Data Mining - CA2

SMS SPAM CLASSIFIER

Submitted by:
Name: Uma Konar
Student Id: 20035958

Submitted to: Prof. Kunwar Madan

Introduction

The ever-increasing volume of spam messages in mobile communication presents a substantial challenge for users and telecom companies alike. With regulatory bodies pressuring service providers to protect their users from these unsolicited messages, the need for automated systems to detect and block spam has never been greater. To meet this demand, telecom companies often rely on machine learning techniques, which can efficiently classify messages as spam or ham (non-spam) based on historical data.

This report focuses on the development of a spam SMS classifier, employing two well-known classification algorithms: Naive Bayes and Support Vector Classifier (SVC). These models are evaluated and compared in terms of their effectiveness in identifying spam messages from a collection of SMS data. The goal is to build a model that can accurately distinguish between ham and spam messages, offering valuable insights into the most suitable model for real-world applications.

1. Data Preparation

Data preparation is the first and most critical step in building any machine learning model. The quality of the data directly impacts the performance of the model. We have used the SMS Spam Collection dataset, which contains labelled messages (either ham or spam). Preprocessing the data is essential because the raw text needs to be cleaned and transformed into a suitable format that machine learning algorithms can process.

1.1 Data Loading and Exploration

The dataset is first loaded using the pandas library to create a DataFrame. The two columns in the dataset represent the message label (either ham or spam) and the content of the message.

```
#Data loading
df = pd.read_csv("SMSSpamCollection", sep='\t', header=None, names=["label", "message"])
```

Upon inspection of the first five rows, we checked the dataset for any inconsistencies, such as missing values or duplicate entries. Fortunately, the dataset was clean and free from any issues that would prevent analysis. The next step was to preprocess the text data, which involved several stages to convert the raw text into a form suitable for machine learning models.

	label	message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

1.2 Text Preprocessing

The primary goal of text preprocessing is to standardize the text so that the model can effectively extract relevant features. The following preprocessing steps were applied to clean and prepare the text data for classification:

1.2.1 Removing HTML Tags

Some of the messages in the dataset contained HTML tags. These were removed using the **BeautifulSoup** library, which extracts only the text content from HTML elements.

```
#Removing HTML tags
soup = BeautifulSoup(text, 'lxml')
souped = soup.get_text()
```

1.2.2 URL Removal

Spam messages often include URLs, which are irrelevant for spam detection. Therefore, all URLs in the dataset were removed.

```
#Removing URLs, user mentions (@), and non-alphabetic characters
souped = re.sub(r"(@|http://|https://|www)\S*", " ", souped)  # Remove URLs
souped = re.sub(r"[^A-Za-z]+", " ", souped)  # Keep only alphabetic characters
```

1.2.3 Tokenization

Tokenization is the process of splitting a sentence into individual words (tokens). This step was necessary to convert the text data into a format that can be analyzed. The nltk library's word_tokenize() function was used for tokenization.

1.2.4 Removing Stopwords

Stopwords are common words like “the,” “and,” “is,” etc., which do not add much value in distinguishing between spam and ham messages. We used the nltk stopwords list to remove these words.

1.2.5 Lemmatization

Lemmatization reduces words to their base form. For instance, “running” becomes “run.” This ensures that different variations of a word are treated as the same feature. The WordNetLemmatizer from the nltk library was used for this process.

1.3 Label Encoding

For machine learning models to work, the target labels (ham and spam) must be converted into numeric values. Therefore, ham was mapped to 0 and spam to 1 using the map() function.

```
#Converting 'label' from categorical to numeric: ham = 0, spam = 1
df['label'] = df['label'].map({'ham': 0, 'spam': 1})
```

1.4 Limitations of Data Preparation

While the preprocessing steps were effective, certain limitations need to be acknowledged:

Text Preprocessing Step	Limitation
Tokenization	Tokenization splits text into individual words, but it may not capture the semantic context of phrases.
Stopword Removal	Removal of common stopwords (e.g., “the”, “and”, “is”) may inadvertently remove important spam-indicating terms (e.g., "free", "offer").
Lemmatization	Lemmatization reduces words to their root forms, but it may fail to handle irregular forms or words with multiple meanings.
Feature Extraction (CountVectorizer)	**CountVectorizer** counts the frequency of words, but it doesn't account for the importance of words across the entire corpus.
Class Imbalance (Dataset Issue)	The dataset contains more ham messages than spam messages, leading to class imbalance.
URL Removal & Special Character Handling	Removing URLs and special characters could eliminate potentially useful features, as spam messages often include suspicious URLs.

2. Model Creation and Evaluation

2.1 Classification Models Chosen

In this task, two classification models were selected for spam SMS classification: **Naive Bayes (MultinomialNB)** and **Support Vector Classifier (SVC)**. These models are widely used in text classification tasks and have demonstrated good performance in previous studies. Here, we evaluate their suitability for spam classification.

2.1.1 Naive Bayes (MultinomialNB)

Naive Bayes is a probabilistic model based on **Bayes' Theorem**, which calculates the posterior probability of each class given the input features (i.e., the words in the messages). It assumes that the features (words) are conditionally independent, given the class label. Despite its simplicity, Naive Bayes often performs surprisingly well in text classification tasks, especially when using discrete features like word counts.

For **Naive Bayes**, we use the **MultinomialNB** variant, which is designed to handle count-based data (such as word counts in text). It calculates the probability of each class by multiplying the individual probabilities of the features (words).

