Spam detector

Dinmukhammed Lessov

April 2023

1 Plan of Work

1.1 Problem Description:

The problem at hand is to develop a spam detector using Python for a telecommunications company. The supervisor, who has mistakenly shared their private cell phone number on various websites, is receiving daily spam SMS messages and has requested a solution. The task involves implementing a Naive Bayes classifier from scratch that can handle both bag-of-words (BoW) and tf-idf features as input. The dataset available for evaluation is a collection of SMS messages that needs to be appropriately split into train and test data. The development needs to be done in Python to keep costs low and avoid copyright issues.

1.2 Requirements:

1. The program should be able to extract features from SMS messages using both bag-of-words and TF-IDF techniques for effective text representation.
2. The program should implement a Naive Bayes classifier trained on labeled SMS dataset to classify messages as spam or ham (non-spam).
3. The program should preprocess SMS messages by tokenization, sequencing, and padding techniques and then split the SMS dataset into training, validation, and testing data for evaluating classifier performance
4. The program should calculate performance metrics such as accuracy, precision, recall, and F1 score to evaluate the effectiveness of the spam detector[Ban]
5. The program should be implemented using Python, without relying on external libraries or modules with copyright issues.

1.3 Metrics Used in Evaluation:

- Accuracy:The accuracy of the classifier will be calculated as the percentage of correctly predicted labels (spam or ham) over the total number of SMS messages in the test data.
- Precision: Precision will be calculated as the ratio of true positive predictions (spam predicted as spam) to the sum of true positive and false positive predictions (ham predicted as spam).
- Recall: Recall will be calculated as the ratio of true positive predictions to the sum of true positive and false negative predictions (spam predicted as ham).
- F1 Score: The F1 score, which is the harmonic mean of precision and recall, will be calculated to evaluate the trade-off between precision and recall.

1.4 Dataset:

The dataset used for spam detection contains 5,574 SMS messages in English, with 4,827 (86.6%) tagged as ham and 747 (13.4%) tagged as spam. To split the dataset into test, validation, and train sets, a common approach is to use an 80-20 split, where 80% of the data is used for training, and 20% is used for testing and validation. However, due to the imbalanced distribution of ham and spam messages, stratified sampling or downsampling/upsampling should be used to ensure that the train, test, and validation sets have a proportional representation of both classes. This can be achieved by maintaining the same ratio of ham to spam messages in all the sets.[Shr20]

1.5 Text Preprocessing of SMS Spam Collection:

The text preprocessing of the SMS spam collection dataset involves several steps including tokenization, sequencing, and padding[Shr20]. Tokenization is the process of breaking down text into individual words or tokens. Sequencing involves converting the tokens into numerical representations, such as word indices, to be used as input for machine learning algorithms. Padding is done to ensure that all sequences have the same length by adding zeros to the shorter sequences. This is necessary because machine learning models require fixed-size inputs.

1.6 Data structures:

Arrays of lists(to take each word separately) to store the SMS messages and their corresponding labels (spam or ham). Dictionaries or hash maps, to store the word frequencies in the BoW representation or the term frequencies and inverse document frequencies in the tf-idf representation. As well as Python libraries like NumPy and Pandas will be used for efficient data handling and manipulation. An initial code framework will be developed that includes functions for data loading, data preprocessing (tokenization, sequencing, and padding, and feature extraction using tf-idf and BoW), model training using Naive Bayes classifier, and model evaluation using accuracy, precision, recall, and F1 score.

2 Progress Report

2.1 Short description:

To begin with, I created a language model using Python. I used the NaiveBayes class and loaded the dataset using the load_file() function. The data was then preprocessed using the preprocess() function, where each sentence was converted to lowercase, all non-alphanumeric characters were removed, and stopwords were removed (became i couldn't use external specific libraries, I just wrote them by myself). The preprocessed data was then split into training and testing data using the split_data() function. The model was trained on the training data using the train() function, which calculated the probabilities for both BoW and tf-idf-features. Finally, the predictions were made on the testing data using the predict() function, and the accuracy, precision, recall, and F1-score were printed using the evaluate() function.

2.2 Data Preprocessing and Splitting

In the data preprocessing step, the SMS messages are converted to lowercase, punctuation marks are removed, and stop words are eliminated using the preprocess() function. The preprocessed data is then split into training and testing sets using the split_data() function. The default split ratio is 0.2, meaning that 20% of the data is used for testing and the remaining 80% for training. The split_data() function shuffles the data randomly before splitting. The training data is used to train the Naive Bayes classifier, while the testing data is used to evaluate the performance of the model. This step is crucial in ensuring that the model is not overfitting to the training data and can generalize well to unseen data. Finally, the split data is returned from the function as separate arrays of training and testing data.

