Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут»

# Проектна робота
# з аналізу кібер інцидентів методами машинного навчання

**Виконали:**
Приходько Андрій
& Шахова Катерина

**Перевірила:**
Наконечна Ю.В.

**Київ 2023**

**Теоретична частина. Класифікація та кластеризація.**

У цій частині навести короткий опис принципу роботи алгоритма.

1. ***Алгоритм Байєса***

Алгоритм Байєса - це метод прогнозування ймовірності події, використовуючи попередні знання або свідчення. Через його основу на умовній ймовірності, алгоритм визначає, що ймовірність події залежить від появи інших подій. В машинному навчанні алгоритм Байєса часто використовується для прогнозування ймовірності конкретної події або результату на основі даних і статистичних моделей. Цей підхід є популярним в наглядового навчання, де модель навчається на закріплених даних, а потім використовується для прогнозування значень нових, невідомих прикладів. Завдяки алгоритму Байєса, ми можемо прогнозувати ймовірність виникнення різних подій на основі наявних доказів чи даних. Наприклад, при використанні спам-фільтрів, ми можемо застосовувати алгоритм Байєса для обчислення ймовірності того, що електронний лист є спамом, враховуючи певні особливості листа, такі як конкретні слова або фрази. Алгоритм Байєса має багато варіацій і він є основним будівельним блоком багатьох алгоритмів та підходів у сфері машинного навчання

2. ***Метод knn***

Алгоритм k-найближчих сусідів (k-NN) - це метод класифікації та регресії, який використовується в машинному навчанні. Принцип роботи алгоритму k-NN базується на тому, що близькі екземпляри даних зазвичай мають подібні значення вихідної змінної.

Для класифікації або прогнозування результатів, алгоритм k-NN використовує набір тренувальних даних з відомими мітками класів. При передбаченні нового екземпляра даних, алгоритм визначає k найближчих сусідів з тренувального набору даних. Класифікація або значення вихідної змінної для нового екземпляра даних визначається за допомогою більшості класів або середнього значення вихідних змінних k найближчих сусідів.

В алгоритмі k-NN, параметр k представляє кількість найближчих сусідів, які використовуються для прийняття рішення. Вибір оптимального значення k є важливим кроком в роботі алгоритму k-NN. Також, для визначення відстані між прикладами використовуються різні метри, такі як евклідова відстань або манхеттенська відстань.

Алгоритм kNN простий для реалізації і може бути ефективним у випадках, коли дані мають просту структуру або вимагають нелінійних відношень між змінними. Однак, важливо вибирати правильні значення k та враховувати особливості даних при застосуванні алгоритму k-NN

### *3. Лінійна/логістична регресія*

Лінійна та логістична регресія - це методи в машинному навчанні, які використовуються для прогнозування значень вихідної змінної на основі вхідних даних.

Лінійна регресія використовує лінійну функцію, щоб побудувати модель, яка найкращим чином підігнатиметься до тренувальних даних. Модель лінійної регресії шукає оптимальні значення ваг і зміщення, щоб мінімізувати помилку між прогнозованими значеннями і справжніми значеннями. Використовуючи цю модель, можна прогнозувати неперервні числові значення.

Логістична регресія використовує логістичну функцію для моделювання ймовірності виникнення певної події. Зазвичай цей метод використовується для задач класифікації, де потрібно визначити, до якого класу належить кожен екземпляр даних. Відповідно до значень вхідних змінних, логістична регресія визначає ймовірність належності до певного класу шляхом використання оптимальних значень ваг і зміщення.

Як лінійна, так і логістична регресія є простими для реалізації і інтерпретації моделей, а також можуть бути ефективними для розв'язку різних задач машинного навчання залежно від природи даних.

### *4. Метод опорних векторів*

Метод опорних векторів (SVM) - це метод машинного навчання, що використовується для класифікації та регресії. Основна мета SVM - побудова гіперплощини (яка може бути лінійною або нелінійною), яка розділяє екземпляри даних різних класів у просторі з максимальними маржиналізаціями. У випадку класифікації, SVM намагається знайти оптимальну гіперплощину. Це досягається шляхом знаходження такої гіперплощини, для якої найближчі точки (опорні вектори) відносяться до різних класів найближчим чином. Це дозволяє забезпечити хорошу узагальнюючу здатність моделі при нових, невідомих прикладах даних.

У випадку регресії, SVM ставить за мету знайти гіперплощину, яка найкращим чином підігнатиметься до тренувальних даних. За допомогою цієї гіперплощини, можна прогнозувати числові значення вихідної змінної для нових прикладів даних.

