Spam-Detector-ThunderBird Report

AAS

Telmo Sauce (104428) Ricardo Covelo (102668)

Spam-Detector-ThunderBird

AAS
DETI
Universidade de Aveiro

Telmo Sauce (104428) Ricardo Covelo (102668)

telmobelasauce@ua.pt, ricardocovelo11@ua.pt

08/01/2023

Index

Chapter 1

Introduction

This report provides an overview of the development process and decision-making in creating our spam detection add-on for Thunderbird. We will explain the tools and machine learning techniques used to train and predict incoming emails. Additionally, we utilized an open-source add-on for Thunderbird and incorporated some code provided by the professor as the foundation for our project.

Chapter 2

Development

2.1 Add-On

While developing our add-on, we explored various tools for Thunderbird that allowed us to implement Python scripts for email analysis.

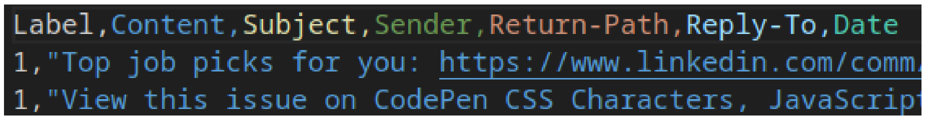
Eventually, we came across an open-source add-on called *Scriptable Notifications* [4]. By examining the source code of this add-on available on the repository *thunderbird-addon-scriptable-notifications* [5], we determined that the add-on utilizes the *webextension api* [6] to establish communication with Thunderbird. With this knowledge, we incorporated additional features to enhance the application. For example, we implemented a response mechanism that allows Thunderbird to classify emails as spam or ham by sending a corresponding message.

2.2 Machine Learning Training

Dataset

To train our model, we collected a diverse set of emails from our email boxes. We gathered 5,000 spam emails and 5,000 ham (non-spam) emails, which were downloaded in a ".eml" format.

To organize these emails, we developed a script called "sort.py" that transformed all the ".eml" files into a single CSV file, as shown in Figure 2.1.

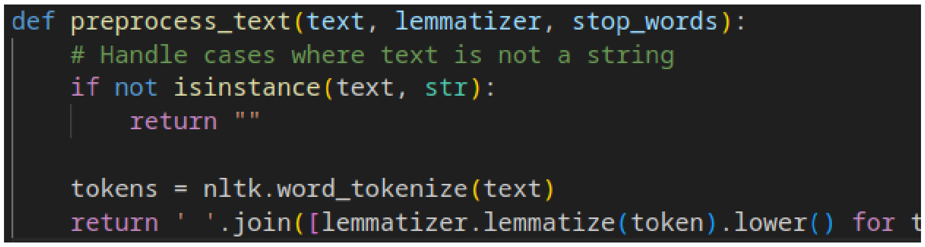


```
Label,Content,Subject,Sender,Return-Path,Reply-To,Date
1,"Top job picks for you: https://www.linkedin.com/comm
1,"View this issue on CodePen CSS Characters, JavaScript
```

Figure 2.1: CSV Dataset

Text Training

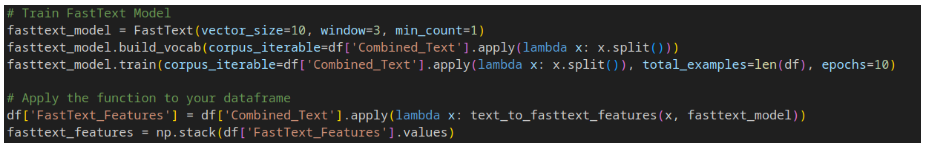
To begin the training process, we combined each content and subject into a single column called "Combined_Text". Subsequently, we implemented text preprocessing techniques, such as tokenization and lemmatization, to simplify the words in the content and subject of the emails, as shown in Figure 2.2.



```python
def preprocess_text(text, lemmatizer, stop_words):
    # Handle cases where text is not a string
    if not isinstance(text, str):
        return ""

    tokens = nltk.word_tokenize(text)
    return ' '.join([lemmatizer.lemmatize(token).lower() for t
```

Figure 2.2: Processed Text Data

Next, we trained a FastText model using Gensim's FastText implementation to convert the text data into numerical features, as shown in figure 2.3. Finally, we stored the trained model in a file called "fasttext.pkl" for future use.



