Building a Smarter AI Powered spam Classifier

Phase-5 Project Documentation and Submission

Introduction:

This problem is to build an AI-powered spam classifier that can accurately distinguish between spam and non-spam messages in emails or text messages.The goal is to reduce the number of false Positives and false negatives while achieving a high level of accuracy.

Developing an effective Spam SMS Detection model involves training a machine learning algorithm to automatically identify and filter out unwanted, unsolicited text messages from legitimate ones.Also the model is created, trained and also tested with a sample output.

Problem Statement:

Develop an AI-powered spam classifier using natural language processing(NLP) and machine learning techniques to accurately distinguish between spam and non-spam messages in emails or text messages.

1. Continuous updates: Regularly update the AI model with new spamming techniques to stay ahead of spammers.

2. Robust training data: Gather a diverse dataset of spam and non-spam messages to improve the accuracy of the classifier.

3. Advanced algorithms: Implement sophisticated algorithms that can adapt and learn from new spam patterns.

4. User feedback: Encourage users to provide feedback on misclassified messages to fine-tune the classifier.

5 .Collaborative efforts: Share insights and collaborate with other organizations and experts to collectively combat spam .

Design thinking process:

- ✧ In the data collection phase, we gather a large dataset of emails or messages labeled as spam or not spam. Then, in the preprocessing phase, we clean and prepare the data for analysis.
- ✧ Next, in the feature extraction phase, we extract relevant information from the messages, like the presence of certain words or patterns. This helps the model understand what distinguishes spam from non-spam.

✧ After that, in the model training phase, we use machine learning algorithms to train the AI model to recognize patterns and make predictions. The model is trained on the labeled dataset we collected earlier.

✧ Finally, in the evaluation phase, we test the performance of the model using a separate set of data. This helps us assess how well the model can classify spam.

✧ The phases of development for an AI spam classifier typically involve data collection, preprocessing, feature extraction, model training, and evaluation.

Phases of Development:

1. Data Collection
2. Preprocessing phase
3. Understanding the model or feature extraction
4. Model Training
5. Model Evaluation

Dataset Used:

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

Program:

Data Preprocessing steps:

1. Importing and loading dataset

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import os
import string
import keras
import nltk
import random
import plotly.express as px
import plotly.figure_factory as ff
import spacy
```

```
from plotly import graph_objs as go
from PIL import Image
from wordcloud import WordCloud , STOPWORDS , ImageColorGenerator
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from tqdm import tqdm
from collections import Counter , defaultdict
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
from keras.callbacks import ModelCheckpoint  , ReduceLROnPlateau
from keras.optimizers import Adam
from keras.models import Sequential
from keras.initializers import Constant
from keras.layers import Dense , LSTM ,Embedding , BatchNormalization ,Dropout ,Bidirectional , Flatten ,GlobalMaxPool1D
from sklearn.metrics import confusion_matrix , classification_report , accuracy_score ,f1_score
```

2. Loading the Data

```
data_path = '/kaggle/input/sms-spam-collection-dataset/spam.csv'

data =  pd.read_csv(data_path , encoding = 'latin')
data = data.drop(['Unnamed: 2' ,'Unnamed: 3' , 'Unnamed: 4'] , axis=1)
data.columns = ['Target', 'Message']
data.reset_index()
data.head()
```

[2]:

	Target	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...

3. EDA [Exploratory Data Analysis]

```
# let's get length of each message

data['message_length'] = data['Message'].apply(lambda x: len(x.split(" ")))
data.head()
```

[3]:

	Target	Message	message_length
0	ham	Go until jurong point, crazy.. Available only ...	20
1	ham	Ok lar... Joking wif u oni...	6
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	28
3	ham	U dun say so early hor... U c already then say...	11
4	ham	Nah I don't think he goes to usf, he lives aro...	13

3.1 Visualizing Imbalanced Data

