Spam Detection Model Development Document

Table of Contents

---

1. Title of the Project

Spam Detection using NLP

2. Brief on the Project

Project Type: NLP Model

Problem Statement:

Spam messages, whether in emails, SMS, or online forums, create numerous problems such as phishing attacks, unsolicited advertisements, and fraudulent activities. An effective spam detection system is essential to automatically classify messages as spam or ham (not spam) and improve user security and experience.

Motivation:

With the increasing volume of digital communication, spam messages have become a significant challenge. Traditional rule-based filtering methods struggle to keep up with the evolving nature of spam messages. A machine learning-based spam detection model can efficiently analyze text patterns and classify messages more accurately.

Previous Work:

- Traditional spam filtering using keyword-based approaches and blacklists.
- Machine learning techniques such as Naïve Bayes, Support Vector Machines (SVM), and deep learning models for text classification.

Tentative Approach:

- Collect and preprocess a dataset containing spam and non-spam messages.
- Explore feature engineering techniques such as TF-IDF and word embeddings.
- Train and evaluate machine learning models including Naïve Bayes, Logistic Regression, Random Forest, and Deep Learning models.
- Optimize model performance using hyperparameter tuning.

- Deploy the final model for real-time spam detection.

3. Deliverables of the Project

General Approach:

1. **Data Collection:** Obtain and clean a dataset of labeled spam and non-spam messages.
2. **Data Preprocessing:** Tokenization, stopword removal, stemming, lemmatization, and vectorization (TF-IDF, etc.).
3. **Model Training & Evaluation:** Train multiple classification models and evaluate performance using accuracy, precision, recall, and F1-score.
4. **Deployment:** Deploy the best-performing model in a real-world environment.

Questions the Model is Designed to Answer:

- Is a given message spam or ham?
- What are the most influential features in spam classification?
- How well does the model generalize to new, unseen data?

Details of the Model & Expected Observations:

- **Models Considered:** Naive Bayes, Logistic Regression, Random Forest Algorithm
- **Important Findings:** Feature selection and preprocessing play a crucial role in model performance
- **Expected Outcome:** A robust and scalable spam detection model with high accuracy and minimal false positives

4. Resources

Dataset Source:

- **SMS Spam Collection Dataset** from UCI Machine Learning Repository: https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

Software & Tools:

- **Programming Language:** Python
- **Libraries:** Pandas, Numpy, Scikit-learn, Beautifulsoup, NLTK, SpaCy, Matplotlib, Seaborn
- **Development Environment:** Jupyter Notebook / Google Colab

References:

1. **"A Comparative Study on Spam Detection Models"** - Research paper discussing various ML-based spam detection techniques.
2. **"Deep Learning for Text Classification"** - Explores LSTM and CNN for NLP tasks.
3. **"Efficient Spam Detection Using Naive Bayes and SVM"** - Discusses classical ML approaches.

5. Model Implementation

Data Preprocessing:

- Loaded dataset and performed data cleaning.
- Tokenized text and removed stopwords.
- Converted text data into numerical vectors using TF-IDF.

Model Training:

- Implemented Naïve Bayes, Logistic Regression, Random Forest, and LSTM models.
- Trained each model on the dataset and optimized hyperparameters.

6. Results and Evaluation

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	98 %	98%	98%	99%
Random Forest	99.8%	100%	99.7%	99.8%
SVM	99.8%	99.8%	99.7%	99.8%

Observations:

- The Random Forest model performed the best with 99% accuracy.
- Traditional models like Naïve Bayes and Logistic Regression performed well .
- Further improvements can be made by fine-tuning deep learning architectures on New set of data and more testing and evolution

7. Deployment Strategy

- **API Development:** Implementing a REST API using Flask or FastAPI.
- **Integration:** Deploying the model in an email or SMS filtering application.
- **Monitoring:** Continuous evaluation with real-world data to improve performance.

