Spam Email Detection

Project Report

Nake Shrivatsa Aparanji
CS Dept. UoFM
Memphis TN - 38152

ABSTRACT

Email communication has become an integral part of our daily lives, but the increasing influx of spam emails poses a significant challenge. Spam emails, characterized by their unsolicited and often irrelevant content, are sent to a multitude of recipients with the intent of promoting commercial products or, in more malicious cases, perpetrating scams. The imperative to filter out these unwanted emails before they inundate users' inboxes has led to the development of various solutions.

In this project, I address the prevailing issue of spam emails by employing diverse Machine Learning algorithms. The implemented models include Naïve Bayes, Decision Tree, Random Forest, Support Vector Machine, Gradient Descent, and Ensemble Techniques. Through the analysis of these techniques, I aim to discern effective strategies for differentiating between spam (unwanted) and ham (desired) emails.

The primary goal is to present a comprehensive evaluation of the performance of each algorithm, considering factors such as accuracy, precision, recall, and F1 score. By leveraging the strengths of various Machine Learning approaches, I endeavor to propose an optimal solution for spam email detection that strikes a balance between precision and efficiency.

1 Introduction

The need to sift through this barrage of spam to identify and separate genuine, or "ham," emails has become an essential aspect of email management.

The ML algorithms implemented in this project include Naïve Bayes, Decision Tree, Random Forest, Support Vector Machine, Gradient Descent, and Ensemble Techniques. Each algorithm brings its own set of strengths and subtlety to the task of classifying emails. Naïve Bayes, for instance, relies on probabilistic models, while Decision Tree and Random Forest leverage tree-based structures. Support Vector Machine focuses on finding optimal hyperplanes, Gradient Descent optimizes parameters iteratively, and Ensemble Techniques combine multiple models for enhanced performance.

However, the journey to implementing these algorithms was not without challenges. The initial hurdle involved pre-processing the data to make it suitable for the algorithms. This included converting the target variable from categorical (spam/ham) to numerical, removing extraneous columns, tokenizing and lowercasing the messages, eliminating punctuation and stop words, and performing stemming. The choices made in these preprocessing steps significantly impacted the performance of the classification models.

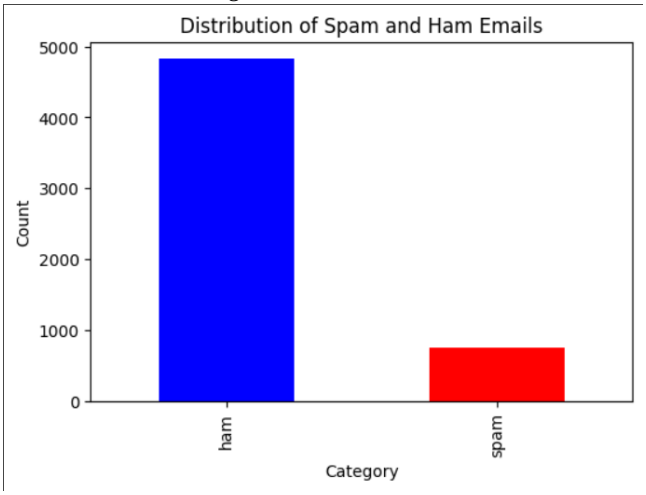
Another pivotal decision in the pre-processing phase was the choice between Count Vectorizer and TF-IDF Vectorizer. Count Vectorizer represents the occurrence of words in a document, while TF-IDF (Term Frequency-Inverse Document Frequency) Vectorizer takes into account the importance of a word in relation to the entire corpus. Selecting the appropriate vectorizer added an additional layer of complexity to the project, as the effectiveness of the chosen vectorizer influences the feature representation and, consequently, the performance of the classification models. Navigating through these challenges and optimizing the pre-processing steps set the foundation for the subsequent exploration and evaluation of classification algorithms in the realm of spam email detection.

