TEAM SPARROW

Building a Smarter AI powered spam classifier

OVERVIEW:

In this project, we will be covering a simple approach to email classification(spam or not spam) using BERT Steps are:

- We will load our data - mainly sentences and labels-span or not spam
- Load these in bert to generate an contextualized embedding vector of length 768
-
  - We will first apply preprocessing using the preprocessor object , refer the documentation
-
  - We will pass this preprocessed text to our model to generate the contexutailized embedding vector
- Finally pass this embedding vector to single neuron in output to do binary classificaton
- For maximizing performance we will be balancing our dataset and use a dropout layer to regularize the model and prevent overfitting

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

Loading Dependencies

Includes

- Tensorflow_hub : Place where all tenseorflow pretrained models are stored.
- Pandas : For data loading, manipulation and wrangling.
- Tensorflow_text : Allows addditional NLP text processing capablities outside scope of tensorflow
- Skelarn : For doing data evaluation and splitting
- Matplotlib : For visualization

```
# installing tensorflow_text
!pip install tensorflow-text
Successfully installed tensorflow-text-2.14.0
import tensorflow_hub as hub
import pandas as pd
import tensorflow_text as text
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import numpy as np
```

Loading Data

- Read Data
- Display data USING PANDAS

```
# load data
df = pd.read_csv('/content/drive/MyDrive/CLG _VK_18_/spam_data.csv')

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 Analysis

- Check the description by grouping by category :
- no of data points for each category - count
- no of unique values in each category - unique

```
# check count and unique and top values and their frequency
df['Category'].value_counts()
ham     4825
spam     747
Name: Category, dtype: int64
```

Clearly dataset is imbalanced - not so much but still it can affect our model. Need to use some type of regulariztion like downsampling dataset for mazority class

Downsampling Dataset

Includes:

- Check percentage of unbalances.
- Creating 2 new dataframes out of existing one.
- Taking any random minority no of samples - `(747)` for majority class `(4825)`.
- Creating a balanced dataset by concating 2 new data frames.

```
# check percentange of data - states how much data needs to be
balanced
str(round(747/4825,2))+'%'
```

```
{"type":"string"}
```

```
# creating 2 new dataframe as df_ham , df_spam

df_spam = df[df['Category']=='spam']
print("Spam Dataset Shape:", df_spam.shape)

df_ham = df[df['Category']=='ham']
print("Ham Dataset Shape:", df_ham.shape)
```

```
Spam Dataset Shape: (747, 2)
Ham Dataset Shape: (4825, 2)
```

```
# downsampling ham dataset - take only random 747 example
# will use df_spam.shape[0] - 747

df_ham_downsampled = df_ham.sample(df_spam.shape[0])
df_ham_downsampled.shape
```

```
(747, 2)
```

```
# concating both dataset - df_spam and df_ham_balanced to create
df_balanced dataset
df_balanced = pd.concat([df_spam , df_ham_downsampled])
df_balanced.head()
```

```
   Category                                            Message
2      spam  Free entry in 2 a wkly comp to win FA Cup fina...
5      spam  FreeMsg Hey there darling it's been 3 week's n...
8      spam  WINNER!! As a valued network customer you have...
9      spam  Had your mobile 11 months or more? U R entitle...
11     spam  SIX chances to win CASH! From 100 to 20,000 po...
```

```
df_balanced['Category'].value_counts()
```

```
spam    747
ham     747
Name: Category, dtype: int64
```

```
df_balanced.sample(10)
```

```
     Category                                            Message
5041     spam  Natalie (20/F) is inviting you to be her frien...
3713      ham                                 Wat u doing there?
1172     spam  Got what it takes 2 take part in the WRC Rally...
1217     spam  You have 1 new voicemail. Please call 08719181...
```

```
403        ham                  The hair cream has not been shipped.
4863      spam  **FREE MESSAGE**Thanks for using the Auction S...
2503       ham  Ola would get back to you maybe not today but ...
5267       ham        Anything lar then ü not going home 4 dinner?
2719      spam  18 days to Euro2004 kickoff! U will be kept in...
3678       ham         Great! So what attracts you to the brothas?
```

Data Prepration

1. Create Numerical Repersentation Of Category - One hot encoding

- Create a new column
- Use `df[col].apply(lambda function)`
- Lambda Function - if spam return 1, else return 0 (for ham) - ternary operators : [`lambda x : value expression else value`]

```
# creating numerical repersentation of category - one hot encoding
df_balanced['spam'] = df_balanced['Category'].apply(lambda x:1 if
x=='spam' else 0)