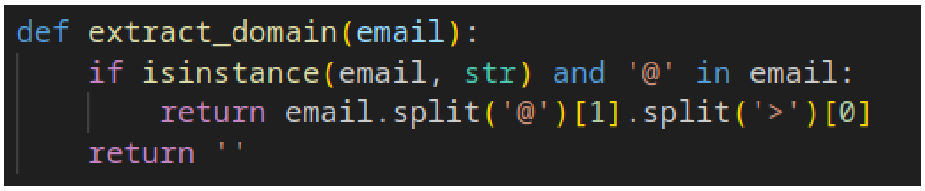
```python
# Train FastText Model
fasttext_model = FastText(vector_size=10, window=3, min_count=1)
fasttext_model.build_vocab(corpus_iterable=df['Combined_Text'].apply(lambda x: x.split()))
fasttext_model.train(corpus_iterable=df['Combined_Text'].apply(lambda x: x.split()), total_examples=len(df), epochs=10)

# Apply the function to your dataframe
df['FastText_Features'] = df['Combined_Text'].apply(lambda x: text_to_fasttext_features(x, fasttext_model))
fasttext_features = np.stack(df['FastText_Features'].values)
```

Figure 2.3: Fast Text Process

Header Fields Training

To train the Header Fileds we used a Hot-encoder from *scikit-learn* [3] but before this, we made modifications using some suggestions from *anatomy-phishing-email* [1] to the fields "From", "Reply-to", and "Return-Path" in the header. Specifically, we removed the email username, retaining only the domain. This process is illustrated in Figure 2.4.

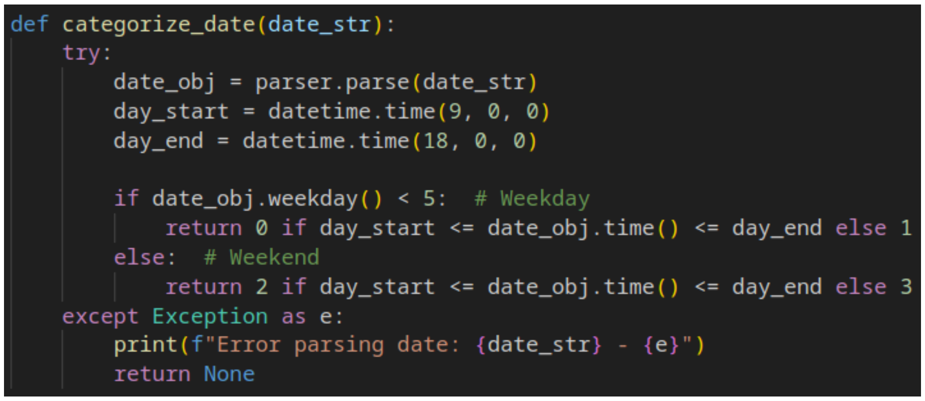
Additionally, we compared the "Sender" and "Return-Path" fields, as it was mentioned in the article that if these two fields differed, it was likely to be a Spoofing email.



```python
def extract_domain(email):
    if isinstance(email, str) and '@' in email:
        return email.split('@')[1].split('>')[0]
    return ''
```

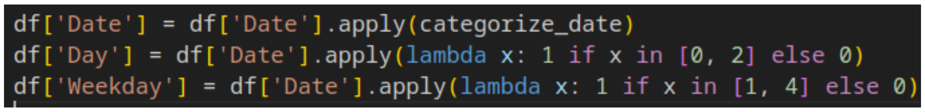
Figure 2.4: Process Emails

In the Date field, we performed further processing by splitting it into two separate columns. The first column specifies whether the date falls on a weekday or weekend, while the second column indicates whether it is daytime or nighttime. This process is displayed on the Figures 2.5 2.6. Afterward, we saved the Hot encoder into a file named "hot_encoder.pkl".



```python
def categorize_date(date_str):
    try:
        date_obj = parser.parse(date_str)
        day_start = datetime.time(9, 0, 0)
        day_end = datetime.time(18, 0, 0)

        if date_obj.weekday() < 5:  # Weekday
            return 0 if day_start <= date_obj.time() <= day_end else 1
        else:  # Weekend
            return 2 if day_start <= date_obj.time() <= day_end else 3
    except Exception as e:
        print(f"Error parsing date: {date_str} - {e}")
        return None
```