My project is divided into 2 parts: (a) Using Count Vectorizer as the vectorization technique and implementing all the stated Machine Learning algorithms without any hyperparameter tuning, meaning, keeping it as simple as possible. (b) Using TF-IDF Vectorization as the vectorization technique and implementing the Machine Learning algorithms along with hyper parameter tuning, along with the additional Ensemble Technique, thus making it more complex. The results for both implementations are later displayed at the end. The following sections will delve deeper into the solution, results, and insights gained from the project.

1.1 Dataset

I used the “SMS Spam Collection” dataset (extracted from Grumble text Web site), the details of which are available publicly at http://archive.ics.uci.edu/dataset/228/sms+spam+collection. There is a collection of 5574 Email/SMS messages, of which 3375 are ham and 425 are spam. This is how the distribution of

the messages looked like:



I can say that the messages (dataset) are biased with ham messages.

Also, below is the description of the dataset:

```
1 # Get basic statistics
2 email.describe()
```

	email	msg	Unnamed: 2	Unnamed: 3	Unnamed: 4
count	5572	5572	50	12	6
unique	2	5169	43	10	5
top	ham	Sorry, I'll call later	bt not his girlfrnd... G o o d n i g h t . . .@"	MK17 92H. 450Ppw 16"	GNT:-)"
freq	4825	30	3	2	2

2 Solution

Supervised Learning, a paradigm within Machine Learning, involves training a model on a labeled dataset, where the desired output (target variable) is provided alongside the input features. Classification, a specific type of supervised learning, focuses on assigning input data to predefined categories or classes. In the context of spam email detection, this entails training a model to differentiate between spam and ham emails based on various features extracted from the email content. The selected classification algorithms for this project operate under the supervised learning paradigm, with each algorithm utilizing distinct methodologies to learn patterns and make predictions.

2.1 Pre-processing

2.1.1 Convert categorical column to numerical column. The target variable ('v1', later changed to 'email') had values 'ham' and 'spam'. These are categorical values and cannot be used as input to most of the Machine Learning algorithms. Therefore, I had to convert them to numerical values of 1 and 0 (1 for spam and 0 for ham). 'LabelEncoder', a library in Python was used to achieve this conversion.

2.1.2 Remove unwanted columns. The dataset contained some unnamed columns which had lots of NAN's, and these columns weren't necessary to classify a message. Only 2 columns – 'v1' and 'v2', later changed to 'email' and 'msg', were needed. Hence, I removed the other columns by using the 'drop()' in Pandas library.

2.1.3 Removal of stop words. The 'stopwords' module from the Natural Language Toolkit (NLTK) is used to obtain a set of common English stop words. Stop words are words that are frequently used in a language but typically do not contribute significant meaning to the context of a sentence. Examples include "the," "and," "is," etc. I have removed them from text data during preprocessing to focus on more meaningful words.

2.1.4 Stemming. Stemming algorithm, represented by the 'PorterStemmer' in my case, is applied to reduce words to their root or base form. It involves removing suffixes from words to retain their essential meaning. For example, the word "running" might be stemmed to "run."

2.1.5 Tokenization. The pre-processed tokens (after removing stop words and applying stemming) are joined back into a string, preserving the original sequence. This step creates a clean, preprocessed representation of the text that can be used for further analysis or modeling.

2.2 Vectorization

Vectorization is a crucial step in natural language processing tasks like spam email detection, transforming textual data into a numerical format that machine learning algorithms can comprehend. This conversion is necessary as algorithms typically operate on numerical data. Vectorization allows the representation of text documents as numerical vectors, enabling the extraction of features and patterns. I had to vectorize the messages which could then be sent as input to the Classification Algorithms.

2.2.1 Count Vectorization. 'CountVectorizer' is a basic technique for vectorization that represents each document as a vector of term frequencies, indicating the count of each word in the document. It builds a vocabulary of unique words in the entire corpus and assigns a numerical value to each word based on its frequency in a document. While simple and efficient, 'CountVectorizer' does not consider the importance of words across the entire corpus.

