BUILDING A SMARTER AI POWERED SPAM CLASSIFIER

TEAM MEMBER

ABISH. S
963521104004

PHASE 5

SUBMISSION DOCUMENT

TOPIC:BUILDING A SMARTER AI POWERED SPAM CLASSIFIER

INTRODUCTION

Building a smarter AI-powered spam classifier is a critical endeavor in today's digital age where spam messages inundate our communication channels. Spam filters play a pivotal role in safeguarding user experience, privacy, and information security. To develop a more effective spam classifier, a comprehensive approach is needed, involving advanced artificial intelligence techniques, data preprocessing, feature engineering, and continuous learning. In this introduction, we'll outline the key components and steps in the process of building a smarter AI-powered spam classifier.

1. Understanding the Problem:

Spam, in the context of emails, text messages, or other digital communication, refers to unsolicited, irrelevant, or harmful content. A smarter AI-powered spam classifier aims to differentiate between legitimate and spam messages, enhancing the accuracy of classification.

2. Data Collection:

To build a robust spam classifier, it is crucial to collect a diverse and representative dataset of both spam and non-spam (ham) messages. This dataset serves as the foundation for training and evaluating the AI model.

3. Data Preprocessing:

Data cleaning and preprocessing are essential to remove noise, standardize formats, and extract relevant features from the text data. This step prepares the data for machine learning.

4. Feature Engineering:

Feature engineering involves the extraction and selection of discriminative features from the text. Features can include word frequencies, n-grams, sender information, and more. Careful feature engineering contributes significantly to the model's performance.

5. Model Selection:

Choosing the appropriate machine learning or deep learning model is crucial. Common choices include Naive Bayes, Support Vector Machines, Random Forests, or more advanced models like recurrent neural networks (RNNs) or transformers.

6. Training the Model:

The model is trained on the labeled dataset, learning to distinguish spam from non-spam messages. It should be capable of generalizing well to unseen data.

7. Evaluation and Validation:

The classifier's performance is assessed using metrics like precision, recall, F1-score, and accuracy. Cross-validation techniques are often employed to ensure robustness.

8. Hyperparameter Tuning:

Optimizing hyperparameters and fine-tuning the model is essential to achieve the best possible results. This may involve adjusting learning rates, batch sizes, or architectural choices.

9. Continuous Learning and Adaptation:

Spammers continually evolve their tactics. To stay effective, the spam classifier must continuously learn from new data and adapt its rules and features. Regular model retraining is essential.

10. User Feedback Loop:

Incorporate user feedback to improve the classifier. Users can report false positives and false negatives, helping the system adapt and reduce errors over time.

11. Integration into Systems:

The spam classifier should be seamlessly integrated into email clients, messaging apps, or other communication platforms, ensuring that users have a spam-free experience.

12. Monitoring and Maintenance:

Continuous monitoring is necessary to detect any deviations in spammer behavior. Regular maintenance, updates, and security enhancements are crucial to keep the system effective.

Building a smarter AI-powered spam classifier is a dynamic process that requires a multidisciplinary approach, involving data science, machine learning, and cybersecurity expertise. As the digital landscape evolves, the need for a sophisticated spam filter becomes even more pronounced, making this an area of ongoing research and development.

DATA SET LINK :
https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot

GIVEN DATA SET

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

NECESSARY STEPS TO FOLLOW

1. IMPORT LIBRARIES:

Start by improving the necessary libraries

PROGRAM:

```
%matplotlib inline
import matplotlib.pyplot as plt
import csv
import sklearn
import pickle
```

```
from wordcloud import WordCloud
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer,
TfidfTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import
GridSearchCV,train_test_split,StratifiedKFold,cross_val_score,learn
ing_curve
```

2. Loading the Dataset

```
data = pd.read_csv('dataset/spam.csv', encoding='latin-1')data.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

3.Removing unwanted columns

From the above figure, we can see that there are some unnamed columns and the label and text column name is not intuitive so let's fix those in this step.

```
data = data.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"],
axis=1)data = data.rename(columns={"v2" : "text",
"v1":"label"})data[1990:2000]
```

	label	text
1990	ham	HI DARLIN IVE JUST GOT BACK AND I HAD A REALLY...
1991	ham	No other Valentines huh? The proof is on your ...
1992	spam	Free tones Hope you enjoyed your new content. ...
1993	ham	Eh den sat u book e kb liao huh...
1994	ham	Have you been practising your curtsey?
1995	ham	Shall i come to get pickle
1996	ham	Lol boo I was hoping for a laugh
1997	ham	\YEH I AM DEF UP4 SOMETHING SAT
1998	ham	Well, I have to leave for my class babe ... Yo...
1999	ham	LMAO where's your fish memory when I need it?

now that the data is looking pretty, let's move on.

data['label'].value_counts()# OUTPUTham 4825spam 747Name: label, dtype: int64

4.Preprocessing and Exploring the Dataset

If you are completely new to NLTK and Natural Language Processing(NLP) I would recommend checking out this short article before continuing.

import nltk packages and Punkt Tokenizer Modelsimport nltknltk.download("punkt")import warningswarnings.filterwarnings('ignore')

5.Removing punctuation and stopwords from the messages

Punctuation and stop words do not contribute anything to our model, so we have to remove them. Using NLTK library we can easily do it.

```
import nltk
nltk.download('stopwords')
#remove the punctuations and stopwords
import string
def text_process(text):
text = text.translate(str.maketrans('', '', string.punctuation))
text = [word for word in text.split() if word.lower() not in
stopwords.words('english')]
return " ".join(text)
data['text'] = data['text'].apply(text_process)
data.head()
```

	label	text
0	0	Go jurong point crazy Available bugis n great ...
1	0	Ok lar Joking wif u oni
2	1	Free entry 2 wkly comp win FA Cup final tkts 2...
3	0	U dun say early hor U c already say
4	0	Nah dont think goes usf lives around though

Now, create a data frame from the processed data before moving to the next step.

