1.764046
148    1.901629
149    1.389666
Name: 0, Length: 150, dtype: float64
0      0.326607
1     -0.169557
2     -0.137346
3     -0.311124
4      0.333925
5      0.747783
6     -0.082105
7      0.170405
8     -0.570798
9     -0.106692
10     0.651935
11     0.021521
12    -0.227740
13    -0.503280
14     1.186195
15     1.344754
16     0.818090
17     0.319137
18     0.879244
19     0.520474
20     0.397868
21     0.440032
22     0.141616
23     0.105523
24    -0.031210
25    -0.139056
26     0.137887
27     0.374685
```

```
28     0.319290
29    -0.190076
          ...
120    0.376782
121   -0.605579
122    0.456773
123   -0.204031
124    0.333387
125    0.558367
126   -0.179137
127   -0.116425
128   -0.210855
129    0.462519
130    0.372743
131    1.370524
132   -0.218326
133   -0.143801
134   -0.501465
135    0.685764
136    0.138907
137    0.048048
138   -0.164502
139    0.371482
140    0.182609
141    0.409271
142   -0.574925
143    0.275975
144    0.303504
145    0.187415
146   -0.375021
147    0.078519
148    0.115877
149   -0.282887
Name: 1, Length: 150, dtype: float64
0     -0.021512
1     -0.203521
2      0.024709
3      0.037672
4      0.096230
5      0.174326
6      0.264251
7     -0.015802
8      0.027335
9     -0.191533
10    -0.069275
11     0.107650
12    -0.200328
13     0.068414
14    -0.144506
```

```
15     0.283731
16     0.145316
17     0.033394
18    -0.114521
19     0.219572
20    -0.233696
21     0.214836
22     0.299619
23     0.045680
24     0.129408
25    -0.247116
26     0.101263
27    -0.072359
28    -0.139253
29     0.046466
         ...
120    0.218649
121    0.512641
122   -0.576910
123   -0.063511
124    0.284678
125   -0.208423
126    0.046978
127    0.231614
128    0.153516
129   -0.452024
130   -0.501032
131   -0.118449
132    0.208422
133   -0.154083
134   -0.175811
135   -0.336423
136    0.734185
137    0.160471
138    0.282461
139    0.027438
140    0.322860
141    0.115493
142    0.296398
143    0.291254
144    0.504303
145    0.179303
146   -0.120636
147    0.130784
148    0.722874
149    0.362318
Name: 2, Length: 150, dtype: float64
```

3. **Based on question 2, using the first, second and third principal component, respectively, to train a SVM model to classify the type of iris and compare their accuracy.**

Code:

```
# Task 2.3
x_train, x_test, y_train, y_test = train_test_split(x_fpc, y,
test_size=0.2)
x_train = np.array(x_train).reshape((-1, 1))
x_test = np.array(x_test).reshape((-1, 1))
clf.fit(x_train, y_train)
y_predict = clf.predict(x_test)
print("First principle accuracy: ", accuracy_score(y_test, y_predict))

x_train, x_test, y_train, y_test = train_test_split(x_spc, y,
test_size=0.2)
x_train = np.array(x_train).reshape((-1, 1))
x_test = np.array(x_test).reshape((-1, 1))
clf.fit(x_train, y_train)
y_predict = clf.predict(x_test)
print("Second principle accuracy: ", accuracy_score(y_test, y_predict))

x_train, x_test, y_train, y_test = train_test_split(x_tpc, y,
test_size=0.2)
x_train = np.array(x_train).reshape((-1, 1))
x_test = np.array(x_test).reshape((-1, 1))
clf.fit(x_train, y_train)
y_predict = clf.predict(x_test)
print("Third principle accuracy: ", accuracy_score(y_test, y_predict))
```

Related output:

```
First principle accuracy:  0.9666666666666667
Second principle accuracy:  0.36666666666666664
Third principle accuracy:  0.3
```

4. **Based on question 2,using any combination of the first, second and third principal components to train a SVM model to classify the type of iris and compare their accuracy.**

Code:

```
# Task 2.4
print(x)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
clf.fit(x_train, y_train)
y_predict = clf.predict(x_test)
print("Three principle components combined accuracy: ",
accuracy_score(y_test, y_predict))
```

Related output

```
            0         1         2
0   -2.684207  0.326607 -0.021512
1   -2.715391 -0.169557 -0.203521
2   -2.889820 -0.137346  0.024709
3   -2.746437 -0.311124  0.037672
4   -2.728593  0.333925  0.096230
5   -2.279897  0.747783  0.174326
6   -2.820891 -0.082105  0.264251
7   -2.626482  0.170405 -0.015802
8   -2.887959 -0.570798  0.027335
9   -2.673845 -0.106692 -0.191533
10  -2.506527  0.651935 -0.069275
11  -2.613143  0.021521  0.107650
12  -2.787434 -0.227740 -0.200328
13  -3.225200 -0.503280  0.068414
14  -2.643543  1.186195 -0.144506
15  -2.383869  1.344754  0.283731
16  -2.622526  0.818090  0.145316
17  -2.648323  0.319137  0.033394
18  -2.199078  0.879244 -0.114521
19  -2.587346  0.520474  0.219572
20  -2.310532  0.397868 -0.233696
21  -2.543235  0.440032  0.214836
22  -3.215858  0.141616  0.299619
23  -2.303129  0.105523  0.045680
24  -2.356171 -0.031210  0.129408
25  -2.507917 -0.139056 -0.247116
26  -2.469056  0.137887  0.101263
27  -2.562391  0.374685 -0.072359
28  -2.639821  0.319290 -0.139253
29  -2.632848 -0.190076  0.046466
..        ...       ...       ...
120  2.428167  0.376782  0.218649
121  1.198097 -0.605579  0.512641
122  3.499265  0.456773 -0.576910
123  1.387668 -0.204031 -0.063511
124  2.275854  0.333387  0.284678
```

```
125  2.614194  0.558367 -0.208423
126  1.257625 -0.179137  0.046978
127  1.290670 -0.116425  0.231614
128  2.122854 -0.210855  0.153516
129  2.387564  0.462519 -0.452024
130  2.840961  0.372743 -0.501032
131  3.232343  1.370524 -0.118449
132  2.158738 -0.218326  0.208422
133  1.443103 -0.143801 -0.154083
134  1.779640 -0.501465 -0.175811
135  3.076522  0.685764 -0.336423
136  2.144987  0.138907  0.734185
137  1.904863  0.048048  0.160471
138  1.168853 -0.164502  0.282461
139  2.107654  0.371482  0.027438
140  2.314303  0.182609  0.322860
141  1.922451  0.409271  0.115493
142  1.414072 -0.574925  0.296398
143  2.563323  0.275975  0.291254
144  2.419391  0.303504  0.504303
145  1.944017  0.187415  0.179303
146  1.525664 -0.375021 -0.120636
147  1.764046  0.078519  0.130784
148  1.901629  0.115877  0.722874
149  1.389666 -0.282887  0.362318