SVM є потужним методом машинного навчання, який може бути ефективним у вирішенні задач класифікації та регресії. Він добре працює як для лінійно роздільних даних, так і для даних з нелінійними залежностями, за допомогою використання ядерних функцій для перетворення вхідних змінних в вищорозмірний простір

### *5. C4.5 та CART*

C4.5 і CART є двома відомими алгоритмами для побудови дерев рішень в машинному навчанні.

C4.5 (Classification and Regression Trees) - це алгоритм, розроблений Россом Куїнланом. Він використовується для класифікації та регресії. C4.5 використовується для побудови дерева рішень, де вузли представляють різні розбиття даних, а листки відповідають класифікації або прогнозуванню значень. Основна ідея C4.5 - вибір оптимального розбиття на кожному кроці, використовуючи показник, відомий як випереджена інформаційна вигода (Gini індекс або ентропія).

CART (Classification and Regression Trees) - це ще один алгоритм побудови дерев рішень, розроблений Лео Бріманом, Джеромом Фрідманом та Ріхардом Олшеджем. CART також використовується для класифікації та регресії. В CART дерево ітеративно розбивається на частини, використовуючи міри чистоти, такі як частка невідповідних класів (для класифікації) або середньоквадратична помилка (для регресії). CART може побудувати як бінарні, так і мультикласові дерева рішень.

Як C4.5, так і CART є популярними і широко використовуваними алгоритмами для побудови дерев рішень. Вони забезпечують просту та зрозумілу модель, яка може бути ефективно використана для класифікації та регресії. Вибір між цими двома алгоритмами з часом залежить від конкретної задачі та характеристик даних

### *6. Метод DBSCAN і A/B тестування*

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) - це алгоритм, що використовується для виявлення кластерів точок даних в наборі даних. Він працює шляхом визначення областей набору даних, які мають високу щільність точок даних, та групує ці точки даних у кластери.

A/B тестування - це статистичний метод, що використовується для порівняння результатів двох різних версій чогось. Воно часто використовується в машинному навчанні для порівняння продуктивності різних моделей або алгоритмів на наборі даних.

Під час A/B тестування набір даних випадковим чином поділяється на дві групи: група A та група B. Версія, яка тестується, застосовується до групи A, тоді як контрольна версія застосовується до групи B. Результати обох груп потім порівнюються

### *7. Аналіз головних компонент*

Аналіз головних компонент - це метод зменшення розмірності даних, який дозволяє знайти головні компоненти у вхідних даних. Головні компоненти є

лінійними комбінаціями вхідних ознак і вказують наявність основних закономірностей у даних.

АГК застосовується для виявлення та видалення зайвої інформації, аналізу впливу окремих ознак на результати дослідження та зменшення розмірності даних. Він розраховує головні компоненти, які зберігають найбільшу кількість дисперсії у вихідних даних, тим самим зберігаючи максимально можливу кількість інформації.

АГК може бути використаний для візуалізації великого обсягу даних, зменшення шуму у даних, полегшення подальшого аналізу та підготовки даних для моделювання. Після застосування АГК, дані можуть бути представлені у зручній для аналізу формі з меншою кількістю ознак, що допомагає виявити тенденції та відношення між даними.

### *8. Ієрархічна кластеризація*

Ієрархічна кластеризація - це метод об'єднання схожих об'єктів у кластери для створення ієрархічної структури. У цьому методі кожен об'єкт спочатку розглядається як окремий кластер, а потім поступово об'єднується з іншими кластерами на основі схожості між ними.

Ієрархічна кластеризація може бути двох типів: агломеративна і дивізійна. В агломеративній кластеризації кожен об'єкт спочатку розглядається як окремий кластер, а потім найбільш схожі кластери поступово об'єднуються до більших кластерів. У дивізійній кластеризації процес відбувається у зворотному напрямку, починаючи з одного великого кластера і розбиваючи його на менші кластери.

Процес ієрархічної кластеризації може бути візуалізований у вигляді дерева, яке називається дендрограмою. Дендрограма представляє послідовність з'єднань кластерів і відображає відстань між ними. Вона може бути використана для визначення оптимального числа кластерів або для подальшого аналізу структури даних

**Практична частина. Реалізація.**

***Постановка задачі:***

Виконати класифікацію/кластеризацію даних датасету за наведеними алгоритмами, додати лістинг програми.