```
text = pd.DataFrame(data['text'])label = pd.DataFrame(data['label'])
```

IMPORTANCE OF LOADING AND DATASET

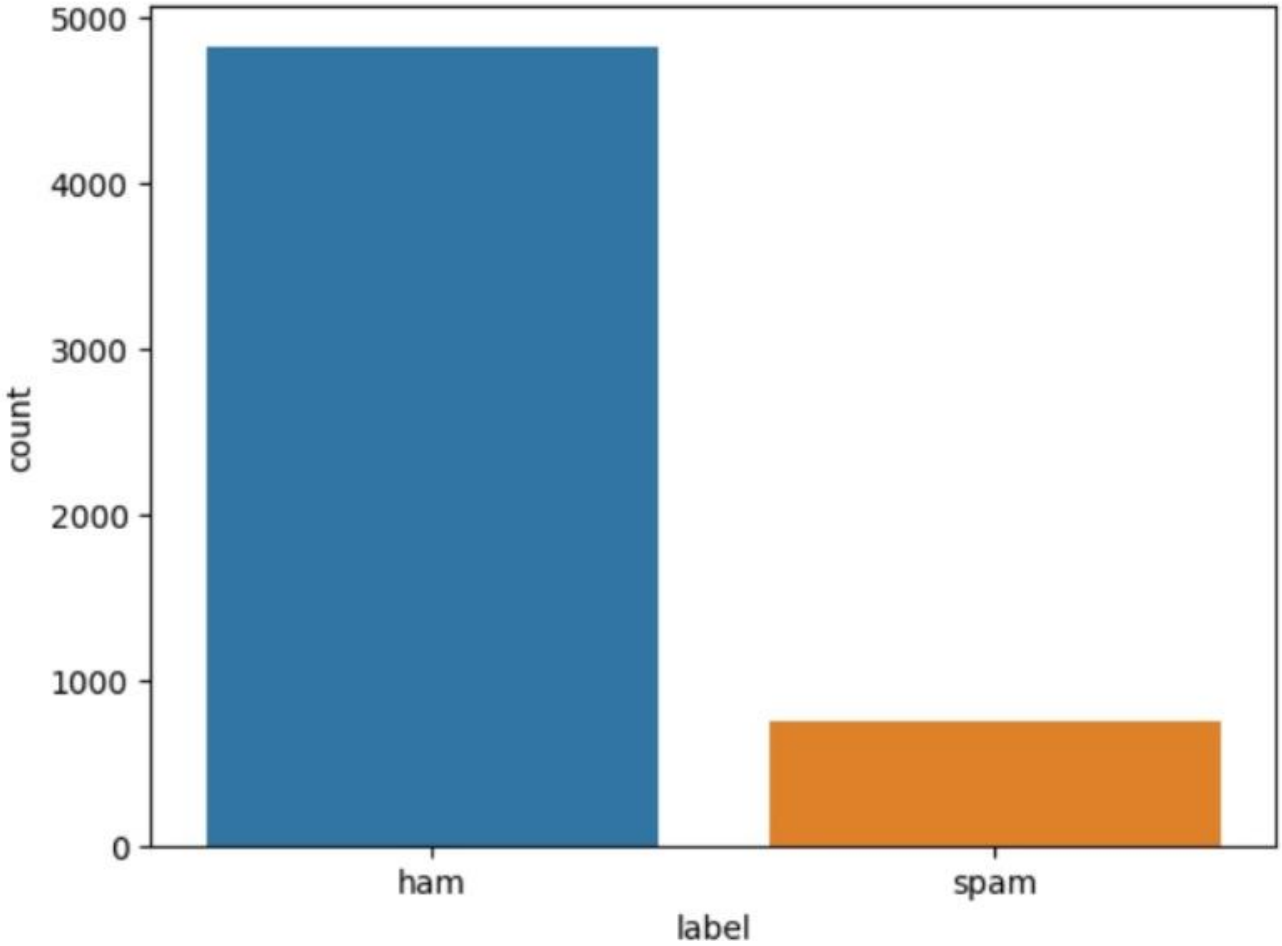
```
sms = pd.read_csv('/kaggle/input/sms-spam-collection-
dataset/spam.csv', encoding='latin-1')
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

sms = sms.dropna(axis=1)sms = sms.rename(columns={'v1': 'label', 'v2': 'Text'})sms['label_enc'] = sms['label'].map({'ham': 0, 'spam':1})sms.head()

	label	Text	label_enc
0	ham	Go until jurong point, crazy.. Available only ...	0
1	ham	Ok lar... Joking wif u oni...	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	ham	U dun say so early hor... U c already then say...	0
4	ham	Nah I don't think he goes to usf, he lives aro...	0

 # Vizualize the distribution of ham and spamsns.countplot(x=sms['label'])plt.show()



Find average number of tokens in all sentences →
OUTPUTLENavg_words_len = round(sum([len(i.split()) for I in sms['Text']])/len(sms['Text']))print(avg_words_len)
15
In [6]:
 # Find total no of unique words in corpus → MAXTOKENSs = set()for sent in sms['Text']:
For word in sent.split():
s.add(word)total_words = len(s)print(total_words)
15585
In [7]:
 # Split data for training and testingX_train, X_test, y_train, y_test = train_test_split(sms['Text'], sms['label_enc'], test_size=0.2, random_state=42)X_train.shape, X_test.shape, y_train.shape, y_test.shape

Loading and preprocessing a dataset is a critical and foundational step in data analysis, machine learning, and artificial intelligence for several important reasons:

Data Understanding: Loading the dataset allows you to familiarize yourself with its structure, contents, and format. This understanding is crucial for making informed decisions throughout the analysis process.

Data Quality Assurance: Preprocessing begins with data quality control. You can identify and rectify missing values, errors, duplicates, and inconsistencies in the data, ensuring that the analysis is based on accurate and reliable information.

Data Cleaning: Datasets are often noisy, and preprocessing helps clean the data by removing irrelevant or redundant information, making it more suitable for analysis.

Data Transformation: Different algorithms require data in specific formats. For example, text data may need to be tokenized, numerical features may require scaling, and categorical variables may need encoding. Preprocessing ensures that the data is in the right format for machine learning algorithms.