[150 rows x 3 columns]
Three principle components combined accuracy:  1.0
```

Task 3 Clustering

Whole code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

iris = pd.read_csv("iris.data.txt", header=None)
x = np.array(iris.iloc[:, 0:4])
y = np.array(iris.iloc[:, 4])

for i in range(1, 7):
    print("\n<--- K means with", i, "clusters --->")
```

```
    k_means = KMeans(n_clusters=i)
    k_means.fit(x)
    # Final centroids
    print("Final centroids:\n", k_means.cluster_centers_)
    # data point labels
    print("Data point labels:\n", k_means.labels_)

print("\n-----------------------------------------------\n"
      "Use PCA to reduce the dimension of features and "
      "combine the first, second and "
      "third principal components to implement k-means algorithm:\n")

pca = PCA(n_components=3)
x = pca.fit_transform(x)

for i in range(1, 7):
    print("\n<--- K means with", i, "clusters --->")
    k_means = KMeans(n_clusters=i)
    k_means.fit(x)
    # Final centroids
    print("Final centroids:\n", k_means.cluster_centers_, "\n")
    # data point labels
    print("Data point labels:\n", k_means.labels_, "\n")

    # Draw the scatter plot
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.scatter(x[:, 0], x[:, 1], x[:, 2], c=k_means.labels_,
cmap='rainbow')

    ax.set_xlabel('first principle component')
    ax.set_ylabel('second principle component')
    ax.set_zlabel('third principle component')
    ax.view_init(elev=10, azim=235)
    plt.show()
```

1. **Using iris.data, select the number (1, 2, 3, 4, 5 and 6) of clustering and implement k-means algorithm (based on public packages or libraries).**

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
```

```python
from sklearn.decomposition import PCA

iris = pd.read_csv("iris.data.txt", header=None)
x = np.array(iris.iloc[:, 0:4])
y = np.array(iris.iloc[:, 4])

for i in range(1, 7):
    print("\n<--- K means with", i, "clusters --->")
    k_means = KMeans(n_clusters=i)
    k_means.fit(x)
    # Final centroids
    print("Final centroids:\n", k_means.cluster_centers_)
    # data point labels
    print("Data point labels:\n", k_means.labels_)
```

Related output:

```
<--- K means with 1 clusters --->
Final centroids:
 [[5.84333333 3.054      3.75866667 1.19866667]]
Data point labels:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]

<--- K means with 2 clusters --->
Final centroids:
 [[6.30103093 2.88659794 4.95876289 1.69587629]
 [5.00566038 3.36037736 1.56226415 0.28867925]]
Data point labels:
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]

<--- K means with 3 clusters --->
Final centroids:
 [[6.85       3.07368421 5.74210526 2.07105263]
 [5.006      3.418      1.464      0.244     ]
 [5.9016129  2.7483871  4.39354839 1.43387097]]
Data point labels:
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 0 2 0 0 0 0
 0 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0
 0 2]
```

```
<--- K means with 4 clusters --->
Final centroids:
 [[5.52962963 2.62222222 3.94074074 1.21851852]
 [5.006      3.418      1.464      0.244     ]
 [6.9125     3.1        5.846875   2.13125   ]
 [6.23658537 2.85853659 4.80731707 1.62195122]]
Data point labels:
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 0 3 0 3 0 3 0 0 0 0 3 0 3 3 0 3 0 3 0 3 3
 3 3 3 3 3 0 0 0 0 3 0 3 3 3 0 0 0 3 0 0 0 0 0 3 0 0 2 3 2 2 2 2 0 2 2 2 3
 3 2 3 3 2 2 2 2 3 2 3 2 3 2 2 3 3 2 2 2 2 2 3 3 2 2 2 3 2 2 2 3 2 2 2 3 3
 2 3]

<--- K means with 5 clusters --->
Final centroids:
 [[6.20769231 2.85384615 4.74615385 1.56410256]
 [5.006      3.418      1.464      0.244     ]
 [7.475      3.125      6.3        2.05      ]
 [6.52916667 3.05833333 5.50833333 2.1625    ]
 [5.508      2.6        3.908      1.204     ]]
Data point labels:
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 4 0 0 0 4 0 4 4 0 4 0 4 0 0 4 0 4 0 4 0 0
 0 0 0 0 0 4 4 4 4 0 4 0 0 0 4 4 4 0 4 4 4 4 4 0 4 4 3 0 2 3 3 2 4 2 3 2 3
 3 3 0 3 3 3 2 2 0 3 0 2 0 3 2 0 0 3 2 2 2 3 0 0 2 3 3 0 3 3 3 0 3 3 3 0 3
 3 0]

<--- K means with 6 clusters --->
Final centroids:
 [[5.25555556 3.67037037 1.5037037  0.28888889]
 [6.24722222 2.84722222 4.775      1.575     ]
 [7.475      3.125      6.3        2.05      ]
 [6.52916667 3.05833333 5.50833333 2.1625    ]
 [5.53214286 2.63571429 3.96071429 1.22857143]
 [4.71304348 3.12173913 1.4173913  0.19130435]]
Data point labels:
 [0 5 5 5 0 0 5 0 5 5 0 5 5 5 0 0 0 0 0 0 0 0 5 0 5 5 0 0 0 5 5 0 0 0 5 5 0
 5 5 0 0 5 5 0 0 5 0 5 0 5 1 1 1 4 1 4 1 4 1 4 4 4 4 1 4 1 4 4 1 4 1 4 1 1
 1 1 1 1 1 4 4 4 4 1 4 1 1 1 4 4 4 1 4 4 4 4 4 1 4 4 3 1 2 3 3 2 4 2 3 2 3
 3 3 1 3 3 3 2 2 1 3 1 2 1 3 2 1 1 3 2 2 2 3 1 1 2 3 3 1 3 3 3 1 3 3 3 1 3
 3 1]
```

2. **Using PCA to reduce the dimension of features and combine the first, second and third principal components to implement k-means algorithm (based on public packages or libraries).**

```
print("\n-----------------------------------------------\n"
      "Use PCA to reduce the dimension of features and "
      "combine the first, second and "
      "third principal components to implement k-means algorithm:\n")

pca = PCA(n_components=3)
x = pca.fit_transform(x)

for i in range(1, 7):
    print("\n<--- K means with", i, "clusters --->")
    k_means = KMeans(n_clusters=i)
    k_means.fit(x)
    # Final centroids
    print("Final centroids:\n", k_means.cluster_centers_, "\n")
    # data point labels
    print("Data point labels:\n", k_means.labels_, "\n")

