```
!pip install wget
```

```
Requirement already satisfied: wget in /usr/local/lib/python3.10/dist-
packages (3.2)
```

Downloading and extracting the dataset.

```
import wget
import tarfile
import ssl
import os

ssl._create_default_https_context = ssl._create_unverified_context

dataset_url =
"http://www.aueb.gr/users/ion/data/lingspam_public.tar.gz"

destination_dir = "ling_spam_dataset"

os.makedirs(destination_dir, exist_ok=True)

# Downloading the dataset
wget.download(dataset_url, out=destination_dir)

dataset_path = os.path.join(destination_dir, "lingspam_public.tar.gz")
with tarfile.open(dataset_path, "r:gz") as tar:
    tar.extractall(destination_dir)

os.remove(dataset_path)

import os
import pandas as pd

# Defining the path to the data directory
data_dir = 'ling_spam_dataset/lingspam_public/lemm_stop'

email_content = []
labels = []
fold_numbers = []

# Traversing through the directories and reading the files
for folder in os.listdir(data_dir):
    folder_path = os.path.join(data_dir, folder)

    if os.path.isdir(folder_path):
        # Infer fold number from folder name
        if folder.startswith("part"):
            fold_number = int(folder[4:])
        else:
            fold_number = int(folder)
```

```
        for file in os.listdir(folder_path):
            file_path = os.path.join(folder_path, file)

            # Reading the content in the file
            with open(file_path, 'r', encoding='latin-1') as f:
                content = f.read()

            # Determining the label
            label = 1 if "spmsg" in file else 0

            email_content.append(content)
            labels.append(label)
            fold_numbers.append(fold_number)

df = pd.DataFrame({
    'text': email_content,
    'label': labels,
    'fold': fold_numbers
})

# Train and test sets
df_test = df[df['fold'] == 10]
df_train = df[df['fold'] != 10]

print("Total dataset length:", len(df))
print("Training set length:", len(df_train))
print("Test set length:", len(df_test))

print("First 5 rows of the test set:")
print(df_test.head())

print("First 5 rows of the train set:")
print(df_train.head())
```

```
Total dataset length: 2893
Training set length: 2602
Test set length: 291
First 5 rows of the test set:
                                                  text  label  fold
579  Subject: vilem mathesius lecture sery 13 - pra...      0    10
580  Subject: anglo - american study\n\nfirst annou...      0    10
581  Subject: sigphon98 workshop - - - extend deadl...      0    10
582  Subject: program & info : workshop comparative...      0    10
583  Subject: whole part\n\nwhole part ( w / p ) bo...      0    10
First 5 rows of the train set:
                                                text  label  fold
0  Subject: summary name day\n\ndear linguists , ...      0     3
1  Subject: summary : adpositional ' eye '\n\nres...      0     3
2  Subject: nlp\n\nreader : recently send several...      0     3
```

```
3  Subject: sum : imperative without subject\n\nc...      0     3
4  Subject: sum : register pre-school age\n\nsumm...      0     3
```