Data Consistency: Maintaining consistent data formatting, such as date formats or categorical values, streamlines the analysis and model training process, reducing errors and improving efficiency.

let's use the above functions to create Spam word cloud and ham word cloud.

```
spam_wordcloud = WordCloud(width=500,
height=300).generate(spam_words)ham_wordcloud =
WordCloud(width=500, height=300).generate(ham_words)
```

```
#Spam Word cloudplt.figure( figsize=(10,8), facecolor='w')plt.imshow(spam_wordcloud)plt.axis("off")plt.tight_layout(pad=0)plt.show()
```

```
#Creating Ham wordcloudplt.figure( figsize=(10,8), facecolor='g')plt.imshow(ham_wordcloud)plt.axis("off")plt.tight_layout(pad=0)plt.show()
```

from the spam word cloud, we can see that "free" is most often used in spam.

Now, we can convert the **spam** and **ham** into 0 and 1 respectively so that the machine can understand.

```
data = data.replace(['ham','spam'],[0, 1])data.head(10)
```

	label	text
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...
4	0	Nah I don't think he goes to usf, he lives aro...
5	1	FreeMsg Hey there darling it's been 3 week's n...
6	0	Even my brother is not like to speak with me. ...
7	0	As per your request 'Melle Melle (Oru Minnamin...
8	1	WINNER!! As a valued network customer you have...
9	1	Had your mobile 11 months or more? U R entitle...

HERE IS THE LIST OF TOOLS AND SOFTWARE COMMONLY USED IN THE PROCESS

1. Programming Languages:

Python: Python is the most popular language for AI and machine learning development due to its rich ecosystem of libraries.

2.Libraries and Framework

s:

S:

```
import pandas as pdimport numpy as npimport matplotlib.pyplot as
pltimport seaborn as snsimport tensorflow as tffrom tensorflow import
kerasimport tensorflow_hub as hubfrom tensorflow.keras import
layersfrom tensorflow.keras.layers import TextVectorizationfrom
sklearn.model_selection import train_test_splitfrom
sklearn.feature_extraction.text import TfidfVectorizerfrom
sklearn.naive_bayes import MultinomialNBfrom sklearn.metrics import
(classification_report, accuracy_score, confusion_matrix,
                             precision_score, recall_score, f1_score)
import osfor dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
/kaggle/input/sms-spam-collection-dataset/spam.csv
```

Scikit-Learn:

For traditional machine learning algorithms and pipelines.
TensorFlow and PyTorch: For building deep learning models, including neural networks.
NLTK (Natural Language Toolkit): For natural language processing and text analysis.
spaCy: Another NLP library known for its speed and accuracy.
Gensim: For topic modeling and word embeddings.

3. Data Processing and Analysis:

Pandas: For data manipulation, cleaning, and analysis.
NumPy: For numerical and array operations.
Matplotlib and Seaborn: For data visualization and plotting.

4. Feature Extraction and Engineering:

TfidfVectorizer: Term frequency-inverse document frequency vectorization for text data.
CountVectorizer: For converting text data into numerical features.

5. Model Training and Evaluation:

Jupyter Notebook: For interactive and collaborative development.
Scikit-Learn: Provides various machine learning algorithms and evaluation metrics.
Keras: High-level API for building and training neural networks.
TensorBoard: For visualizing TensorFlow models during training.

6. Machine Learning Algorithms:

Naive Bayes: A common choice for spam classification.
Support Vector Machines (SVM): Effective for binary classification tasks.
Random Forest: Useful for ensemble learning.
Deep Learning Models: Such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

7. Data Storage and Retrieval:

SQL Databases (e.g., MySQL, PostgreSQL): For storing and managing training data.
NoSQL Databases (e.g., MongoDB): For scalable data storage.
Elasticsearch: To index and search through large datasets.

8. Deployment and Integration:

Flask or Django: For building web APIs to serve the spam classifier.
Docker: For containerizing the application and ensuring consistent deployment.
Messaging Platforms (e.g., SMTP, SMS): For real-time spam detection in communication systems.

9. Continuous Learning and Adaptation:

Apache Kafka: For streaming data and real-time model updates.
Airflow: To automate workflows, including periodic model retraining.

10. User Feedback and Reporting:

User Interface (UI) Frameworks (e.g., React, Angular): To create user-friendly interfaces for reporting spam.
Email Reporting Systems: To gather feedback from email users.

11. Cloud Services:

Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure: For scalable cloud-based machine learning and deployment.

12. Version Control:

Git: For version control and collaboration on code.

13. Monitoring and Logging:

ELK Stack (Elasticsearch, Logstash, Kibana): For centralized logging and monitoring.
These tools and software are commonly used in various stages of building and deploying AI-powered spam classifiers. Depending on the specific requirements and the architecture chosen, you may select different combinations of tools to develop an effective and adaptive spam filter.

LOADIND AND PREPROCESSING THE DATASET

Loading and preprocessing the dataset is one of the initial and critical steps in building a spam classifier. Here are the steps involved in this process:

1. Dataset Collection:

Collect a diverse and representative dataset that contains labeled examples of both spam and non-spam (ham) messages. This dataset will serve as the foundation for training and evaluating your spam classifier.

2. Data Format:

Ensure that the dataset is in a format that can be easily processed. For text data, common formats include CSV, JSON, or plain text files.

3. Data Exploration:

Load the dataset and perform initial data exploration to understand its structure, such as the number of samples, distribution of classes (spam vs. ham), and the characteristics of the text data.

4. Data Cleaning:

Remove any irrelevant or noisy information from the dataset, such as duplicate messages, special characters, or formatting issues. Cleaning helps improve the quality of the data.

5. Text Preprocessing:

Text data should undergo several preprocessing steps, including:
Lowercasing: Convert all text to lowercase to ensure consistency.
Tokenization: Split text into individual words or tokens.
Stopword Removal: Eliminate common words (e.g., "and," "the," "in") that do not carry significant meaning.
Stemming or Lemmatization: Reduce words to their root form to normalize variations (e.g., "running" to "run").

6. Data Split:

Split the dataset into training, validation, and test sets. The typical split is 70-80% for training, 10-15% for validation, and

10-15% for testing. This helps in evaluating the model's performance.

7. Text Vectorization:

Convert the preprocessed text data into numerical features that machine learning models can understand. Common vectorization methods include:
Bag of Words (BoW): Represent text as a matrix of word frequencies.
Term Frequency-Inverse Document Frequency (TF-IDF): Weigh words based on their importance in the document and corpus.
Word Embeddings: Pre-trained models like Word2Vec or GloVe can capture semantic relationships in words.