1. ***Алгоритм Байєса***

## Lab 1 (Класифікатор Наївний Баєс)

In [18]:
```
mnb_classifier = MultinomialNB()#створимо класифікатор Наївного Баєса
mnb_classifier.fit(X_train_features, y_train)#вчимо класифікатор
```

Out[18]:
```
MultinomialNB()
```
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [19]:
```
y_pred = mnb_classifier.predict(X_test_features)#прогнозуємо тестові дані
```

In [20]:
```
print(classification_report(y_test,y_pred))# виводимо звіт класифікації

print('Accuracy of MNB classifier: {} %'.format(round(accuracy_score(y_test, y_pred) * 100),2))#виводимо загальну
```

```
              precision    recall  f1-score   support

           0       0.99      0.70      0.82       311
           1       0.95      1.00      0.98      1918

    accuracy                           0.96      2229
   macro avg       0.97      0.85      0.90      2229
weighted avg       0.96      0.96      0.95      2229

Accuracy of MNB classifier: 96 %
```

2. ***Метод knn***

In [25]:
```
knc = KNeighborsClassifier(n_neighbors=1)#створимо класифікатор кнн
knc.fit(X_train_features, y_train)#вчимо класифікатор
```

Out[25]:
```
KNeighborsClassifier(n_neighbors=1)
```
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [26]:
```
y_pred = knc.predict(X_test_features)#прогнозуємо тестові дані
```

In [27]:
```
print(classification_report(y_test,y_pred))# виводимо звіт класифікації

print('Accuracy of MNB classifier: {} %'.format(round(accuracy_score(y_test, y_pred) * 100),2))#виводимо загальну
```

```
              precision    recall  f1-score   support

           0       0.99      0.65      0.78       311
           1       0.95      1.00      0.97      1918

    accuracy                           0.95      2229
   macro avg       0.97      0.82      0.88      2229
weighted avg       0.95      0.95      0.95      2229

Accuracy of MNB classifier: 95 %
```

## 3. *Лінійна/логістична регресія*

# Logistic Regression

In [17]:
```
logistic_reg = LogisticRegression()
logistic_reg.fit(X_train_features, y_train.values)
```

Out[17]:
```
LogisticRegression()
```
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [18]:
```
y_pred_log = logistic_reg.predict(X_test_features)
```

In [19]:
```
print(classification_report(y_test,y_pred_log))
```

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      1465
           1       0.97      0.73      0.83       207

    accuracy                           0.96      1672
   macro avg       0.97      0.86      0.91      1672
weighted avg       0.96      0.96      0.96      1672
```

# Linear regression

In [21]:
```
linear_reg = LinearRegression()
linear_reg.fit(X_train_features, y_train.values)
```

Out[21]:
```
LinearRegression()
```
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [22]:
```
y_pred_lin = linear_reg.predict(X_test_features)
y_pred_lin = np.round(y_pred_lin).astype(int)
```

In [59]:
```

```

In [60]:
```
print(classification_report(y_test,y_pred_lin))
```

```
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.97      0.96      0.96      1465
           1       0.79      0.79      0.79       207
           2       0.00      0.00      0.00         0

    accuracy                           0.94      1672
   macro avg       0.44      0.44      0.44      1672
weighted avg       0.95      0.94      0.94      1672
```

## 4. *Метод опорних векторів*

### SVM

In [26]:
```
clf = svm.SVC()
```

In [27]:
```
clf.fit(X_train_features, y_train.values)
```

Out[27]:
```
SVC()
```
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [28]:
```
y_pred_svm = clf.predict(X_test_features)
```

In [29]:
```
print(classification_report(y_test,y_pred_svm))
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      1465
           1       1.00      0.86      0.92       207

    accuracy                           0.98      1672
   macro avg       0.99      0.93      0.96      1672
weighted avg       0.98      0.98      0.98      1672
```

## 5. *C4.5 та CART*

In [17]:
```
from sklearn.tree import DecisionTreeClassifier #CART

DT_classifier = DecisionTreeClassifier()#створимо класифікатор дерева рішень, що працює на основі CART
DT_classifier.fit(X_train_features, y_train)#вчимо класифікатор
```

Out[17]:
```
DecisionTreeClassifier()
```
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [20]:
```
y_pred = DT_classifier.predict(X_test_features)#прогнозуємо тестові дані

print(classification_report(y_test,y_pred))# виводимо звіт класифікації

print('Accuracy of MNB classifier: {} %'.format(round(accuracy_score(y_test, y_pred) * 100),2))#виводимо загальну
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      1475
           1       0.88      0.82      0.85       197

    accuracy                           0.97      1672
   macro avg       0.93      0.90      0.92      1672
weighted avg       0.97      0.97      0.97      1672