Figure 2.5: Process the Date



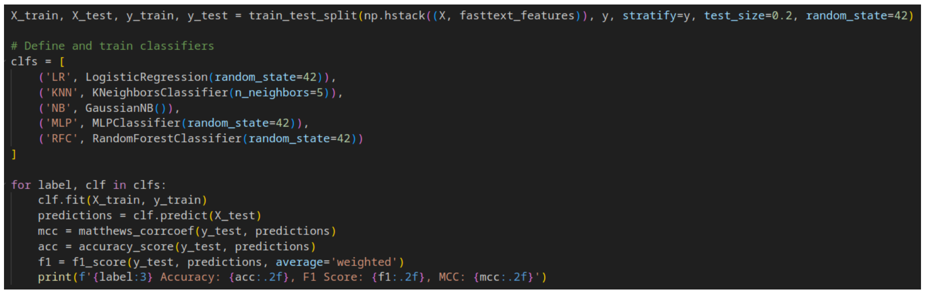
```python
df['Date'] = df['Date'].apply(categorize_date)
df['Day'] = df['Date'].apply(lambda x: 1 if x in [0, 2] else 0)
df['Weekday'] = df['Date'].apply(lambda x: 1 if x in [1, 4] else 0)
```

Figure 2.6: Process the date into two columns

Model Selection

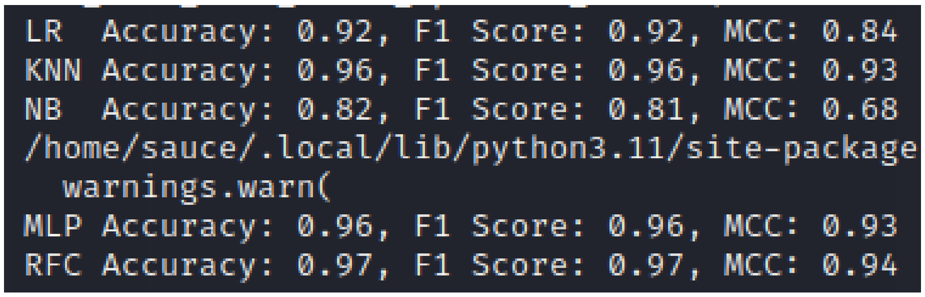
In our project, after incorporating and processing these features, We trained the model using five different types of Machine Learning classifiers, as depicted in Figure 2.7.

Each classifier was assigned accuracy, F1 score, and MMC values after the training process. These metrics were then analyzed to determine the best classifier for our case. After careful evaluation, we identified the Random Forest Classifier as the top-performing model, as demonstrated in Figure 2.8.



```python
X_train, X_test, y_train, y_test = train_test_split(np.hstack((X, fasttext_features)), y, stratify=y, test_size=0.2, random_state=42)

# Define and train classifiers
clfs = [
    ('LR', LogisticRegression(random_state=42)),
    ('KNN', KNeighborsClassifier(n_neighbors=5)),
    ('NB', GaussianNB()),
    ('MLP', MLPClassifier(random_state=42)),
    ('RFC', RandomForestClassifier(random_state=42))
]

for label, clf in clfs:
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_test)
    mcc = matthews_corrcoef(y_test, predictions)
    acc = accuracy_score(y_test, predictions)
    f1 = f1_score(y_test, predictions, average='weighted')
    print(f'{label:3} Accuracy: {acc:.2f}, F1 Score: {f1:.2f}, MCC: {mcc:.2f}')
```

