C3_W1_Anomaly_Detection

October 25, 2023

```
[ ]: # Anomaly Detection

     In this exercise, you will implement the anomaly detection algorithm and apply␣
      ↪it to detect failing servers on a network.



     # Outline
     - [ 1 - Packages ](#1)
     - [ 2 - Anomaly detection](#2)
       - [ 2.1 Problem Statement](#2.1)
       - [ 2.2  Dataset](#2.2)
       - [ 2.3 Gaussian distribution](#2.3)
         - [ Exercise 1](#ex01)
         - [ Exercise 2](#ex02)
       - [ 2.4 High dimensional dataset](#2.4)
```

NOTE: *To prevent errors from the autograder, you are not allowed to edit or delete non-graded cells in this lab. Please also refrain from adding any new cells.* **Once you have passed this assignment** *and want to experiment with any of the non-graded code, you may follow the instructions at the bottom of this notebook.*

1 - Packages

First, let's run the cell below to import all the packages that you will need during this assignment. - numpy is the fundamental package for working with matrices in Python. - matplotlib is a famous library to plot graphs in Python. - utils.py contains helper functions for this assignment. You do not need to modify code in this file.

```
[1]: import numpy as np
     import matplotlib.pyplot as plt
     from utils import *

     %matplotlib inline
```

2 - Anomaly detection

2.1 Problem Statement

In this exercise, you will implement an anomaly detection algorithm to detect anomalous behavior in server computers.

The dataset contains two features - * throughput (mb/s) and * latency (ms) of response of each server.

While your servers were operating, you collected $m = 307$ examples of how they were behaving, and thus have an unlabeled dataset $\{x^{(1)}, \ldots, x^{(m)}\}$. * You suspect that the vast majority of these examples are "normal" (non-anomalous) examples of the servers operating normally, but there might also be some examples of servers acting anomalously within this dataset.

You will use a Gaussian model to detect anomalous examples in your dataset. * You will first start on a 2D dataset that will allow you to visualize what the algorithm is doing. * On that dataset you will fit a Gaussian distribution and then find values that have very low probability and hence can be considered anomalies. * After that, you will apply the anomaly detection algorithm to a larger dataset with many dimensions.

2.2 Dataset

You will start by loading the dataset for this task. - The `load_data()` function shown below loads the data into the variables `X_train`, `X_val` and `y_val` - You will use `X_train` to fit a Gaussian distribution - You will use `X_val` and `y_val` as a cross validation set to select a threshold and determine anomalous vs normal examples

```
[2]:  # Load the dataset
      X_train, X_val, y_val = load_data()
```

View the variables Let's get more familiar with your dataset.
- A good place to start is to just print out each variable and see what it contains.

The code below prints the first five elements of each of the variables

```
[3]:  # Display the first five elements of X_train
      print("The first 5 elements of X_train are:\n", X_train[:5])
```

```
The first 5 elements of X_train are:
 [[13.04681517 14.74115241]
 [13.40852019 13.7632696 ]
 [14.19591481 15.85318113]
 [14.91470077 16.17425987]
 [13.57669961 14.04284944]]
```

```
[4]:  # Display the first five elements of X_val
      print("The first 5 elements of X_val are\n", X_val[:5])
```

```
The first 5 elements of X_val are
 [[15.79025979 14.9210243 ]
 [13.63961877 15.32995521]
 [14.86589943 16.47386514]
```

```
[13.58467605 13.98930611]
[13.46404167 15.63533011]]
```

[5]:
```python
# Display the first five elements of y_val
print("The first 5 elements of y_val are\n", y_val[:5])
```

```
The first 5 elements of y_val are
 [0 0 0 0 0]
```

Check the dimensions of your variables Another useful way to get familiar with your data is to view its dimensions.

The code below prints the shape of X_train, X_val and y_val.

