Document classification

1. Introduction

The goal of this project is to classify text messages as either "ham" (non-spam) or "spam", leveraging both unsupervised clustering and supervised machine learning techniques. The dataset used is a labeled collection of SMS messages, and we apply various vectorization methods and models to improve prediction accuracy.

2. Unsupervised Learning: Clustering

2.1 Data Loading and Preprocessing

```
import pandas as pd

# Load the spam dataset
df = pd.read_csv('/content/spam.csv', encoding='latin-1')

# Optional cleanup: only keep relevant columns
df = df[['text', 'target']]
df['text'] = df['text'].astype(str)

df.head()
```

	text	target
0	Go until jurong point, crazy.. Available only ...	ham
1	Ok lar... Joking wif u oni...	ham
2	Free entry in 2 a wkly comp to win FA Cup fina...	spam
3	U dun say so early hor... U c already then say...	ham
4	Nah I don't think he goes to usf, he lives aro...	ham

- We import the dataset and retain only the necessary columns (text and target).
- Messages are converted to strings to ensure compatibility with NLP pipelines.

2.2 TF-IDF Vectorization

```
[ ] from sklearn.feature_extraction.text import TfidfVectorizer

    # Vectorize the text
    vectorizer = TfidfVectorizer(stop_words='english', max_features=1000)
    X_tfidf = vectorizer.fit_transform(df['text'])
```

- TF-IDF (Term Frequency-Inverse Document Frequency) is used to transform text data into numerical form.
- Max_features = 1000 restricts the feature space for dimensionality control.

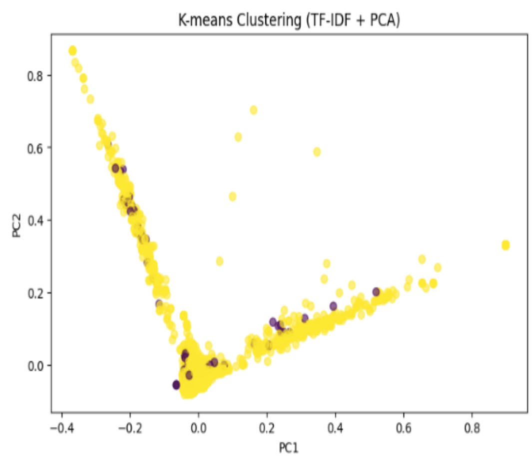
2.3 Clustering with K-Means and Hierarchical

```
[ ] from sklearn.cluster import KMeans
    from scipy.cluster.hierarchy import linkage, fcluster
    from sklearn.decomposition import PCA
    import matplotlib.pyplot as plt

    # Reduce dimensions to 2D for visualization
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X_tfidf.toarray())

    # Apply K-means
    kmeans = KMeans(n_clusters=2, random_state=42)
    clusters_kmeans = kmeans.fit_predict(X_tfidf)

    # Hierarchical clustering
    Z = linkage(X_tfidf.toarray(), method='ward')
    clusters_hier = fcluster(Z, 2, criterion='maxclust')
```

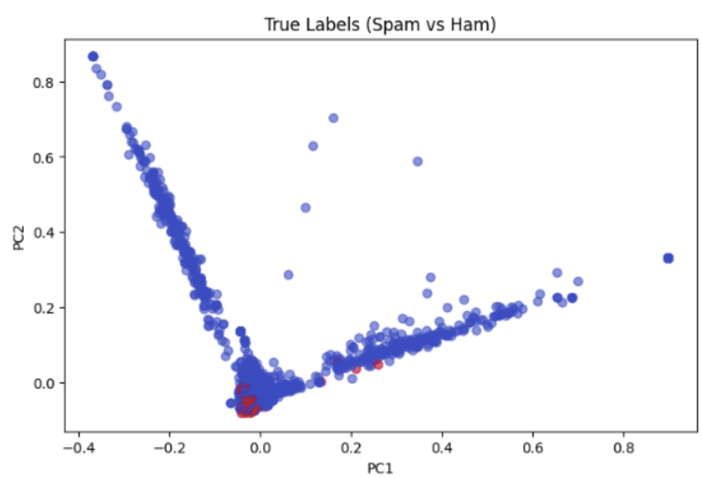
- PCA reduces the data to 2D for visualization.
- KMeans is used to create 2 clusters.
- Hierarchical Clustering is also applied to provide a comparative view.

