Final Project- Natural Language Processing

INFO7390
Fall 2023

Presented by Neha Agnihotri & Sejal Chandak

Outline

- Dataset: Spam Message Detection Dataset- https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset
- Goal: To discern the most effective machine learning model for text classification by assessing various algorithms on the SMS Spam Collection Dataset, leveraging accuracy, precision, recall, and F1-score as key performance indicators
- Result: Concluded with the best suited model and designed Streamlit application

Tools used

Python

Jupyter Notebook

Dataset

Dataset Link: https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

	Data Set Characteristics	Attribute Characteristics	Associated Tasks	Number of Instances	Number of Attributes
Dataset	Multivariate	Real	Classification	5621	5000

- The SMS Spam Collection Dataset contains over 5,500 messages labeled as 'spam' or 'ham', providing a binary classification challenge.
- It includes a diverse range of message content, from typical day-to-day communications to various forms of commercial and unsolicited spam.
- This dataset is widely used for natural language processing and machine learning tasks, serving as a benchmark for spam detection algorithms.

Methodology

Data Exploration

```
# Basic Data Exploration
print("Number of Rows:", len(df))
print("\nColumn Names:")
print(df.columns)
print("\nLabel Distribution:")
print(df['v1'].value_counts())

df.columns = ['label', 'message']
df['label'] = df['label'].apply(lambda x: 1 if x == 'spam' else 0)


stop_words = set(stopwords.words('english'))
ps = PorterStemmer()

def preprocess_text(text):
    text = re.sub('[^a-zA-Z]', ' ', text)
    text = text.lower()
    words = word_tokenize(text)
    words = [ps.stem(word) for word in words if word not in stop_words]
    return ' '.join(words)

df['message'] = df['message'].apply(preprocess_text)
```

```
# Step 2: TF-IDF Vectorization
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
X_tfidf = tfidf_vectorizer.fit_transform(df['message'])
y = df['label']
```

Methodology

Data Exploration

Displayed the dataset's size, column names, and distribution of spam vs. ham labels.

```
Number of Rows: 5572

Column Names:
Index(['v1', 'v2'], dtype='object')

Label Distribution:
ham      4825
spam      747
Name: v1, dtype: int64
```

Renamed columns for clarity and encodes labels as binary values (1 for spam, 0 for ham).

Implemented Porter Stemmer to reduce words to their root form, helping standardize message text for analysis.

Defined a function to preprocess text by removing non-alphabetic characters, converting to lowercase, tokenizing, removing stopwords.

Applied TF-IDF vectorization to the messages, converting them into numerical data with a maximum of 5,000 features for model training.

Methodology

Splitting the Data:

```
# Step 3: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42)
```

Splits the TF-IDF vectorized data into training and testing sets, allocating 20% of the data for testing, with a consistent random state for reproducibility

Methodology

Model Selection

- Naive Bayes: A probabilistic classifier known for its simplicity and effectiveness in text classification, particularly useful for its speed and baseline performance.
- Support Vector Machine (SVM): Offers robustness and high accuracy in high-dimensional spaces, ideal for feature-rich text data.
- Convolutional Neural Network (CNN): Excels in capturing local patterns within data, making it effective for identifying key textual features in spam detection.
- Long Short-Term Memory (LSTM): Capable of understanding long-term dependencies in text, beneficial for capturing contextual nuances in message classification.

