IT 7143 – CLOUD ANALYTICS TECHNOLOGY

Lab Assignment – 7

Saketh_Tatipally

KSU ID – 001119926

Dataset and source:

This dataset is collected from Kaggle.com. This dataset is about messages which are classified in ham or spam. This dataset contains two columns. One column contains the actual message and other column contains whether message is ham or spam.

Link to dataset:

https://www.kaggle.com/datasets/team-ai/spam-text-message-classification

Link to Base Notebook:

https://www.kaggle.com/code/aruneshhh/text-classification-spam

Goal, tasks & Motivation:

Goal:

The goal of this analysis is collecting the dataset, analyzing it and develop a robust machine learning model that can accurately predict whether a message is ham or spam.

Tasks:

This project contains following tasks performed:

- Data collection from source.
- Data Cleaning.
- Data Analysis & Visualization
- Machine learning modeling.

Motivation:

The motivation behind this project is to develop a robust and high accurate machine learning classification model that can classify a message is ham or spam.

Data Collection, Visualization & Samples:

Data Collection:

Reading Data File

```
[61] # Reading datafile and showing top 10 rows
     df = pd.read_csv(dataset_path)
     df.head()
```

	Category	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...

Data overview:

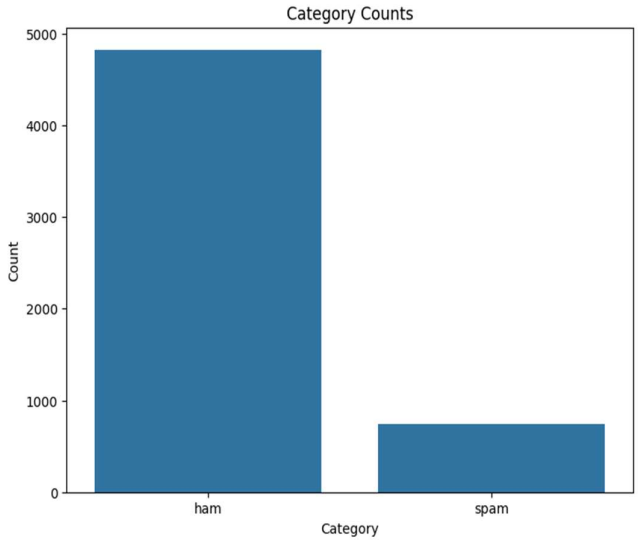
This dataset is collected from Kaggle.com. The source of dataset is https://www.kaggle.com/datasets/team-ai/spam-text-message-classification. I downloaded dataset from this link and saved it on my computer storage and then loaded it into google drive.

Data samples:

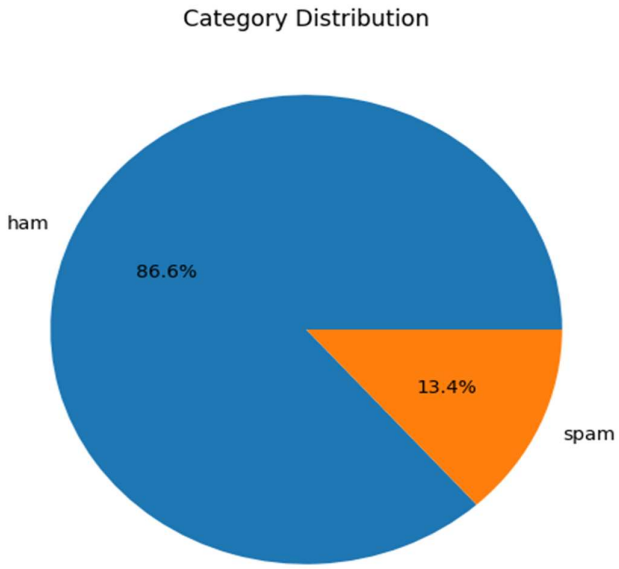
```
[ ]  # Shape of pandas dataframe
     df.shape

     (5572, 2)
```

This dataset contains total 5572 rows and two columns. The first column is about the category of message and second column is the actual message.