Information gain and Entropy.

```
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer

# Information gain
def calculate_information_gain(texts, labels):

    # Converting the texts to term frequency
    vectorizer = CountVectorizer(binary=True)
    term_frequency = vectorizer.fit_transform(texts)
    terms = vectorizer.get_feature_names_out()

    #  Entropy
    def calculate_entropy(prob_spam):
        prob_legitimate = 1 - prob_spam
        if prob_spam == 0 or prob_legitimate == 0:
            return 0
        return -prob_legitimate * np.log2(prob_legitimate) - prob_spam
* np.log2(prob_spam)

    # Overall entropy
    prob_spam = np.mean(labels)
    total_entropy = calculate_entropy(prob_spam)

    information_gain_values = {}

    # Iterating through each term
    for i, term in enumerate(terms):
        term_presence = term_frequency[:, i].toarray().flatten() == 1

        # Separating labels into spam and legitimate based on term
presence
        labels_with_term = labels[term_presence]
        labels_without_term = labels[~term_presence]

        # Calculating conditional entropy for labels with and without
the term
        prob_with_term = np.mean(term_presence)
        prob_without_term = 1 - prob_with_term
        entropy_with_term =
calculate_entropy(np.mean(labels_with_term))
        entropy_without_term =
calculate_entropy(np.mean(labels_without_term))

        # Calculate information gain
        conditional_entropy = prob_with_term * entropy_with_term +
```

```
prob_without_term * entropy_without_term
        information_gain = total_entropy - conditional_entropy

        information_gain_values[term] = information_gain

    return information_gain_values

X_train = df_train['text']
y_train = df_train['label']

information_gain_values = calculate_information_gain(X_train, y_train)

# Rank terms based on information gain
sorted_terms = sorted(information_gain_values,
key=information_gain_values.get, reverse=True)

print("Top 10 terms based on Information Gain:")
for term in sorted_terms[:10]:
    print(term)
```

```
Top 10 terms based on Information Gain:
language
remove
free
linguistic
click
business
advertise
company
sell
english
```

```
# Top terms for different N values (10, 100, and 1000)
top_10_features = sorted_terms[:10]
top_100_features = sorted_terms[:100]
top_1000_features = sorted_terms[:1000]

# Top 10 terms with their information gain values
top_10_ft = [(term, information_gain_values[term]) for term in
sorted_terms[:10]]

for term, info_gain in top_10_ft:
    print(f"Term: {term}, Information Gain: {info_gain:.4f}")
```

```
Term: language, Information Gain: 0.2060
Term: remove, Information Gain: 0.1689
Term: free, Information Gain: 0.1632
Term: linguistic, Information Gain: 0.1532
Term: click, Information Gain: 0.1023
Term: business, Information Gain: 0.0868
Term: advertise, Information Gain: 0.0780
```

```
Term: company, Information Gain: 0.0770
Term: sell, Information Gain: 0.0768
Term: english, Information Gain: 0.0747
```

Naive Bayes Classifiers.

```
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from sklearn.metrics import precision_score, recall_score
import time
from tabulate import tabulate

def evaluate_classifier(classifier, X_train, y_train, X_test, y_test):
    # Timing the training and prediction process
    start_time = time.time()
    clf = classifier.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    latency = time.time() - start_time

    # Calculating precision and recall
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    return precision, recall, latency

train_data = df[df['fold'] < 10]
test_data = df[df['fold'] == 10]

N_values = [10, 100, 1000]
top_feature_sets = [top_10_features, top_100_features,
top_1000_features]
classifiers = [BernoulliNB(), MultinomialNB(), MultinomialNB()]
classifier_names = ['BernoulliNB', 'MultinomialNB_bin',
'MultinomialNB_tf']

results = []

# Looping over different feature sizes (N)
for i, N in enumerate(N_values):
    vectorizer_bin = CountVectorizer(binary=True,
vocabulary=top_feature_sets[i])
    X_train_bin = vectorizer_bin.fit_transform(train_data['text'])
    X_test_bin = vectorizer_bin.transform(test_data['text'])

    vectorizer_tf = CountVectorizer(vocabulary=top_feature_sets[i])
    X_train_tf = vectorizer_tf.fit_transform(train_data['text'])
    X_test_tf = vectorizer_tf.transform(test_data['text'])

    y_train = train_data['label']
    y_test = test_data['label']

    # Evaluating multiple classifiers for each feature set
```

```
    for j, classifier in enumerate(classifiers):
        precision, recall, latency = evaluate_classifier(classifier,
X_train_bin, y_train, X_test_bin, y_test)
        results.append((classifier_names[j], N, precision, recall,
latency))

table_headers = ["Classifier", "Top N Features", "Precision",
"Recall", "Latency (seconds)"]

table = tabulate(results, headers=table_headers, tablefmt="grid")

print(table)
```

```
+--------------------+-------------------+-------------+----------
+---------------------+
| Classifier         |   Top N Features |   Precision |   Recall |
Latency (seconds) |
+====================+===================+=============+==========+======
================+
| BernoulliNB        |                10 |    0.804348 | 0.755102 |
0.002846   |
+--------------------+-------------------+-------------+----------
+---------------------+
| MultinomialNB_bin |                10 |    0.804348 | 0.755102 |
0.00178576 |
+--------------------+-------------------+-------------+----------
+---------------------+
| MultinomialNB_tf  |                10 |    0.804348 | 0.755102 |
0.00167608 |
+--------------------+-------------------+-------------+----------
+---------------------+
| BernoulliNB        |               100 |    1         | 0.673469 |
0.00286269 |
+--------------------+-------------------+-------------+----------
+---------------------+
| MultinomialNB_bin |               100 |    0.976744 | 0.857143 |
0.00201845 |
+--------------------+-------------------+-------------+----------
+---------------------+
| MultinomialNB_tf  |               100 |    0.976744 | 0.857143 |
0.00188994 |
+--------------------+-------------------+-------------+----------
+---------------------+
| BernoulliNB        |              1000 |    1         | 0.591837 |
0.00398445 |
+--------------------+-------------------+-------------+----------
+---------------------+
| MultinomialNB_bin |              1000 |    1         | 0.938776 |
0.00236535 |
+--------------------+-------------------+-------------+----------
```

```
+--------------------+
| MultinomialNB_tf  |              1000 |    1         | 0.938776 |
0.00226092 |
+------------------+----------------+------------+----------
+--------------------+
```