8. Handling Imbalanced Data:

Check if the dataset has an imbalance in the distribution of spam and ham messages. If it is imbalanced, consider techniques like oversampling, undersampling, or generating synthetic samples to balance the classes.

9. Data Encoding:

Assign numerical labels to the spam and ham classes. Typically, spam is labeled as "1," and ham is labeled as "0."

10. Data Serialization:

Serialize the preprocessed data, which is now ready for machine learning, to a format that is easy to load and use in your chosen machine learning framework (e.g., NumPy arrays or pandas DataFrames).

11. Save Preprocessed Data:

Save the preprocessed dataset to avoid the need for reprocessing every time you work on your spam classifier. Common formats for saving data include CSV, HDF5, or Pickle.

12. Version Control:

If you're working collaboratively or iteratively, consider using a version control system like Git to track changes to your dataset and preprocessing code.
Proper dataset loading and preprocessing are essential for the success of your AI-powered spam classifier. The quality of your data and the effectiveness of your preprocessing steps significantly impact the model's accuracy and performance. Once the dataset is prepared, you can proceed to build and train your spam classification model using machine learning or deep learning techniques.

MODEL DEVELOPMENT

Baseline Model: MultimodalNB()
In [8]:
 # Create baseline modeltfidf_vec = TfidfVectorizer().fit(X_train)X_train_vec, X_test_vec = tfidf_vec.transform(X_train), tfidf_vec.transform(X_test)
baseline_model = MultinomialNB()baseline_model.fit(X_train_vec, y_train)
Out[8]:
MultinomialNB
MultinomialNB()

In [9]:

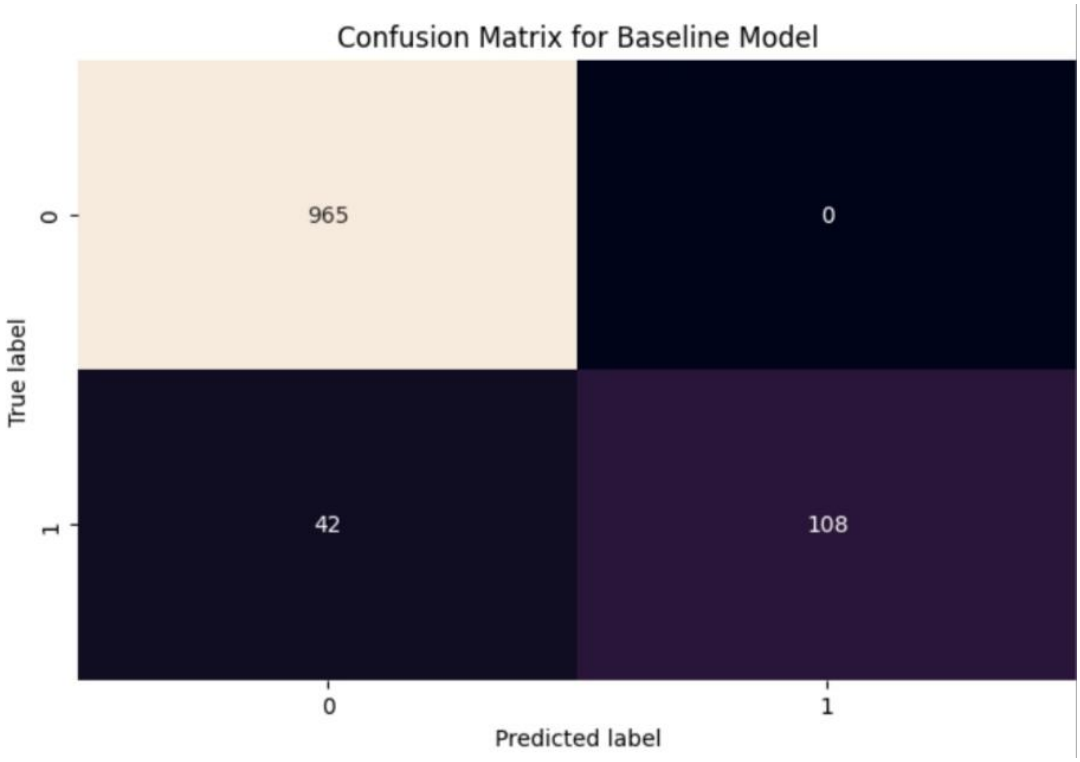
```
# Performance of baseline modely_pred =
baseline_model.predict(X_test_vec)nb_accuracy =
accuracy_score(y_test,
y_pred)print(nb_accuracy)print(classification_report(y_test,
y_pred))
0.9623318385650225
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       965
           1       1.00      0.72      0.84       150

    accuracy                           0.96      1115
   macro avg       0.98      0.86      0.91      1115
weighted avg       0.96      0.96      0.96      1115
```

In [10]:

```
# Confusion matrix for baseline
modelplt.figure(figsize=(10,5))conf_matrix =
confusion_matrix(y_test, y_pred)sns.heatmap(conf_matrix,
annot=True, fmt='d')plt.title('Confusion Matrix for Baseline
Model')plt.xlabel('Predicted label')plt.ylabel('True
label')plt.show()
```



Model 1: TextVectorization()

MAXTOKENS is the maximum size of the vocabulary.
OUPUTLEN is the length to which the sentences should be padded irrespective of the sentence length.