Figure 2.7: Model Classifiers



```
LR  Accuracy: 0.92, F1 Score: 0.92, MCC: 0.84
KNN Accuracy: 0.96, F1 Score: 0.96, MCC: 0.93
NB  Accuracy: 0.82, F1 Score: 0.81, MCC: 0.68
/home/sauce/.local/lib/python3.11/site-package
  warnings.warn(
MLP Accuracy: 0.96, F1 Score: 0.96, MCC: 0.93
RFC Accuracy: 0.97, F1 Score: 0.97, MCC: 0.94
```

Figure 2.8: Model Training Results

2.3 Prediction Results

After training our model, we performed our accuracy tests using the test dataset generated during the training phase. This test dataset consisted of 2,000 emails. With this, we calculated the accuracy and obtained the same value as that achieved during the training process.

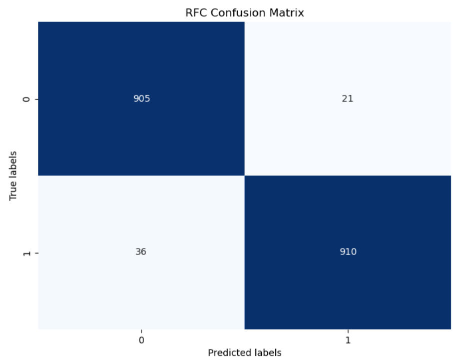


Figure 2.9: Confusion Matrix Calculated During Training

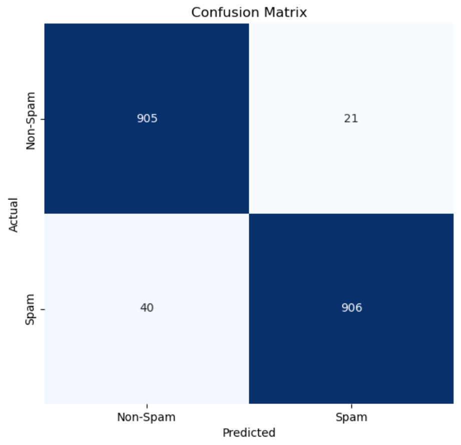


Figure 2.10: Confusion Matrix Calculated During Testing

The accuracy obtained was 96.5%, whereas, during training, the accuracy reached 97%. In an attempt to improve accuracy, we explored additional features, such as Authentication-Results, as suggested in *how-to-read-email-headers* [2]. However, these modifications did not show any improvement. We believe these features would help the model if we had a larger dataset with a more diverse range of data.

Chapter 3

Conculsion

In conclusion, our project has yielded successful results that we are satisfied with. We have been able to identify spam using our current dataset effectively, however, we acknowledge that there is room for improvement. By enhancing the accuracy of our spam detection algorithms, we can further enhance the effectiveness of our project.

Furthermore, expanding our dataset to include a wider range of spam formats would be beneficial. Currently, our dataset is tailored to our specific use case, but by incorporating a more diverse and extensive dataset, we can identify different types of spam with varying formats.

Therefore, our plans involve improving the accuracy of our spam detection algorithms and acquiring a larger and more diverse dataset. This will enable us to continue making strides in spam identification and contribute towards a safer online environment.

Bibliography

[1] *anatomy-phishing-email.* URL: `https://www.linkedin.com/pulse/anatomy-phishing-email-whats-header-penelope-raquel-bise-/`.

[2] *how-to-read-email-headers.* URL: `https://netcorecloud.com/blog/how-to-read-email-headers-to-identify-spam/`.

[3] *scikit-learn.* URL: `https://scikit-learn.org/stable/index.html`.

[4] *Scriptable Notifications.* URL: `https://addons.thunderbird.net/en-us/thunderbird/addon/scriptable-notifications/?src=cb-dl-created`.

[5] *thunderbird-addon-scriptable-notifications.* URL: `https://github.com/electrotype/thunderbird-addon-scriptable-notifications`.

[6] *webextension api.* URL: `https://webextension%20api.thunderbird.net/en/stable/`.