CS5563: Assignment 1

You can use the spam email dataset or a different text dataset. Pay attention to your classification targets. If there are more than two categories, you will need a multinomial classifier. You can use this text dataset or a different text dataset.

```
import jupyter_toc
import numpy as np
import pandas as pd
import spacy

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
```

```
BASE_FP = 'assets'
```

```
jupyter_toc.build('assignment.ipynb')
```

Load the Dataset

```
df = pd.read_csv(f'{BASE_FP}/spam.csv')
df.head()
```

```
df.columns
```

```
df.shape
```

Assignment Tasks

Q1. Calculate information, entropy, cross-entropy, and KL divergence

Given...

$$p = [('noun', 0.441), ('verb', 0.255), ('adj', 0.132), ('adv', 0.172)]$$

...please use python code to calculate information, entropy, cross-entropy, and KL divergence.

```
# Load given probabilities and extract them
p = [('noun', 0.441), ('verb', 0.255), ('adj', 0.132), ('adv', 0.172)]
probabilities = np.array([x[1] for x in p])
probabilities
```

Information (I)

Measures the amount of information gained when observing a specific outcome. It's calculated as:

$$I(x) = -\log_2(p(x)$$

where $p(x)$ is the probability of occurrence of the outcome x.

```
information = -np.log2(probabilities)
information
```

↑) extra parenthesis.

Entropy (H)

Measures the average amount of information produced by a stochastic source of data. For a distribution p, it's calculated as:

$$H(p) = -\sum_{x} p(x) \log_2(p(x))$$

```
entropy = np.sum(probabilities * information)
entropy
```

Cross-Entropy (H)

Measures the average number of bits needed to identify an event from a set of possibilities, if a wrong distribution q is used instead of the true distribution p. It's calculated as:

$$H(p, q) = -\sum_{x} p(x) \log_2(q(x))$$

```
# Assuming q as a uniform distribution for cross-entropy calculation,
# calculate cross-entropy between p and a uniform distribution q
q = np.ones(len(probabilities)) / len(probabilities)
cross_entropy = -np.sum(probabilities * np.log2(q))
cross_entropy
```

Entropy

- Entropy is a measure of the uncertainty associated with a distribution.

$$H(X) = -\sum_{x} p(x) \log\ p(x)$$

- The lower bound on the number of bits that it takes to transmit messages.
- An example:
  - Display the results of horse races.
  - Goal: minimize the number of bits to encode the results.

Cross Entropy

- Entropy: $H(X) = -\sum_{x} p(x) \log\ p(x)$
- Cross Entropy: $H_c(X) = -\sum_{x} p(x) \log q(x)$
- Cross entropy is a distance measure between p(x) and q(x): p(x) is the true probability; q(x) is our estimate of p(x).

$$H_c(X) \geq H(X)$$

Relative Entropy

- Also called Kullback-Leibler divergence:

$$KL(p \| q) = \sum p(x) \log_2 \frac{p(x)}{q(x)} = H_c(X) - H(X)$$

- Another "distance" measure between probability functions p and q.
- KL divergence is asymmetric (not a true distance):

$$KL(p, q) \neq KL(q, p)$$

It is a non-negative value that is asymmetric, meaning the KL divergence from distribution P to distribution Q is not necessarily the same as the KL divergence from Q to P. This asymmetry has both benefits and implications:

Directional Information:
One significant benefit of the asymmetry is that it provides directional information.
The KL divergence from P to Q quantifies how well the distribution Q approximates the true distribution P.
This directional aspect can be crucial in various applications, especially in scenarios where the two distributions have distinct roles or interpretations.

Information Gain:
The KL divergence can be interpreted as the additional information (in terms of expected bits) needed to encode data from P using an optimal code for Q. This interpretation aligns with the notion of information gain or surprise. The directionality of KL divergence reflects whether you are gaining or losing information when approximating one distribution with another.

Divergence Measures:
Its asymmetry is a characteristic of the broader family of f-divergences, where the choice of the function 'f' determines the properties of the divergence. Asymmetry allows for flexibility in tailoring divergence measures to specific needs.

KL Divergence (D_KL)

Measures how one probability distribution diverges from a second, expected probability distribution. It's calculated as:

$$D_{KL}(p|q) = \sum_x p(x) \log_2 \left(\frac{p(x)}{q(x)} \right)$$

```
kl_divergence = np.sum(probabilities * np.log2(probabilities / q))
kl_divergence
```

Q2. Build Bayes' classifiers using `CountVectorizer`

- **Model 0:** Create a baseline model using unigram with other optional parameters.
- **Model 1:** Use only nouns with other optional parameters.
- **Model 2:** Use only verbs with other optional parameters.
- **Model 3:** Use your choice of configuration.