Accuracy of MNB classifier: 97 %
```

In [21]:
```
#побудуємо матрицю плутанини

plt.rcParams["figure.figsize"] = (8,8)

cm = confusion_matrix(y_test, y_pred, labels=DT_classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=data.Category.unique())
disp.plot()
plt.title("ConfusionMatrix for MNB-classifier")

plt.show()
```

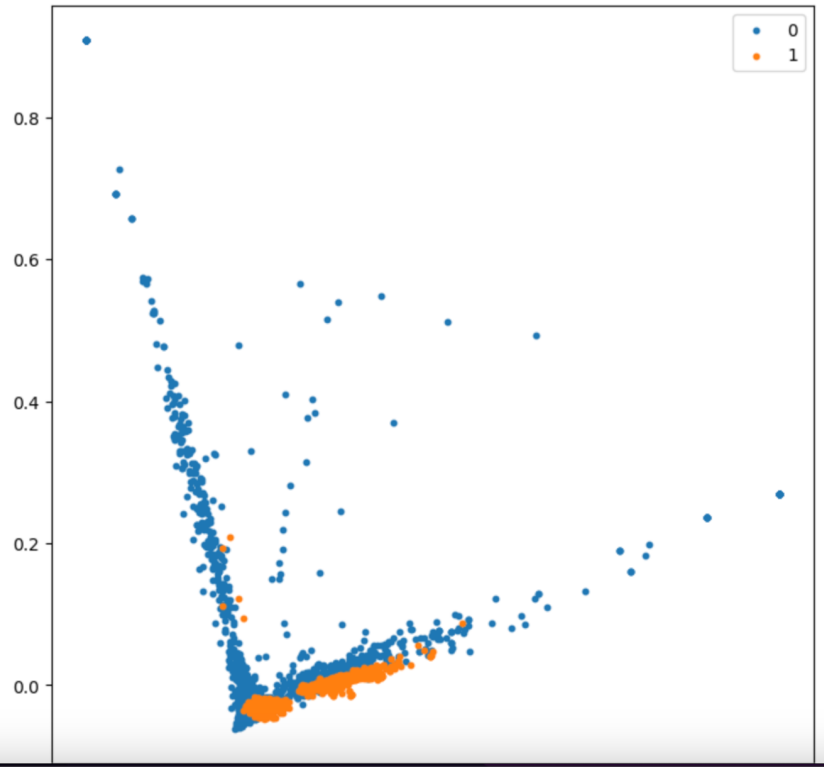
## 6. *Метод DBSCAN і A/B тестування*

## DB Scan

In [22]:
```
vectorizer = TfidfVectorizer()#створимо векторизатор
X_features = vectorizer.fit_transform(X).toarray()#векторизуємо дані

# Перетворимо всі дані у розмірність 2

from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_features)
```

In [23]:
```
#Візуалізуємо розподіл точок у просторі


for i in data.Category.unique():
    plt.scatter(X_pca[data['Category'] == i, 0],
                X_pca[data['Category'] == i, 1],
                s=10, label = i)
plt.legend()
plt.show()
```

In [23]:
```
#Візуалізуємо розподіл точок у просторі


for i in data.Category.unique():
    plt.scatter(X_pca[data['Category'] == i, 0],
                X_pca[data['Category'] == i, 1],
                s=10, label = i)
plt.legend()
plt.show()
```



In [25]:

```
BUCKETS = 64 #Розмір пакету

def get_v_measure_score(df):
    return metrics.v_measure_score(df['Spam_label'], df['model_A']),metrics.v_measure_score(df['Spam_label'], df[

df = pd.DataFrame()
df['id'] = range(y.shape[0])
df['Spam_label'] = y.replace(1,-1) #бо DBSCAN повертає викиди як -1 (СПАМ)

#  модель 1
model_A = DBSCAN(eps=1, min_samples = 3)
model_A.fit(X_features)
df['model_A'] = model_A.labels_
df['model_A'] = df['model_A'].replace()

