MACHINE LEARNING

Project-2

Program: SMS Spam Classification using N-Gram Models

Name: **NIKHIL CHEZIAN**

Matriculation Number: **11027706**

Professor: **Dr. MILAN GNJATOVIC**

Date: **10th May 2023**

Abstract

This report presents a program that utilizes character n-gram models for the purpose of SMS spam detection. The primary objective of this task is to develop an efficient spam filtering system by employing machine learning techniques. The approach undertaken in this project draws inspiration from the concepts covered in Chapter 3 of the Lecture notes in Machine Learning. To train and evaluate the model, the SMS Spam Collection Data Set from the UCI Machine Learning Repository is utilized. The report provides an overview of the program's design, implementation details, performance evaluation, and key insights obtained from the experimentation process. The findings of this study demonstrate the efficacy of the character n-gram model in accurately identifying and filtering SMS spam, thus highlighting its potential for real-world applications in mitigating unwanted text-based communications.

Introduction

The objective of this project was to develop a text classification model to identify SMS messages as either Spam or Non-Spam (Ham). The SMS Spam Collection dataset was utilized, which contains labeled SMS messages for training and evaluation. The classification model was built using the Naive Bayes algorithm and the TF-IDF vectorization technique.

TF-IDF (Term Frequency-Inverse Document Frequency) is a widely used technique in natural language processing and information retrieval for converting textual data into numerical feature vectors. It aims to capture the importance of words in a document relative to a corpus of documents.

TF-IDF vectorization calculates a weight for each word in a document based on two factors: term frequency (TF) and inverse document frequency (IDF).

Term frequency (TF) measures the frequency of a word in a document. It assigns a higher weight to words that appear more frequently within the document. The intuition is that more frequent words are likely to be more important for representing the document's content.

Inverse document frequency (IDF) measures the rarity of a word across the entire corpus of documents. It assigns a higher weight to words that are less common in the corpus. The idea is that words that are less common tend to carry more discriminative information about the document.

The TF-IDF weight for a word in a document is calculated as the product of its term frequency (TF) and inverse document frequency (IDF). This weight represents the relative importance of the word in the document and the corpus.

TF-IDF vectorization involves the following steps:

Tokenization: Breaking down the text into individual words or tokens.

Counting Term Frequencies: Calculating the frequency of each word in a document.

Calculating Inverse Document Frequency: Computing the IDF value for each word across the corpus.

Normalization: Applying optional normalization techniques to the TF-IDF weights, such as L2 normalization, to ensure that the vectors are comparable.

The resulting TF-IDF vectors can be used as input features for machine learning algorithms or other downstream tasks such as text classification, information retrieval, and clustering. By capturing both local and global word importance, TF-IDF vectorization helps in representing documents effectively and enabling meaningful analysis of textual data.

It is important to note that TF-IDF vectorization is a bag-of-words approach, meaning it disregards the order and context of words within a document. However, it remains a powerful and widely used technique for transforming text data into numerical representations, facilitating various text mining and analysis tasks.

The N-gram model is a statistical language model that captures the frequency and sequence of N consecutive words (or characters) in a text. It is widely used in natural language processing tasks. The model assumes that the probability of a word depends on the previous (N-1) words, allowing it to generate text, predict word sequences, and assess sentence fluency. The model involves tokenization, N-gram generation, counting frequencies, probability estimation, and smoothing. The choice of N depends on the task and available data. The N-gram model is a powerful tool for language modeling and analysis, despite its limitations in capturing long-range dependencies and contextual nuances.

Data Preprocessing

The SMS Spam Collection dataset was loaded into a Pandas dataframe. The dataset consisted of two columns: "label" and "message." The labels were converted to binary values, where "spam" was encoded as 1 and "ham" as 0.

Model Development

To build the classification model, the dataset was split into training and test sets using a test size of 20% and a random state of 42. TF-IDF vectorization was applied to convert the text messages into numerical feature vectors. The TF-IDF vectorizer calculated the importance of each word in the messages based on its frequency and inverse document frequency.

Next, a grid of hyperparameters was defined for the Multinomial Naive Bayes classifier. GridSearchCV was utilized to perform cross-validation and find the best hyperparameters, specifically tuning the "alpha" parameter. Model evaluation was conducted using 5-fold cross-validation.

Model Evaluation

After training the Multinomial Naive Bayes classifier with the best hyperparameters, the labels for the test data were predicted. The accuracy of the classifier was computed by comparing the predicted labels with the true labels of the test set.

Results

The trained classifier achieved an accuracy of **0.989237668161435** on the test set, indicating its effectiveness in distinguishing spam and non-spam messages.