2.2.2 TF-IDF Vectorization. TF-IDF (Term Frequency-Inverse Document Frequency) Vectorizer is a more sophisticated vectorization technique that considers the importance of words in the context of the entire corpus. It not only takes into account the frequency of a term in a document but also penalizes terms that are common across multiple documents. This is achieved by multiplying the term frequency by the inverse document frequency. TF-IDF helps in identifying words that are distinctive to a document, making it particularly useful in tasks like spam email detection where distinguishing features are crucial.

In comparison, Count Vectorizer treats all words equally, considering only their frequency in individual documents. TF-IDF Vectorizer, on the other hand, provides a more nuanced representation by highlighting words that are both frequent in a document and distinctive across the entire corpus. While Count Vectorizer is simpler and computationally less intensive, TF-IDF Vectorizer often yields more meaningful features, especially in tasks where distinguishing between documents is paramount.

2.3 Classification Algorithms

```
1 # Naive Bayes
2 naive_bayes = MultinomialNB()
3
4 # Decision Tree
5 decision_tree = DecisionTreeClassifier(random_state=1)
6
7 # Random Forest
8 random_forest = RandomForestClassifier(random_state=1)
9
10 # SVM
11 svm_model = SVC(random_state=1)
12
13 # Gradient Boosting
14 gradient_boosting = GradientBoostingClassifier(random_sta
```

2.3.1 Naïve Bayes. It is a probabilistic algorithm based on Bayes' theorem, which calculates the probability of a hypothesis given observed evidence. In the context of spam filtering, Naïve Bayes assumes independence between features, considering each feature's impact on the classification independently. It is computationally efficient and particularly effective when dealing with high-dimensional data. MultinomialNB() initializes a Multinomial Naive Bayes classifier, specifically suitable for discrete data, such as word counts in text classification. This line (Line 2 from the above screenshot) creates an instance of the Multinomial Naive Bayes classifier named 'naive_bayes' for later use in training and predictions.

2.3.2 Decision Tree. Decision Trees are hierarchical structures where nodes represent decisions based on input features, and branches lead to subsequent decisions or outcomes. In spam filtering, decision trees recursively split the data based on features, creating a tree structure that aids in classification. The 'decision_tree' (Line 5 from the above screenshot) variable holds an instance of the Decision Tree classifier, which can be further trained on labeled data for classification tasks.

2.3.3 Random Forest. Random Forest, an ensemble technique, builds multiple decision trees and combines their outputs for improved accuracy and generalization. The 'random_forest' (Line 8 from the above screenshot) variable is an instance of the Random Forest classifier, capable of ensemble learning using multiple decision trees.

2.3.4 Support Vector Machine. SVM aims to find the optimal hyperplane that separates data into different classes. In spam filtering, SVM works by mapping input data into a higher-dimensional space and finding the hyperplane that maximally separates spam and ham emails. It is particularly effective in high-dimensional spaces and situations where clear class boundaries exist. The 'svm_model' (Line 11 from the above screenshot) variable represents an instance of the SVM classifier, suitable for binary and multiclass classification tasks.

2.3.5 Gradient Descent. Gradient Descent is an optimization algorithm used to minimize the error or loss function iteratively. In the context of spam filtering, gradient descent is applied to optimize parameters of the model for accurate classification. It is versatile and widely used for training various machine learning models. The 'gradient_boosting' (Line 14 from the above screenshot) variable holds an instance of the Gradient Boosting classifier, which builds an ensemble of weak learners to create a robust model.