#  модель 2
model_B = DBSCAN(eps=3, min_samples=5)
model_B.fit(X_features)
df['model_B'] = model_B.labels_


# рахуємо v_measure_score по бакетам
df['bucket'] = pd.util.hash_pandas_object(df['id'], index=False) % BUCKETS
tmp = df.groupby(['bucket']).apply(get_v_measure_score).reset_index().rename(columns={0:'v_measure_score'})

tmp[['v_measure_score_A', 'v_measure_score_B']] = tmp['v_measure_score'].tolist()
stat, p_val = ttest_ind(tmp['v_measure_score_B'], tmp['v_measure_score_A'], equal_var=False)

print('stat: ', stat, '\np_val:', round(p_val, 5))
```

```
/var/folders/_c/hfddj3lj4gg6_1zrnr5bsbb40000gp/T/ipykernel_2001/3576424259.py:14: FutureWarning: Series.replace wit
hout 'value' and with non-dict-like 'to_replace' is deprecated and will raise in a future version. Explicitly speci
fy the new values instead.
  df['model_A'] = df['model_A'].replace()
stat:  -6.263532349885574
p_val: 0.0
```

In [26]:

```
# Визначимо дані, які можуть бути спамом
spam_data = df[df['model_A'] == -1].shape[0]
print("Number of spam messages:", spam_data)
```

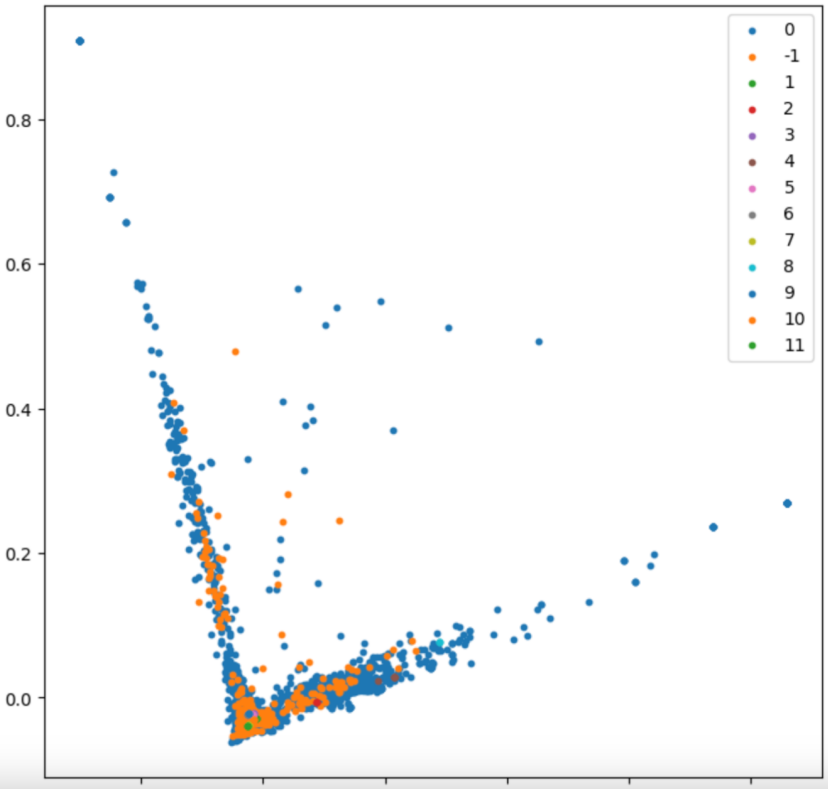
```
Number of spam messages: 729
```

In [27]:

```
# Визначимо дані, які можуть є спамом і розпізнані як спам
spam_data = df[(df['model_A']==-1) & (df['Spam_label']== -1)].shape[0]
print("Number of detected spam messages:", spam_data)
```

```
Number of detected spam messages: 74
```

In [28]:
```
# візуалізуємо. -1 – це спам, інші кластери – якісь специфічні поідомлення (можливо одна теми ітд)