Accuracy
0.989237668161435

Table-1: Accuracy

To further analyze the performance, a confusion matrix was generated. The confusion matrix displayed the number of true positives, true negatives, false positives, and false negatives, providing insights into the classifier's ability to correctly classify spam and non-spam messages.

Confusion Matrix		Predicted Class	
		Ham	Spam
Actual Class	Ham	965	1
	Spam	11	138

Table-2: Confusion Matrix

In this case, the confusion matrix is as follows:

True Positives (TP): 965

True Negatives (TN): 138

False Positives (FP): 1

False Negatives (FN): 11

This means that the model correctly classified 965 non-spam (ham) messages as ham and 138 spam messages as spam. It made 1 false positive prediction, incorrectly classifying a non-spam message as spam, and 11 false negative predictions, incorrectly classifying spam messages as non-spam.

Additionally, the precision, recall, F1 score, and support for each class were calculated using the classification_report function. The classification report provides a comprehensive summary of different evaluation metrics for each class (spam and ham). It includes precision, recall, F1-score, and support.

Classification Report	Precision	Recall	F1-Score	Support
0	0.99	1.00	0.99	966
1	0.99	0.93	0.96	149

Table-3: Classification Report consists of Precision, recall, F1 Score and support.

Precision: It represents the proportion of correctly classified instances for a class out of all instances predicted as that class. In this case, the precision for spam (class 1) is 0.99, indicating that 99% of the predicted spam messages are actually spam. The precision for ham (class 0) is also 0.99.

Recall: Also known as sensitivity or true positive rate, it represents the proportion of correctly classified instances for a class out of all actual instances of that class. The recall for spam (class 1) is 0.93, meaning that the model correctly identified 93% of the actual spam messages. The recall for ham (class 0) is 1.00, indicating that the model identified all non-spam messages correctly.

F1-score: The F1-score is the harmonic mean of precision and recall, providing a single metric that considers both precision and recall. It balances the trade-off between precision and recall. The F1-score for spam (class 1) is 0.96, and for ham (class 0) is 0.99.

Support: It represents the number of samples in each class. In this case, the support for spam (class 1) is 149, and for ham (class 0) is 966.

Conclusion

In this project, a text classification model was developed using the Multinomial Naive Bayes algorithm and TF-IDF vectorization. The model demonstrated **98.92%** accuracy in distinguishing between spam and non-spam SMS messages. Overall, the output indicates that the text classification model performed very well with high accuracy, effectively distinguishing between spam and non-spam (ham) messages. It achieved a high precision, recall, and F1-score for both classes, demonstrating its effectiveness in identifying spam messages. The results indicate its potential for automated spam detection, which can be applied to real-world applications to protect users from unwanted messages.

Additional Questions

1. **What is the order of your n-gram models (e.g., bigram models, trigram models, etc.)?**

   In this project, I have used the default order of the 'TfidfVectorizer' class. The default order is 1, thereby considering unigrams.

2. **How do you define and handle out-of-vocabulary symbols?**

   In this code I have used the functionality the 'tfidfVectorizer' class to handle Out-Of-Vocabulary (OOV) symbols. Basically, it ignores the OOV symbols and assigns a zero weight to out-of-vocabulary words when transforming text data into TF-IDF feature vectors.

   The following are steps involved in handling OOV symbols by 'tfidfVectorizer'

   - The dataset is split into training data and test data using train_test_split.
   - The TfidfVectorizer is instantiated without explicitly setting the vocabulary parameter. This means that the vocabulary will be automatically learned from the training data.
   - The fit_transform method of the TfidfVectorizer is applied on the training data (X_train), which learns the vocabulary and transforms the training data into TF-IDF vectors.
   - The transform method of the TfidfVectorizer is applied on the test data (X_test). This method transforms the test data into TF-IDF vectors using the vocabulary learned from the training data. During this transformation, any OOV words in the test data are ignored and not considered in the vectorization process.
   - The transformed training data (X_train_tfidf) and test data (X_test_tfidf) are used for training the Multinomial Naive Bayes classifier and making predictions.

   In summary, the TfidfVectorizer class automatically handles OOV symbols by ignoring them during the vectorization process. This allows the model to handle new or unseen words during prediction without raising errors.

