PROGRAMMING ASSIGNMENT 3

Issue Date : 11.04.2025 - Friday
Due Date : 25.04.2025 - Friday (23:59)
Advisor: Res. Asst. İsmail Furkan Atasoy
Programing Language : Python

This project aims to explore and practice some text classification techniques. To achieve this, binary sentiment classification (positive/negative) will be conducted using the "IMDB Movie Review Dataset"[1]. The solutions will involve both Naive Bayes and Logistic Regression models.

Dataset

You can load the dataset using `load_dataset("imdb")` function from the `datasets` library. IMDB dataset is split into two parts: "train" and "test," each containing 25,000 samples. Each has two colums: "text" and "label". `label` contains 0 (negative) and 1 (positive).

After loading the dataset, it should be converted into two pandas dataframes named `train_df` and `test_df`, each containing 25,000 samples. Each dataframe will have the features "text" and "label".

1 Preprocessing

In this part, a function `preprocess_text(text)` should be defined, which processes the given text and returns the new processed text. The text processing steps are as follows:

Removing Punctuation: The punctuation characters in all texts should be removed with respect to the `string.punctuation`, and a space character should be added in their place.

Removing Numbers: All digits from 0 to 9 should be removed and replaced with a space character.

Converting to Lowercase: All characters should be converted to lowercase.

Removing Stop Words: The stopwords from `nltk.corpus.stopwords.words("english")` should be removed from all texts and replaced with a space character.

After all preprocessing steps are applied with respect to their order, multiple consecutive spaces should be replaced with a single space, leading or trailing spaces should be removed from the text.

Let's walk through the preprocessing step by step using an example sentence `"Hello world!!! I love the <AIN442> and <BBM497> courses."`

[1] https://huggingface.co/datasets/stanfordnlp/imdb

After applying the "Removing Punctuation" and removing unnecessary white spaces:

`"Hello world I love the AIN442 and BBM497 courses"`

After applying the "Removing Numbers" and removing unnecessary white spaces:

`"Hello world I love the AIN and BBM courses"`

After applying the "Converting to Lowercase" and removing unnecessary white spaces:

`"hello world i love the ain and bbm courses"`

After applying the "Removing Stop Words" and removing unnecessary white spaces:

`"hello world love bbm courses"`

The text column of both dataframes (`train_df` and `test_df`) should be updated by applying the preprocessing steps with respect to their order.

2 Naive Bayes

2.1 Training

In this section, only the updated dataframe `train_df` will be used.

First, a class named `NaiveBayesClassifier` should be created. This class should have the following properties:

total_pos_words: Total word count in the texts with a positive label in the `train_df`.

total_neg_words: Total word count in the texts with a negative label in the `train_df`.

vocab_size: Total number of unique words in `train_df`.

prior_pos: The ratio of the positive samples to the total number of samples in `train_df`.

prior_neg: The ratio of the negative samples to the total number of samples in `train_df`.

pos_counter: Frequency of each word belonging to the positive class in `train_df`. It should be created using the `Counter` class from the `collections` library.

neg_counter: Frequency of each word belonging to the negative class in `train_df`. It should be created using the `Counter` class from the `collections` library.

`NaiveBayesClassifier` should also have three methods named `fit()` and `predict()`.

def fit(self, train_df): When this method is called, the values for all the above properties should be calculated and assigned to them. It will not return anything.

def predict(self, text): This method can be called with both the sample texts in `test_df` and with different texts. Therefore, it is necessary to first apply the same preprocessing steps to the `text` argument. Then, it should calculate the log probabilities according to the following steps and formulas.

To prevent the floating point underflow caused by multiplications, we will take the logarithm of both sides. This way, the multiplication will be transformed into the sum of logarithms, as shown below. (You can use the `math.log()` function).

$$P(y = 1|\text{words}) = P(y = 1) \cdot P(w_1|y = 1) \cdot P(w_2|y = 1) \cdots P(w_n|y = 1)$$

where:

- $P(y = 1)$: the prior probability of the positive class (`prior_pos`),
- $P(w_i|y = 1)$: the probability of word w_i given that the class is positive.