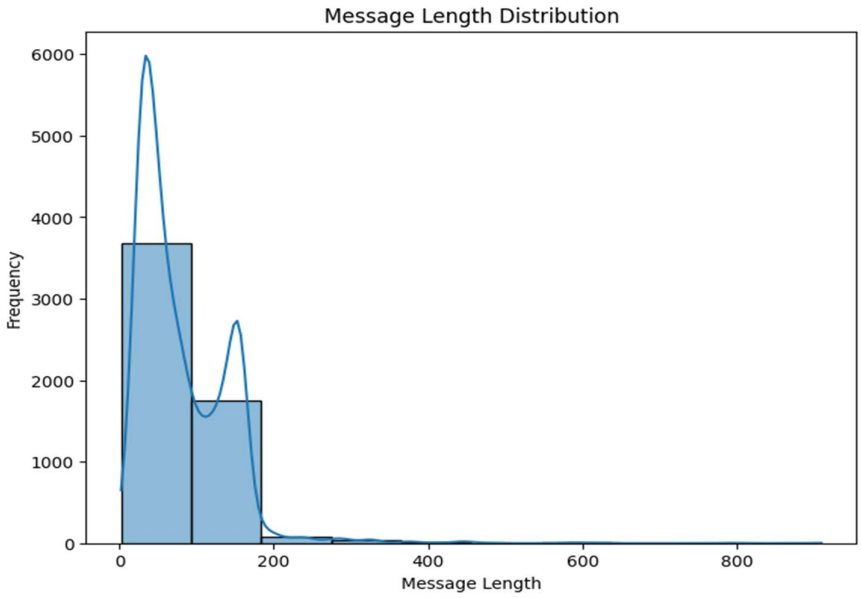
Data Visualization:



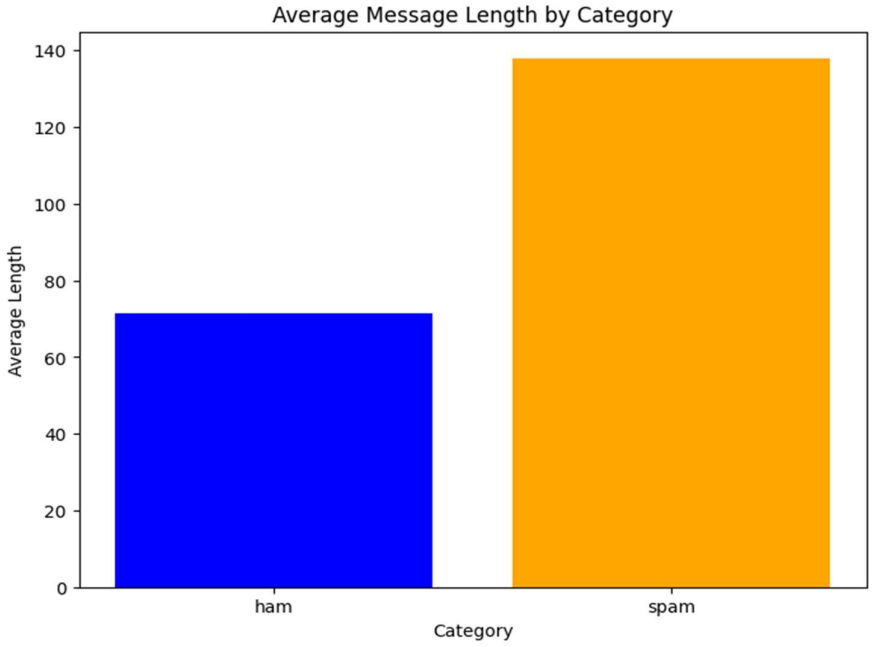
This visualization shows the number of hams and spams based on the dataset there are 4825 ham messages and 747 spam messages.



This visualization shows the %of hams and spams There are 13.4% of spams and 86.6% hams.



This visualization shows the message length distribution. Most of the messages have length between 0 to 100 followed by 100 to 200.



The spam messages on average have more length as compared to ham messages. The average length of spam messages are about 140 words and of ham are about 70 words.

This word cloud shows the top words used in the messages. Bigger the bubble shows the most frequently word is used in the messages. Smaller the words shows it is less frequently used in the messages.