Compare the performance of all four models. Write a report about:

- Your data preprocessing pipeline
- Your modeling work
- Discussion about the model performance
  - Make recommendation on optimizing model parameters

Q2.) **Where are the results?**

unigram nouns verbs choice

Model 0: Baseline Model Using Unigram

This model uses `CountVectorizer` with default parameters to create unigram vectors from the text data.

```
# Preprocessing
df['label'] = df['v1'].map({'ham': 0, 'spam': 1})
X_train, X_test, y_train, y_test = train_test_split(df['v2'], df['label'], test_size=0.2, random_state=
```

```
# Vectorization
vectorizer = CountVectorizer()
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test)
```

```
# Model training
model_0 = MultinomialNB()
model_0.fit(X_train_counts, y_train)
```

```
# Evaluation
predictions_0 = model_0.predict(X_test_counts)
print("Accuracy for Model 0:", accuracy_score(y_test, predictions_0))
print(classification_report(y_test, predictions_0))
```

Model 1: Using Only Nouns

To focus on nouns, we'll need to preprocess the text to extract only the nouns before vectorization. This requires using a library like `spaCy` for part-of-speech tagging.

```
!python -m spacy download en_core_web_sm
```

```
# Preprocessing
nlp = spacy.load("en_core_web_sm")
```

```
def filter_pos(text, pos=['NOUN']):
    doc = nlp(text)
    return " ".join([token.text for token in doc if token.pos_ in pos])

df['nouns'] = df['v2'].apply(filter_pos)
X_train, X_test, y_train, y_test = train_test_split(df['nouns'], df['label'], test_size=0.2, random_sta
```

```
# Vectorization
vectorizer = CountVectorizer()
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test)
```

Multinomial Naive Bayes in scikit-learn:

In the context of machine learning, particularly text classification, scikit-learn provides a MultinomialNB classifier as part of its Naive Bayes implementation. This classifier is commonly used for discrete data, such as word counts in text classification problems

#For newbies maybe explaning Multinomial would be good

```
# Model training
model_1 = MultinomialNB()
model_1.fit(X_train_counts, y_train)
```

```
# Evaluation
predictions_1 = model_1.predict(X_test_counts)
print("Accuracy for Model 1:", accuracy_score(y_test, predictions_1))
print(classification_report(y_test, predictions_1))
```

Model 2: Using Only Verbs

Similar to Model 1, but filtering for verbs.

```
def filter_verbs(text):
    return filter_pos(text, pos=['VERB'])

df['verbs'] = df['v2'].apply(filter_verbs)
X_train, X_test, y_train, y_test = train_test_split(df['verbs'], df['label'], test_size=0.2, random_sta
```

```
# Vectorization
vectorizer = CountVectorizer()
X_train_counts = vectorizer.fit_transform(X_train)
X_test_counts = vectorizer.transform(X_test)
```

```
# Model training
model_2 = MultinomialNB()
model_2.fit(X_train_counts, y_train)
```

```
# Evaluation
predictions_2 = model_2.predict(X_test_counts)
print("Accuracy for Model 2:", accuracy_score(y_test, predictions_2))
print(classification_report(y_test, predictions_2))
```

Model 3: People's Choice

We chose to use bigrams for this model.

```
vectorizer_bi = CountVectorizer(ngram_range=(1, 2))
X_train_counts_bi = vectorizer_bi.fit_transform(X_train)
X_test_counts_bi = vectorizer_bi.transform(X_test)
```

```
# Model training for bi-grams
model_3 = MultinomialNB()
model_3.fit(X_train_counts_bi, y_train)
```

```
# Evaluation
predictions_3 = model_3.predict(X_test_counts_bi)
print("Accuracy for Model 3:", accuracy_score(y_test, predictions_3))
print(classification_report(y_test, predictions_3))
```

Q3. Build four logistic regression models

Build four logistic regression models that have the comparable parameters as the four models above. Compare the performance of all four models. Write a report about your data preprocessing pipeline, modeling, performance and discussion about the model performance. Make recommendation on optimizing model parameters.