Taking the log of both sides:

$$\log P(y = 1|\text{words}) = \log P(y = 1) + \log P(w_1|y = 1) + \log P(w_2|y = 1) + \cdots + \log P(w_n|y = 1)$$

Similarly, for the negative class:

$$\log P(y = 0|\text{words}) = \log P(y = 0) + \log P(w_1|y = 0) + \log P(w_2|y = 0) + \cdots + \log P(w_n|y = 0)$$

The value of $\log P(w_i \mid y = 1)$ can be computed using the following formula:

$$\log P(w_i \mid y = 1) = \log \left(\frac{\text{count}(w_i, y = 1) + 1}{\text{total\_pos\_words} + \text{vocab\_size}} \right)$$

where:

- $\text{count}(w_i, 1)$: the count of the word in the positive labeled samples.

Similarly, for the negative class:

$$\log P(w_i \mid y = 0) = \log \left(\frac{\text{count}(w_i, y = 0) + 1}{\text{total\_neg\_words} + \text{vocab\_size}} \right)$$

After calculating log probabilities for both classes, the class with the greater log probability can be determined as the predicted class as below.

$$\text{y\_predicted} = \begin{cases} 1 & \text{if log\_prob\_pos} > \text{log\_prob\_neg} \\ 0 & \text{otherwise} \end{cases}$$

where:

- log_prob_pos: The log probability of `text` belonging to class 1, $\log P(y = 1|\text{words})$,
- log_prob_neg: The log probability of `text` belonging to class 0, $\log P(y = 0|\text{words})$.

The `predict()` method should return a tuple with the values (y_predicted, log_prob_pos, log_prob_neg) at the end.

2.2 Testing with Examples

```
nb = NaiveBayesClassifier()
nb.fit(train_df)
```

```
print(nb.total_pos_words)
print(nb.total_neg_words)
print(nb.vocab_size)
print(nb.prior_pos)
print(nb.prior_neg)
print(nb.pos_counter["great"])
print(nb.neg_counter["great"])
```

Output:
1575152
1516208
74002
0.5
0.5
6419
2642

```
prediction1 = nb.predict(test_df.iloc[0]["text"])

prediction2 = nb.predict("This movie will be place at 1st in my favourite
    ↪ movies!")

prediction3 = nb.predict("I couldn't wait for the movie to end, so I
    ↪ turned it off halfway through. :D It was a complete disappointment.
    ↪ ")

print(f"{'Positive' if prediction1[0] == 1 else 'Negative'}")
print(prediction1)

print(f"{'Positive' if prediction2[0] == 1 else 'Negative'}")
print(prediction2)

print(f"{'Positive' if prediction3[0] == 1 else 'Negative'}")
print(prediction3)
```

Output:
Negative
(0, -1167.5758675517511, -1146.4479616999306)
Positive
(1, -36.43364380516184, -37.068841883770205)
Negative
(0, -57.05497089563332, -53.21115758896025)

```
print(preprocess_text("This movie will be place at 1st in my favourite
    ↪ movies!"))

print(preprocess_text("I couldn't wait for the movie to end, so I turned
    ↪ it off halfway through. :D It was a complete disappointment."))
```

Output:
movie place st favourite movies
wait movie end turned halfway complete disappointment

```
y_true = test_df['label'].values

y_pred = [nb.predict(text)[0] for text in test_df['text']]

# from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_true, y_pred)

print(f"Accuracy: {accuracy}")
```

Output:
Accuracy: 0.82464

3 Logistic Regression

3.1 Training

In this section, preprocessed dataframes `train_df` and `test_df` will be used.

To perform Logistic Regression, a feature vector is needed. Thus, a custom feature vector will be created specifically for the training dataset.

First, the `bias_scores(train_df)` function should be defined. This function will take `train_df` as a parameter. Then, it will calculate the bias scores according to the formulas below.

1. For each word w in the dataset, compute:
   - $f_p(w)$: frequency of word w in positive class.
   - $f_n(w)$: frequency of word w in negative class.
   - $f_t(w) = f_p(w) + f_n(w)$: total frequency in both classes.
2. Compute the **bias score** as:

$$\text{score}(w) = \left| \frac{f_p(w) - f_n(w)}{f_t(w)} \right| \cdot \log(f_t(w))$$

3. Sort all words by their bias score in descending order (if bias score is the same, then sort in ascending alphabetical order) and keep only the top 10,000 in a list of tuples. The tuple should be as below.

$$(w, f_p(w), f_n(w), f_t(w), \text{score}(w))$$

The function should return a list consisting of the top 10,000 tuples based on the highest bias scores.