```
data['Target'].value_counts()
```

```
[4]: Target
     ham     4825
     spam     747
     Name: count, dtype: int64
```

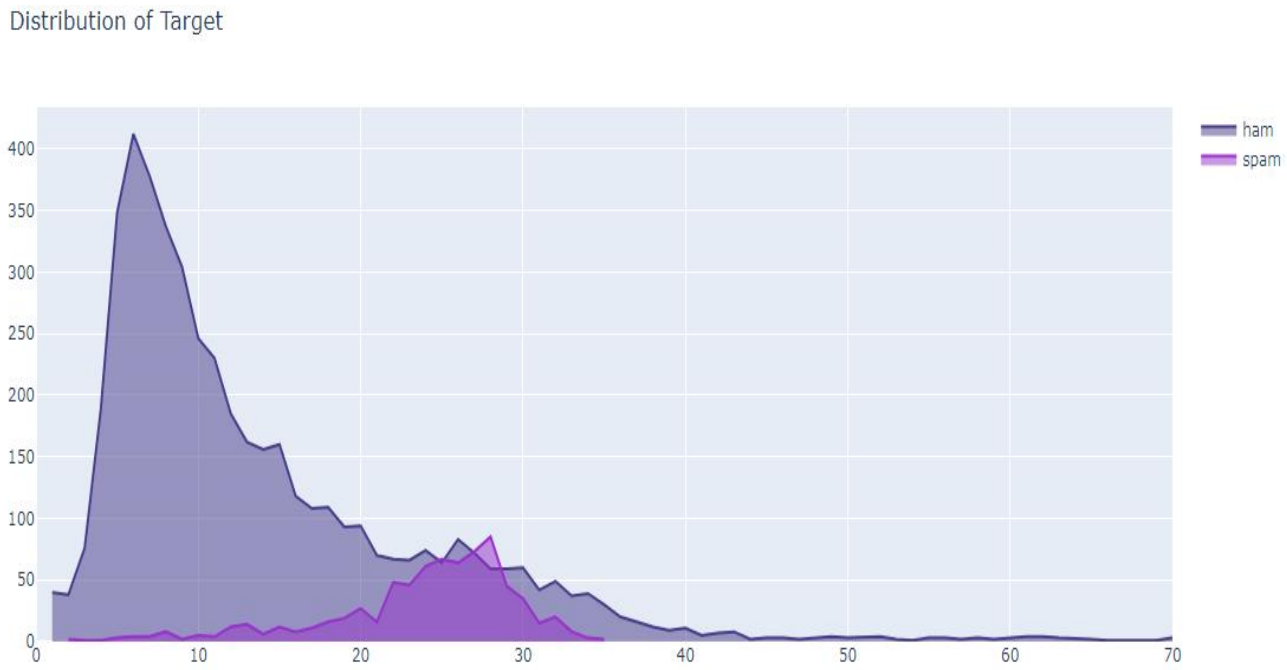
```
Ham_len = data[data['Target']=='ham']['message_length'].value_counts().sort_index()
Spam_len= data[data['Target']=='spam']['message_length'].value_counts().sort_index()


fig = go.Figure()
fig.add_trace(go.Scatter(
    x = Ham_len.index ,
    y = Ham_len.values ,
    name= 'ham' ,
    fill= 'tozeroy',
    marker_color = 'darkslateblue',

    ))
fig.add_trace(go.Scatter(
    x = Spam_len.index ,
    y = Spam_len.values ,
    name = 'spam' ,
    fill = 'tozeroy',
    marker_color = 'darkorchid' ,
    ))
```

```
fig.update_layout( title = 'Distribution of Target')
fig.update_xaxes(range =[0,70])
fig.show()
```

The below diagram shows the amount ham and spam messages in the given dataset



4. Preprocessing the Dataset

```
stop_words = stopwords.words('english') + ['u', 'im', 'c']
stemmer = nltk.SnowballStemmer('english')


def clean_text(text):
    '''Do lowercase, remove text in square brackets,links, punctuation
    and words containing numbers.'''
    text = str(text).lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text
```

```
def preprocessing(text):

    cleaned_text = clean_text(text)
    # remove stopwords
    cleaned_text  = ' '.join(word for word in cleaned_text.split(' ') if word not in stop_words)
    # do stem method
    cleaned_text  = ' '.join(stemmer.stem(word) for word in cleaned_text.split(' '))
    return cleaned_text
```

```
data['Cleaned_Message'] = data['Message'].apply(preprocessing)
# let's show new length after process

data['New_length'] = data['Cleaned_Message'].apply(lambda x: len(x.split(' ')))
data.head()
```

[8]:

	Target	Message	message_length	Cleaned_Message	New_length
0	ham	Go until jurong point, crazy.. Available only ...	20	go jurong point crazi avail bugi n great world...	16
1	ham	Ok lar... Joking wif u oni...	6	ok lar joke wif oni	5
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	28	free entri wkli comp win fa cup final tkts m...	23
3	ham	U dun say so early hor... U c already then say...	11	dun say earli hor alreadi say	6
4	ham	Nah I don't think he goes to usf, he lives aro...	13	nah dont think goe usf live around though	8

5. Tokens Visualisation

```
plt.figure(figsize = (16,5))
plt.title('Top Words For Ham Message')

Word_ham = WordCloud(
    background_color=None, mode="RGBA" ,
    width=800,
    height=300,
    )