Data Preprocessing:

```
def update(cat):
    if cat == "spam":
        return 1
    elif cat == "ham":
        return 0
    return cat
df.loc[:, "Category"] = df["Category"].apply(update)
df.head()
```

I defined a function called **update** that changes values in the "Category" column of a DataFrame based on certain conditions: if the value is "spam", it becomes 1; if it's "ham", it becomes 0. Then, it applies this function to every element in the "Category" column using the **apply** method.

```
# Text cleaning function
def clean_text(text):
    # Convert to lowercase
    text = text.lower()
    # Remove non-alphanumeric characters and digits
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Tokenize words
    words = word_tokenize(text)
    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in stop_words]
    # Join words back into sentence
    cleaned_text = ' '.join(words)
    return cleaned_text

# Apply text cleaning to the 'review' column
df['Message'] = df['Message'].apply(clean_text)
```

Then I defined this function named **clean_text** to process text data. It converts the text to lowercase, removes non-alphanumeric characters and digits, tokenizes the words, removes stopwords, and then reassembles the cleaned text. After defining the function, it applies this cleaning process to the "Message" column of a DataFrame called **df** using the **apply** method, updating the column with the cleaned text.

```
from sklearn.preprocessing import LabelEncoder

# Convert labels to integers
label_encoder = LabelEncoder()
df['Category'] = label_encoder.fit_transform(df['Category'])
```

```
# TF-IDF vectorization
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(df['Message'])
y = df['Category']
```

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

The code snippet utilizes scikit-learn for preprocessing text data. It uses **LabelEncoder** to convert categorical labels into integers and **TfidfVectorizer** for TF-IDF vectorization, creating numerical features from text. It then splits the dataset into training and testing sets using **train_test_split**.

Then baseline notebook does not have clean text function. To clean text first. Also, does not had label encoder function utilized.

Machine Learning:

```
accuracy_scores = {}

# Defining models
models = {
    'Multinomial Naive Bayes': MultinomialNB(),
    'Linear SVM': LinearSVC(),
    'Logistic Regression': LogisticRegression(),
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(n_estimators=100),
    'Gradient Boosting': GradientBoostingClassifier(),
    'XGBoost': XGBClassifier(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Multilayer Perceptron': MLPClassifier(),
}
```

The code sets up a dictionary to store accuracy scores, then defines a variety of machine learning models including

- Multinomial Naive Bayes
- Linear SVM
- Logistic Regression
- Decision Tree
- Random Forest
- Gradient Boosting
- XGBoost
- K-Nearest Neighbors
- Multilayer Perceptron.

I also defined deep learning model as well.

```python
for model_name, model in models.items():
    # Train the model
    model.fit(X_train, y_train)

    # Evaluate the model
    y_pred = model.predict(X_test)

    # Calculate accuracy score
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores[model_name] = accuracy

    # Print classification report
    print(f"Classification Report for {model_name}:")
    print(classification_report(y_test, y_pred))


    # Plot confusion matrix as heatmap
    plot_confusion_matrix(y_test, y_pred, f'Confusion Matrix for {model_name}')

# Print accuracy scores
plt.figure(figsize=(10, 6))
plt.bar(accuracy_scores.keys(), accuracy_scores.values(), color='skyblue')
plt.xlabel('Model')
plt.ylabel('Accuracy Score')
plt.title('Accuracy Scores of Different Models')
plt.xticks(rotation=45)
plt.show()
```

This code iterates over each model defined in the models dictionary. For each model, it trains the model on the training data (X_train, y_train), evaluates its performance on the test data (X_test, y_test), calculates the accuracy score using accuracy_score from scikit-learn, and stores the accuracy score in the accuracy_scores dictionary under the corresponding model name.

After that, it prints the classification report for each model using classification_report from scikit-learn, and calls the plot_confusion_matrix function to plot the confusion matrix heatmap for each model.

Evaluation Metrics used:

Accuracy, Classification report and Confusion matrix.

Hyperparameter optimization for Algorithms used:

Multinomial Naive Bayes: For Multinomial Naive Bayes, you could optimize the alpha smoothing parameter through techniques like grid search or randomized search.

Linear SVM: Linear SVM's regularization parameter (C) can be optimized using techniques like grid search or randomized search.

Logistic Regression: Logistic Regression's regularization parameter (C) can also be optimized using techniques like grid search or randomized search.

Decision Tree: Decision Tree's hyperparameters such as maximum depth, minimum samples split, and minimum samples leaf can be optimized using techniques like grid search or randomized search.

Random Forest: Random Forest's hyperparameters including the number of trees (n_estimators), maximum depth, minimum samples split, and minimum samples leaf can be optimized using techniques like grid search or randomized search.