2.3.6 Ensemble Technique. Ensemble techniques combine the predictions of multiple base models to produce a more robust and accurate final prediction. Ensemble methods reduce overfitting and improve the model's ability to generalize to new, unseen data. The below provided code demonstrates the creation and utilization of a Voting Classifier, an ensemble learning method that combines predictions from multiple individual classifiers to make a final decision. Here, 'VotingClassifier' is initialized with a list of tuples, where each tuple contains a string identifier for a classifier ('nb' for Naive Bayes, 'dt' for Decision Tree, etc.) and the corresponding classifier instance. The parameter voting='hard' indicates that the majority vote (hard voting) will be used to make the final decision.

```
1 # Combining predictions from all models using a majority vote.
2 voting_classifier = VotingClassifier(estimators=[
3     ('nb', nb_classifier),
4     ('dt', dt_classifier),
5     ('rf', rf_classifier),
6     ('svm', svm_classifier),
7     ('gb', gb_classifier)
8 ], voting='hard')
```

2.4 Methodology (ML Steps)

2.4.1 Data Pre-processing. Explained in Section 2.1. Along with that the dataset is split into training and testing sets using 'train_test_split' method from ScikitLearn library. The split is in the ratio 80:20, meaning training data is 80% and testing data is 20%.

2.4.2 Vectorization. Explained in Section 2.2.

2.4.3 Model Selection. Explained in Section 2.3.

2.4.4 Model Training. Here, the created Machine Learning models are trained, meaning, each of the models are applied to 'fit()'.

2.4.5 Hyperparameter Tuning. Parameter tuning is a crucial aspect of Machine Learning model development, involving the adjustment of hyperparameters to optimize a model's

performance. Hyperparameters are configuration settings external to the model that cannot be learned from training data. I have used RandomizedSearchCV, a method for hyperparameter tuning that conducts a randomized search over specified hyperparameter values. Instead of exhaustively trying all possible combinations, it randomly samples a defined number of parameter combinations from the given hyperparameter space. This approach is particularly useful when the search space is large, as it helps in efficiently narrowing down the possibilities. The number of iterations is controlled by the parameter 'n_iter', and the cross-validation strategy is defined by the parameter 'cv'. The best-performing set of hyperparameters is then selected based on the specified scoring metric. Below is the code snippet for hyperparameter tuning of Decision Trees:

```
# Decision Tree
dt_param_dist = {
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': randint(2, 11),
    'min_samples_leaf': randint(1, 5),
    'criterion': ['gini', 'entropy']
}
dt_random_search = RandomizedSearchCV(DecisionTreeClassifier(random_state=1), dt_param_dist, n_iter=10, cv=3)
dt_random_search.fit(x_train_tfidf, y_train)
dt_classifier = dt_random_search.best_estimator_
print(dt_classifier)
```

2.4.6 Model Evaluation. Explained later in Section 3.

2.5 Assumptions

- The selected features (words) are assumed to be relevant indicators for distinguishing between spam and non-spam emails.
- The assumption is made that the characteristics of spam emails remain relatively static during the model training and testing periods.
- The provided dataset is assumed to be representative of the general characteristics of spam and non-spam emails.

3 Empirical Experiments

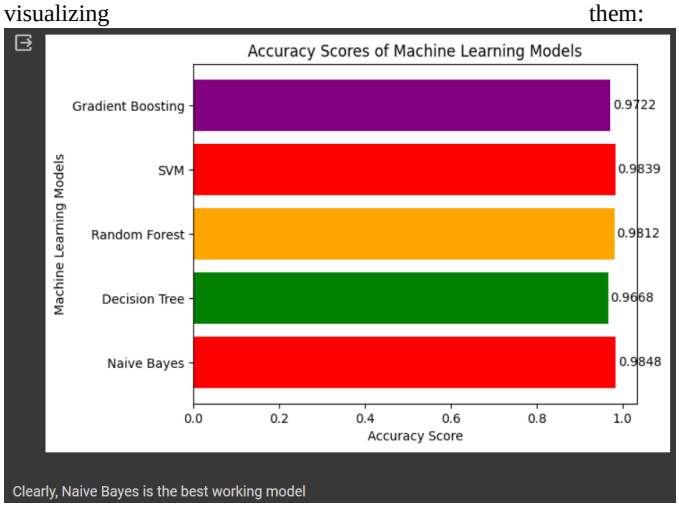
My As mentioned earlier, my project has 2 parts – one with Count Vectorizer and the other with TFIDF Vectorizer (along with Hyperparameter tuning and Ensemble Technique).

