Anomaly detection

Anomaly detection

During this exercise we are going to implement an anomaly detection algorithm for server computers. The features measure the throughput (mb/s) and latency (ms) of response of each server.

Firstly, load the dataset

ex8data1.mat

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.io import loadmat


data = loadmat('ex8data1.mat')
X = data['X']
# TODO: amount of data
```

Using `X.shape` analyse the amount of data and features.

Visualise the dataset using `scatter` plot

```
# TODO: visualise the dataset and look for anomalies
```

Using `plt.hist` plot the histogram of the data and check wether it is necessary to scale it.

```
# TODO: plot the histogram for features: throughput (mb/s) and latency (ms).
```

Gaussian distribution

Now, you will estimate a Gaussian distribution for each feature in the data.
To accomplish this we'll create a simple function that calculates the mean and standard deviation for each feature in our data set:

```
def estimate_gaussian(X):

# TODO: calculate mu and sigma

    return mu, sigma

# TODO: run the function for your dataset
```

Correct outcome: `(array([14.11222578, 14.99771051]), array([1.35374717, 1.3075723 ]))`

Calculate probability

Now that we have our model parameters, we need to determine a probability threshold which indicates that an example should be considered an anomaly. To do this, we need to use a set of labeled validation data:

```
Xval = data['Xval']
yval = data['yval']

# TODO: check the number of data (X.shape)
```

Use the biult-in function `stats.norm.pdf()` (with parameters: `loc` as mean and `scale` as standard deviation) to calculate the probability that a data (firstly X not Xval) point belongs to a normal distribution (for both features). Calculate the probability that each of the first 50 instances of our data set's first dimension belong to the distribution:

```
from scipy import stats
# TODO: calculate the probability for X data
```

You just calculated the probability that a data point belongs to a normal distribution that we defined earlier by calculating the mean and std for that dimension (feature 1 and 2). It's computing how far each instance is from the mean.
Now, we also need to do this for the validation set (**using the same model parameters**). We'll use these probabilities combined with the true label to determine the optimal probability threshold to assign data points as anomalies.

```
pval = np.zeros((Xval.shape[0], Xval.shape[1]))
# TODO: calculate the probability for Xval data
# TODO: pval[:,0] =
# TODO: pval[:,1] =
```

Selecting the threshold

The aim of this part is to implement a function that finds the best threshold value given the probability density values and true labels.
The function should return two values: selected threshold and F1 score (import `f1_score` from `sklearn.metrics`), which tells you how well you’re doing on finding the ground truth anomalies given a certain threshold.
Epsilon value should be iterated with a small step between 'p.min' and 'p.max' value (about 10000 steps).

```
def select_threshold(pval, yval):
TODO: select the best epsilon based on the F1 score
```

```
return best_epsilon, best_f1
```

Possible outcome (for validation data!):
`(0.00039, 0.875)`

Anomaly detection result

run your anomaly detection code and circle the anomalies in the plot:

```
# TODO: find indexes where the p value is lower than epsilon. Use the
np.where() function
# TODO: plot the data and analyse the outcome. Use plt.scatter() function
```

When you finish, please upload your results on the UPEL platform.

If you have any problems with the task, check the

example provided by the author