Methodology

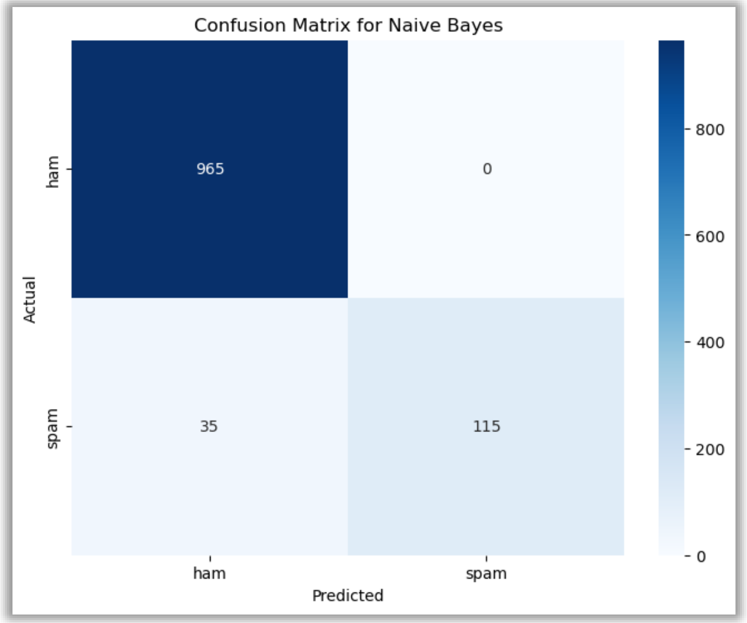
Model Selection

Naive Bayes:

```python
# Step 4: Naive Bayes Model
nb_model = MultinomialNB()
nb_model.fit(X_train, y_train)
nb_preds = nb_model.predict(X_test)

nb_accuracy = accuracy_score(y_test, nb_preds)
nb_precision = precision_score(y_test, nb_preds)
nb_recall = recall_score(y_test, nb_preds)
nb_f1 = f1_score(y_test, nb_preds)

print(f'Naive Bayes Model:')
print(f'Accuracy: {nb_accuracy}, Precision: {nb_precision}, Recall: {nb_recall}, F1 Score: {nb_f1}')
```



The Naive Bayes model, trained on the dataset, shows high accuracy in classifying messages, with the confusion matrix indicating strong true positive and true negative rates, yet some spam messages are misclassified as ham. The calculated accuracy, precision, recall, and F1 score metrics further quantify the model's performance.

```
Naive Bayes Model:
Accuracy: 0.968609865470852, Precision: 1.0, Recall: 0.7666666666666667, F1 Score: 0.8679245283018869
```

Methodology

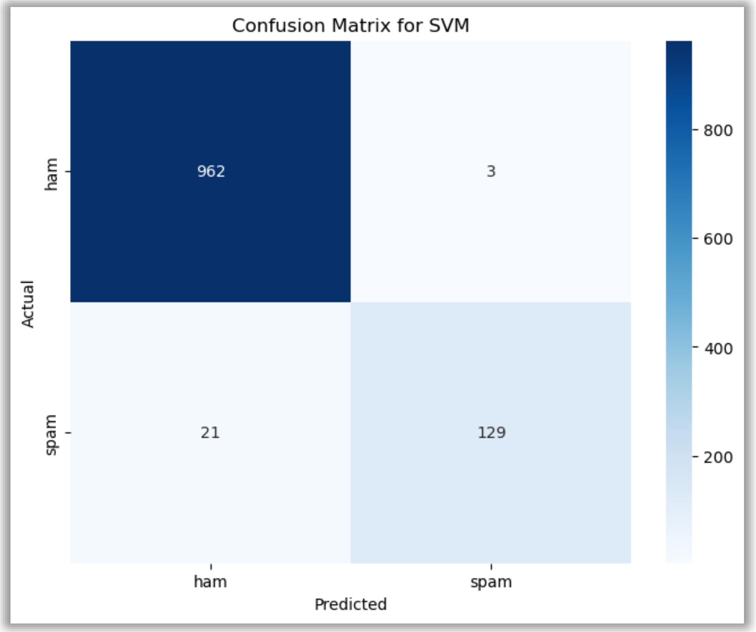
Model Selection

SVM:

```
# Step 5: SVM Model
svm_model = SVC(kernel='linear')
svm_model.fit(X_train, y_train)
svm_preds = svm_model.predict(X_test)

svm_accuracy = accuracy_score(y_test, svm_preds)
svm_precision = precision_score(y_test, svm_preds)
svm_recall = recall_score(y_test, svm_preds)
svm_f1 = f1_score(y_test, svm_preds)

print(f'SVM Model:')
print(f'Accuracy: {svm_accuracy}, Precision: {svm_precision}, Recall: {svm_recall}, F1 Score: {svm_f1}')
```