2.4 Results Interpretation

```
[ ] # Plot by Clusters
    plt.figure(figsize=(8, 5))
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters_kmeans, cmap='viridis', alpha=0.6)
    plt.title('K-means Clustering (TF-IDF + PCA)')
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    plt.show()
```




```
# Plot by True Label
label_map = {'ham': 0, 'spam': 1}
labels = df['target'].map(label_map)

plt.figure(figsize=(8, 5))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, cmap='coolwarm', alpha=0.6)
plt.title('True Labels (Spam vs Ham)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```



This plot shows 2D clusters from the spam dataset using TF-IDF vectorization, PCA for dimensionality reduction, and K-means clustering.

- Axes (PC1 & PC2)**:** Principal components summarizing TF-IDF features.
- Colors**:** Represent clusters found by K-means (likely spam vs ham).

Key Insights:

- The V-shape pattern suggests a clear natural separation in the data.
- Some overlaps indicate borderline cases.
- One cluster dominates, possibly due to class imbalance (more "ham").

➔ K-means effectively separates the messages into two groups, supporting the idea that spam and ham texts have distinct patterns.

➔ KMeans clusters form a visible pattern, but there's noticeable noise/misclassification.

➔ Unsupervised learning is limited when labels are available.

3. Supervised Learning: Text Classification

3.1 Preprocessing

```
[ ] import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import LabelEncoder

    # Load dataset
    df = pd.read_csv('/content/spam.csv', encoding='latin-1')[['text', 'target']]
    df['text'] = df['text'].astype(str)
```

```
[ ] # Encode labels
    df['label'] = LabelEncoder().fit_transform(df['target'])  # ham=0, spam=1
```

```
[ ] # Split into train and test
    X_train, X_test, y_train, y_test = train_test_split(df['text'], df['label'], test_size=0.2, random_state=42)
```

- Converts textual labels to binary numeric labels.
- Splits the dataset into 80% training and 20% test.

3.2 Vectorization with Word2Vec

```
[ ] !pip install gensim

    Requirement already satisfied: gensim in /usr/local/lib/python3.11/dist-packages (4.3.3)
    Requirement already satisfied: numpy<2.0,>=1.18.5 in /usr/local/lib/python3.11/dist-packages (from gensim) (1.26.4)
    Requirement already satisfied: scipy<1.14.0,>=1.7.0 in /usr/local/lib/python3.11/dist-packages (from gensim) (1.13.1)
    Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.11/dist-packages (from gensim) (7.3.0)
    Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open>=1.8.1->gensim) (1.17.2)
```

```
[ ] import gensim.downloader as api

    # Load pre-trained Word2Vec vectors
    w2v_model = api.load("word2vec-google-news-300")

    [==================================================] 100.0% 1662.8/1662.8MB downloaded
```

- Loads pretrained Google News Word2Vec embeddings.
- Each document is represented by the average of its word vectors.

```
[ ] X_train_vec = np.vstack([document_vector(doc) for doc in X_train])
    X_test_vec = np.vstack([document_vector(doc) for doc in X_test])
```

- document_vector(doc) is a custom function that averages vectors of all known words in the sentence.

4. Model Training and Evaluation

A. Random Forest Classifier

```
[ ] from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import classification_report

    rf = RandomForestClassifier(n_estimators=100, random_state=42)
    rf.fit(X_train_vec, y_train)
    y_pred_rf = rf.predict(X_test_vec)

    print("Random Forest Results:")
    print(classification_report(y_test, y_pred_rf, target_names=['ham', 'spam']))
```