3.1 Part 1

These are the screenshots of Classification Report for all the ML models:

```
Naive Bayes Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       976
           1       0.96      0.92      0.94       139

    accuracy                           0.98      1115
   macro avg       0.97      0.96      0.96      1115
weighted avg       0.98      0.98      0.98      1115


Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       976
           1       0.86      0.87      0.87       139

    accuracy                           0.97      1115
   macro avg       0.92      0.93      0.92      1115
weighted avg       0.97      0.97      0.97      1115


Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       976
           1       0.98      0.87      0.92       139

    accuracy                           0.98      1115
   macro avg       0.98      0.93      0.95      1115
weighted avg       0.98      0.98      0.98      1115

SVM Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       976
           1       1.00      0.87      0.93       139

    accuracy                           0.98      1115
   macro avg       0.99      0.94      0.96      1115
weighted avg       0.98      0.98      0.98      1115


Gradient Boosting Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       976
           1       0.94      0.83      0.88       139

    accuracy                           0.97      1115
   macro avg       0.96      0.91      0.93      1115
weighted avg       0.97      0.97      0.97      1115
```

To get a better visual of the accuracy scores, below is the graph visualizing them:



Clearly, Naïve Bayes is the best working model with an accuracy score of 98.48%.

3.2 Part 2

These are the screenshots of Classification Report for all the ML models:

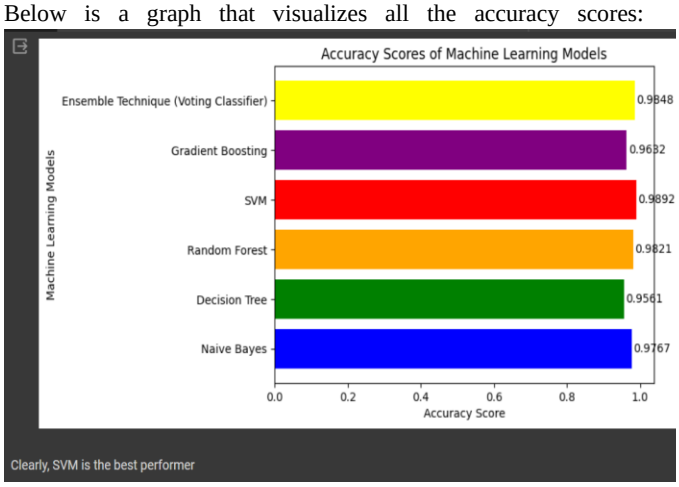
```
Naive Bayes (TF-IDF) Accuracy: 0.9766816143497757
Naive Bayes (TF-IDF) Classification Report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       976
           1       0.99      0.82      0.90       139

    accuracy                           0.98      1115
   macro avg       0.98      0.91      0.94      1115
weighted avg       0.98      0.98      0.98      1115


Decision Tree (TF-IDF) Accuracy: 0.9560538116591928
Decision Tree (TF-IDF) Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.97      0.97       976
           1       0.82      0.83      0.82       139

    accuracy                           0.96      1115
   macro avg       0.90      0.90      0.90      1115
weighted avg       0.96      0.96      0.96      1115
```

```
Random Forest (TF-IDF) Accuracy: 0.9820627802690582
Random Forest (TF-IDF) Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       976
           1       0.98      0.88      0.92       139

    accuracy                           0.98      1115
   macro avg       0.98      0.94      0.96      1115
weighted avg       0.98      0.98      0.98      1115


SVM (TF-IDF) Accuracy: 0.989237668161435
SVM (TF-IDF) Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       976
           1       0.99      0.92      0.96       139

    accuracy                           0.99      1115
   macro avg       0.99      0.96      0.97      1115
weighted avg       0.99      0.99      0.99      1115

 Gradient Boosting (TF-IDF) Accuracy: 0.9632286995515695
 Gradient Boosting (TF-IDF) Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98       976
           1       0.87      0.83      0.85       139

    accuracy                           0.96      1115
   macro avg       0.92      0.91      0.91      1115
weighted avg       0.96      0.96      0.96      1115
```