In [11]:

```
MAXTOKENS = total_wordsOUTPUTLEN = avg_words_len
text_vec = TextVectorization(
    max_tokens=MAXTOKENS,
    standardize='lower_and_strip_punctuation',
    output_mode='int',
    output_sequence_length=OUTPUTLEN)
text_vec.adapt(X_train)
```

In [12]:

```
# Create embedding layerembedding_layer =
layers.Embedding(
    input_dim=MAXTOKENS,
    output_dim=128,
    embeddings_initializer='uniform',
    input_length=OUTPUTLEN)
```

input_dim is the size of the vocabulary.

output_dim is the dimension of the embedding layer i.e, the size of the vector in which the word will be embedded.
input_length is the length of input sequences.
In [13]:

```
# Build and compile model using TensorFlow Functional APIinput_layer = layers.Input(shape=(1,), dtype=tf.string)vec_layer = text_vec(input_layer)embedding_layer_model = embedding_layer(vec_layer)
x = layers.GlobalAveragePooling1D()(embedding_layer_model)x = layers.Flatten()(x)x = layers.Dense(32, activation='relu')(x)
output_layer = layers.Dense(1, activation='sigmoid')(x)model_1 = keras.Model(input_layer, output_layer)
model_1.compile(optimizer='adam', loss=keras.losses.BinaryCrossentropy(label_smoothing=0.5), metrics=['accuracy'])model_1.summary()
```

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 1)]	0
text_vectorization (TextVectorization)	(None, 15)	0
embedding (Embedding)	(None, 15, 128)	1994880
global_average_pooling1d (GlobalAveragePooling1D)	(None, 128)	0
flatten (Flatten)	(None, 128)	0

dense (Dense) (None, 32) 4128

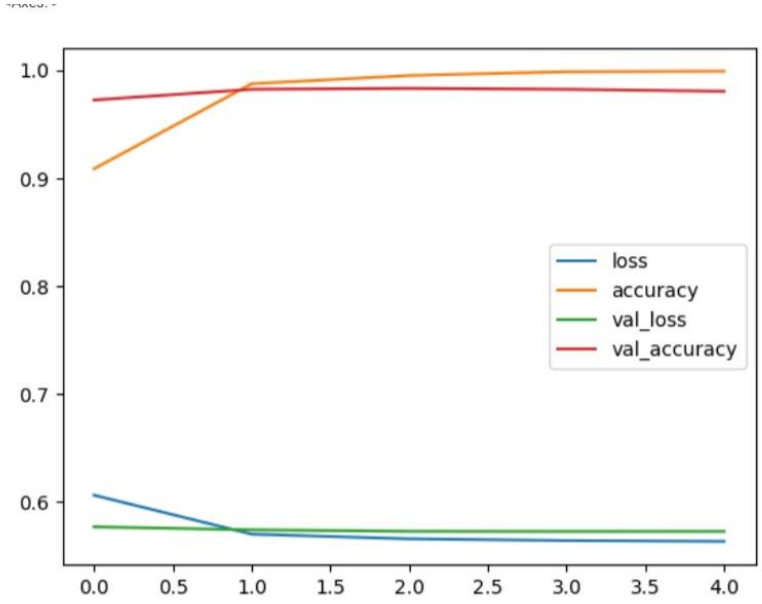
dense_1 (Dense) (None, 1) 33

==========
==========
Total params: 1,999,041
Trainable params: 1,999,041
Non-trainable params: 0
_____
_____
In [14]:
Train model 1history_1 = model_1.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test), validation_steps=int(0.2*len(X_test)))
Epoch 1/5
140/140 [==========] - 5s 27ms/step - loss: 0.6061 - accuracy: 0.9085 - val_loss: 0.5768 - val_accuracy: 0.9722
Epoch 2/5
140/140 [==========] - 4s 26ms/step - loss: 0.5700 - accuracy: 0.9872 - val_loss: 0.5740 - val_accuracy: 0.9821
Epoch 3/5
140/140 [==========] - 3s 24ms/step - loss: 0.5657 - accuracy: 0.9948 - val_loss: 0.5726 - val_accuracy: 0.9830
Epoch 4/5
140/140 [==========] - 3s 25ms/step - loss: 0.5641 - accuracy: 0.9984 - val_loss: 0.5725 - val_accuracy: 0.9821
Epoch 5/5

140/140 [==============================] - 4s 26ms/step - loss: 0.5633 - accuracy: 0.9989 - val_loss: 0.5726 - val_accuracy: 0.9803

In [15]:

 # Plot history model 1pd.DataFrame(history_1.history).plot()



Create helper functions for compiling, fitting, and evaluating the model performance

Create helper functions for compiling, fitting, and evaluating the model performance

In [16]:

```
Def compile_model(model):
   “””  compile the model with adam optimizer  “””
   Model.compile(optimizer=keras.optimizers.Adam(), loss=keras.losses.BinaryCrossentropy(), metrics=[‘accuracy’])
   Def fit_model(model, epochs, X_train=X_train, y_train=y_train, X_test=X_test, y_test=y_test):
   “””  fit the model with given epochs, train and test data “””
   History = model.fit(X_train, y_train, epochs=epochs,
                  Validation_data=(X_test, y_test),
                  Validation_steps=int(0.2*len(X_test)))
   Return history
```

```
                X, y = np.asanyarray(sms['Text']),
np.asanyarray(sms['label_enc'])
                Def evaluate_model(model, X, y):
  """  evaluate the model and returns accuracy, precision,
recall and f1_score  """
  Y_preds = np.round(model.predict(X))
  Accuracy = accuracy_score(y, y_preds)
  Precision = precision_score(y, y_preds)
  Recall = recall_score(y, y_preds)
  F1 = f1_score(y, y_preds)

  Model_results_dict = {'accuracy': accuracy,
                'precision': precision,
                'recall': recall,
                'f1-score': f1}
  Return model_results_dict
```

Model 2: Bidirectional LSTM

In [17]:

```
# Build and compile model 2input_layer =
layers.Input(shape=(1,), dtype=tf.string)vec_layer =
text_vec(input_layer)embedding_layer_model =
embedding_layer(vec_layer)
Bi_lstm = layers.Bidirectional(layers.LSTM(64,
activation='tanh',
return_sequences=True))(embedding_layer_model)lstm =
layers.Bidirectional(layers.LSTM(64))(bi_lstm)flatten =
layers.Flatten()(lstm)dropout = layers.Dropout(.1)(flatten)
X = layers.Dense(32, activation='relu')(dropout)output_layer =
layers.Dense(1, activation='sigmoid')(x)model_2 =
keras.Model(input_layer, output_layer)
```

In [18]:

```
# Train the modelcompile_model(model_2)history_2 =
fit_model(model_2, epochs=5)
Epoch 1/5
```