The SVM model demonstrates robust classification capabilities with a confusion matrix showing high true positives and negatives, and minimal misclassifications, underscored by strong accuracy, precision, recall, and F1 score metrics reflecting its efficacy in text categorization.

```
SVM Model:
Accuracy: 0.97847533632287, Precision: 0.9772727272727273, Recall: 0.86, F1 Score: 0.9148936170212766
```

Methodology

Model Selection

CNN:

```
# Step 6: CNN Model
max_words = 1000
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(df['message'])
X_seq = tokenizer.texts_to_sequences(df['message'])
max_len = max(len(x) for x in X_seq)
X_seq = pad_sequences(X_seq, maxlen=max_len)

X_train_seq, X_test_seq, y_train_seq, y_test_seq = train_test_split(X_seq, y, test_size=0.2, random_state=42)

cnn_model = Sequential([
    Embedding(input_dim=max_words, output_dim=128, input_length=max_len),
    Conv1D(filters=64, kernel_size=3, activation='relu'),
    GlobalMaxPooling1D(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

cnn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history = cnn_model.fit(X_train_seq, y_train_seq, epochs=10, batch_size=32, validation_split=0.2)
```

The CNN model is configured with text tokenization and padding, and its architecture includes an embedding layer, convolutional layer, global max pooling, and dense layers, optimized for binary classification. It is trained and validated on tokenized message data, demonstrating its ability to capture local textual patterns for effective spam detection.