[6]:
```python
print ('The shape of X_train is:', X_train.shape)
print ('The shape of X_val is:', X_val.shape)
print ('The shape of y_val is: ', y_val.shape)
```

```
The shape of X_train is: (307, 2)
The shape of X_val is: (307, 2)
The shape of y_val is:  (307,)
```

Visualize your data Before starting on any task, it is often useful to understand the data by visualizing it. - For this dataset, you can use a scatter plot to visualize the data (X_train), since it has only two properties to plot (throughput and latency)

- Your plot should look similar to the one below

[7]:
```python
# Create a scatter plot of the data. To change the markers to blue "x",
# we used the 'marker' and 'c' parameters
plt.scatter(X_train[:, 0], X_train[:, 1], marker='x', c='b')

# Set the title
plt.title("The first dataset")
# Set the y-axis label
plt.ylabel('Throughput (mb/s)')
# Set the x-axis label
plt.xlabel('Latency (ms)')
# Set axis range
plt.axis([0, 30, 0, 30])
plt.show()
```

The first dataset
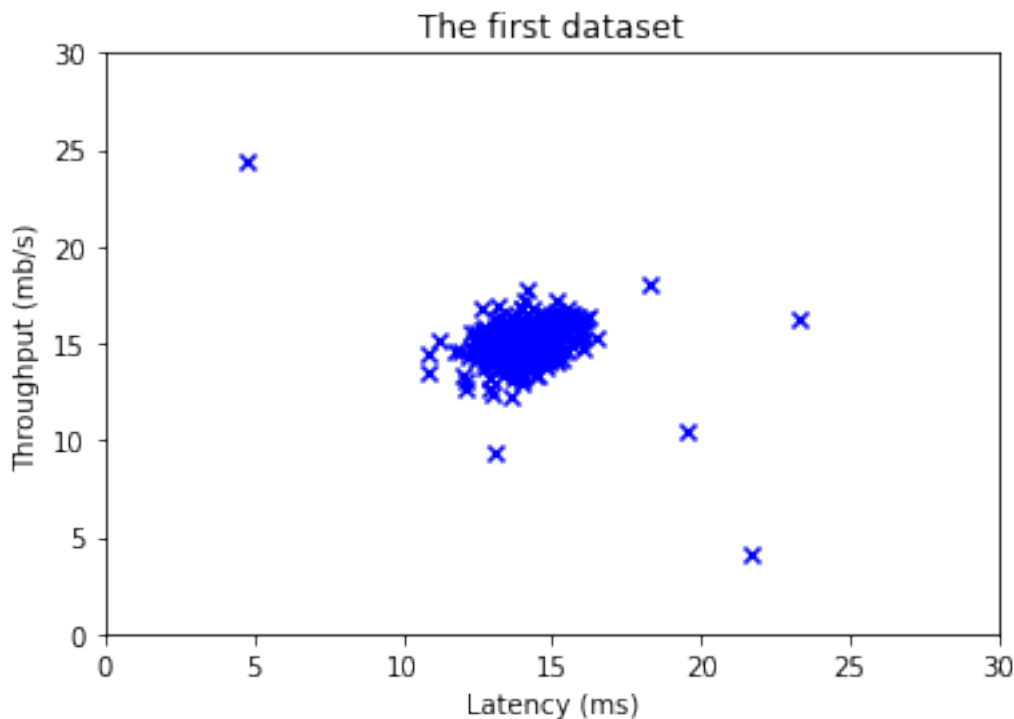
2.3 Gaussian distribution

To perform anomaly detection, you will first need to fit a model to the data's distribution.

- Given a training set $\{x^{(1)}, ..., x^{(m)}\}$ you want to estimate the Gaussian distribution for each of the features x_i.

- Recall that the Gaussian distribution is given by

$$p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

  where μ is the mean and σ^2 is the variance.

- For each feature $i = 1 \ldots n$, you need to find parameters μ_i and σ_i^2 that fit the data in the i-th dimension $\{x_i^{(1)}, ..., x_i^{(m)}\}$ (the i-th dimension of each example).

0.0.1   2.3.1 Estimating parameters for a Gaussian distribution

Implementation:

Your task is to complete the code in `estimate_gaussian` below.

Exercise 1

Please complete the `estimate_gaussian` function below to calculate `mu` (mean for each feature in X) and `var` (variance for each feature in X).

You can estimate the parameters, (μ_i, σ_i^2), of the i-th feature by using the following equations. To estimate the mean, you will use:

$$\mu_i = \frac{1}{m} \sum_{j=1}^{m} x_i^{(j)}$$

and for the variance you will use:

$$\sigma_i^2 = \frac{1}{m} \sum_{j=1}^{m} (x_i^{(j)} - \mu_i)^2$$

If you get stuck, you can check out the hints presented after the cell below to help you with the implementation.