Support Vector Machine (SVM)

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import precision_score, recall_score
import time
from tabulate import tabulate

# Function to train and evaluate an SVM classifier
def evaluate_svm_classifier(X_train, y_train, X_test, y_test,
param_grid, N):
    svm = SVC(kernel='linear')
    clf = GridSearchCV(svm, param_grid, cv=5)  # using 5-fold cross-
validation
    clf.fit(X_train, y_train)
    best_C = clf.best_params_['C']
    svm_best = SVC(kernel='linear', C=best_C)
    start_time = time.time()
    svm_best.fit(X_train, y_train)
    y_pred = svm_best.predict(X_test)
    latency = time.time() - start_time
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    return best_C, precision, recall, latency

svm_results = []

N_values = [10, 100, 1000]
top_features = [top_10_features, top_100_features, top_1000_features]

for i, N in enumerate(N_values):
    vectorizer = CountVectorizer(binary=True,
vocabulary=top_features[i])
    X_train_bin = vectorizer.fit_transform(df_train['text'])
    X_test_bin = vectorizer.transform(df_test['text'])

    y_train = df_train['label']
    y_test = df_test['label']

    # Defining hyperparameters grid
    param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

    best_C, precision, recall, latency =
evaluate_svm_classifier(X_train_bin, y_train, X_test_bin, y_test,
```

```
param_grid, N)

    svm_results.append((f"My SVM having top {N} binary features",
best_C, precision, recall, latency))

table_headers = ["SVM Configuration", "Best C", "Precision", "Recall",
"Latency (seconds)"]
table = tabulate(svm_results, headers=table_headers, tablefmt="grid")
print(table)
```