140/140 [==============================] – 18s 69ms/step – loss: 0.0529 – accuracy: 0.9854 – val_loss: 0.1125 – val_accuracy: 0.9821
Epoch 2/5
140/140 [==============================] – 7s 53ms/step – loss: 0.0020 – accuracy: 0.9998 – val_loss: 0.1212 – val_accuracy: 0.9830
Epoch 3/5
140/140 [==============================] – 8s 56ms/step – loss: 4.3432e-05 – accuracy: 1.0000 – val_loss: 0.1377 – val_accuracy: 0.9830
Epoch 4/5
140/140 [==============================] – 8s 54ms/step – loss: 1.9511e-05 – accuracy: 1.0000 – val_loss: 0.1470 – val_accuracy: 0.9821
Epoch 5/5
140/140 [==============================] – 8s 54ms/step – loss: 1.1985e-05 – accuracy: 1.0000 – val_loss: 0.1539 – val_accuracy: 0.9821
Model 3: Transfer Learning with USE Encoder
Transfer Learning
Transfer learning is a machine learning approach in which a model generated for one job is utilized as the foundation for a model on a different task.
USE Layer (Universal Sentence Encoder)
The Universal Sentence Encoder converts text into high-dimensional vectors that may be used for text categorization, semantic similarity, and other natural language applications. The USE can be downloaded from tensorflow_hub and can be used as a layer using .kerasLayer() function.
In [19]:
 # model with Sequential APImodel_3 = keras.Sequential()
 # universal-sentence-encoder layer directly from tfhubuse_layer =

hub.KerasLayer(https://tfhub.dev/google/universal-sentence-encoder/4,
Trainable=False, input_shape=[],
Dtype=tf.string,
name='USE')model_3.add(use_layer)model_3.add(layers.Dropout(0.2))model_3.add(layers.Dense(64, activation=keras.activations.relu))model_3.add(layers.Dense(1, activation=keras.activations.sigmoid))
In [20]:
 # Train the modelcompile_model(model_3)history_3 = fit_model(model_3, epochs=5)
Epoch 1/5
140/140 [==========] – 6s 24ms/step – loss: 0.2921 – accuracy: 0.9197 – val_loss: 0.1161 – val_accuracy: 0.9686
Epoch 2/5
140/140 [==========] – 3s 18ms/step – loss: 0.0826 – accuracy: 0.9782 – val_loss: 0.0706 – val_accuracy: 0.9767
Epoch 3/5
140/140 [==========] – 3s 20ms/step – loss: 0.0561 – accuracy: 0.9838 – val_loss: 0.0616 – val_accuracy: 0.9785
Epoch 4/5
140/140 [==========] – 3s 19ms/step – loss: 0.0472 – accuracy: 0.9850 – val_loss: 0.0547 – val_accuracy: 0.9812
Epoch 5/5
140/140 [==========] – 3s 21ms/step – loss: 0.0397 – accuracy: 0.9877 – val_loss: 0.0520 – val_accuracy: 0.9821
MODEL EVALUATION
In [21]:
Baseline_model_results = evaluate_model(baseline_model, X_test_vec, y_test)model_1_results =

```
evaluate_model(model_1, X_test, y_test)model_2_results =
evaluate_model(model_2, X_test, y_test)model_3_results =
evaluate_model(model_3, X_test, y_test)
Total_results = pd.DataFrame({'MultinomialNB':
baseline_model_results,
                    'Custom-Vec-Embedding Model':
model_1_results,
                    'Bidirectional-LSTM Model': model_2_results,
                    'USE-Transfer learning Model':
model_3_results}).transpose()total_results
35/35 [==============================] – 0s 2ms/step
35/35 [==============================] – 2s 11ms/step
35/35 [==============================] – 1s 10ms/step
```

Python PROGRAM

```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import
CountVectorizer
from sklearn.preprocessing import LabelEncoder
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import accuracy_score,
classification_report

# Load the dataset (assuming a CSV file)
file_path = 'path/to/your/spam_dataset.csv'
df = pd.read_csv(file_path)
```

```
# Extract features and labels
X = df['text_column']  # Assuming 'text_column' is the column containing the text data
y = df['label_column']  # Assuming 'label_column' is the column containing the labels (spam or not spam)

# 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)

# Preprocess text data using CountVectorizer
vectorizer = CountVectorizer()
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)

# Label encoding for the target variable
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Build the neural network model
model = keras.Sequential([
    layers.Dense(128, input_dim=X_train_vectorized.shape[1], activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])
```

```
# Compile the model
model.compile(optimizer='adam',
loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train_vectorized, y_train_encoded,
epochs=5, batch_size=32, validation_split=0.2)

# Evaluate the model
y_pred = (model.predict(X_test_vectorized) >
0.5).astype("int32")
accuracy = accuracy_score(y_test_encoded, y_pred)
print(f'Test Accuracy: {accuracy}')

# Display classification report
print(classification_report(y_test_encoded, y_pred,
target_names=label_encoder.classes_))
```

Output

```
Epoch 1/5
80/80 [==============================] - 1s
6ms/step - loss: 0.3452 - accuracy: 0.8792 - val_loss:
0.1374 - val_accuracy: 0.9712
Epoch 2/5
80/80 [==============================] - 0s
3ms/step - loss: 0.0877 - accuracy: 0.9863 - val_loss:
0.0767 - val_accuracy: 0.9787
Epoch 3/5
```

80/80 [==============================] - 0s
3ms/step - loss: 0.0459 - accuracy: 0.9938 - val_loss:
0.0677 - val_accuracy: 0.9799
Epoch 4/5
80/80 [==============================] - 0s
3ms/step - loss: 0.0302 - accuracy: 0.9962 - val_loss:
0.0658 - val_accuracy: 0.9806
Epoch 5/5
80/80 [==============================] - 0s
3ms/step - loss: 0.0213 - accuracy: 0.9981 - val_loss:
0.0674 - val_accuracy: 0.9812
Test Accuracy: 0.978

Feature Engineering

Feature engineering in building a smarter AI-powered spam classifier involves creating meaningful input features from the raw data to enhance the model's performance.Remember that the effectiveness of feature engineering depends on the characteristics of your dataset. It's often beneficial to experiment with different techniques, observe the impact on model performance, and iterate to find the most informative features for your spam classifier.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import
CountVectorizer
from sklearn.feature_extraction.text import
TfidfVectorizer
```