for i in df['model_A'].unique():
    plt.scatter(X_pca[df['model_A'] == i, 0],
                X_pca[df['model_A'] == i, 1],
                s=10, label = i)
plt.legend()
plt.show()
```




## 7. *Аналіз головних компонент*

### PCA

In [12]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

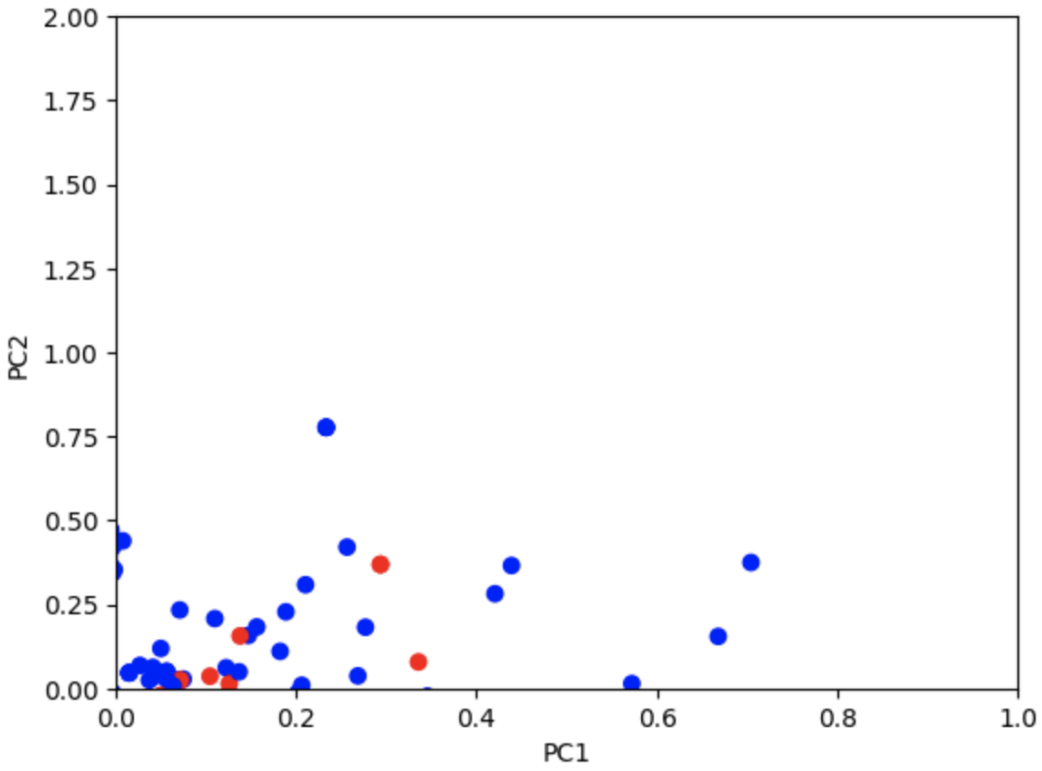
In [13]:
```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(data['Message'])
```

In [14]:
```
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X.toarray())
```

In [15]:
```
pca = PCA(n_components=2)  # Choose the number of components you want to retain
X_pca = pca.fit_transform(X_normalized)
```

In [16]:
```
pca_df = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])
pca_df['Category'] = data['Category']
```

In [17]:
```
plt.scatter(pca_df['PC1'], pca_df['PC2'], c=pca_df['Category'].map({'ham': 'blue', 'spam': 'red'}))
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.xlim(0, 1)
plt.ylim(0, 2)
plt.show()
```


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
PC2
0.0
0.2
0.4
0.6
0.8
1.0
PC1

## 8. *Ієрархічна кластеризація*

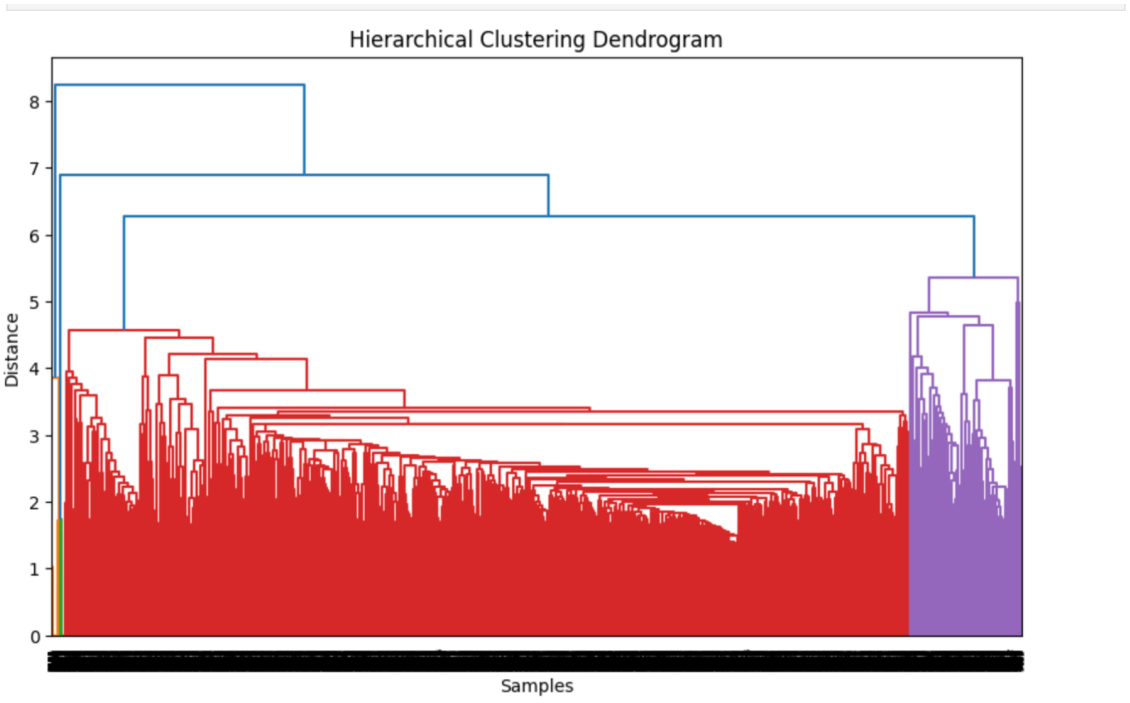
### Hierarchical Clustering

In [18]:
```
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [19]:
```
# Vectorize the text messages using TF-IDF
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(data['Message'])
```