Word_ham.generate(' '.join(text for text in  data.loc[ data['Target']=='ham' , 'Cleaned_Message']))
plt.imshow(Word_ham, interpolation='bilinear')
plt.axis('off')
plt.show()


plt.figure(figsize = (16,5))
plt.title('Top Words For Spam Message')
```

```
plt.figure(figsize = (16,5))
plt.title('Top Words For Spam Message')

Word_spam = WordCloud(
    background_color=None, mode="RGBA" ,
    width=800,
    height=300,
    )

Word_spam.generate(' '.join(text for text in  data.loc[ data['Target']=='spam' , 'Cleaned_Message']))
plt.imshow(Word_spam, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Top Words For Ham Message

Top Words For Spam Message

6.Building the Model and Predicting the Data

Machine learning Algorithm : **BERT**[**B**idirectional **E**ncoder **R**epresentations from **T**ransformers]

6.1 BERT:

Encoder-Only Models:

Function: Encoder-only models are designed to process input sequences and encode them into a fixed-length representation, often referred to as a context vector or latent representation. They do not have a decoding mechanism.

Usage Example: BERT is a popular encoder-only model. It's pre-trained on a massive corpus of text and used for various downstream NLP tasks like text classification, named entity recognition, and question answering. Decoder-Only Models.

```
import tensorflow as tf
from tensorflow import keras
from keras.layers import Dense  , Input
from keras.optimizers import Adam
from keras.models import Model
from keras.callbacks import ModelCheckpoint

import transformers
from transformers import BertTokenizer , TFBertModel
```

```
tokenizer = BertTokenizer.from_pretrained('bert-large-uncased')
bert_model= TFBertModel.from_pretrained('bert-large-uncased')
```

let's create a function to encode our input text to extact 'input_ids', and 'attention_maske'

```
def encode_(data , max_len):
    ids_list = []
    mask_list= []

    for text in data:
        text_encode = tokenizer.encode_plus(

            text ,
            max_length = max_len ,
            add_special_tokens=True ,
            truncation=True,
            padding='max_length' ,
            return_attention_mask=True
        )
        ids_list.append(text_encode['input_ids'])
        mask_list.append(text_encode['attention_mask'])
    return np.array(ids_list) , np.array(mask_list)
```

```
train_input_ids , train_attention_mask = encode_(train_data ,60)
```

6.2Create BERT Model

```
def create_bert_model(bert_model):
    input_ids     = Input(shape=(60,) , dtype='int32')
    attention_mask= Input(shape=(60,) , dtype='int32')
    ### layers
    out = bert_model(input_ids ,attention_mask)[1]
    out = Dense(32 , activation='relu')(out)
    out = Dropout(0.2)(out)
    out = Dense(1 , activation ='sigmoid')(out)
    ### initiate the model
    model = Model(inputs = [input_ids , attention_mask] , outputs = out)

    model.compile(Adam(learning_rate=1e-5) , loss =  'binary_crossentropy'
, metrics = ['accuracy'])

    return model
Bert_model = create_bert_model(bert_model)
Bert_model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 60)]         0           []

 input_2 (InputLayer)           [(None, 60)]         0           []

 tf_bert_model (TFBertModel)    TFBaseModelOutputWi  335141888   ['input_1[0][0]',
                                thPoolingAndCrossAt               'input_2[0][0]']
                                tentions(last_hidde
                                n_state=(None, 60,
                                1024),
                                 pooler_output=(Non
                                e, 1024),
                                 past_key_values=No
                                ne, hidden_states=N
                                one, attentions=Non
                                e, cross_attentions
                                =None)

 dense (Dense)                  (None, 32)           32800       ['tf_bert_model[0][1]']

 dropout_73 (Dropout)           (None, 32)           0           ['dense[0][0]']

 dense_1 (Dense)                (None, 1)            33          ['dropout_73[0][0]']

==================================================================================================
Total params: 335,174,721
Trainable params: 335,174,721
Non-trainable params: 0
__________________________________________________________________________________________________
```