Methodology

Model Selection

CNN:

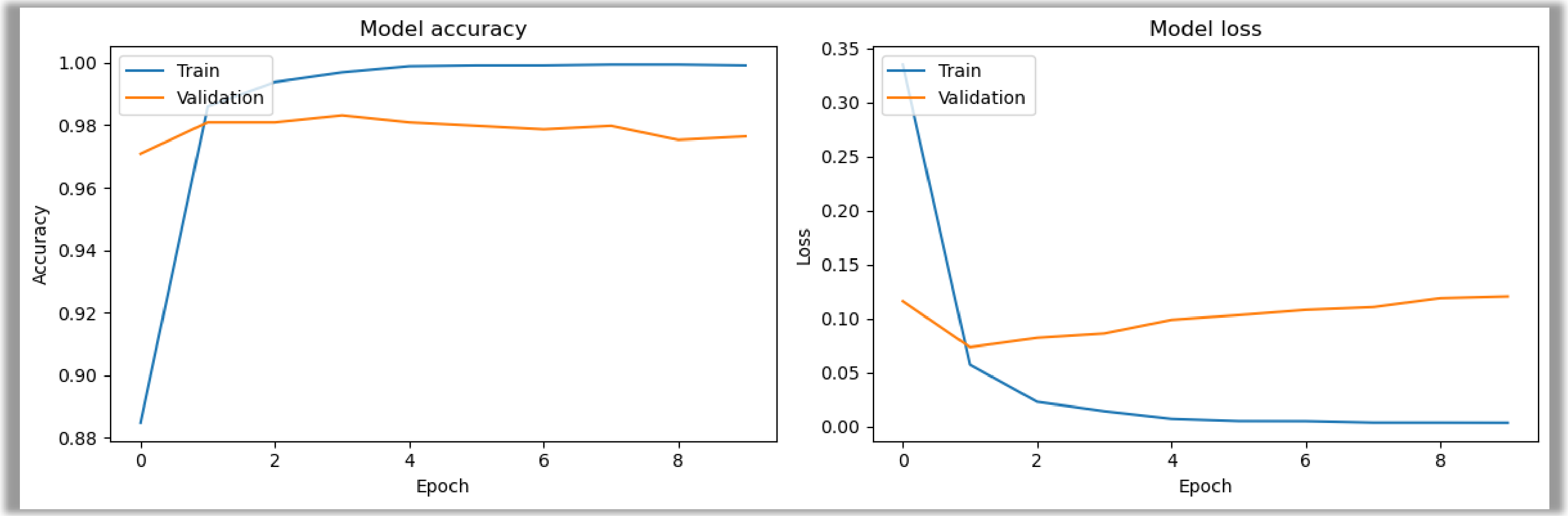
```
Epoch 1/10
112/112 [==============================] - 7s 30ms/step - loss: 0.3350 - accuracy: 0.8847 - val_loss: 0.1158 - val_accuracy: 0.
9709
Epoch 2/10
112/112 [==============================] - 3s 24ms/step - loss: 0.0570 - accuracy: 0.9860 - val_loss: 0.0733 - val_accuracy: 0.
9809
Epoch 3/10
112/112 [==============================] - 3s 27ms/step - loss: 0.0227 - accuracy: 0.9938 - val_loss: 0.0819 - val_accuracy: 0.
9809
Epoch 4/10
112/112 [==============================] - 3s 23ms/step - loss: 0.0139 - accuracy: 0.9969 - val_loss: 0.0860 - val_accuracy: 0.
9832
Epoch 5/10
112/112 [==============================] - 2s 20ms/step - loss: 0.0069 - accuracy: 0.9989 - val_loss: 0.0985 - val_accuracy: 0.
9809
Epoch 6/10
112/112 [==============================] - 2s 19ms/step - loss: 0.0048 - accuracy: 0.9992 - val_loss: 0.1030 - val_accuracy: 0.
9798
Epoch 7/10
112/112 [==============================] - 2s 19ms/step - loss: 0.0047 - accuracy: 0.9992 - val_loss: 0.1079 - val_accuracy: 0.
9787
Epoch 8/10
112/112 [==============================] - 2s 20ms/step - loss: 0.0033 - accuracy: 0.9994 - val_loss: 0.1104 - val_accuracy: 0.
9798
Epoch 9/10
112/112 [==============================] - 2s 19ms/step - loss: 0.0033 - accuracy: 0.9994 - val_loss: 0.1187 - val_accuracy: 0.
9753
Epoch 10/10
112/112 [==============================] - 2s 19ms/step - loss: 0.0032 - accuracy: 0.9992 - val_loss: 0.1203 - val_accuracy: 0.
9765
```