[8]:
```python
# UNQ_C1
# GRADED FUNCTION: estimate_gaussian

def estimate_gaussian(X):
    """
    Calculates mean and variance of all features
    in the dataset

    Args:
        X (ndarray): (m, n) Data matrix

    Returns:
        mu (ndarray): (n,) Mean of all features
        var (ndarray): (n,) Variance of all features
    """

    m, n = X.shape

    ### START CODE HERE ###
    mu = np.mean(X,axis=0)
    var = np.var(X,axis=0)
    ### END CODE HERE ###

    return mu, var
```

Click for hints

- You can implement this function in two ways:
    - 1 - by having two nested for loops - one looping over the **columns** of X (each feature) and then looping over each data point.
    - 2 - in a vectorized manner by using `np.sum()` with `axis = 0` parameter (since we want the sum for each column)

5

– Here's how you can structure the overall implementation of this function for the vectorized implementation:

```python
def estimate_gaussian(X):
    m, n = X.shape

    ### START CODE HERE ###
    mu = # Your code here to calculate the mean of every feature
    var = # Your code here to calculate the variance of every feature
    ### END CODE HERE ###

    return mu, var
```

If you're still stuck, you can check the hints presented below to figure out how to calculate `mu` and `var`.

Hint to calculate mu   You can use np.sum to with `axis = 0` parameter to get the sum for each column of an array

More hints to calculate mu   You can compute mu as mu = 1 / m * np.sum(X, axis = 0)

Hint to calculate var   You can use np.sum to with `axis = 0` parameter to get the sum for each column of an array and **2 to get the square.

More hints to calculate var   You can compute var as var = 1 / m * np.sum((X - mu) ** 2, axis = 0)

You can check if your implementation is correct by running the following test code:

```python
[9]:  # Estimate mean and variance of each feature
      mu, var = estimate_gaussian(X_train)

      print("Mean of each feature:", mu)
      print("Variance of each feature:", var)

      # UNIT TEST
      from public_tests import *
      estimate_gaussian_test(estimate_gaussian)
```

```
Mean of each feature: [14.11222578 14.99771051]
Variance of each feature: [1.83263141 1.70974533]
All tests passed!
```

Expected Output:

Mean of each feature:

[14.11222578 14.99771051]

Variance of each feature:

[1.83263141 1.70974533]

Now that you have completed the code in `estimate_gaussian`, we will visualize the contours of the fitted Gaussian distribution.

You should get a plot similar to the figure below.

From your plot you can see that most of the examples are in the region with the highest probability, while the anomalous examples are in the regions with lower probabilities.

```
[11]:  # Returns the density of the multivariate normal
       # at each data point (row) of X_train
       p = multivariate_gaussian(X_train, mu, var)

       #Plotting code
       visualize_fit(X_train, mu, var)
```
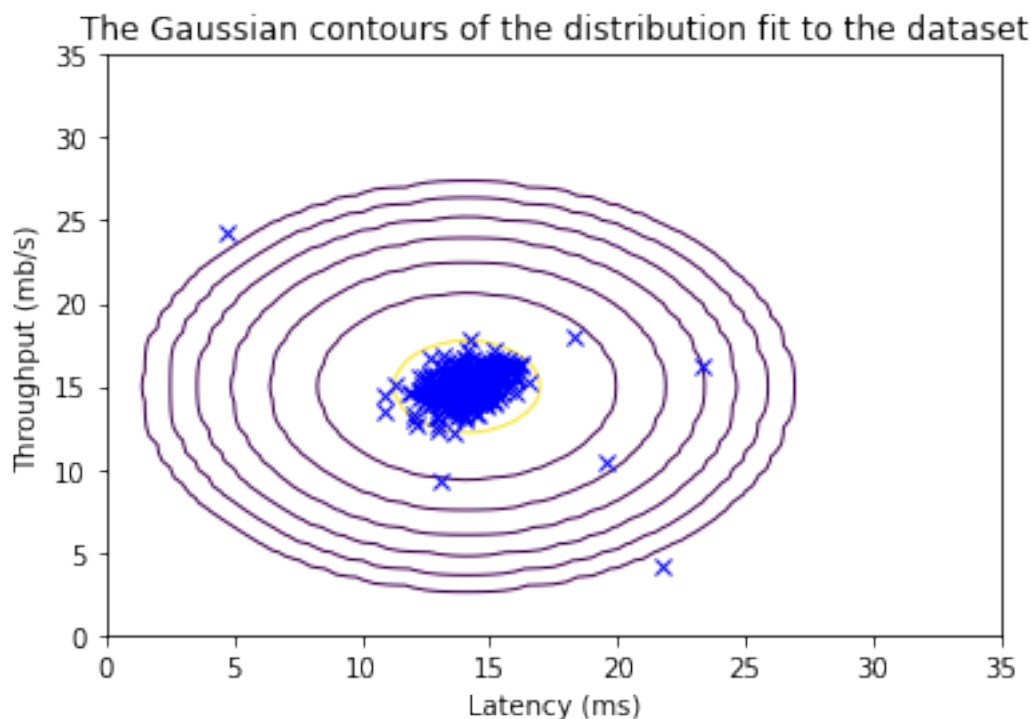

0.0.2 2.3.2 Selecting the threshold ϵ

Now that you have estimated the Gaussian parameters, you can investigate which examples have a very high probability given this distribution and which examples have a very low probability.