    # Draw the scatter plot
    fig = plt.figure()
    ax = Axes3D(fig)
    ax.scatter(x[:, 0], x[:, 1], x[:, 2], c=k_means.labels_,
cmap='rainbow')

    ax.set_xlabel('first principle component')
    ax.set_ylabel('second principle component')
    ax.set_zlabel('third principle component')
    ax.view_init(elev=10, azim=235)
    plt.show()
```

Related output:

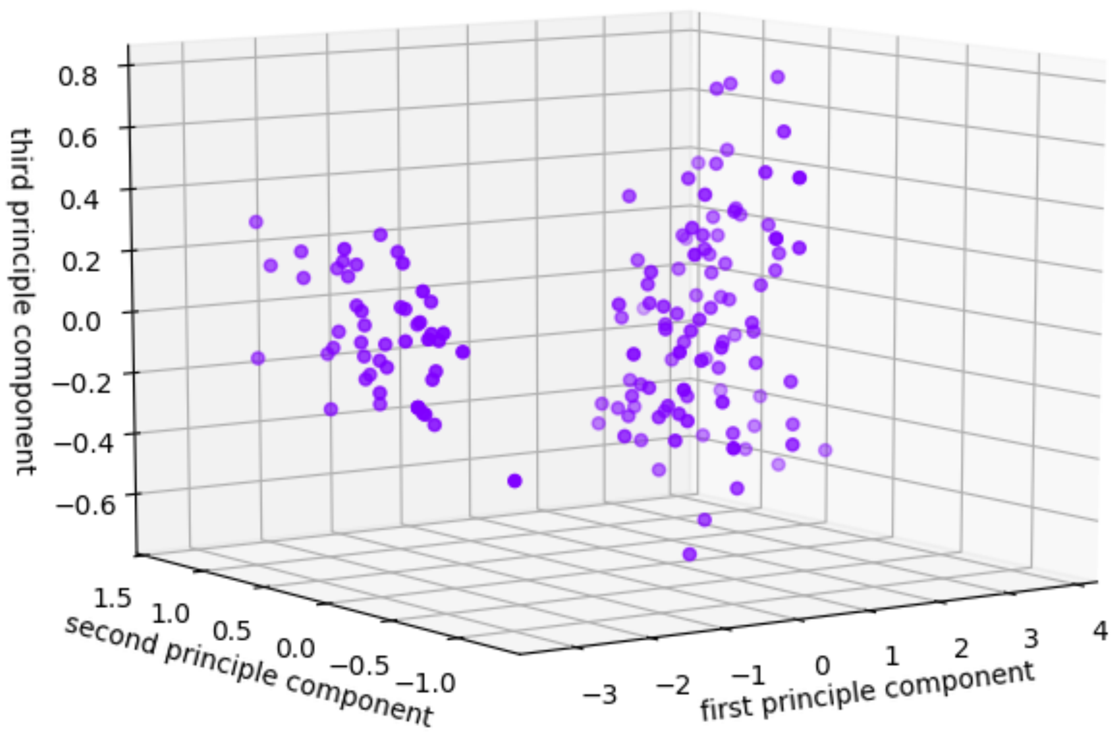
```
-----------------------------------------------
Use PCA to reduce the dimension of features and combine the first, second
and third principal components to implement k-means algorithm:



<--- K means with 1 clusters --->
Final centroids:
 [[ 6.63173220e-16 -5.41418762e-16  3.29366164e-17]]

Data point labels:
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]

<--- K means with 2 clusters --->
```

```
Final centroids:
 [[ 1.38566031 -0.0697412  -0.00572844]
 [-2.53601981  0.12763956  0.01048412]]

Data point labels:
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]

<--- K means with 3 clusters --->
Final centroids:
 [[ 2.37438946  0.2614839   0.04970951]
 [-2.64084076  0.19051995  0.01299584]
 [ 0.67443933 -0.31390945 -0.04094764]]

Data point labels:
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 0 0 0 0 2 0 0 0 0
 0 0 2 2 0 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 0 2 0 0 0 0 2 0 0 0 2 0 0 0 2 0
 0 2]

<--- K means with 4 clusters --->
Final centroids:
 [[ 1.22213475 -0.09409735 -0.04571935]
 [-2.64084076  0.19051995  0.01299584]
 [ 2.50616813  0.29880836  0.06974274]
 [ 0.10568813 -0.54728468 -0.03759949]]

Data point labels:
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 3 0 3 0 3 0 3 3 3 3 0 3 0 3 3 0 3 0 3 0 0
 0 0 0 0 0 3 3 3 3 0 3 0 0 0 3 3 3 0 3 3 3 3 3 0 3 3 2 0 2 2 2 2 3 2 2 2 0
 0 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 0 2 2 2 0 2 2 2 0 2 2 2 0 0
 2 0]

<--- K means with 5 clusters --->
Final centroids:
 [[ 2.07971215  0.05088548  0.24277236]
 [-2.64084076  0.19051995  0.01299584]
 [ 1.10798128 -0.09734498 -0.05572464]
 [ 3.06648377  0.61278015 -0.2539065 ]
 [ 0.04592561 -0.57809549 -0.06218747]]

Data point labels:
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 4 2 2 2 4 2 4 4 2 4 2 4 2 2 4 2 4 2 4 2 2
```

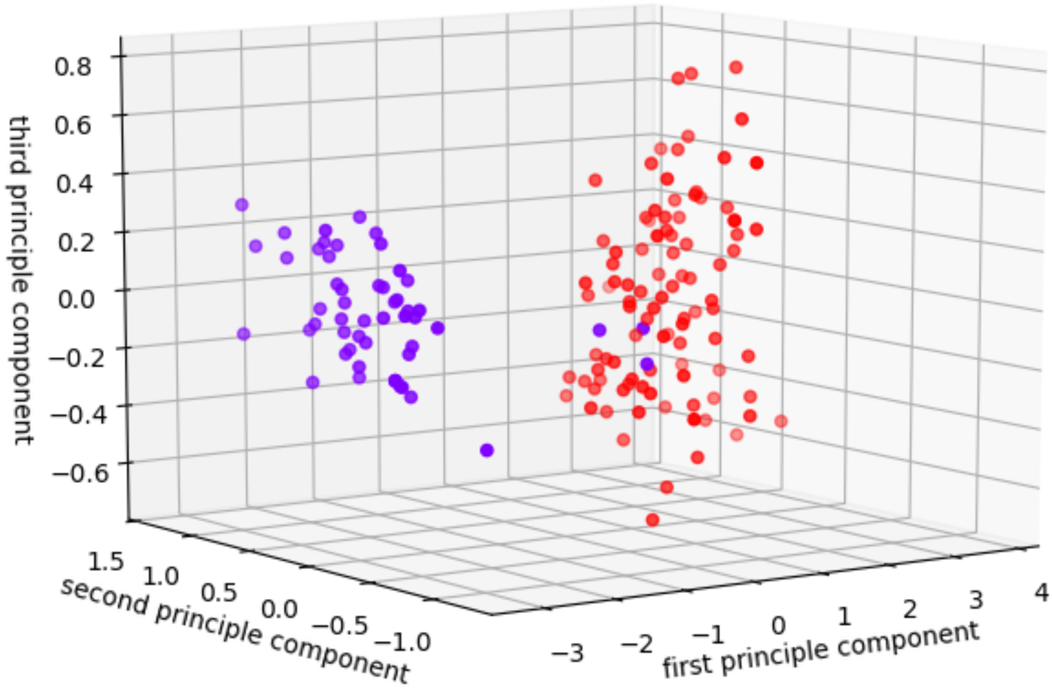
```
 2 2 2 2 2 4 4 4 4 2 4 2 2 2 4 4 4 2 4 4 4 4 4 2 4 4 0 2 3 0 0 3 4 3 0 3 0
 0 0 2 0 0 0 3 3 2 0 2 3 2 0 3 2 2 0 3 3 3 0 2 0 3 0 0 2 0 0 0 2 0 0 0 2 0
 0 2]

