ICT607: Artificial Intelligence for Cybersecurity

Experiment 11

1 Clustering domain URLs using K-means clustering algorithm

Spam domains are malicious or suspicious domain names used for various illicit activities, including phishing, malware distribution and spam email campaigns. Clustering these domains helps identify patterns, similarities and clusters of spam domains, enabling the development of more effective detection and mitigation strategies.

By applying clustering algorithms to spam domain datasets, we can group similar domains together based on various features such as domain names, registration dates, IP addresses or textual content. This clustering process helps in identifying common characteristics and patterns that are prevalent among spam domains.

K-means clustering is a popular unsupervised machine learning algorithm used for partitioning data into distinct groups based on similarities. It aims to minimise the variance within each cluster by iteratively assigning data points to the nearest cluster centroid and updating the centroids. K-means is a simple and efficient algorithm that works well for large datasets. However, it requires specifying the number of clusters (K) in advance.

1.1 Dataset

To demonstrate both the cluster generation and cluster scoring steps, we will work with a labeled dataset of internet domain names. The good names are the top 500,000 Alexa sites from May 2014, and the bad names are 13,789 "toxic domains".

Download the datasets and keep it on your Google drive folder (e.g., `Colab Notebooks/AICS/` folder)

[1]:
```
import numpy as np
```

[2]:
```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

[3]:
```
good_domains = []
toxic_domains = []
```

[4]:
```
# Read good_domains.txt
with open('/content/drive/MyDrive/Colab Notebooks/AICS/good_domains.txt', 'r')␣
↪as f:
```

```
  good_domains = f.read().splitlines()

# Read toxic_domains_whole.txt
with open('/content/drive/MyDrive/Colab Notebooks/AICS/toxic_domains_whole.
 ↪txt', 'r') as f:
  toxic_domains = f.read().splitlines()
```

[5]:
```
print("Ten samples of good domains: ", good_domains[1000:1005])
print("Ten samples of toxic domains: ", toxic_domains[1000:1005])
```

```
Ten samples of good domains:  ['101prikaz.ru', '101razasdeperros.com',
'101recipe.com', '101secureonline.com', '101shans.ru']
Ten samples of toxic domains:  ['allaboutemarketing.info',
'allaboutlabyrinths.com', 'alladyn.unixstorm.org', 'allairjordanoutlet.us',
'allairmaxsaleoutlet.us']
```

[6]:
```
print("Number of good domains: ", len(good_domains))
print("Number of toxic domains: ", len(toxic_domains))
```

```
Number of good domains:  500000
Number of toxic domains:  13789
```

1.2 Feature extraction

Convert the domain URLs into numerical features using a bag-of-words representation. In this representation, we'll consider each unique word in the domain URLs as a feature. We'll use scikit-learn's CountVectorizer for this purpose.

[7]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

[8]:
```
# Combine the good and toxic domain lists, because the algorithm doesn't␣
 ↪suppose to know beforehand which one is good which one is toxic
all_domains = good_domains + toxic_domains
```

[9]:
```
# Create an instance of CountVectorizer
vectoriser = CountVectorizer()
```

[10]:
```
# Fit and transform the data to obtain the feature matrix
features = vectoriser.fit_transform(all_domains)
```

1.3 Applying K-means Clustering

Now, we can apply the K-means clustering algorithm to the feature matrix obtained in the previous step. We'll use scikit-learn's KMeans class for this task.

[11]:
```
from sklearn.cluster import KMeans
```

[12]:
```
# Specify the number of clusters (K)
K = 2
```

[13]:
```
# Create an instance of KMeans
kmeans = KMeans(n_clusters=K, random_state=0)
```

[14]:
```
# Fit the model to the feature matrix
kmeans.fit(features)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
```

[14]:
```
KMeans(n_clusters=2, random_state=0)
```

1.4 Analysing the Clustering Results

After clustering, we can analyse the results to see which domains belong to which cluster.

[15]:
```
# Get the cluster labels for each domain
labels = kmeans.labels_
```

[16]:
```
# Separate the domains into their respective clusters
good_domains_cluster = []
toxic_domains_cluster = []
```

[17]:
```
for i, domain in enumerate(all_domains):
  if labels[i] == 0:
    good_domains_cluster.append(domain)
  else:
    toxic_domains_cluster.append(domain)
```

1.5 Saving clusters

[18]:
```
# Specify the output file paths
good_domains_file = "good_domains_cluster.txt"
toxic_domains_file = "toxic_domains_cluster.txt"
```

[19]:
```
# Open the output files in write mode
with open(good_domains_file, 'w') as f:
  # Write the good domains to the file
  f.write('\n'.join(good_domains_cluster))

with open(toxic_domains_file, 'w') as f:
  f.write('\n'.join(toxic_domains_cluster))
```

1.6 Calculate clustering performacne

In this code snippet, we use the `accuracy_score` function from scikit-learn's metrics module to calculate the accuracy. We pass the ground truth labels (ground_truth_labels) and the cluster labels obtained from K-means (labels) as input to the `accuracy_score` function.

[20]:
```
from sklearn.metrics import accuracy_score

# Create ground truth labels
ground_truth_labels = [0] * len(good_domains) + [1] * len(toxic_domains)

# Calculate accuracy
accuracy = accuracy_score(ground_truth_labels, labels)

# Print the accuracy value
print("Accuracy:", accuracy)
```

```
Accuracy: 0.4107386495234425
```

1.7 Practice task

Instead of the whole dataset, use only the last 5000 domains from each dataset (`good_domains.txt` and `toxic_domains_whole.txt`). Cluster the good and toxic domains and report accuracy, precision and recall.