- An epoch in machine learning is a complete pass through the entire training dataset, consisting of both forward and backward propagation, with the number of epochs being a hyperparameter that determines how many times the data is iterated for model training.
- The use of epochs in training machine learning models is to iteratively optimize the model's weights to minimize error, with each epoch providing the model an opportunity to learn and improve its predictions on the dataset.

Methodology

Model Selection

CNN:



The left graph shows the model's accuracy on the training and validation datasets increasing with each epoch, indicating the model's improving performance on both datasets over time.

The right graph illustrates the model's loss decreasing sharply with the initial epochs and then stabilizing, suggesting that the model is effectively learning from the data with diminishing returns on improvement after a certain number of epochs

Methodology

Model Selection

RNN-LSTM:

```python
# Defining the LSTM Model
lstm_model = Sequential([
    Embedding(input_dim=max_words, output_dim=128, input_length=max_len),
    LSTM(64, dropout=0.2, recurrent_dropout=0.2),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])

lstm_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Training the LSTM Model
lstm_history = lstm_model.fit(X_train_seq, y_train_seq, epochs=10, batch_size=32, validation_split=0.2)

# Saving the LSTM Model
lstm_model.save('lstm_model.h5')
```

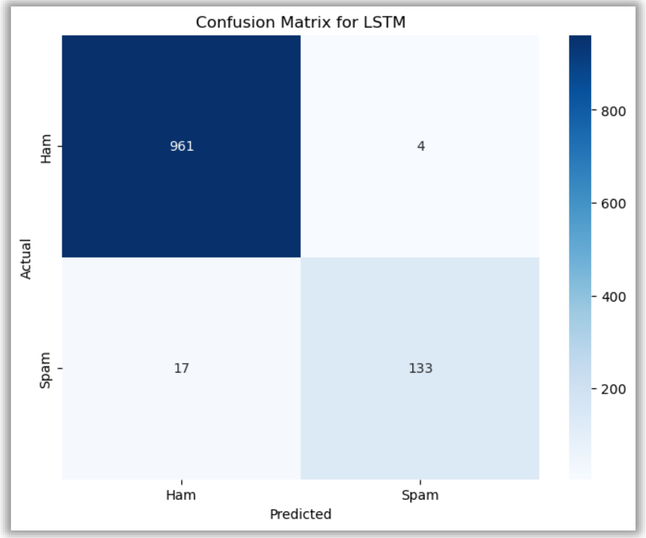
```
Epoch 1/10
112/112 [==============================] - 26s 112ms/step - loss: 0.2006 - accuracy: 0.9332 - val_loss: 0.0627 - val_accuracy:
0.9843
Epoch 2/10
112/112 [==============================] - 11s 100ms/step - loss: 0.0428 - accuracy: 0.9891 - val_loss: 0.0577 - val_accuracy:
0.9854
Epoch 3/10
112/112 [==============================] - 11s 100ms/step - loss: 0.0219 - accuracy: 0.9955 - val_loss: 0.0704 - val_accuracy:
0.9832
Epoch 4/10
112/112 [==============================] - 11s 100ms/step - loss: 0.0144 - accuracy: 0.9964 - val_loss: 0.0775 - val_accuracy:
0.9798
Epoch 5/10
112/112 [==============================] - 11s 95ms/step - loss: 0.0099 - accuracy: 0.9983 - val_loss: 0.0839 - val_accuracy:
0.9843
Epoch 6/10
112/112 [==============================] - 10s 91ms/step - loss: 0.0092 - accuracy: 0.9972 - val_loss: 0.0897 - val_accuracy:
0.9843
Epoch 7/10
112/112 [==============================] - 10s 89ms/step - loss: 0.0054 - accuracy: 0.9994 - val_loss: 0.1272 - val_accuracy:
0.9809
Epoch 8/10
112/112 [==============================] - 10s 89ms/step - loss: 0.0054 - accuracy: 0.9989 - val_loss: 0.1390 - val_accuracy:
0.9809
Epoch 9/10
112/112 [==============================] - 10s 89ms/step - loss: 0.0068 - accuracy: 0.9980 - val_loss: 0.1399 - val_accuracy:
0.9809
Epoch 10/10
112/112 [==============================] - 10s 88ms/step - loss: 0.0039 - accuracy: 0.9992 - val_loss: 0.1384 - val_accuracy:
0.9809
```

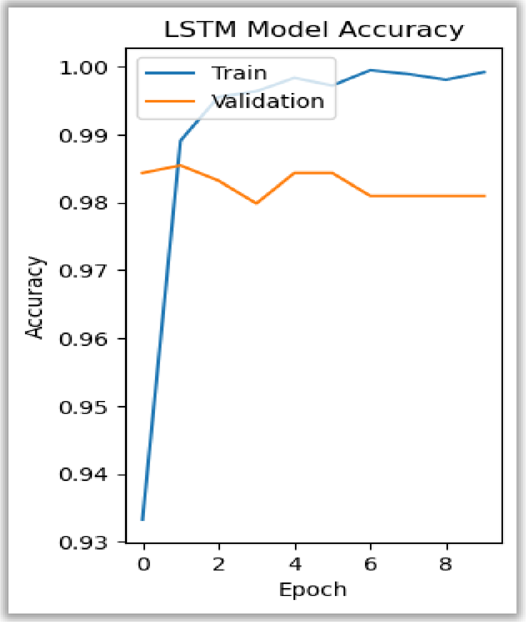
The LSTM model, structured with embedding, LSTM, and dense layers, is trained over 10 epochs, showing a rapid decrease in loss and increase in accuracy, indicating effective learning. However, the validation loss starts increasing after the initial epochs, suggesting overfitting to the training data.