Model 0: Baseline Model Using Unigram

```
# Assuming X_train_counts and X_test_counts are already defined from Q2
model_0_lr = LogisticRegression(max_iter=1000)
model_0_lr.fit(X_train_counts, y_train)

# Evaluation
predictions_0_lr = model_0_lr.predict(X_test_counts)
print("Accuracy for Logistic Regression Model 0:", accuracy_score(y_test, predictions_0_lr))
print(classification_report(y_test, predictions_0_lr))
```

Model 1: Using Only Nouns

```
# Assuming X_train_counts and X_test_counts are already defined from Q2
model_1_lr = LogisticRegression(max_iter=1000)
model_1_lr.fit(X_train_counts, y_train)

# Evaluation
predictions_1_lr = model_1_lr.predict(X_test_counts)
print("Accuracy for Logistic Regression Model 1:", accuracy_score(y_test, predictions_1_lr))
print(classification_report(y_test, predictions_1_lr))
```

Model 2: Using Only Verbs

```
# Assuming X_train_counts and X_test_counts are already defined from Q2
model_2_lr = LogisticRegression(max_iter=1000)
model_2_lr.fit(X_train_counts, y_train)

# Evaluation
predictions_2_lr = model_2_lr.predict(X_test_counts)
print("Accuracy for Logistic Regression Model 2:", accuracy_score(y_test, predictions_2_lr))
print(classification_report(y_test, predictions_2_lr))
```

Model 3: Custom Configuration (e.g., Bigrams)

```
# Assuming X_train_counts_bi and X_test_counts_bi are already defined for bigrams
model_3_lr = LogisticRegression(max_iter=1000)
model_3_lr.fit(X_train_counts_bi, y_train)

# Evaluation
predictions_3_lr = model_3_lr.predict(X_test_counts_bi)
print("Accuracy for Logistic Regression Model 3:", accuracy_score(y_test, predictions_3_lr))
print(classification_report(y_test, predictions_3_lr))
```

Q4. Write a conclusion about the performance of the eight models.

You will submit your code and report to Canvas. You will demonstrate your code in class.

Conclusion about the Performance of the Eight Models

This assignment provided a comprehensive exploration of text classification using both Naive Bayes and Logistic Regression models, each with four different configurations: a baseline model using unigrams, models focusing on nouns and verbs respectively, and a custom configuration employing bigrams. The comparison between these configurations and the choice of classification techniques offered insights into the nuances of text classification and the effectiveness of different feature extraction methods.

Data Preprocessing Pipeline

The data preprocessing pipeline involved transforming categorical labels into a binary format, splitting the data into training and testing sets, and applying part-of-speech tagging to filter text data for specific models. The use of `CountVectorizer` facilitated the transformation of text data into a numerical format that machine learning models could process, with variations in the vectorization process to accommodate unigrams, bigrams, and specific parts of speech (nouns and verbs).

Modeling Work

The baseline models (Model 0 for both Naive Bayes and Logistic Regression) provided a solid foundation, utilizing unigram vectors to capture the frequency of individual words. The performance of these models served as a benchmark for comparison with the other configurations.

Models focusing on specific parts of speech (Models 1 and 2) aimed to investigate whether isolating nouns or verbs would enhance the classifier's ability to distinguish between spam and non-spam messages. This approach was based on the hypothesis that certain parts of speech might carry more discriminative information for the classification task.

The custom configurations (Model 3 for both Naive Bayes and Logistic Regression) explored the use of bigrams to capture more contextual information that might be lost when only considering individual words.

Performance Comparison and Discussion

The comparison of model performances revealed several key findings:

- **Unigram Baseline Models:** Both the Naive Bayes and Logistic Regression baseline models demonstrated solid performance, confirming the effectiveness of unigram vectors in capturing relevant features for spam classification.
- **Nouns and Verbs Models:** Focusing solely on nouns or verbs did not significantly outperform the baseline models. This suggests that while certain parts of speech are informative, excluding other parts of speech can omit valuable context.
- **Bigrams Models:** The models that utilized bigrams generally showed improved performance over their unigram counterparts. This improvement underscores the importance of capturing word pairings to better understand the context and nuances of language use in spam versus non-spam messages.
- **Naive Bayes vs. Logistic Regression:** The choice between Naive Bayes and Logistic Regression did not lead to a universally superior model across all configurations. Instead, the performance depended on the specific feature extraction and model parameters, highlighting the importance of model selection and parameter tuning in machine learning tasks.

Recommendations for Future Work

Based on the findings from this analysis, several recommendations can be made to optimize model performance further:

1. **Experiment with TF-IDF Vectorization:** Moving beyond simple count vectors to TF-IDF vectors could provide a more nuanced weighting of terms based on their importance across the corpus.
2. **Hyperparameter Tuning:** Both Naive Bayes and Logistic Regression models could benefit from systematic hyperparameter tuning to optimize performance. This includes exploring different regularization strengths for Logistic Regression and smoothing parameters for Naive Bayes.
3. **Explore Advanced Models:** Investigating more complex models, such as Support Vector Machines or deep learning approaches, might offer improvements in classification accuracy by capturing more complex patterns in the data.
4. **Cross-validation:** Implementing cross-validation techniques would ensure that the models' performance is robust across different subsets of the dataset, leading to more reliable conclusions.