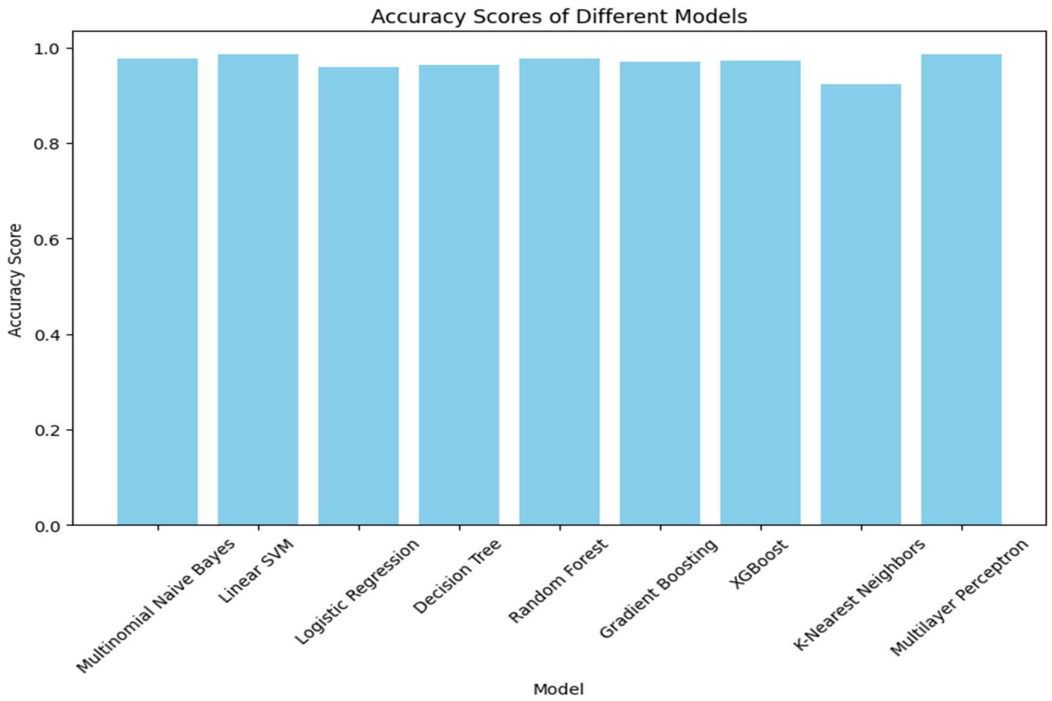
Gradient Boosting: Gradient Boosting's hyperparameters such as the learning rate, maximum depth, and number of trees can be optimized using techniques like grid search or randomized search.

XGBoost: XGBoost's hyperparameters including learning rate, maximum depth, and number of trees can be optimized using techniques like grid search or randomized search.

K-Nearest Neighbors: For K-Nearest Neighbors, you can optimize the number of neighbors (n_neighbors) using techniques like grid search or randomized search.

Multilayer Perceptron: For Multilayer Perceptron, you can optimize hyperparameters like the number of hidden layers, number of neurons per layer, and activation functions using techniques like grid search or randomized search.

Final Results:



Based on accuracy the best model to predict whether message is spam or ham is multilayer perceptron.

Summary:

Findings & Conclusions:

- The project aimed to develop a robust machine learning model for classifying messages as spam or ham.
- Data collection from Kaggle and subsequent cleaning revealed a dataset with 5572 messages categorized as ham or spam.
- Visualization highlighted the distribution of ham and spam messages, as well as message length characteristics.
- Preprocessing involved text cleaning and numerical encoding of labels using LabelEncoder.

- Various machine learning models, including traditional algorithms and deep learning, were trained and evaluated.
- Hyperparameter optimization was suggested for each algorithm to enhance model performance.
- Multilayer Perceptron emerged as the best-performing model based on accuracy for this classification task.

Insights & Challenges:

- Understanding data distributions and characteristics aided in feature selection and model interpretation.
- Challenges included ensuring robustness of text preprocessing and selecting appropriate hyperparameters for each algorithm.
- Interpretability of deep learning models like Multilayer Perceptron might be limited compared to traditional algorithms.

Recommendations for Future Work:

- Experimentation with advanced text preprocessing techniques, such as word embeddings or more sophisticated cleaning methods, could further enhance model performance.
- Exploring ensemble methods or advanced deep learning architectures could provide additional performance gains.
- Continuous monitoring and updating of the model with new data could help maintain its effectiveness over time.