6.3Training the Model

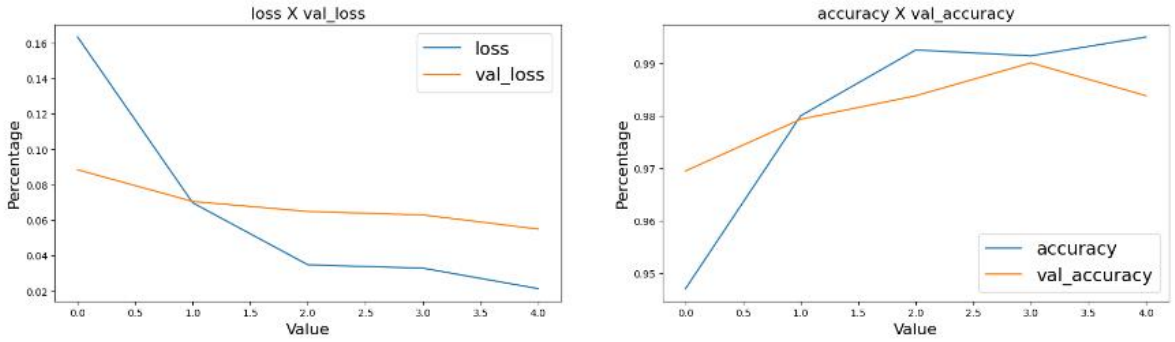
```
history = Bert_model.fit(
    [train_input_ids , train_attention_mask] ,
    y_label ,
    validation_split = 0.2,
    epochs = 5 ,
    batch_size=16

)
```

```
Epoch 1/5
279/279 [==============================] - 161s 468ms/step - loss: 0.1634 - accuracy: 0.9470 - val
_loss: 0.0884 - val_accuracy: 0.9695
Epoch 2/5
279/279 [==============================] - 135s 485ms/step - loss: 0.0698 - accuracy: 0.9800 - val
_loss: 0.0706 - val_accuracy: 0.9794
Epoch 3/5
279/279 [==============================] - 135s 485ms/step - loss: 0.0348 - accuracy: 0.9926 - val
_loss: 0.0649 - val_accuracy: 0.9839
Epoch 4/5
279/279 [==============================] - 135s 485ms/step - loss: 0.0329 - accuracy: 0.9915 - val
_loss: 0.0629 - val_accuracy: 0.9901
Epoch 5/5
279/279 [==============================] - 135s 485ms/step - loss: 0.0213 - accuracy: 0.9951 - val
_loss: 0.0550 - val_accuracy: 0.9839
```

```
def plot_history(history, arr):
    fig, ax = plt.subplots(1, 2, figsize=(20, 5))
    for index in range(2):
        ax[index].plot(history.history[arr[index][0]])
        ax[index].plot(history.history[arr[index][1]])
        ax[index].legend([arr[index][0], arr[index][1]],fontsize=18)
        ax[index].set_xlabel('Value',fontsize=16)
        ax[index].set_ylabel('Percentage',fontsize=16)
        ax[index].set_title(arr[index][0] + ' X ' + arr[index][1],fontsize=16)
```

```
plot_history(history , [['loss', 'val_loss'],['accuracy', 'val_accuracy'] ])
```



6.4Test the Model with Zero shot inference

```
preamble = "Classify the following SMS message as either 'ham' or 'spam':"
end_prompt    = 'Answer:\n'

template =  Template('$preamble\n\n$prompt\n\n$end_prompt')



def create_prompt(data , index):
    prompt = data.loc[index , 'Message']

    input_text = template.substitute(preamble=preamble , prompt = prompt ,end_prompt=end_prompt)
    return input_text
```

```
print(create_prompt(data , 0))
```

```
print(create_prompt(data , 0))
```

```
Classify the following SMS message as either 'ham' or 'spam':

Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got
amore wat...

Answer:
```

```
import random
index = random.randint(0,200)

zero_shot_message = create_prompt(data , index)
zero_shot_class   = data.loc[index , 'Target']


inputs  = tokenizer(zero_shot_message , return_tensors='pt')
generate= Flan_model.generate(inputs['input_ids'] )[0]
model_answer = tokenizer.decode(generate , skip_special_tokens=True)


line_dash = '-'.join('' for _ in range(100))
print(line_dash)
print(line_dash)
print(f'Prompt:\n{zero_shot_message}')
print(line_dash)
```

```
print(f'Acual Answer:\n{zero_shot_class}')
print(line_dash)
print(f'Model Answer:\n{model_answer}')
```

```
------------------------------------------------------------------------------------------------------------
-
------------------------------------------------------------------------------------------------------------
-
Prompt:
Classify the following SMS message as either 'ham' or 'spam':

FreeMsg Why haven't you replied to my text? I'm Randy, sexy, female and live local. Luv to hear fr
om u. Netcollex Ltd 08700621170150p per msg reply Stop to end

    Answer:


    ------------------------------------------------------------------------------------------------------------
    -
    Acual Answer:
    spam
    ------------------------------------------------------------------------------------------------------------
    -
    Model Answer:
    spam
```

Thus the project is build , trained and also it is tested with some sample input.

Conclusion:

The project for building a Smarter AI Powered Spam Classifier is done by preprocessing the dataset ,model training ,building and by testing the model with a sample output .