Also, the Classification Report of the Ensemble Technique looks like this:

```
Voting Classifier (TF-IDF) Accuracy: 0.9847533632286996
Voting Classifier (TF-IDF) Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       976
           1       1.00      0.88      0.93       139

    accuracy                           0.98      1115
   macro avg       0.99      0.94      0.96      1115
weighted avg       0.99      0.98      0.98      1115
```

Below is a graph that visualizes all the accuracy scores:



Clearly, SVM is the best performer.

3.3 Predicting Spam/Ham in real-time (using SVM with TF-IDF)

I am using the 'joblib' library to save and load a Support Vector Machine (SVM) classifier trained on a TF-IDF (Term Frequency-Inverse Document Frequency) representation. First step is to save the trained SVM classifier (svm_classifier) to a file named 'svm_model_tfidf.pkl' using the 'joblib.dump' function. Then, I loaded the previously saved SVM model from the file 'svm_model_tfidf.pkl' into a new variable 'loaded_model' using the 'joblib.load' function. Below, is the screenshot of the output where I am using the 'loaded_model' to predict if the user given sentence is spam/ham:

```
# Take input from the user in real-time
user_input = input("Enter the email text: ")

# Use the predict_spam_or_ham function to make a prediction with the loaded model
result = predict_spam_or_ham(user_input, loaded_model)

# Display the result
print(f"The email is predicted as: {result}")
```

```
Enter the email text: congratulations. you have just won a cash prize of $10000
The email is predicted as: spam
```

4 Conclusion

Naïve Bayes is a probabilistic classification algorithm that works well for text classification tasks like spam detection. With the use of the Count Vectorizer, it converts text data into a numerical format by counting the frequency of each word. This combination is effective for capturing the presence and frequency of words in emails, making it suitable for spam detection.

Support Vector Machines (SVM) are known for their effectiveness in high-dimensional spaces, making them suitable for text classification. In my project, SVM's preference for TF-IDF (Term Frequency-Inverse Document Frequency) representation indicates the importance of considering the significance of words in the entire corpus. Additionally, hyperparameter tuning likely played a crucial role in optimizing the SVM's performance, emphasizing the need to fine-tune parameters for better results.

5 Discussion

5.1 Limitations

5.1.1 Imbalanced Dataset. The "SMS Spam Collection" dataset exhibited a noticeable bias towards ham messages, with 3375 ham messages and only 425 spam messages. This imbalance poses a challenge in training robust models, as the algorithms may become skewed towards predicting ham.

5.1.2 Lack of Temporal Context. The analysis focused on the current state of the dataset without considering temporal aspects of spam email patterns. Incorporating temporal features and trends could improve the models' adaptability to evolving spam techniques and enhance their real-world applicability.

5.2 Future Works

5.2.1 Advanced Ensemble Techniques. Exploring advanced ensemble techniques and model stacking may yield improvements in overall classification performance.

5.2.2 Deep Learning Architectures. The application of deep learning architectures, such as recurrent neural networks (RNNs) or transformers, could provide a fresh perspective on spam email detection. These architectures have shown promise in capturing intricate patterns and dependencies within sequences of data, which might be advantageous in the context of email content.

REFERENCES

[1] http://archive.ics.uci.edu/dataset/228/sms+spam+collection
[2] https://www.semanticscholar.org/paper/An-Immunological-Based-Simulation%3A-A-Case-Study-of-Zainal-Jali/d2088475356b8c3f154e021cee205a2bd29b8f89
[3] https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing
[4] https://scikit-learn.org/stable/modules/grid_search.html