<--- K means with 6 clusters --->
Final centroids:
 [[-2.78123098 -0.20587391 -0.02580593]
 [ 2.07971215  0.05088548  0.24277236]
 [ 0.04592561 -0.57809549 -0.06218747]
 [-2.52124909  0.5281888   0.04604921]
 [ 1.10798128 -0.09734498 -0.05572464]
 [ 3.06648377  0.61278015 -0.2539065 ]]

Data point labels:
 [3 0 0 0 3 3 0 3 0 0 3 0 0 0 3 3 3 3 3 3 3 3 0 3 0 0 3 3 3 0 0 3 3 3 0 0 3
 0 0 3 3 0 0 3 3 0 3 0 3 0 4 4 4 2 4 4 4 2 4 2 2 4 2 4 2 4 4 2 4 2 4 2 4 4
 4 4 4 4 4 2 2 2 2 4 2 4 4 4 2 2 2 4 2 2 2 2 2 4 2 2 1 4 5 1 1 5 2 5 1 5 1
 1 1 4 1 1 1 5 5 4 1 4 5 4 1 5 4 4 1 5 5 5 1 4 1 5 1 1 4 1 1 1 4 1 1 1 4 1
 1 4]
```

The generated scatter plot with clusters from 1 to 6:

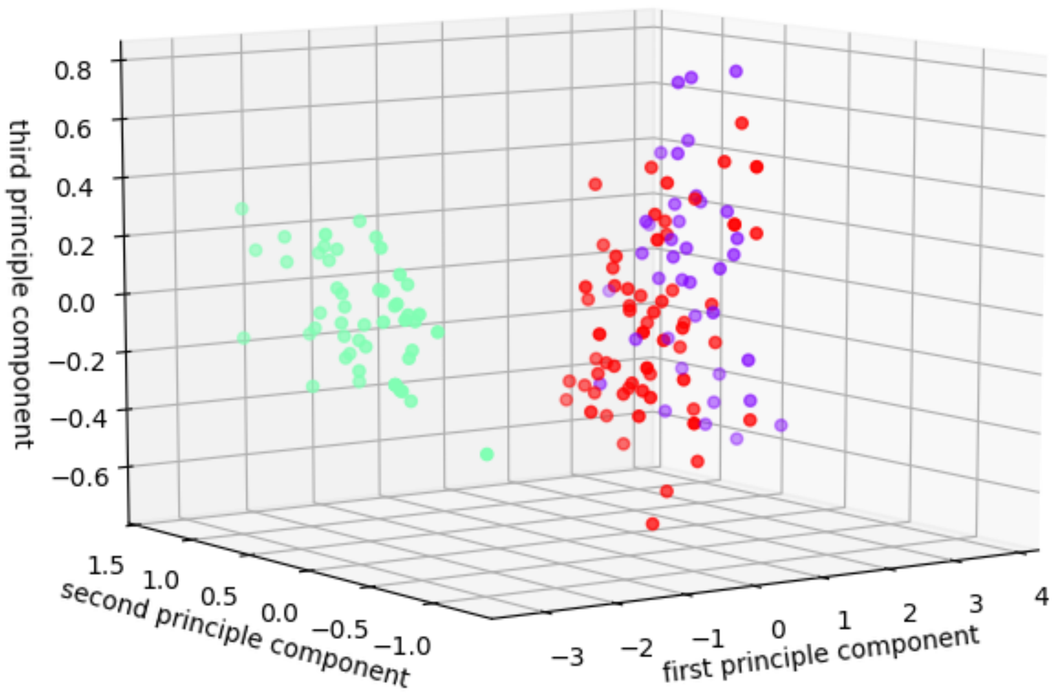
1. K-means with 1 cluster:



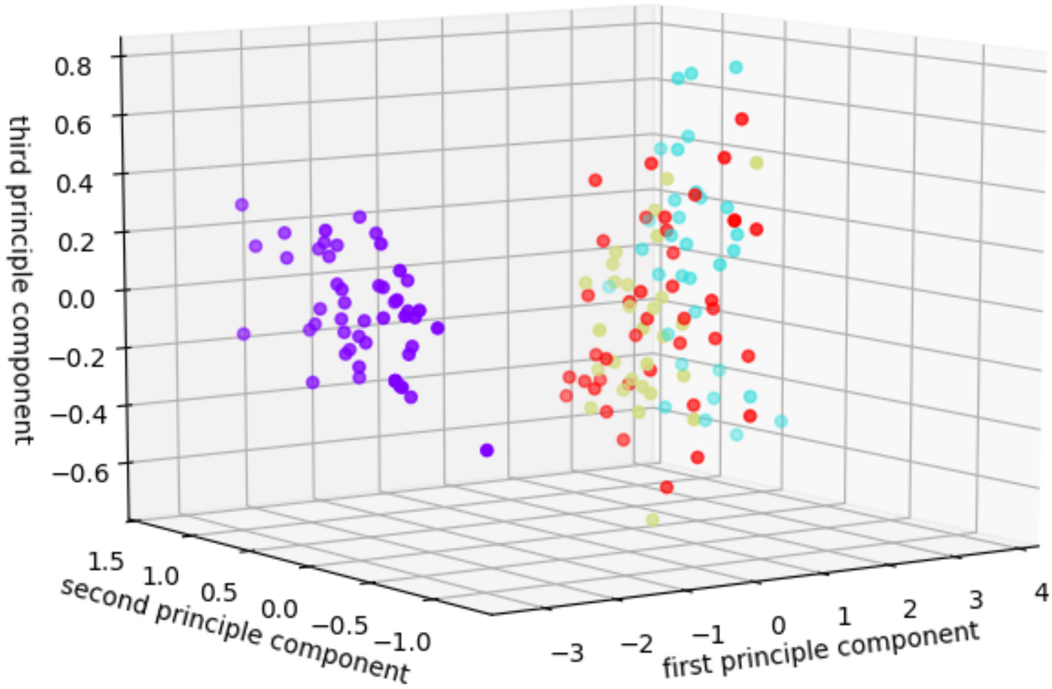
2. K-means with 2 clusters:



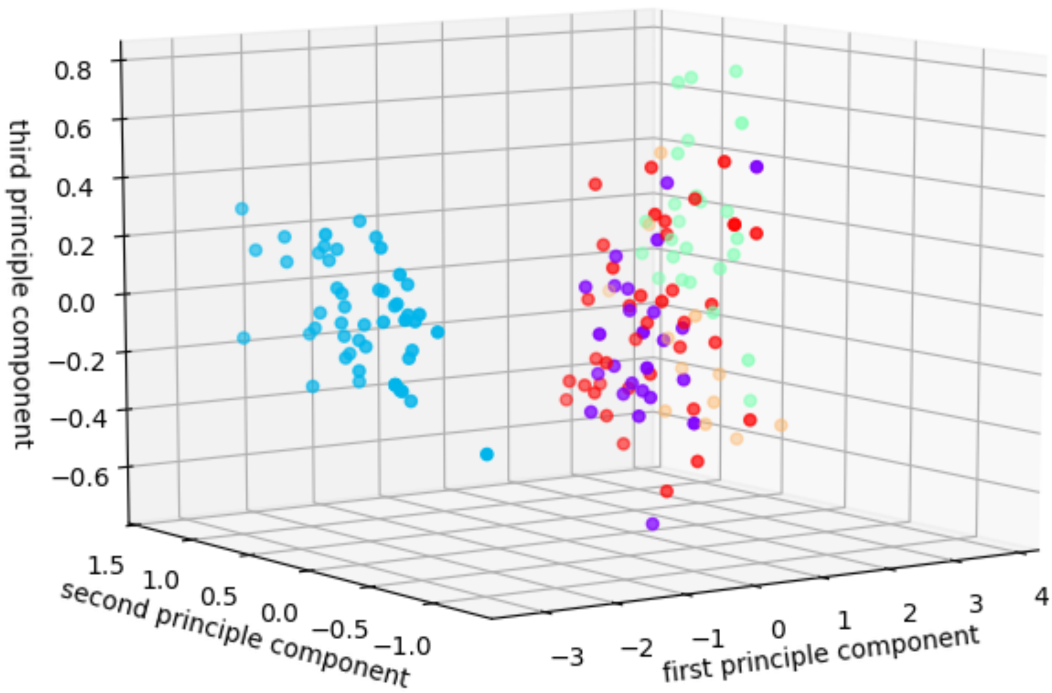
3. K-means with 3 clusters:



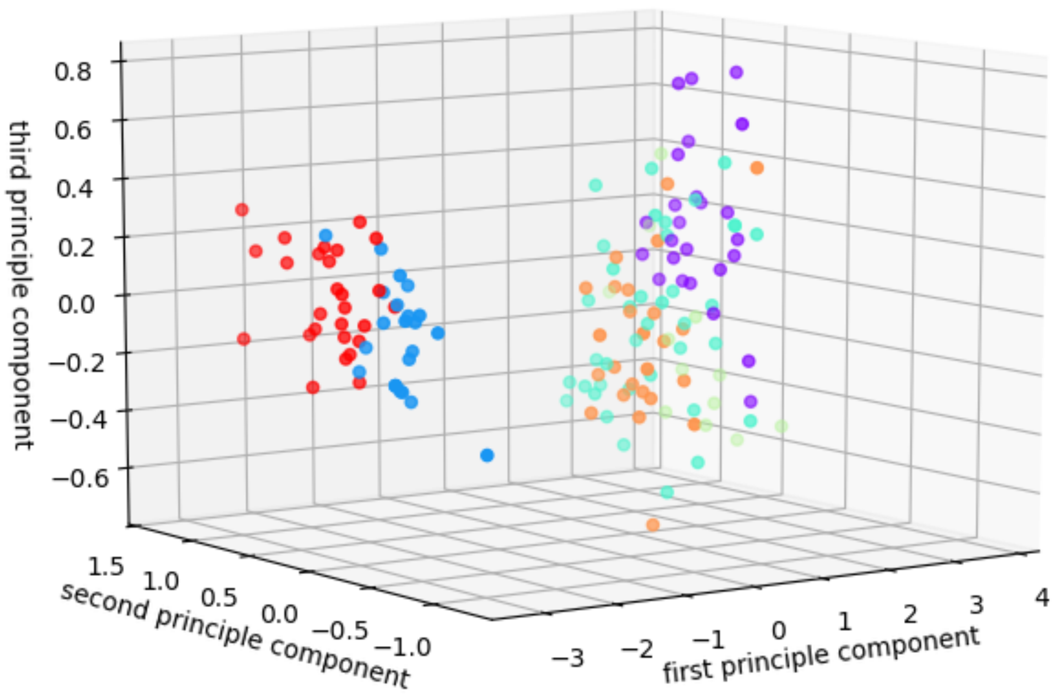
4. K-means with 4 clusters:



5. K-means with 5 clusters:



6. K-means with 6 clusters:



Task 4 The Application of Machine Learning

1. **Finish the Coursera Programming Exercises 8 (Anomaly Detection and Recommender Systems).**

1. estimateGaussian.m

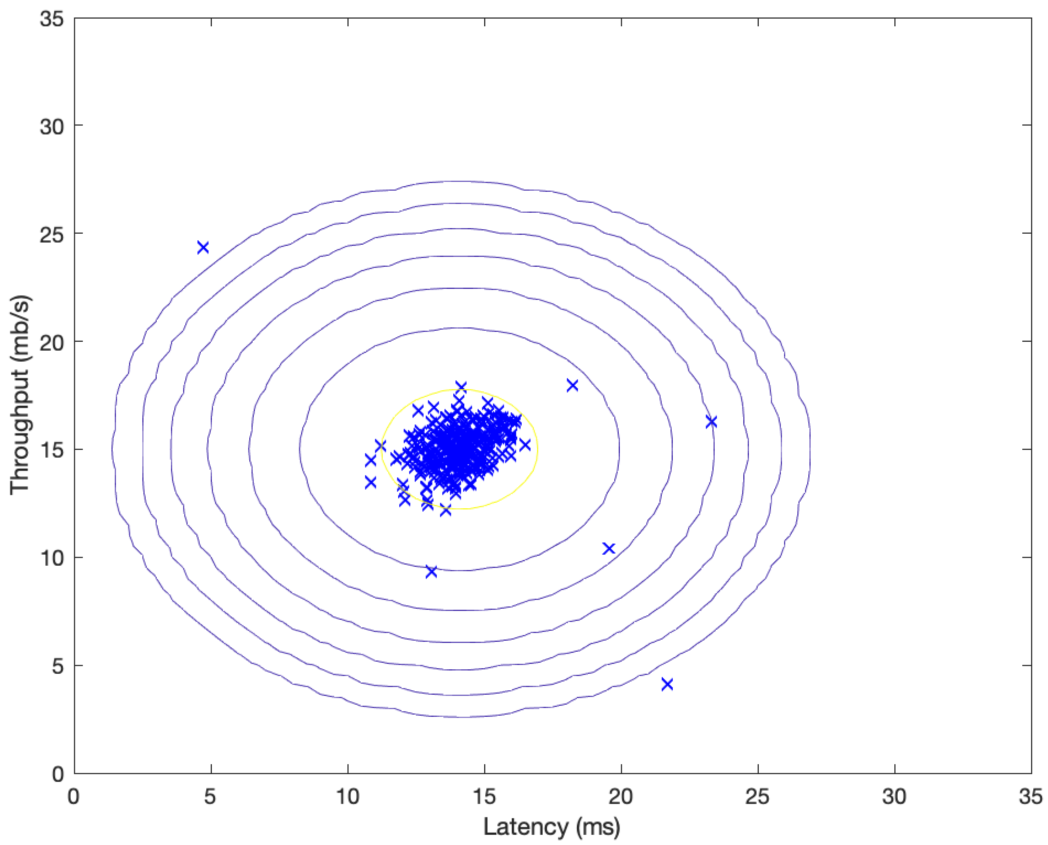
```
function [mu sigma2] = estimateGaussian(X)
%ESTIMATEGAUSSIAN This function estimates the parameters of a
%Gaussian distribution using the data in X
%   [mu sigma2] = estimateGaussian(X),
%   The input X is the dataset with each n-dimensional data point in one
row
%   The output is an n-dimensional vector mu, the mean of the data set
%   and the variances sigma^2, an n x 1 vector
%

