Anomaly Detection to detect failing servers on a network

An anomaly detection algorithm will be implemented to detect anomalous behavior in server computers.

The dataset contains two features :

- throughput (mb/s) and
- latency (ms) of the response of each server.

While servers were operating, $m = 307$ examples of how they were behaving are collected, and thus have an unlabeled dataset $\{x^{(1)}, \ldots, x^{(m)}\}$ (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).

A Gaussian model will be used to detect anomalous examples in the dataset.

- A Gaussian distribution will be fitted on the dataset and then values that have very low probability will be finded and hence can be considered anomalies.
- The anomaly detection algorithm will be applied to a larger dataset with many dimensions.

steps followed by code:

- 1- Dataset
- 2- Gaussian distribution
- 3- High dimensional dataset

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

%matplotlib inline
```

1- Dataset:

- X_train will be used to fit a Gaussian distribution
- X_val and y_val will be used as a cross validation set to select a threshold and determine anomalous vs normal examples

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

In [3]:
```
# Displaying 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:
 [[13.04853517 14.74115241]
 [13.40852019 13.7632696 ]
 [14.19591481 15.85318113]
 [14.91470077 16.17501648]
 [13.57660993 14.04284944]]
```

In [4]:
```
# Displaying 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 14.9218243 ]
 [13.63961877 15.32905521]
 [14.86589043 16.47386514]
 [13.58467805 13.98930611]
 [13.46404167 15.63538311]]
```

In [5]:
```
# Displaying 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]
```

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

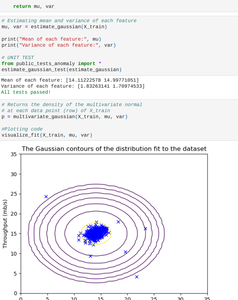
In [6]:
```
print ('The shape of X_train is:', X_train.shape)
print ('The shape of X_val is:', X_val.shape)
print ('The shape of y_val is: ', y_val.shape)
```

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

visualize the data X_train:

The dataset has only two properties to plot (throughput and latency).

In [7]:
```
# Creating a scatter plot of the data.
plt.scatter(X_train[:, 0], X_train[:, 1], marker='x', c='b')

# Set the title
plt.title("The first dataset")
# Set the x-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()
```



2- Gaussian distribution

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

- Given a training set $\{x^{(1)}, \ldots, x^{(m)}\}$ --> estimating the Gaussian distribution for each of the features x_i.
- the Gaussian distribution:

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

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

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

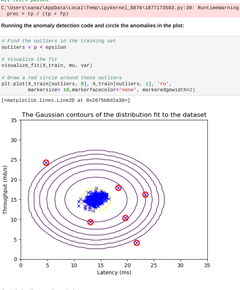
Estimating parameters for a Gaussian distribution --> estimate_gaussian:

In [8]:
```
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

    mu = 1 / m * np.sum(X, axis = 0)
    var = 1 / m * np.sum((X - mu) ** 2, axis = 0)

    return mu, var
```

In [11]:
```
# Estimating 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_anomaly 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!
```

In [12]:
```
# 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)
```



In the plot above, most of the examples are in the region with the highest probability, while the anomalous examples are in the regions with lower probabilities.

Selecting the threshold ϵ:

After the Gaussian parameters have been estimated , we 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.

So, select_threshold is used to select the threshold ε using the F_1 score on a cross validation set.

- For this, a cross validation set will be used $\{(x_{cv}^{(1)}, y_{cv}^{(1)}), \ldots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})\}$, where the label $y = 1$ corresponds to an anomalous example, and $y = 0$ corresponds to a normal example.
- For each cross validation example, $p(x_{cv}^{(1)})$ will be computed. 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.

In 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.
- If an example x has a low probability $p(x) < \varepsilon$, then it is classified as an anomaly.
  - precision and recall:

$$prec = \frac{tp}{tp + fp}$$

$$rec = \frac{tp}{tp + fn}$$

where tp is the number of true positives, fp is the number of false positives, and fn is the number of false negatives

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

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

In [13]:
```
def select_threshold(y_val, p_val):
    """
    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):

        predictions = (p_val < epsilon)

        tp = np.sum((predictions == 1) & (y_val == 1))
        fp = np.sum((predictions == 1) & (y_val == 0))
        fn = np.sum((predictions == 0) & (y_val == 1))

        prec = tp / (tp + fp)
        rec = tp / (tp + fn)

        F1 = 2 * prec * rec / (prec + rec)


        if F1 > best_F1:
            best_F1 = F1
            best_epsilon = epsilon

    return best_epsilon, best_F1
```

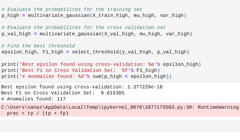
In [15]:
```
p_val = multivariate_gaussian(X_val, mu, var)
epsilon, F1 = select_threshold(y_val, p_val)

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!
C:\Users\sanaz\AppData\Local\Temp\ipykernel_8876\1877173503.py:30: RuntimeWarning: invalid value encountered in scalar divide
  prec = tp / (tp + fp)
```

Running the anomaly detection code and circle the anomalies in the plot:

In [16]:
```
# 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)
```

Out[16]:
```
[<matplotlib.lines.Line2D at 0x2675b6d3a30>]
```



3- High dimensional dataset

In this dataset that the anomaly detection algorithm will be implemented on it, each example is described by 11 features, capturing many more properties of compute servers.

Let's start by loading the dataset.

- _high is meant to distinguish these variables from the ones used in the previous part
- X_train_high will be used to fit Gaussian distribution
- X_val_high and y_val_high will be used as a cross validation set to select a threshold and determine anomalous vs normal examples

In [17]:
```
# Load the dataset
X_train_high, X_val_high, y_val_high = load_data_multi()
```

In [18]:
```
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,)
```

Running the nomaly detection algorithm on this new dataset:

- Estimate the Gaussian parameters (μ_i and σ_i^2)
- Evaluate the probabilities for both the training data X_train_high as well as for the the cross-validation set X_val_high.
- Using select_threshold to find the best threshold ε.

In [19]:
```
# Apply the same steps to the larger dataset

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

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

# Evaluate the probabilities 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
C:\Users\sanaz\AppData\Local\Temp\ipykernel_8876\1877173503.py:30: RuntimeWarning: invalid value encountered in scalar divide
  prec = tp / (tp + fp)
```

In [ ]:
```
# Find the outliers in the training set
outliers = p_high < epsilon_high

# Visualize the fit
visualize_fit(X_train_high, mu_high, var_high)

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