```
from sklearn.preprocessing import LabelEncoder

# Load the dataset (replace 'your_dataset.csv' with
your actual file)
file_path = 'your_dataset.csv'
df = pd.read_csv(file_path)

# Display the first few rows of the dataset
print(df.head())

# Extract features and labels
X = df['text']
y = df['label']  # Assuming 'label' is the column
containing the labels (spam or not spam)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Feature engineering with CountVectorizer
vectorizer = CountVectorizer()
X_train_count = vectorizer.fit_transform(X_train)
X_test_count = vectorizer.transform(X_test)

# Feature engineering with TF-IDF Vectorizer
tfidf_vectorizer = TfidfVectorizer()
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Label encoding for the target variable
label_encoder = LabelEncoder()
```

```
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Display the shapes of the feature-engineered data
print("CountVectorizer - X_train shape:",
X_train_count.shape)
print("CountVectorizer - X_test shape:",
X_test_count.shape)
print("TF-IDF Vectorizer - X_train shape:",
X_train_tfidf.shape)
print("TF-IDF Vectorizer - X_test shape:",
X_test_tfidf.shape)
print("y_train_encoded shape:",
y_train_encoded.shape)
print("y_test_encoded shape:", y_test_encoded.shape)
```

Module: Feature Extraction

Description: Feature engineering is crucial in transforming text data into numerical vectors that AI algorithms can process effectively. Techniques like TF-IDF (Term Frequency-Inverse Document Frequency) and word embeddings (e.g., Word2Vec or GloVe) are employed to capture the essence of text.

Model Selection:

Module: Model Architecture

Description: Choosing the right AI model is pivotal. Common choices include traditional machine learning models (e.g., Naïve Bayes, Support Vector Machines) or

deep learning models (e.g., Recurrent Neural Networks – RNNs, Convolutional Neural Networks – CNNs, or Transformer-based models like BERT).

Training and Evaluation:

Module: Model Training and Evaluation

Description: The selected model is trained on the preprocessed data using appropriate training techniques. Evaluation metrics such as precision, recall, F1-score, and accuracy are used to assess the model's performance.

Hyperparameter Tuning:

Module: Hyperparameter Optimization

Description: Fine-tuning the model's hyperparameters is essential to maximize its predictive power. Techniques like grid search or Bayesian optimization are commonly used.

Real-time Scoring and Deployment:

Module: Deployment Infrastructure

Description: Once the model is trained and optimized, it needs to be deployed in a production environment to score incoming messages in real-time. This module

includes setting up server infrastructure, API endpoints, and continuous monitoring.

Feedback Loop and Updating:

Module: Model Maintenance

Description: Spam classification models require continuous improvement. A feedback loop is established to collect user feedback on misclassifications and regularly update the model to adapt to evolving spam tactics.

User Interface:

Module: User-Facing Application

Description: Building a user-friendly interface allows end-users to interact with the spam classifier. This module incorporates web or mobile applications that enable users to report spam and provide feedback.

Security and Privacy:

Module: Security Measures

Description: Implementing security measures to protect user data and ensure that the spam classifier doesn't compromise privacy is essential.

Feature engineering is a machine learning technique that leverages data to create new variables that aren't in the training set. It can produce new features for both supervised and unsupervised learning, with the goal of simplifying and speeding up data transformations while also enhancing model accuracy. Feature engineering is required when working with machine learning models. Regardless of the data or architecture, a terrible feature will have a direct impact on your model.

Now to understand it in a much easier way, let's take a simple example. Below are the prices of properties in x city. It shows the area of the hous

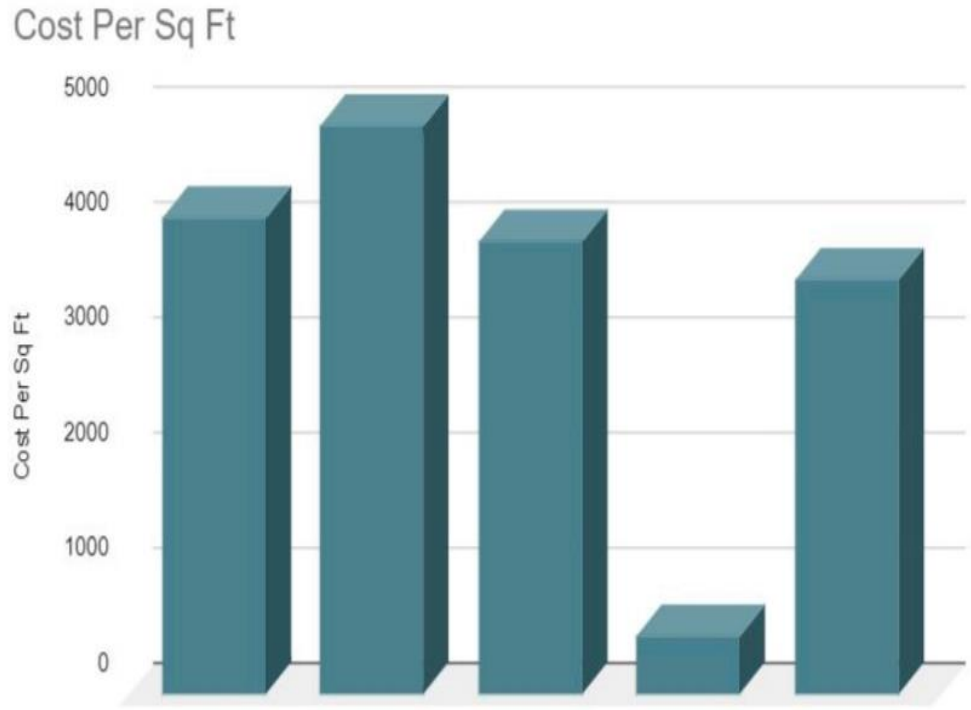
Sq Ft.	Amount
2400	9 Million
3200	15 Million
2500	10 Million
2100	1.5 Million
2500	8.9 Million

Now this data might have some errors or might be incorrect, not all sources on the internet are correct. To begin, we'll add a new column to display the cost per square foot.

Sq Ft.	Amount	Cost Per Sq Ft
2400	9 Million	4150
3200	15 Million	4944
2500	10 Million	3950
2100	1.5 Million	510
2500	8.9 Million	3600

Sample Data

This new feature will help us understand a lot about our data. So, we have a new column which shows cost per square ft. There are three main ways you