% Useful variables
% [m, n] = size(X);

% You should return these values correctly
% mu = zeros(n, 1);
% sigma2 = zeros(n, 1);

% ====================== YOUR CODE HERE ======================
```

```
% Instructions: Compute the mean of the data and the variances
%               In particular, mu(i) should contain the mean of
%               the data for the i-th feature and sigma2(i)
%               should contain variance of the i-th feature.
%
a = mean(X);
mu = a';
b = std(X,1,1);
b = b.^2;
sigma2 = b';

% =============================================================


end
```

Related output:

```
Program paused. Press enter to continue.
Visualizing Gaussian fit.
```

Related plot:



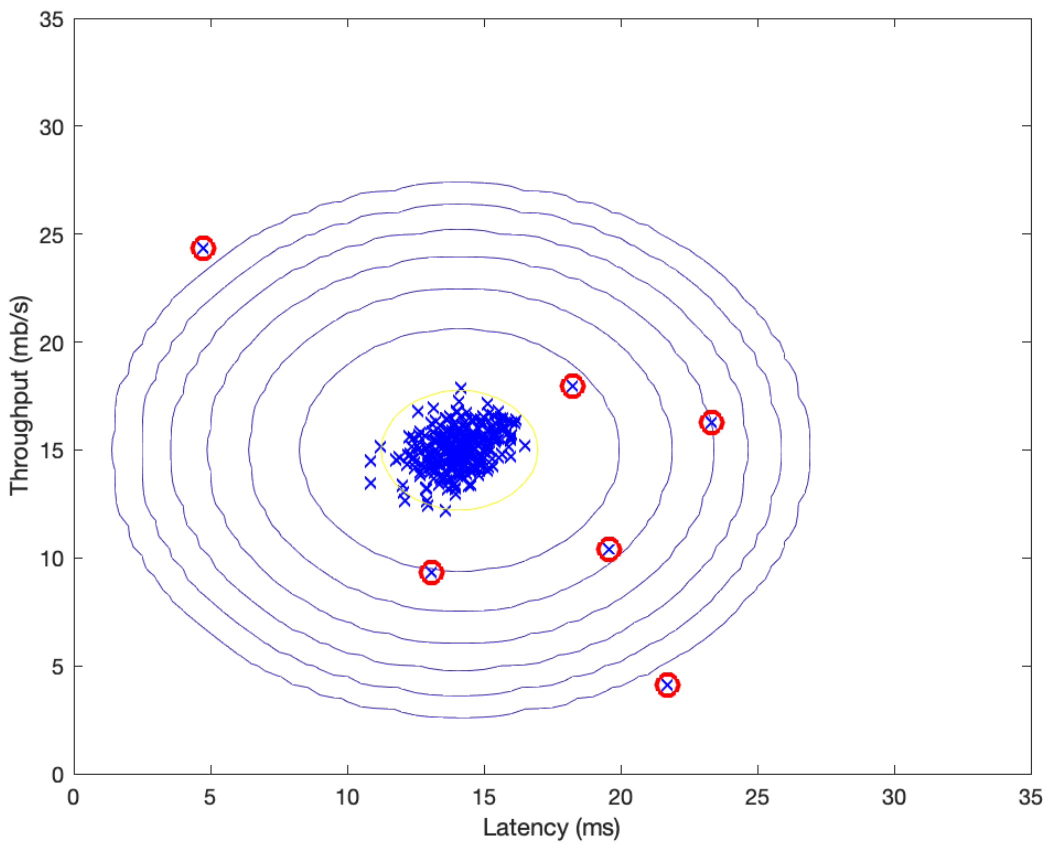
2. selectThreshold.m

```
function [bestEpsilon bestF1] = selectThreshold(yval, pval)
%SELECTTHRESHOLD Find the best threshold (epsilon) to use for selecting
%outliers
%   [bestEpsilon bestF1] = SELECTTHRESHOLD(yval, pval) finds the best
%   threshold to use for selecting outliers based on the results from a
%   validation set (pval) and the ground truth (yval).
%

bestEpsilon = 0;
bestF1 = 0;
F1 = 0;

stepsize = (max(pval) - min(pval)) / 1000;
for epsilon = min(pval):stepsize:max(pval)

    % ====================== YOUR CODE HERE ======================
    % Instructions: Compute the F1 score of choosing epsilon as the
    %               threshold and place the value in F1. The code at the
    %               end of the loop will compare the F1 score for this
    %               choice of epsilon and set it to be the best epsilon if
    %               it is better than the current choice of epsilon.
    %
    % Note: You can use predictions = (pval < epsilon) to get a binary
vector
    %       of 0's and 1's of the outlier predictions

    % yval:label    pval:probability
    cvPredictions = (pval < epsilon);
    tp = sum((cvPredictions == 1) & (yval == 1));
    fp = sum((cvPredictions == 1) & (yval == 0));
    fn = sum((cvPredictions == 0) & (yval == 1));
    prec = tp / (tp + fp);
    rec = tp / (tp + fn);
    F1 = 2 * prec * rec / (prec + rec);
    % =============================================================

    if F1 > bestF1
       bestF1 = F1;
       bestEpsilon = epsilon;
    end
end

end
```

Related output:

```
Program paused. Press enter to continue.
Best epsilon found using cross-validation: 8.990853e-05
Best F1 on Cross Validation Set:  0.875000
   (you should see a value epsilon of about 8.99e-05)
   (you should see a Best F1 value of  0.875000)
```

Related plot:



Whole output:

```
Visualizing example dataset for outlier detection.

Program paused. Press enter to continue.
Visualizing Gaussian fit.