Reason for choosing Naive Bayes:

- Simple and fast to train, making it suitable for large datasets.
- Performs well when the features are conditionally independent (as is often the case in text classification tasks).
- Less prone to overfitting due to its simplicity.

2.1.2 Support Vector Classifier (SVC)

The **Support Vector Classifier (SVC)** is a powerful classification algorithm that aims to find the hyperplane that best separates the data points of two classes. It works by maximizing the margin between the classes, which helps achieve better generalization. SVC performs well in high-dimensional spaces, which makes it especially useful for text classification, where each unique word in the dataset corresponds to a feature.

SVC uses kernel functions to transform the input features into a higher-dimensional space, allowing it to model more complex decision boundaries. In this task, we use a **linear kernel** for simplicity, as we are working with high-dimensional text data where a linear separator may be sufficient.

Reason for choosing SVC:

- Effective in high-dimensional spaces, making it suitable for text classification tasks.
- Capable of modelling complex decision boundaries when used with non-linear kernels.
- Robust to overfitting, especially when using the linear kernel.

2.2 Evaluation Metrics and Prioritization

To assess the models' performance, we used several evaluation metrics:

- **Accuracy**: Measures the proportion of correctly classified instances.
- **Precision**: The proportion of true positive predictions among all predicted positives.
- **Recall**: The proportion of true positives among all actual positives.
- **F1-Score**: The harmonic mean of precision and recall, providing a balanced measure of both.
- **Confusion Matrix**: A table that helps visualize true positives, true negatives, false positives, and false negatives.

Metric	Priority	Explanation
Recall	Top Priority	Ensures the model detects as many spam messages as possible, minimizing false negatives (missed spam).
Precision	Second Priority	Reduces false positives, ensuring that flagged messages are truly spam.
F1-Score	Moderate Priority	Balances precision and recall, especially important in the context of an imbalanced dataset.
Accuracy	Lowest Priority	Can be misleading in imbalanced datasets and was therefore not prioritized in the final model evaluation.

2.3 Model Evaluation and Results

Metric	Naive Bayes	SVC
Precision	0.99	0.96
Recall	0.82	0.99
F1-score	0.90	0.98
Accuracy	0.91	0.98

The SVC model outperformed Naive Bayes, achieving an accuracy of 98%. It also had a more balanced precision (96%) and recall (99%) for spam, making it the superior model for spam detection.

2.4 Conclusion

In spam SMS classification, precision and recall are the most important metrics, with a strong focus on minimizing false positives and false negatives. The F1-score combines these two metrics, making it the best overall indicator of model performance. Other metrics like AUC and log-loss provide additional insights but are less critical in this case. The confusion matrix is invaluable for understanding specific errors, helping fine-tune the model further. Prioritizing these evaluation metrics ensures that the spam detection model balances both the cost of missing spam and incorrectly flagging legitimate messages.

3. Recommendations

3.1 Recommended Model for Real-World Use

After evaluating both Naive Bayes and Support Vector Classifier (SVC) models, SVC is recommended for real-world deployment in spam SMS classification due to its superior performance. SVC outperformed Naive Bayes in key metrics like precision and recall, achieving 98% accuracy and effectively identifying a high proportion of spam messages with minimal false positives. The model demonstrated a balanced precision of 96% and recall of 99%, ensuring that spam messages are accurately flagged while minimizing errors. Additionally, SVC's robust performance on imbalanced datasets (where ham messages outnumber spam) further highlights its suitability for real-world applications, where minimizing false negatives is crucial for protecting users from harmful messages. Given these factors, SVC is the most reliable model for effective and accurate spam detection in practical scenarios.

3.2 Improvements

While **SVC** is the recommended model, there are several improvements and optimizations that could further enhance its performance:

3.2.1 Handling Class Imbalance

To address this imbalance, techniques such as SMOTE (Synthetic Minority Over-sampling Technique) could be applied to oversample the spam class and make the dataset more balanced. Alternatively, undersampling of the ham class could be explored. This would allow the model to learn more effectively from the minority (spam) class, improving recall and overall performance.

3.2.2 Hyperparameter Tuning

Using RandomizedSearchCV or Bayesian optimization can explore the hyperparameter space more efficiently. In addition to C, other hyperparameters like the kernel (e.g., RBF, polynomial) and gamma can be optimized to achieve even better performance.

3.2.3 Advanced Feature Engineering

More advanced techniques such as word embeddings (e.g., Word2Vec or GloVe) could be explored to capture semantic meanings of words and relationships between them. Word embeddings represent words in dense vector spaces, capturing deeper meaning beyond word

frequency. These methods could improve classification accuracy, especially in complex spam detection scenarios where understanding the context is crucial.

3.2.4 Advanced Text Preprocessing (N-grams)

Capturing combinations of words (e.g., “free gift” or “win money”) rather than single words can provide better context, especially in spam classification. This approach helps capture relationships between words that may be important indicators of spam.

3.2.5 Handling URLs and Special Characters

Rather than completely removing URLs, it might be more beneficial to capture their presence as a binary feature indicating whether a message contains a URL. This would help the model distinguish between spam messages containing links and legitimate messages.

3.3 Justification for Recommendations

In real-world applications, detecting spam messages efficiently and accurately is paramount. SVC provides an excellent balance of precision and recall, ensuring that spam messages are caught without overwhelming users with false positives. These recommendations focus on enhancing the model's capacity to handle diverse and noisy text data, including informal language, typos, and variations in expression. By incorporating more sophisticated techniques like semantic understanding and feature optimization, these improvements aim to increase the model’s sensitivity to spam while maintaining its precision, striking the right balance for an effective and reliable spam detection system in real-world scenarios.