```
+-----------------------------------------+----------+-------------
+----------+--------------------+
| SVM Configuration                       |   Best C |   Precision |
Recall |   Latency (seconds) |
+=========================================+==========+=============+===
=======+====================+
| My SVM having top 10 binary features    |     10   |    0.804348 |
0.755102 |           0.0188777 |
+-----------------------------------------+----------+-------------
+----------+--------------------+
| My SVM having top 100 binary features   |      1   |    0.973684 |
0.755102 |           0.047395  |
+-----------------------------------------+----------+-------------
+----------+--------------------+
| My SVM having top 1000 binary features  |      0.1 |    0.974359 |
0.77551  |           0.175972  |
+-----------------------------------------+----------+-------------
+----------+--------------------+
```

In this assignment, I achieved the following important goals:

1. Classifier Evaluation for Spam Precision and Recall: I assessed three distinct classifiers, specifically the Bernoulli Naive Bayes (utilizing binary features), the Multinomial Naive Bayes (with binary features), and the Multinomial Naive Bayes (incorporating term frequencies). My evaluation comprised of metrics like precision, recall, and the latency of these classifiers. This analysis was performed across different feature sets, including the top 10, 100, and 1000 features.

2. Latency Measurement: I quantified the latency for each classifier and feature set. The latency measurement covers the time taken from the training phase to the prediction phase.

3. Design and Evaluation of a Support Vector Machine (SVM) Spam Filter: I designed an SVM-based spam filter. My approach involved using binary features (BF) for this filter. To determine the most suitable hyperparameters, especially the regularization parameter (C), I utilized GridSearchCV along with cross-validation, specifically on the training dataset. Going ahead, I rigorously assessed the performance of the SVM on the test dataset and reported the key metrics, including

precision, recall, and latency. These evaluations were conducted for the top 10, 100, and 1000 features.

4. Feature Selection Methodology: In my SVM implementation, I made a choice to employ binary features (BF), driven by their strong relevance in the domain of spam classification. To select the most important features, I used a method focused on identifying the top N words. This N value was varied between 10, 100, and 1000 based on the significance of these terms, often measured by their frequency within the dataset.

5. Hyperparameter Selection Protocol: I adhered to not use the test dataset for hyperparameter selection. Instead, I rigorously followed the recommended best practice of leveraging GridSearchCV with a 5-fold cross-validation strategy applied exclusively to the training dataset. This approach ensured the optimal hyperparameters were determined systematically and independently from the test dataset.

**Results**

---

**Top 10 words:**

["remove", "free", "linguistic", "click", "business", "advertise", "company", "sell", "english"]

**Spam Precision and Spam Recall for Naive Bayes Classifiers:**

Methodology

- Classifiers: BernoulliNB and two variants of MultinomialNB (binary features and term frequency).
- Evaluation: Measured precision and recall for each classifier.
- Feature Sizes (N): Evaluated three feature sizes: 10, 100, and 1000.
- Feature Representation: Used binary and term frequency representations.
- Timing: Recorded training and prediction time (latency) for each classifier.

Results:

| > | Classifier | Top N Features | Precision | Recall |
|---|---|---|---|---|
| BernoulliNB | 10 | 0.804348 | 0.755102 | 0.00380373 |
| MultinomialNB_bin | 10 | 0.804348 | 0.755102 | 0.00289392 |
| MultinomialNB_tf | 10 | 0.804348 | 0.755102 | 0.00244927 |
| BernoulliNB | 100 | 1 | 0.673469 | 0.00285673 |
| MultinomialNB_bin | 100 | 0.976744 | 0.857143 | 0.00178242 |
| MultinomialNB_tf | 100 | 0.976744 | 0.857143 | 0.00179315 |

| > | Classifier | Top N Features | Precision | Recall |
|---|---|---|---|---|
| BernoulliNB | 1000 | 1 | 0.591837 | 0.00389504 |
| MultinomialNB_bin | 1000 | 1 | 0.938776 | 0.0026834 |
| MultinomialNB_tf | 1000 | 1 | 0.938776 | 0.002352 |

**Support Vector Machine (SVM)**

Methodology:

- Features Used: Binary features indicating the presence or absence of terms in emails.
- Number of Features: Evaluated with three settings: 10, 100, and 1000 top features based on information gain.
- Feature Selection: Selected top-N features using information gain.
- SVM Parameters: Linear SVM with 'C' hyperparameter grid search.
- Evaluation: 5-fold cross-validation for 'C' selection, then training on the full dataset.

Results:

| > | SVM Configuration | Best C | Precision | Recall |
|---|---|---|---|---|
| My SVM having top 10 binary features | 10 | 0.804348 | 0.755102 | 0.0219417 |
| My SVM having top 100 binary features | 1 | 0.973684 | 0.755102 | 0.0663161 |
| My SVM having top 1000 binary features | 0.1 | 0.974359 | 0.77551 | 0.167927 |