Program paused. Press enter to continue.
Best epsilon found using cross-validation: 8.990853e-05
Best F1 on Cross Validation Set:  0.875000
   (you should see a value epsilon of about 8.99e-05)
   (you should see a Best F1 value of  0.875000)

Program paused. Press enter to continue.
Best epsilon found using cross-validation: 1.377229e-18
Best F1 on Cross Validation Set:  0.615385
   (you should see a value epsilon of about 1.38e-18)
```

```
    (you should see a Best F1 value of 0.615385)
# Outliers found: 117
```

The output shows that the first part of this exercise is implemented correctly.

3. cofiCostFunc.m

```
function [J, grad] = cofiCostFunc(params, Y, R, num_users, num_movies, ...
                                  num_features, lambda)
%COFICOSTFUNC Collaborative filtering cost function
%   [J, grad] = COFICOSTFUNC(params, Y, R, num_users, num_movies, ...
%   num_features, lambda) returns the cost and gradient for the
%   collaborative filtering problem.
%

% Unfold the U and W matrices from params
X = reshape(params(1:num_movies*num_features), num_movies, num_features);
Theta = reshape(params(num_movies*num_features+1:end), ...
                num_users, num_features);


% You need to return the following values correctly
J = 0;
X_grad = zeros(size(X));
Theta_grad = zeros(size(Theta));

% ====================== YOUR CODE HERE ======================
% Instructions: Compute the cost function and gradient for collaborative
%               filtering. Concretely, you should first implement the cost
%               function (without regularization) and make sure it is
%               matches our costs. After that, you should implement the
%               gradient and use the checkCostFunction routine to check
%               that the gradient is correct. Finally, you should implement
%               regularization.
%
% Notes: X - num_movies  x num_features matrix of movie features
%        Theta - num_users  x num_features matrix of user features
%        Y - num_movies x num_users matrix of user ratings of movies
%        R - num_movies x num_users matrix, where R(i, j) = 1 if the
%            i-th movie was rated by the j-th user
%
% You should set the following variables correctly:
%
%        X_grad - num_movies x num_features matrix, containing the
%                 partial derivatives w.r.t. to each element of X
```

```
%        Theta_grad - num_users x num_features matrix, containing the
%                     partial derivatives w.r.t. to each element of Theta
%
% Original implementation of cost.
% sum = 0;
% for i = 1:num_movies
%     for j = 1:num_users
%         if R(i,j) == 1
%             a = (X(i,:) * Theta(j,:)'  - Y(i,j))^2;
%             sum = sum + a;
%         end
%     end
% end
% J = 1 / 2 * sum;

% Vectorized implementation of cost.
a = (X * Theta' - Y).^2;
b = lambda / 2 * sum(sum(Theta.^2));
c = lambda / 2 * sum(sum(X.^2));
J = 1 / 2 * sum(sum(R .* a)) + b + c;

% Original implementation of X_grad.
% for i = 1:num_movies
%     for k = 1:3
%         g = 0;
%         for j = 1:num_users
%             if R(i,j) == 1
%                 a = (X(i,:) * Theta(j,:)'  - Y(i,j)) * Theta(j,k);
%                 g = g + a;
%             end
%         end
%         X_grad(i,k) = g;
%     end
% end

% Vectorized implementation of X_grad.
for i = 1:num_movies
                  idx = find(R(i, :)==1);
                  Theta_temp = Theta(idx, :);
                  Y_temp = Y(i, idx);
                  X_grad(i,:) = (X(i,:) * Theta_temp' - Y_temp) *
Theta_temp + lambda * X(i,:);
end

% Original implementation of Theta_grad.
% for j = 1:num_users
%     for k = 1:3
%         g = 0;
%         for i = 1:num_movies
```

```
%             if R(i,j) == 1
%                 a = (X(i,:) * Theta(j,:)'  - Y(i,j)) * X(i,k);
%                 g = g + a;
%             end
%         end
%         Theta_grad(j,k) = g;
%     end
% end

% Vectorized implementation of Theta_grad.
for j = 1:num_users
                  idx = find(R(:, j)==1);
                  X_temp = X(idx, :);
                  Y_temp = Y(idx, j);
                  Theta_grad(j,:) = (X_temp' * (X_temp * Theta(j,:)' -
Y_temp))' + lambda * Theta(j,:);
end

% =============================================================
% disp(X_grad);
% disp(Theta_grad);
grad = [X_grad(:); Theta_grad(:)];

end
```

Whole output:

```
Loading movie ratings dataset.

Average rating for movie 1 (Toy Story): 3.878319 / 5


Program paused. Press enter to continue.
Cost at loaded parameters: 22.224604
(this value should be about 22.22)

Program paused. Press enter to continue.

Checking Gradients (without regularization) ...
    5.5335    5.5335
    3.6186    3.6186
    5.4422    5.4422
   -1.7312   -1.7312
    4.1196    4.1196
   -1.4833   -1.4833
   -6.0734   -6.0734
    2.3490    2.3490
    7.6341    7.6341
    1.8651    1.8651
```

```
    4.1192    4.1192
   -1.5834   -1.5834
    1.2828    1.2828
   -6.1573   -6.1573
    1.6628    1.6628
    1.1686    1.1686
    5.5630    5.5630
    0.3050    0.3050
    4.6442    4.6442
   -1.6691   -1.6691
   -2.1505   -2.1505
   -3.6832   -3.6832
    3.4067    3.4067
   -4.0743   -4.0743
    0.5567    0.5567
   -2.1056   -2.1056
    0.9168    0.9168

The above two columns you get should be very similar.
(Left-Your Numerical Gradient, Right-Analytical Gradient)

If your cost function implementation is correct, then
the relative difference will be small (less than 1e-9).

Relative Difference: 1.7768e-12

Program paused. Press enter to continue.
Cost at loaded parameters (lambda = 1.5): 31.344056
(this value should be about 31.34)

Program paused. Press enter to continue.

Checking Gradients (with regularization) ...
    2.2223    2.2223
    0.7968    0.7968
   -3.2924   -3.2924
   -0.7029   -0.7029
   -4.2016   -4.2016
    3.5969    3.5969
    0.8859    0.8859
    1.0523    1.0523
   -7.8499   -7.8499
    0.3904    0.3904
   -0.1347   -0.1347
   -2.3656   -2.3656
    2.1066    2.1066
    1.6703    1.6703
    0.8519    0.8519
   -1.0380   -1.0380
```

```
    2.6537    2.6537
    0.8114    0.8114
   -0.8604   -0.8604
   -0.5884   -0.5884
   -0.7108   -0.7108
   -4.0652   -4.0652
    0.2494    0.2494
   -4.3484   -4.3484
   -3.6167   -3.6167
   -4.1277   -4.1277
   -3.2439   -3.2439

The above two columns you get should be very similar.
(Left-Your Numerical Gradient, Right-Analytical Gradient)

If your cost function implementation is correct, then
the relative difference will be small (less than 1e-9).

Relative Difference: 1.82991e-12

Program paused. Press enter to continue.


New user ratings:
Rated 4 for Toy Story (1995)
Rated 3 for Twelve Monkeys (1995)
Rated 5 for Usual Suspects, The (1995)
Rated 4 for Outbreak (1995)
Rated 5 for Shawshank Redemption, The (1994)
Rated 3 for While You Were Sleeping (1995)
Rated 5 for Forrest Gump (1994)
Rated 2 for Silence of the Lambs, The (1991)
Rated 4 for Alien (1979)
Rated 5 for Die Hard 2 (1990)
Rated 5 for Sphere (1998)

Program paused. Press enter to continue.