In [20]:
```
# Perform hierarchical clustering
linked = linkage(X.toarray(), 'ward')
```

In [21]:
```
plt.figure(figsize=(10, 6))
dendrogram(linked)
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.title('Hierarchical Clustering Dendrogram')
plt.show()
```



**Практична частина. Аналіз та інтерпретація результатів.**

У цій частині навести переваги та недоліки алгоритмів, можливості використання, на яких даних працює краще/гірше, випадки не застосовності алгоритма (якщо є), які проблеми виникали в процесі роботи і шляхи їх розв’язання, що може вплинути на точність роботи алгоритма та яким чином.

1. ***Алгоритм Байєса***

Алгоритм Байєса - статистичний алгоритм, простий та інтерпретований, працює добре з невеликою кількістю чітких ознак. Залежить від припущень про незалежність ознак і може мати проблеми з класифікацією сумішей класів. Використовується в багатьох сферах, може бути непридатним без достатньої кількості даних або відсутністю незалежності ознак. Розв'язання проблем включає покращення оцінки ймовірностей та використання інших алгоритмів. Точність залежить від якості даних, припущень, ознак і апріорних ймовірностей

2. ***Метод knn***

Алгоритм k-найближчих сусідів (k-NN) - простий алгоритм класифікації, який заснований на пошуку k найбільш близьких сусідів для нових даних. Він не робить припущень про розподіл даних та може працювати з різними типами ознак. Однак, він може бути чутливим до шуму, вимагає багато додаткової пам'яті та потенційно повільний на великих даних. Краще працює з великою кількістю даних і більш узгодженими класами. Вимагає відповідного підготування даних та настроїв k-параметра. Точність залежить від вибору відстані та k-параметра, а також якості даних та унікальності класів.

3. ***Лінійна/логістична регресія***

Лінійна регресія - статистичний алгоритм, що встановлює лінійну залежність між вхідними ознаками і вихідними значеннями. Вона проста у реалізації та інтерпретованість, а також працює добре з великою кількістю ознак, коли є лінійна залежність. Однак, вона може бути неефективною, коли залежність не є лінійною.

Логістична регресія - алгоритм класифікації, який використовує логістичну функцію для прогнозування ймовірностей приналежності до класу. Вона також проста у реалізації та інтерпретованість, і працює добре з бінарними класами. Однак, вона може мати проблеми з вирішенням задач з багатьма класами або коли немає логістичної залежності між ознаками і класами.

Точність лінійної та логістичної регресії залежить від якості даних, залежностей між ознаками та вихідними значеннями, а також від відповідних параметрів моделі. Важливо враховувати особливості даних та природу задачі при виборі між цими алгоритмами

### *4. Метод опорних векторів*

Переваги SVM:

- Добра точність класифікації, особливо при використанні ядерних функцій.
- Можливість працювати з великою кількістю ознак.
- Універсальність - використання для бінарної та багатокласової класифікації.

Недоліки SVM:

- Вимогливість до обчислювальних ресурсів, особливо для великих наборів даних.
- Складності з інтерпретованістю результатів.
- Потенційна проблема з вибором та налаштуванням параметрів.
- Не ефективний у випадках, де є багато шуму або накладання класів.

SVM часто працює добре на данних з чітко визначеними границями між класами та коли дані мають багато ознак. Однак, він може бути менш ефективним на даних з багато шуму або коли границі між класами не є чіткими.