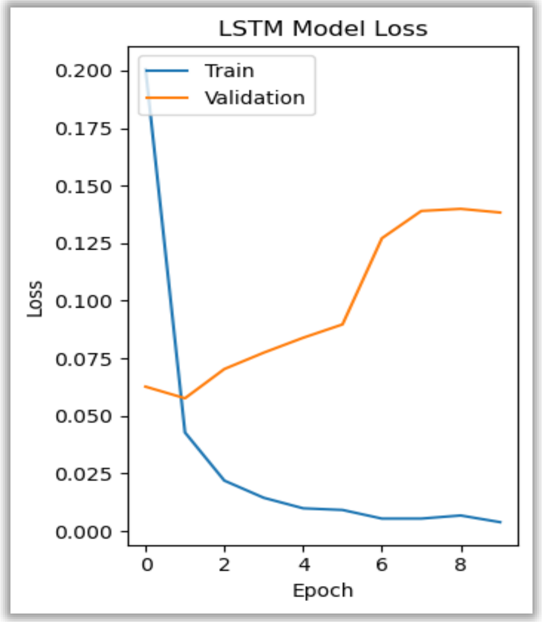
Methodology

Model Selection

RNN-LSTM:







The LSTM model's confusion matrix shows high accuracy in classifying 'ham' and 'spam', with few misclassifications. The accuracy and loss graphs indicate that the model quickly learns to a high degree of accuracy but begins to show signs of overfitting as the validation loss increases after an initial decrease, despite high accuracy on the validation set.

```
LSTM Model Evaluation:
Accuracy: 0.9811659192825112, Precision: 0.9708029197080292, Recall: 0.8866666666666667, F1 Score: 0.926829268292683
```

Result

- In order to display the results we used Streamlit to display the UI for Naive Bayes and SVM models

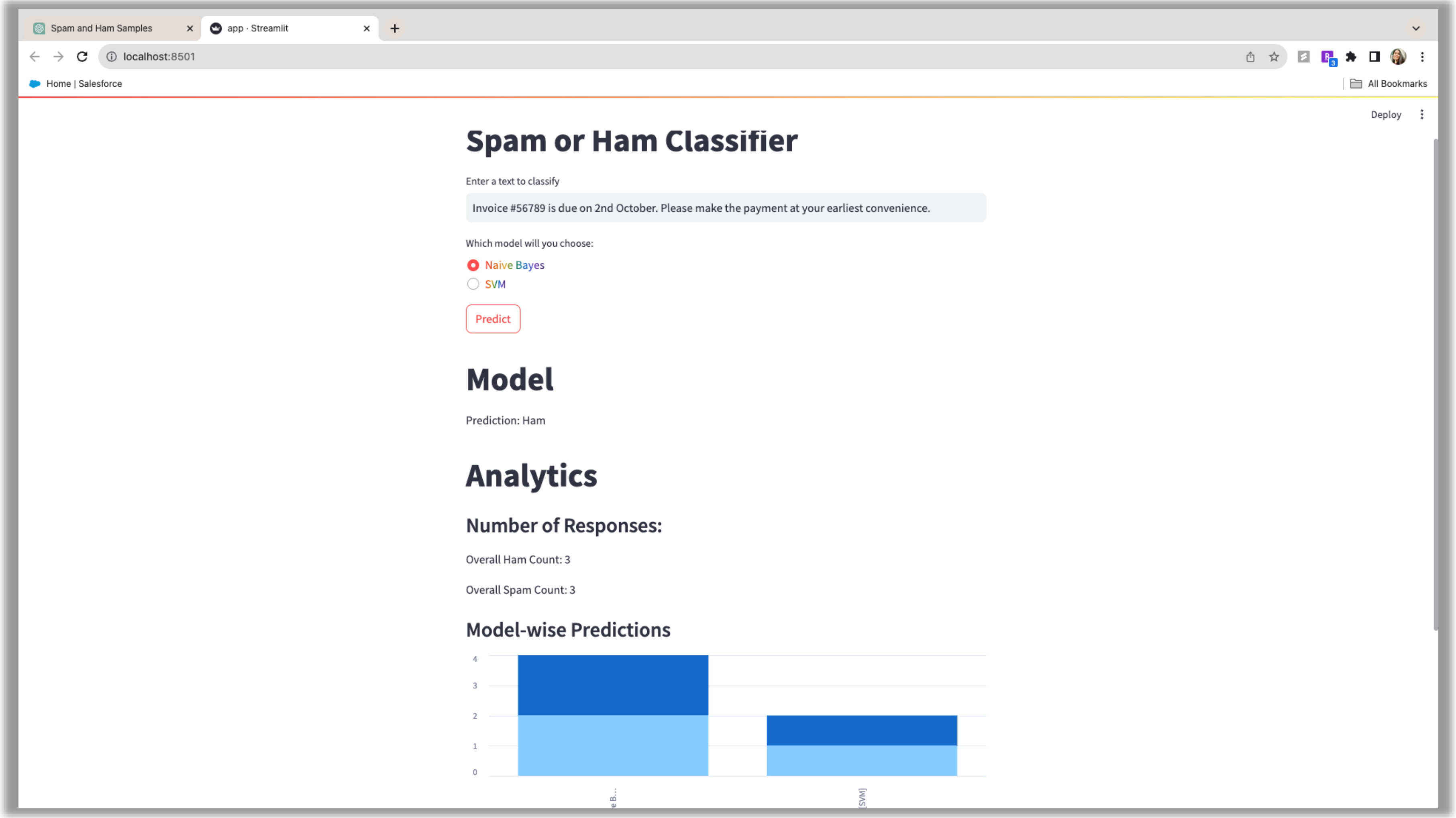
```python
# Save both the model and the TF-IDF vectorizer
file_name = 'saved_model_with_vectorizer.pkl'
with open(file_name, 'wb') as file:
    pickle.dump(svm_model, file)
    pickle.dump(tfidf_vectorizer, file)
svm_preds = svm_model.predict(X_test)
```