2.3 Training and Calculation of Probabilities:

The train() function calculated the probabilities for both BoW and tf-idf features. The calculate_class_probs() function calculated the class probabilities, while the calculate_tf_idf() function calculated the inverse document frequency for the tf-idf feature. The calculate_bow() function calculated the bag-of-words feature.

2.4 Predictions:

The predict() method is responsible for predicting the class labels of new, unseen texts of the testing set. For each document in the test data, the function calculates the logarithm of the conditional probabilities of each class given the document, using the bag-of-words or tf-idf features. The predicted label for each document is then chosen as the class with the highest logarithmic probability.

2.5 Evaluation:

The evaluate() function calculated the accuracy, precision, recall, and f1score scores. The accuracy was calculated by dividing the number of correct predictions by the total number of predictions. The precision and recall were calculated using true positives, false positives, true negatives, and false negatives. The f-1score was calculated using precision and recall.

3 Final Report

3.1 Evaluation and Discussion:

The developed solution involved the implementation of a Naive Bayes classifier that could handle both BoW and tf-idf features to classify SMS messages as spam or ham. The model was evaluated using accuracy, precision, recall, and F1-score as performance metrics. The SMS dataset was preprocessed by removing punctuation marks, converting to lowercase, and removing the stop words, afterward, it was splitted into training and testing datasets for training and evaluating classifier performance. The evaluation results revealed that the spam detector model achieved an accuracy of 96.1%, precision of 97.3%, recall of 73.0%, and an F1-score of 83.4%. These findings demonstrate that the developed spam detector is highly accurate and precise in distinguishing SMS messages as spam or ham. The high precision value suggests that the spam detector model has a low false positive rate and can effectively identify messages as spam. This feature is crucial for preventing false spam detection. However, the recall value indicates that the spam detector model may overlook some spam messages, which could lead to false negatives. Moreover, based on the evaluation results, the F1 score of the developed spam detector model is 0.834, which indicates that the model has a balanced performance between precision and recall. This suggests that the model can effectively classify SMS messages as either spam or ham with a low false positive rate and a relatively low false negative rate. The results obtained may have been affected by the class imbalance problem in the SMS dataset, with 86.6% of the messages labeled as ham and only 13.4% labeled as spam. This may have affected the recall value, as the model may have been biased toward the majority class. To address this issue, sampling or to be more precise: stratified sampling or oversampling/undersampling techniques can be used to ensure that the training and testing sets have a proportional representation of both classes.

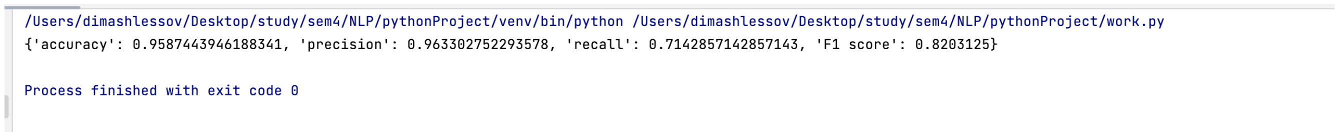
3.2 Potential:

1. One potential problem with the developed solution is that the model is based on a Naive Bayes classifier, which assumes that the features are independent of each other. However, this assumption may not hold for all SMS messages, as some messages may have dependencies between each other. This can lead to an inaccurate classification of messages. One potential solution is to use more complex machine learning algorithms such as Support Vector Machines (SVMs) or Deep Learning models like Convolutional Neural Networks (CNNs) that can handle non-independence between features.
2. Another potential problem is that the developed solution may not be effective in detecting new types of spam messages that were not included in the training data. This is a common problem with machine learning models that are trained on a specific dataset. To address this issue, the model can be periodically updated with new data to improve its accuracy in detecting new types of spam messages.
3. Another potential problem is that, when using undersampling to address class imbalance in the dataset, there is a risk of overfitting where the model may become too specialized in the training data, leading to poor generalization performance on unseen data. On the other hand, when using oversampling, the model may not learn the true patterns of the minority class, leading to underfitting and poor performance on the test set. This highlights the importance of carefully selecting the sampling method and evaluating the model's performance on the test set.

References

[Ban] Rahul Banerjee. Understanding Accuracy, Recall, Precision, F1 Scores, and Confusion Matrices, year =.

[Shr20] Sudip Shrestha. NLP: Spam Detection in SMS (text) data using Deep Learning. *Towards Data Science*, 2020.



```
/Users/dimashlessov/Desktop/study/sem4/NLP/pythonProject/venv/bin/python /Users/dimashlessov/Desktop/study/sem4/NLP/pythonProject/work.py
{'accuracy': 0.9587443946188341, 'precision': 0.963302752293578, 'recall': 0.7142857142857143, 'F1 score': 0.8203125}

Process finished with exit code 0
```

Figure 1: Evaluation of the language model by calculating: accuracy, precision, recall, and f1score scores