У процесі роботи з SVM можуть виникати проблеми, такі як медіоцизиращі неправильно класифіковані точки або неякісні дані. Для вирішення цих проблем можна виконати налаштування параметрів, збільшити кількість потрібних даних чи застосувати методи попередньої обробки даних.

Точність SVM може впливати на кілька способів, таких як вибір ядра, налаштування параметрів C та gamma, якість та однорідність даних. Важливо провести належний аналіз та налаштування алгоритму, щоб забезпечити оптимальні результати

### *5. C4.5 та CART*

CART є простим та легко інтерпретованим, що дозволяє легко розуміти та пояснювати отримане дерево. Він може працювати з різними типами даних та ознак і відображати складні залежності та взаємодії. Однак він схильний до перенавчання, особливо при використанні глибоких дерев або великої кількості листків. Він також чутливий до шуму та незначних змін в даних. Вибір оптимального критерію поділу може бути складним. CART працює найкраще з великими наборами даних і коли залежності між ознаками є добре узгодженими. Проблеми включають перенавчання, проблеми з точністю.

### *6. Метод DBSCAN і A/B тестування*

Він не вимагає вказування кількості кластерів перед виконанням і може виявити як стандартні кластери з однаковими щільностями, так і зшивання та шум.

Переваги DBSCAN включають незалежність від кількості кластерів, здатність знаходити кластери різної форми та перетини між ними, інтегрування шуму та дозвіл на автоматичну визначення щільності.

Недоліки DBSCAN включають залежність від вибору параметрів, таких як радіус і мінімальна кількість сусідів, труднощі з роботою з даними з високою розрідженістю, а також вразливість до неоднорідності щільності даних.

DBSCAN є корисним алгоритмом для виявлення кластерів у даних, зокрема у випадках, коли форма та кількість кластерів невідомі або коли є шум. Важливо належним чином встановити параметри для досягнення оптимальних результатів при використанні DBSCAN

### *7. Аналіз головних компонент*

Переваги PCA включають:

- Зменшення розмірності даних, що дозволяє простіше візуалізувати та аналізувати дані.

- Видалення зайвої кореляції між ознаками.

- Збереження найбільшої частини інформації.

Недоліки PCA включають:

- Втрата неналежного розуміння змінних, оскільки нові головні компоненти є лінійними комбінаціями початкових ознак.

- Неможливість пояснити головні компоненти людям, оскільки вони є абстрактними конструкціями, які важко зв'язати з реальними ознаками.

### *8. Ієрархічна кластеризація*

Переваги ієрархічної кластеризації включають:

- Здатність візуалізувати кластерну структуру у вигляді дендрограми.

- Відсутність необхідності визначати кількість кластерів заздалегідь.

- Здатність працювати з різноманітними типами даних та враховувати різні міри відстані або схожості.

Недоліки ієрархічної кластеризації включають:

- Висока обчислювальна складність на великих наборах даних.

- Висока чутливість до видалення або додавання нових об'єктів.

- Важкість інтерпретації результатів, зокрема при великому числі кластерів

**Висновки**

Як і де можна застосувати результати вашої роботи.

Розробка спам-детектора методами машинного навчання - це надзвичайно цікава і корисна задача, яка має кілька важливих переваг.

По-перше, використання машинного навчання дозволяє побудувати модель, яка може ефективно аналізувати великі обсяги текстових даних та впевнено розпізнавати спам. Це дозволяє забезпечити швидку та ефективну фільтрацію небажаних повідомлень. Крім того, машинне навчання також надає велику гнучкість у виборі методів та алгоритмів, що дозволяє оптимально налаштувати спам-детектор під конкретні потреби.

По-друге, спам-детектори, можуть бути персоналізованими під потреби конкретного користувача. Це означає, що, враховуючи статистику та відмітки користувача, модель може вчитися визначати, які повідомлення він вважає спамом, а які - ні. Це забезпечує більш точну та індивідуалізовану фільтрацію.
Використання спам-детектора, розробленого методами машинного навчання, може бути широко застосоване. Наприклад, такі спам-детектори можуть бути використані в електронній пошті, соціальних мережах, на веб-сайтах, а також в мобільних додатках. Вони допоможуть користувачам фільтрувати небажані коментарі, повідомлення, рекламний спам та інші форми небажаної активності.

Отже, спам-детектор методами машинного навчання є потужним інструментом, який забезпечує безпеку та комфорт користувачів у сфері електронних комунікацій та онлайн-платформ. Використання таких спам-детекторів може стати важливою складовою для боротьби зі спамом та забезпечення безпеки в інтернет-середовищі.