3. **Which smoothing method do you apply? How do you avoid zero probabilities?**
   In this code I have used the Multinomial Naive Bayes classifier (MultinomialNB), which applies Laplace smoothing (also known as additive smoothing) by default. Laplace smoothing helps avoid zero probabilities by adding a small constant (alpha) to the observed counts of each feature.
   Naive Bayes classifiers calculate probabilities based on the occurrence of features (words) in the training data. However, in some cases, certain words may not appear in either the spam or ham (non-spam) messages, leading to zero probabilities when calculating conditional probabilities. This can cause issues during classification when encountering new or unseen words in the test data.
   To address this problem, additive smoothing is employed. Additive smoothing involves adding a small positive constant (usually denoted as alpha or Laplace parameter) to both the numerator and denominator of the probability calculation. By doing this, even if a feature has not been observed in the training data, it will still have a non-zero probability.
   In the code, the smoothing parameter alpha is set to a range of values [0.1, 0.5, 1, 2] in the params dictionary. Later, GridSearchCV is used to find the best value of alpha through cross-validation. This process helps in selecting the optimal smoothing parameter that results in better classification performance.
   During the training of the Multinomial Naive Bayes classifier (clf.fit(X_train_tfidf, y_train)), the smoothing parameter specified in params is utilized. The MultinomialNB class in scikit-learn implements the additive smoothing technique by default.

By applying additive smoothing, zero probabilities are avoided, and the classifier becomes more robust to unseen or rare words in the test data. It helps in preventing overfitting and improves the generalization capability of the model.

4. Do you consider n-gram models of different orders? If so, how does the n-gram order affect the classification accuracy?

For the order of **N-gram (2,2)**, the following output is obtained:

Accuracy: 0.9713004484304932

Confusion matrix:

```
[[966   0]
 [ 32 117]]
```

Classification report:

	precision	recall	f1-score	support
0	0.97	1.00	0.98	966
1	1.00	0.79	0.88	149
accuracy			0.97	1115
macro avg	0.98	0.89	0.93	1115
weighted avg	0.97	0.97	0.97	1115

For the order of **N-gram (3,3)**, the following output is obtained:

Accuracy: 0.9587443946188341

Confusion matrix:

```
[[966   0]
 [ 46 103]]
```

Classification report:

	precision	recall	f1-score	support
0	0.95	1.00	0.98	966
1	1.00	0.69	0.82	149
accuracy			0.96	1115
macro avg	0.98	0.85	0.90	1115
weighted avg	0.96	0.96	0.96	1115

For the order of **N-gram (2,4)**, the following output is obtained:

Accuracy: 0.9614349775784753

Confusion matrix:

```
[[966   0]
 [ 43 106]]
```

Classification report:

	precision	recall	f1-score	support
0	0.96	1.00	0.98	966
1	1.00	0.71	0.83	149
accuracy			0.96	1115
macro avg	0.98	0.86	0.90	1115
weighted avg	0.96	0.96	0.96	1115

Now, though the order of the Ngram is increased, the accuracy is seen to gradually reduce significantly by order 2-3%

The gradual reduction in accuracy can be attributed to the following factors.

Increased Dimensionality: Including higher n-grams expands the number of features used by the classifier. With more features, the model needs to estimate more parameters, which can be challenging, especially if the dataset is limited. It can lead to overfitting, where the model becomes too specialized to the training data and performs poorly on unseen data.

Sparse Data: As the n-gram range increases, the occurrence of specific n-grams becomes less frequent. This can result in sparse feature vectors, where many of the features have zero values. Sparse data can make it harder for the model to generalize and find meaningful patterns, potentially reducing accuracy.

Increased Noise: Including higher n-grams may introduce more noise into the feature representation. Longer n-grams can capture specific sequences that are not relevant or meaningful in the context of the task, in this case SMS spam detection, leading to decreased accuracy.

Loss of Generalization: Increasing the n-gram range can make the model more dependent on specific patterns and sequences in the training data. This can reduce its ability to generalize to new, unseen data, resulting in lower accuracy on the test set.

References

1. https://chat.openai.com/
   - Concept of reading zip file using pandas and zipfile library Lines: 11-12, 16 & 20.
   - Concept of data vectorization using sklearn model: Line 30, 33-35 & 38.
   - Concept of training the NB classifier using grid search and function of fitting: Lines 45&46.

**** The above lines of code were extracted from ChatGPT to adhere to syntax, however the methods used were determined after due diligence.**

2. https://scikit-learn.org/
3. https://docs.python.org/
4. https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c
5. https://scikit-learn.org/stable/modules/naive_bayes.html
6. http://gnjatovic.info/misc/ngram.models.pdf
7. http://gnjatovic.info/misc/naive.bayesian.classification.pdf
8. Input data: https://archive.ics.uci.edu/ml/machine-learning-databases/00228/

¶¶¶ Thank You for your Patience ¶¶¶