A Bag-of-Words vector should be constructed using these 10,000 words, which are assumed to have the most influence on sentiment analysis within the `train_df` dataset. You can use `CountVectorizer` from the `sklearn.feature_extraction.text` module for this purpose.

After that, you should apply this vectorizer to transform all the texts in `train_df` and `test_df` by using it's `transform` method. This will create `X_train` and `X_test`, which are the feature matrices for the training and test datasets, respectively.

You should also assign the corresponding labels to `y_train` and `y_test`, which correspond to the feature matrices in `X_train` and `X_test`, respectively.

For the training, you should use `LogisticRegression` class from `sklearn.linear_model` and create a model named `lr_model`. The only parameter to be used is `max_iter`. You should change `max_iter` from 1 to 25, creating 25 different models. For each model, record the accuracy scores of prediction results for both the training and test sets, and plot them on the same graph. The x-axis should represent the number of iterations, and the y-axis should represent the accuracy score. The graph should contain two lines: one for the training results and one for the test results. (You can use `matplotlib.pyplot`)

At the end of your code, you should include a small analysis in a comment block. Discuss which model you would prefer to use and explain why, based on the results from the graph.

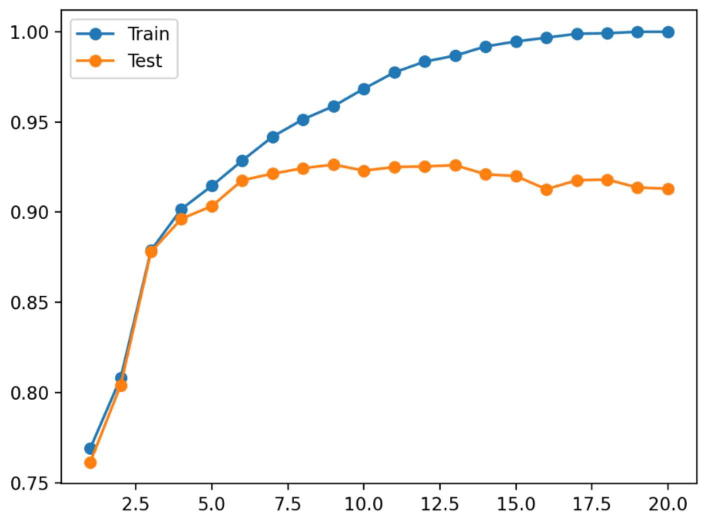

Figure 1: Example Graph for Results

Note: The graph above is provided as an example for guidance purposes. The values it contains **do not** reflect the actual results.

3.2 Examples

```
scores = bias_scores(train_df)

print(scores[:2])
print(scores[-2:])
```

Output:
[('worst', 252, 2480, 2732, 6.453036011602796), ('waste', 99, 1359, 1458, 6.295524245429657)]

[('complimented', 10, 3, 13, 1.3811265770946735), ('conformity', 10, 3, 13, 1.3811265770946735)]

Submission

- You will submit your programming assignment using the **HADI** system. You have to upload a single **zip file** (`.zip`, `.gzip` or `.rar`) holding 2 different Python program (`.py` file). First one is for Naive Bayes and the second is for Logistic Regression solutions. In both solutions, you should load the dataset from scratch and use the `preprocess_text()` function.
- The name of your Python file should be: `hw03_NameLastname_nb.py` for your Naive Bayes solution and `hw03_NameLastname_lr.py` for your Logistic Regression solution, replacing `NameLastname` with your actual first name and last name. The name of your zip file should match `hw03_NameLastname` with a corresponding extension (`.zip`, `.gzip`, or `.rar`).
- Your Naive Bayes solutions will be tested with some examples taken from `test_df` and any other examples.
- Your Logistic Regression solutions will be tested with only `test_df`. You are expected to implement the score function correctly and produce a meaningful training.

Late Policy:

- You must submit your programming assignment **before its due date**.
- You may submit your assignment up to **three days late**, but with a penalty:
  - **1 day late:** 10% penalty
  - **2 days late:** 20% penalty
  - **3 days late:** 30% penalty

DO YOUR PROGRAMMING ASSIGNMENTS YOURSELF!

- **Do not share your programming assignments with your friends.**
- **Do not complete your entire assignment using AI tools and comment on the parts where you have received support from AI tools.**
- **Cheating will be punished.**