- The low probability examples are more likely to be the anomalies in our dataset.
- One way to determine which examples are anomalies is to select a threshold based on a cross validation set.

In this section, you will complete the code in `select_threshold` to select the threshold ε using the F_1 score on a cross validation set.

- For this, we will use a cross validation set $\{(x_{cv}^{(1)}, y_{cv}^{(1)}), \ldots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})\}$, where the label $y = 1$ corresponds to an anomalous example, and $y = 0$ corresponds to a normal example.
- For each cross validation example, we will compute $p(x_{cv}^{(i)})$. The vector of all of these probabilities $p(x_{cv}^{(1)}), \ldots, p(x_{cv}^{(m_{cv})})$ is passed to `select_threshold` in the vector `p_val`.
- The corresponding labels $y_{cv}^{(1)}, \ldots, y_{cv}^{(m_{cv})}$ are passed to the same function in the vector `y_val`.

Exercise 2 Please complete the `select_threshold` function below to find the best threshold to use for selecting outliers based on the results from the validation set (`p_val`) and the ground truth (`y_val`).

- In the provided code `select_threshold`, there is already a loop that will try many different values of ε and select the best ε based on the F_1 score.

- You need to implement code to calculate the F1 score from choosing `epsilon` as the threshold and place the value in `F1`.

  - Recall that if an example x has a low probability $p(x) < \varepsilon$, then it is classified as an anomaly.

  - Then, you can compute precision and recall by:

  $$prec = \frac{tp}{tp + fp}$$
  $$rec = \frac{tp}{tp + fn},$$

  where

  * tp is the number of true positives: the ground truth label says it's an anomaly and our algorithm correctly classified it as an anomaly.
  * fp is the number of false positives: the ground truth label says it's not an anomaly, but our algorithm incorrectly classified it as an anomaly.
  * fn is the number of false negatives: the ground truth label says it's an anomaly, but our algorithm incorrectly classified it as not being anomalous.

  - The F_1 score is computed using precision ($prec$) and recall (rec) as follows:

  $$F_1 = \frac{2 \cdot prec \cdot rec}{prec + rec}$$

Implementation Note: In order to compute tp, fp and fn, you may be able to use a vectorized implementation rather than loop over all the examples.

If you get stuck, you can check out the hints presented after the cell below to help you with the implementation.

```
[28]:  # UNQ_C2
       # GRADED FUNCTION: select_threshold

       def select_threshold(y_val, p_val):
```

```python
    """
    Finds the best threshold to use for selecting outliers
    based on the results from a validation set (p_val)
    and the ground truth (y_val)

    Args:
        y_val (ndarray): Ground truth on validation set
        p_val (ndarray): Results on validation set

    Returns:
        epsilon (float): Threshold chosen
        F1 (float):      F1 score by choosing epsilon as threshold
    """

    best_epsilon = 0
    best_F1 = 0
    F1 = 0

    step_size = (max(p_val) - min(p_val)) / 1000

    for epsilon in np.arange(min(p_val), max(p_val), step_size):

        ### START CODE HERE ###
        tp = np.sum(np.logical_and(y_val==1 , p_val < epsilon))
        fp = np.sum(np.logical_and(y_val==0 , p_val < epsilon))
        fn = np.sum(np.logical_and(y_val==1 , p_val >= epsilon))
        prec = tp / (tp + fp)
        rec = tp / (tp + fn)
        F1 = 2 * prec * rec / (prec + rec)

        ### END CODE HERE ###

        if F1 > best_F1:
            best_F1 = F1
            best_epsilon = epsilon

    return best_epsilon, best_F1
```