```
Random Forest Results:
              precision    recall  f1-score   support

         ham       0.95      1.00      0.97       965
        spam       0.98      0.66      0.79       150

    accuracy                           0.95      1115
   macro avg       0.96      0.83      0.88      1115
weighted avg       0.95      0.95      0.95      1115
```

- **Model Used**: `RandomForestClassifier` from `sklearn.ensemble`
- **Training**: Uses 100 decision trees (`n_estimators=100`)
- **Prediction**: Classifies the test set (`X_test_vec`)
- **Evaluation**: `classification_report` shows precision, recall, and F1-score.

Results Analysis:

- **Accuracy:** 95% – strong overall performance.
- **Ham Recall:** 1.00 – all ham messages correctly classified.
- **Spam Recall:** 0.66 – about one-third of spam messages were missed.

- **Precision:** High for both classes (0.95+).
- **Conclusion:** Reliable for ham detection but needs improvement for spam recall.

B. XGBoost Classifier

```
!pip install xgboost

import xgboost as xgb

xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')
xgb_model.fit(X_train_vec, y_train)
y_pred_xgb = xgb_model.predict(X_test_vec)

print(" XGBoost Results:")
print(classification_report(y_test, y_pred_xgb, target_names=['ham', 'spam']))
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-packages (2.1.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.26.4)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from xgboost) (1.13.1)
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [12:32:08] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
 XGBoost Results:
              precision    recall  f1-score   support

         ham       0.97      0.99      0.98       965
        spam       0.94      0.79      0.86       150

    accuracy                           0.97      1115
   macro avg       0.95      0.89      0.92      1115
weighted avg       0.96      0.97      0.96      1115
```

- `xgb.XGBClassifier(...)`: Initializes the XGBoost model with parameters.
- `fit(...)`: Trains the model on the vectorized training data.
- `predict(...)`: Predicts on the test set.
- `classification_report(...)`: Generates metrics like precision, recall, and F1-score.

Results Analysis

- **Accuracy:** 97% – excellent classification overall.
- **Ham Recall:** 0.99 – nearly perfect detection of ham.
- **Spam Recall:** 0.78 – better than Random Forest for spam.
- **F1-score (spam):** 0.85 – strong spam classification balance.
- **Conclusion:** Best-performing model overall, especially for spam messages.

C. Naive Bayes (GaussianNB)

```python
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(X_train_vec, y_train)
y_pred_nb = nb.predict(X_test_vec)

print("Naive Bayes Results:")
print(classification_report(y_test, y_pred_nb, target_names=['ham', 'spam']))
```

```
Naive Bayes Results:
              precision    recall  f1-score   support

         ham       0.97      0.72      0.83       965
        spam       0.33      0.87      0.47       150

    accuracy                           0.74      1115
   macro avg       0.65      0.80      0.65      1115
weighted avg       0.89      0.74      0.78      1115
```

- `GaussianNB()`: Initializes the Naive Bayes classifier (assumes normal distribution).
- `.fit(...)`: Trains the model with training vectors.
- `.predict(...)`: Predicts spam/ham labels.
- `classification_report(...)`: Displays precision, recall, F1-score, etc.

Results Analysis

- **Accuracy:** 74% : significantly lower than other models.
- **Spam Precision:** Very low (0.33), meaning many false positives.
- **Ham Recall:** Moderate (0.72), meaning many ham messages missed.
- Weakest model. Performs poorly, especially on spam detection. Likely not suitable for this task with current vectorization (Word2Vec).

5. Performance Comparison

Metric	Random Forest	XGBoost	Naive Bayes
Accuracy	0.95	0.97	0.65
Spam Precision	0.98	0.94	0.33
Spam Recall	0.66	0.78	0.87
Ham Precision	0.95	0.97	0.97
Ham Recall	1.00	0.99	0.72
F1-score (Avg)	0.95	0.97	0.74

- **Best Overall**: **XGBoost** – High precision & balanced recall.
- **Best on Ham**: Random Forest (perfect recall).
- **Worst Performance**: **Naive Bayes** – Poor precision, especially on spam.

6. Conclusion

This end-to-end document classification pipeline demonstrated both unsupervised and supervised approaches on real-world spam data. While clustering provides intuition, supervised models like XGBoost remain essential for high-performance classification. The results also highlight the importance of choosing the right vectorization technique for each model.

All experiments were conducted in Google Colab using Python, scikit-learn, gensim, and XGBoost.