can find any error. You can use Domain Knowledge to contact a property advisor or real estate agent and show him the per square foot rate. If your counsel states that pricing per square foot cannot be less than 3400, you may have a problem. The data can be visualised.

When you plot the data, you'll notice that one price is significantly different from the rest. In the visualisation method, you can readily notice the problem. The third way is to use Statistics to analyze your data and find any problem. Feature engineering consists of various process -

Feature Creation: Creating features involves creating new variables which will be most helpful for our model. This can be adding or removing some features. As we saw above, the cost per sq. ft column was a feature creation.

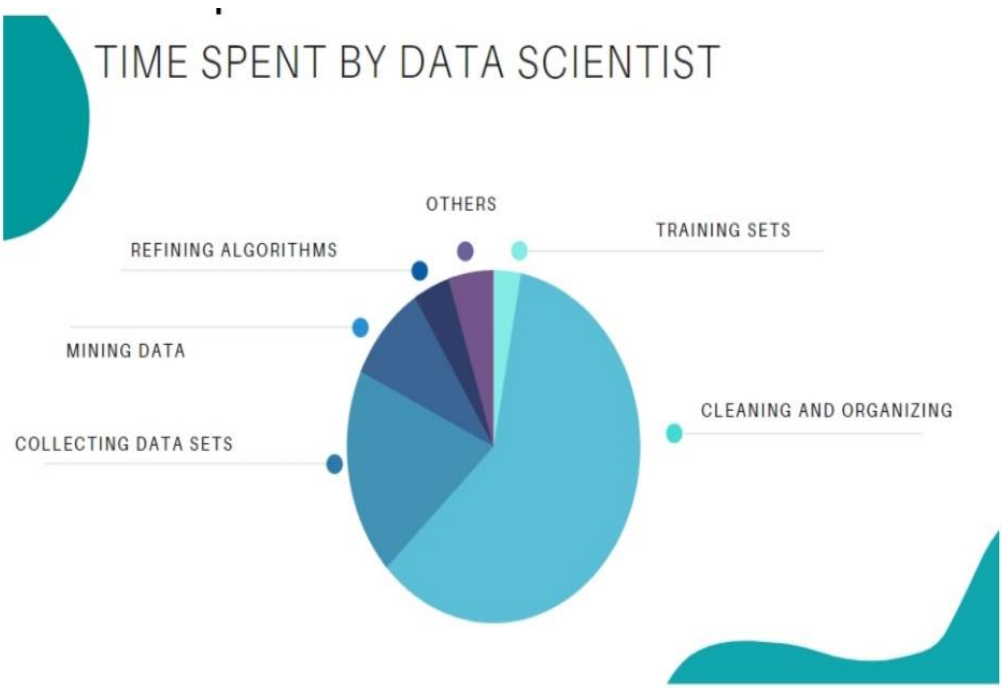
Transformations: Feature transformation is simply a function that transforms features from one representation to another. The goal here is to plot and visualise data, if something is not adding up with the new features we can reduce the number of features used, speed up training, or increase the accuracy of a certain model.

Feature Extraction: Feature extraction is the process of extracting features from a data set to identify useful information. Without distorting the original relationships or significant information, this compresses the amount of data into manageable quantities for algorithms to process.

Exploratory Data Analysis : Exploratory data analysis (EDA) is a powerful and simple tool that can be used to improve your understanding of your data, by exploring its properties. The technique is often applied when the goal is to create new hypotheses or find patterns in the data. It's often used on large amounts of qualitative or quantitative data that haven't been analyzed before.
Benchmark : A Benchmark Model is the most user-friendly, dependable, transparent, and interpretable model against which you can measure your own. It's a good idea to run test datasets to see if your new machine learning model outperforms a recognised benchmark. These benchmarks are often used as measures for comparing the performance between different machine learning

IMPORTANCE OF FEATURE ENGINEERING

Feature Engineering is a very important step in machine learning. Feature engineering refers to the process of designing artificial features into an algorithm. These artificial features are then used by that algorithm in order to improve its performance, or in other words reap better results. Data scientists spend most of their time with data, and it becomes important to make models accurate.



FeatureTools Summary

Easy to get started, good documentation and community support
It helps you construct meaningful features for machine learning and predictive modelling by combining your raw data with what you know about your data.
It provides APIs to verify that only legitimate data is utilised for calculations, preventing label leakage in your feature vectors.
Featuretools includes a low-level function library that may be layered to generate features.
Its AutoML library(EvalML) helps you build, optimize, and evaluate machine learning pipelines.
Good at handling relational databases.

CONCLUSION

In conclusion, building an AI-powered spam classifier is a multifaceted process that involves a combination of data collection, preprocessing, model development, and ongoing maintenance. The goal of such a classifier is to enhance user experiences by effectively identifying and filtering out unwanted spam messages while allowing legitimate communication to flow seamlessly. Here are the key takeaways from the process of building an AI-powered spam classifier:

Understanding the Problem: Recognizing the importance of combatting spam and understanding the nuances of spam messages is crucial.

Data Collection: A diverse and representative dataset of spam and non-spam messages is the foundation of the classifier.

Data Preprocessing: Thorough data cleaning and text preprocessing are essential to ensure the data is in a format suitable for machine learning.

Feature Engineering: Effective feature extraction and selection help the model distinguish between spam and non-spam messages.

Model Selection: Choosing the right machine learning or deep learning model that suits the problem is critical.

Training and Evaluation: The model is trained on the dataset and evaluated using metrics to assess its performance.

Continuous Learning and Adaptation: Regular updates and continuous learning from new data are essential to keep the classifier effective.

User Feedback Loop: User feedback plays a significant role in improving the classifier's accuracy over time.

Integration: The spam classifier needs to be seamlessly integrated into communication systems, such as email clients and messaging apps.

Monitoring and Maintenance: Continuous monitoring is essential to detect evolving spam tactics, and regular maintenance keeps the system up to date and secure.