Training collaborative filtering...
Iteration     1 | Cost: 3.108511e+05
Iteration     2 | Cost: 1.475959e+05
Iteration     3 | Cost: 1.000321e+05
Iteration     4 | Cost: 7.707565e+04
Iteration     5 | Cost: 6.153638e+04
Iteration     6 | Cost: 5.719300e+04
Iteration     7 | Cost: 5.239113e+04
Iteration     8 | Cost: 4.771435e+04
Iteration     9 | Cost: 4.559863e+04
Iteration    10 | Cost: 4.385394e+04
```

```
Iteration    11 | Cost: 4.263562e+04
Iteration    12 | Cost: 4.184598e+04
Iteration    13 | Cost: 4.116751e+04
Iteration    14 | Cost: 4.073297e+04
Iteration    15 | Cost: 4.032577e+04
Iteration    16 | Cost: 4.009203e+04
Iteration    17 | Cost: 3.986428e+04
Iteration    18 | Cost: 3.971337e+04
Iteration    19 | Cost: 3.958890e+04
Iteration    20 | Cost: 3.949630e+04
Iteration    21 | Cost: 3.940187e+04
Iteration    22 | Cost: 3.934142e+04
Iteration    23 | Cost: 3.930822e+04
Iteration    24 | Cost: 3.926063e+04
Iteration    25 | Cost: 3.922334e+04
Iteration    26 | Cost: 3.920956e+04
Iteration    27 | Cost: 3.917145e+04
Iteration    28 | Cost: 3.914804e+04
Iteration    29 | Cost: 3.913479e+04
Iteration    30 | Cost: 3.910882e+04
Iteration    31 | Cost: 3.908992e+04
Iteration    32 | Cost: 3.908209e+04
Iteration    33 | Cost: 3.907380e+04
Iteration    34 | Cost: 3.906903e+04
Iteration    35 | Cost: 3.906437e+04
Iteration    36 | Cost: 3.905754e+04
Iteration    37 | Cost: 3.905112e+04
Iteration    38 | Cost: 3.904531e+04
Iteration    39 | Cost: 3.904023e+04
Iteration    40 | Cost: 3.903390e+04
Iteration    41 | Cost: 3.902800e+04
Iteration    42 | Cost: 3.902367e+04
Iteration    43 | Cost: 3.902195e+04
Iteration    44 | Cost: 3.902007e+04
Iteration    45 | Cost: 3.901780e+04
Iteration    46 | Cost: 3.901699e+04
Iteration    47 | Cost: 3.901489e+04
Iteration    48 | Cost: 3.901190e+04
Iteration    49 | Cost: 3.900929e+04
Iteration    50 | Cost: 3.900742e+04
Iteration    51 | Cost: 3.900630e+04
Iteration    52 | Cost: 3.900485e+04
Iteration    53 | Cost: 3.900348e+04
Iteration    54 | Cost: 3.900283e+04
Iteration    55 | Cost: 3.900208e+04
Iteration    56 | Cost: 3.900118e+04
Iteration    57 | Cost: 3.899982e+04
Iteration    58 | Cost: 3.899860e+04
Iteration    59 | Cost: 3.899710e+04
```

```
Iteration    60 | Cost: 3.899381e+04
Iteration    61 | Cost: 3.899242e+04
Iteration    62 | Cost: 3.899094e+04
Iteration    63 | Cost: 3.898986e+04
Iteration    64 | Cost: 3.898908e+04
Iteration    65 | Cost: 3.898811e+04
Iteration    66 | Cost: 3.898754e+04
Iteration    67 | Cost: 3.898736e+04
Iteration    68 | Cost: 3.898712e+04
Iteration    69 | Cost: 3.898687e+04
Iteration    70 | Cost: 3.898673e+04
Iteration    71 | Cost: 3.898634e+04
Iteration    72 | Cost: 3.898524e+04
Iteration    73 | Cost: 3.898369e+04
Iteration    74 | Cost: 3.898322e+04
Iteration    75 | Cost: 3.898257e+04
Iteration    76 | Cost: 3.898194e+04
Iteration    77 | Cost: 3.898141e+04
Iteration    78 | Cost: 3.898077e+04
Iteration    79 | Cost: 3.898025e+04
Iteration    80 | Cost: 3.897962e+04
Iteration    81 | Cost: 3.897908e+04
Iteration    82 | Cost: 3.897861e+04
Iteration    83 | Cost: 3.897736e+04
Iteration    84 | Cost: 3.897610e+04
Iteration    85 | Cost: 3.897534e+04
Iteration    86 | Cost: 3.897490e+04
Iteration    87 | Cost: 3.897472e+04
Iteration    88 | Cost: 3.897418e+04
Iteration    89 | Cost: 3.897392e+04
Iteration    90 | Cost: 3.897377e+04
Iteration    91 | Cost: 3.897367e+04
Iteration    92 | Cost: 3.897336e+04
Iteration    93 | Cost: 3.897322e+04
Iteration    94 | Cost: 3.897312e+04
Iteration    95 | Cost: 3.897305e+04
Iteration    96 | Cost: 3.897281e+04
Iteration    97 | Cost: 3.897268e+04
Iteration    98 | Cost: 3.897265e+04
Iteration    99 | Cost: 3.897250e+04
Iteration   100 | Cost: 3.897237e+04

Recommender system learning completed.

Program paused. Press enter to continue.

Top recommendations for you:
Predicting rating 5.0 for movie Saint of Fort Washington, The (1993)
Predicting rating 5.0 for movie Great Day in Harlem, A (1994)
```

```
Predicting rating 5.0 for movie Someone Else's America (1995)
Predicting rating 5.0 for movie Santa with Muscles (1996)
Predicting rating 5.0 for movie Entertaining Angels: The Dorothy Day Story
(1996)
Predicting rating 5.0 for movie Aiqing wansui (1994)
Predicting rating 5.0 for movie Prefontaine (1997)
Predicting rating 5.0 for movie They Made Me a Criminal (1939)
Predicting rating 5.0 for movie Marlene Dietrich: Shadow and Light (1996)
Predicting rating 5.0 for movie Star Kid (1997)


Original ratings provided:
Rated 4 for Toy Story (1995)
Rated 3 for Twelve Monkeys (1995)
Rated 5 for Usual Suspects, The (1995)
Rated 4 for Outbreak (1995)
Rated 5 for Shawshank Redemption, The (1994)
Rated 3 for While You Were Sleeping (1995)
Rated 5 for Forrest Gump (1994)
Rated 2 for Silence of the Lambs, The (1991)
Rated 4 for Alien (1979)
Rated 5 for Die Hard 2 (1990)
Rated 5 for Sphere (1998)
```

The output in part 2 is equal to the document, which shows that the part 2 is implemented correctly.