8. Challenges and Limitations

- **Imbalanced Data:** Spam datasets often contain more ham messages than spam messages.
- **Evolving Spam Tactics:** Spammers constantly change tactics to bypass filters.
- **Computational Cost:** Deep learning models require significant resources.

9. Future Improvements

- **Hybrid Models:** Combining traditional and deep learning models for better performance.
- **Real-time Classification:** Implementing low-latency models for real-time detection.
- **Multi-language Support:** Expanding the model to classify spam in multiple languages.

10. Model Code with Explanation

Data Processing

```
def clean_and_preprocess_text(text):
  # Remove HTML tags
  text = BeautifulSoup(text, "html.parser").get_text()

  # Convert to lowercase
  text = text.lower()

  # Remove punctuation
  text = re.sub(f"[{re.escape(string.punctuation)}]", "", text)

  # Tokenization
  tokens = word_tokenize(text)

  # Remove stop words
  stop_words = set(stopwords.words("english"))
  tokens = [token for token in tokens if token not in stop_words]

  # Stemming
  stemmer = PorterStemmer()
  tokens = [stemmer.stem(token) for token in tokens]

  # Join tokens into a single text string
  cleaned_text = " ".join(tokens)
```

```
# Load spaCy language model
nlp = spacy.load('en_core_web_sm')


def lemm(data):
  lemmanized = []
  for i in range(len(data)):
    lemmed = []
    doc = nlp(data['Message'].iloc[i])
    for token in doc:
      lemmed.append(token.lemma_)
    lemmanized.append(lemmed)


  data['lemmanized'] = lemmanized
  data['text'] = data['lemmanized'].apply(' '.join)
  data = data.drop("lemmanized", axis=1)
  data = data.drop("Message", axis=1)
  return data


# Call the lemm function with your data
df = lemm(df)
```

```
# Initialize the TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer()


# Transform the training data using TF-IDF
X_train_transformed = tfidf_vectorizer.fit_transform(X_train)


# Transform the test data using the same TF-IDF configuration
X_test_transformed = tfidf_vectorizer.transform(X_test)
```

```
# ROC-AUC Score
roc_auc = roc_auc_score(y_test,
logistic_regression_model.predict_proba(X_test_transformed)[:, 1])
print("ROC-AUC Score:", roc_auc)


# Create a DataFrame for evaluation metrics
evaluation_metrics = {
   "Accuracy": accuracy_score(y_test, predictions ),
   "Precision": precision_score(y_test, predictions),
   "Recall": recall_score(y_test, predictions),
   "F1-Score": f1_score(y_test, predictions),
   "ROC-AUC Score": roc_auc_score(y_test,
logistic_regression_model.predict_proba(X_test_transformed)[:, 1])
}

metrics_df = pd.DataFrame([evaluation_metrics])
print("\nEvaluation Metrics Table:")

print(metrics_df)
# Transform the training data using TF-IDF
X_train_transformed = tfidf_vectorizer.fit_transform(X_train)
```

```
# Analysing Using confusion Matrix

cm = confusion_matrix(y_test, predictions)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Ham', 'Spam'], yticklabels=['Ham', 'Spam'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()
```

```
# Initialize a Random Forest Classifier
random_forest_classifier = RandomForestClassifier()

# Train the classifier using the training data
random_forest_classifier.fit(X_train_transformed, y_train)

# Make predictions on the test data
predictions = random_forest_classifier.predict(X_test_transformed)

# Print the accuracy score of the predictions
print("Accuracy Score:", accuracy_score(predictions, y_test))
```

10.Conclusion

the spam detection model has achieved an accuracy of 99%, indicating excellent performance. The high precision and recall scores suggest that the model effectively identifies spam messages while minimizing false positives and false negatives. However, further testing on real-world data is recommended to validate robustness. The model can be further optimized by fine-tuning parameters and incorporating more diverse training data**.**

11. Individual Details

- **Name:** Arun Gourh
- **Email:** arungourh12935@gmail.com
- **Phone Number:** +918819909979

*****