- Streamlit is an open-source Python library that allows for the creation of web applications for data exploration and visualization. It simplifies the process of building web apps for machine learning, data analysis, and other projects, enabling developers and data scientists to create interactive and customizable web applications with minimal effort.

- To create the streamlit application, we used Visual Studio Code to and created a app.py file to write the code to display the models in the streamlit and use pickle library to save and load trained machine learning model

- For analytics we stored the result of Spam / Ham in a database using sqlite and displayed the bar chart analytics based on number of Spam and Ham responses received on the inputs

Result


Spam and Ham Samples
app · Streamlit
localhost:8501
Home | Salesforce
All Bookmarks
Deploy
Spam or Ham Classifier
Enter a text to classify
Invoice #56789 is due on 2nd October. Please make the payment at your earliest convenience.
Which model will you choose:
Naive Bayes
SVM
Predict
Model
Prediction: Ham
Analytics
Number of Responses:
Overall Ham Count: 3
Overall Spam Count: 3
Model-wise Predictions
4
3
2
1
0
[SVM]

Conclusion

- The project centered around the development of an SMS spam detection system using machine learning models and the creation of an interactive interface through Streamlit. The primary models utilized for this task were Convolutional Neural Network (CNN), Support Vector Machine (SVM), RNN-LSTM and Naive Bayes.
- The choice of machine learning models was strategic and aimed at leveraging the strengths of each:
- **Convolutional Neural Network (CNN):** Employed for its process in understanding intricate patterns within text sequences.
- **RNN-LSTM :** Model was chosen for spam-ham detection due to its inherent capability to effectively capture sequential patterns and long-term dependencies within text sequences.
- **Support Vector Machine (SVM):** Chosen for its effectiveness in high-dimensional feature space and capability in text classification.
- **Naive Bayes:** Used as a simple baseline model, given its efficiency in text classification tasks.
- **Streamlit** was the interface used to implement and showcase the functionality of these models. The system allowed users to input text messages and receive predictions indicating whether the message was spam or non-spam.

Learning Outcome

- **Model Understanding:** Deeper comprehension of CNN's ability to recognize patterns, SVM's performance in high-dimensional spaces, and Naive Bayes' reliance on conditional probabilities for text classification.
- **Data Preprocessing:** Familiarization with text preprocessing techniques, including handling textual data, numerical representation, and stemming, for machine learning readiness.
- **Streamlit Implementation:** Skill development in building interactive and user-friendly applications that showcase machine learning model predictions.

Thank you :)