Click for hints

- Here's how you can structure the overall implementation of this function for the vectorized implementation: "'python
  def select_threshold(y_val, p_val): best_epsilon = 0 best_F1 = 0 F1 = 0

  ```python
  step_size = (max(p_val) - min(p_val)) / 1000

  for epsilon in np.arange(min(p_val), max(p_val), step_size):
  ```

```
    ### START CODE HERE ###
    predictions = # Your code here to calculate predictions for each example using epsilo

    tp = # Your code here to calculate number of true positives
    fp = # Your code here to calculate number of false positives
    fn = # Your code here to calculate number of false negatives

    prec = # Your code here to calculate precision
    rec = # Your code here to calculate recall

    F1 = # Your code here to calculate F1
    ### END CODE HERE ###

    if F1 > best_F1:
        best_F1 = F1
        best_epsilon = epsilon

 return best_epsilon, best_F1
```

" '

If you're still stuck, you can check the hints presented below to figure out how to calculate each variable.

Hint to calculate predictions   If an example has a low probability $p(x) < \epsilon$, then it is classified as an anomaly. To get predictions for each example (0/ False for normal and 1/True for anomaly), you can use predictions = (p_val < epsilon)

Hint to calculate tp, fp, fn

If you have several binary values in an n-dimensional binary vector, you can find out how many values in this vector are 0 by using: `np.sum(v == 0)`

You can also apply a logical *and* operator to such binary vectors. For instance, `predictions` is a binary vector of the size of your number of cross validation set, where the i-th element is 1 if your algorithm considers $x_{cv}^{(i)}$ an anomaly, and 0 otherwise.

You can then, for example, compute the number of false positives using:
fp = sum((predictions == 1) & (y_val == 0)).

        More hints to calculate tp, fn

You can compute tp as tp = np.sum((predictions == 1) & (y_val == 1))

You can compute tn as fn = np.sum((predictions == 0) & (y_val == 1))

Hint to calculate precision   You can calculate precision as prec = tp / (tp + fp)

Hint to calculate recall   You can calculate recall as rec = tp / (tp + fn)

Hint to calculate F1   You can calculate F1 as F1 = 2 * prec * rec / (prec + rec)

You can check your implementation using the code below

```
[29]:  p_val = multivariate_gaussian(X_val, mu, var)
       epsilon, F1 = select_threshold(y_val, p_val)

       print('Best epsilon found using cross-validation: %e' % epsilon)
       print('Best F1 on Cross Validation Set: %f' % F1)

       # UNIT TEST
       select_threshold_test(select_threshold)
```

Best epsilon found using cross-validation: 8.990853e-05
Best F1 on Cross Validation Set: 0.875000
All tests passed!

Expected Output:

Best epsilon found using cross-validation:

8.99e-05
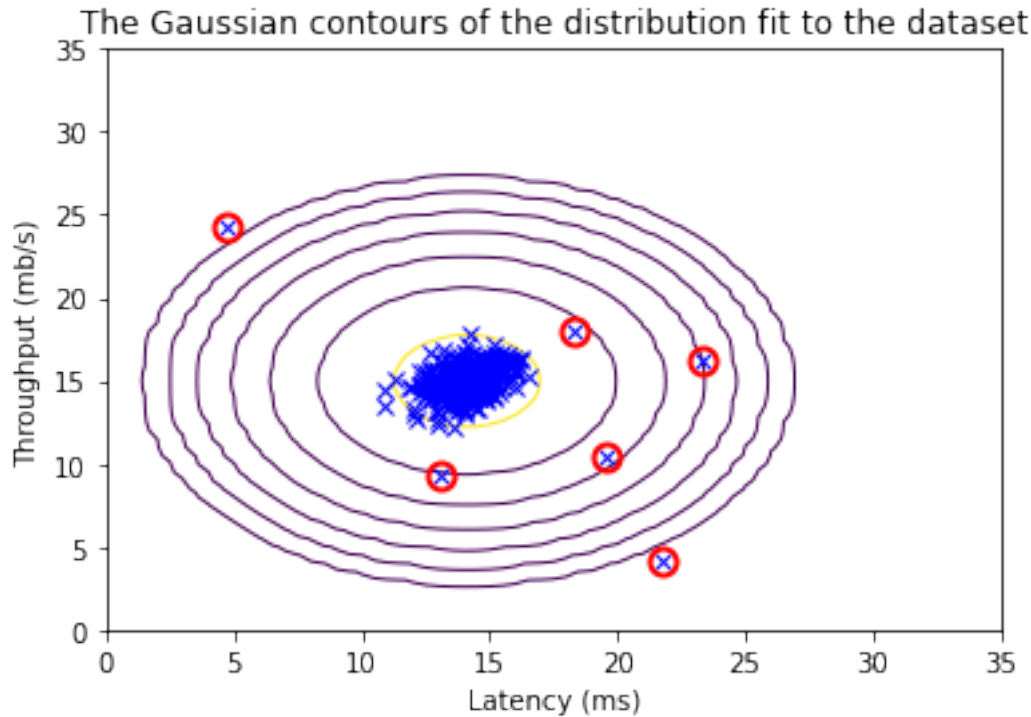
Best F1 on Cross Validation Set:

0.875

Now we will run your anomaly detection code and circle the anomalies in the plot (Figure 3 below).

```
[30]:  # Find the outliers in the training set
       outliers = p < epsilon

       # Visualize the fit
       visualize_fit(X_train, mu, var)

       # Draw a red circle around those outliers
       plt.plot(X_train[outliers, 0], X_train[outliers, 1], 'ro',
                markersize= 10,markerfacecolor='none', markeredgewidth=2)
```

[30]: [<matplotlib.lines.Line2D at 0x7f30a156e3d0>]

The Gaussian contours of the distribution fit to the dataset

2.4 High dimensional dataset

Now, we will run the anomaly detection algorithm that you implemented on a more realistic and much harder dataset.

In this dataset, each example is described by 11 features, capturing many more properties of your compute servers.

Let's start by loading the dataset.

- The `load_data()` function shown below loads the data into variables `X_train_high`, `X_val_high` and `y_val_high`
  - `_high` is meant to distinguish these variables from the ones used in the previous part
  - We will use `X_train_high` to fit Gaussian distribution
  - We will use `X_val_high` and `y_val_high` as a cross validation set to select a threshold and determine anomalous vs normal examples

```
[31]:  # load the dataset
       X_train_high, X_val_high, y_val_high = load_data_multi()
```

Check the dimensions of your variables   Let's check the dimensions of these new variables to become familiar with the data

```
[32]:  print ('The shape of X_train_high is:', X_train_high.shape)
       print ('The shape of X_val_high is:', X_val_high.shape)
```

```
print ('The shape of y_val_high is: ', y_val_high.shape)
```

```
The shape of X_train_high is: (1000, 11)
The shape of X_val_high is: (100, 11)
The shape of y_val_high is:  (100,)
```

Anomaly detection Now, let's run the anomaly detection algorithm on this new dataset.

The code below will use your code to * Estimate the Gaussian parameters (μ_i and σ_i^2) * Evaluate the probabilities for both the training data **X_train_high** from which you estimated the Gaussian parameters, as well as for the the cross-validation set **X_val_high**. * Finally, it will use **select_threshold** to find the best threshold ε.

[33]:
```
# Apply the same steps to the larger dataset

# Estimate the Gaussian parameters
mu_high, var_high = estimate_gaussian(X_train_high)

# Evaluate the probabilites for the training set
p_high = multivariate_gaussian(X_train_high, mu_high, var_high)

# Evaluate the probabilites for the cross validation set
p_val_high = multivariate_gaussian(X_val_high, mu_high, var_high)

# Find the best threshold
epsilon_high, F1_high = select_threshold(y_val_high, p_val_high)

print('Best epsilon found using cross-validation: %e'% epsilon_high)
print('Best F1 on Cross Validation Set:  %f'% F1_high)
print('# Anomalies found: %d'% sum(p_high < epsilon_high))
```

```
Best epsilon found using cross-validation: 1.377229e-18
Best F1 on Cross Validation Set:  0.615385
# Anomalies found: 117
```

Expected Output:

Best epsilon found using cross-validation:

1.38e-18

Best F1 on Cross Validation Set:

0.615385

anomalies found:

117

Please click here if you want to experiment with any of the non-graded code.

Important Note: Please only do this when you've already passed the assignment to avoid problems with the autograder.

On the notebook's menu, click "View" > "Cell Toolbar" > "Edit Metadata"

Hit the "Edit Metadata" button next to the code cell which you want to lock/unlock

Set the attribute value for "editable" to:

"true" if you want to unlock it

"false" if you want to lock it

```
    </li>
    <li> On the notebook's menu, click "View" > "Cell Toolbar" > "None" </li>
</ol>
<p> Here's a short demo of how to do the steps above:
    <br>
    <img src="https://drive.google.com/uc?export=view&id=14Xy_Mb17CZVgzVAgq7NCjMVBvSae3xO1" al
```