# displaying data - spam -1 , ham-0
df_balanced.sample(4)
     Category                                          Message  spam
928       ham                    K:)i will give my kvb acc details:)     0
435       ham  You available now? I'm like right around hills...     0
2760      ham  Can meh? Thgt some will clash... Really ah, i ...     0
4682      ham                             Are you staying in town ?     0
```

1. Do train-test split

- split dataset into 80-20 ratio with 80% train and remaing as test
- for eveness of data we will use `stratify` agrument which ensures same ratio of both

category is loaded for each case, even if one categoy has more training samples - prevents overfitting Store our data in:

- `X_train, y_train` - traininge set(training_data and labels respectively)
- `X_test,, y_test` - testing set(testing_data and labels)

```
# loading train test split
from sklearn.model_selection import train_test_split

X_train, X_test , y_train, y_test =
train_test_split(df_balanced['Message'],          df_balanced['spam'],
stratify = df_balanced['spam'])

# check for startification y_train.value_counts()
```

```
1    560
0    560
Name: spam, dtype: int64
560/560
1.0
y_test.value_counts()
1    187
0    187
Name: spam, dtype: int64
187/187

1.0
```
-> Almost similar, means data is downsampled now

Model Creation

Our Model is BERT , which will do 2 thing:

- Preporcess our training data that will be feeded - includes **adding additional token CLF , PAD and SEP** to genrate `input_mask`, `input_type_ids`, `input_word_ids(token given to each word in  sentences)`
- Note: no of words in sentence - 128/ max length of sentence can be 128

Downloading BERT

Model specification :

- Layers - 12
- Hidden layers - 768 - embedding size
- Attention - 12 Name - Bert Small --- This model has 2 parts:
- Bert_preprocessor - preprocess the text to be BERT ready
- ## Bert_encoder - do the actual encoding

  Steps:

  Preprocessor
- create a keras hub layer from the preprocessing url

Encoder

- create a keras hub layer from the encoder/ model url

Awesome functionality provided by Tf hub API

```
                                          +
```

Creating our own model using functional model api- link old layers to new layers rather than building it(in a sequential way) and allows sharing of layers too Info:

- Text the embedding as input - text_input
- Create a Sinlge output dense layer • Add dropout to reduce overfitting

```
# downloading preprocessing files and model
bert_preprocessor =
hub.KerasLayer('https://tfhub.dev/tensorflow/bert_en_uncased_preproces
s/3') bert_encoder =
hub.KerasLayer('https://tfhub.dev/tensorflow/bert_en_uncased_L-
12_H768_A-12/4')
```

Process And Encode Data

Use functional API to process and encode data in the layers itself

- Create a input layers with shape() , type - tf.string, and layer name - text - `TEXT_INPUT`
- Pass TEXT_INPUT into bert_prerocessor - `PREPROCESSED TEXT[*]`
- Pass the above[*] to encoder - `EMBEED`
- pass pooled_outputs of EMBEED to dropout layer - `DROPOUT`
- create a dense layer with activation as `sigmoid` `OUTPUTS`
- Create out MODEL (inputs - text_input, outputs - dropout)

```
import tensorflow as tf

text_input = tf.keras.layers.Input(shape = (), dtype = tf.string, name
= 'Inputs')
preprocessed_text = bert_preprocessor(text_input)
embeed = bert_encoder(preprocessed_text)
dropout = tf.keras.layers.Dropout(0.1, name = 'Dropout')
(embeed['pooled_output'])
outputs = tf.keras.layers.Dense(1, activation = 'sigmoid', name =
'Dense')(dropout)
```

```
# creating final model
model = tf.keras.Model(inputs = [text_input], outputs = [outputs])

# check summary of model model.summary()
Model: "model_2"

__________________________________________________________________________________
__________________________________
 Layer (type)                  Output Shape                     Param
Connected t
==================================================================================
==================================
 Inputs (InputLayer)           [(None,)]                        0           []

 keras_layer_2 (KerasLayer)    {'input_type_ids': (None,        0
['Inputs[0][0]'
                               128),
                                                 'input_mask': (None, 128)
                                                , 'input_word_ids': (None,
                                                                     128)}

 keras_layer_3 (KerasLayer)    {'encoder_outputs': [(None       1094822
['keras_layer_2[0][0]',
                               , 128, 768),                     41
'keras_layer_2[0][1]',
                                (None, 128, 768),
'keras_layer_2[0][2]']
                                            (None, 128, 768),
                            (None, 128, 768),
                                            (None, 128, 768),
                            (None, 128, 768),
                                            (None, 128, 768),
                            (None, 128, 768),
                                            (None, 128, 768),
                            (None, 128, 768),
```

```
                                    (None, 128, 768),

                                    (None, 128, 768)],

'default': (None, 768),

                                    'sequence_output': (None,

128, 768),

                                    'pooled_output': (None, 7

68)}


 Dropout (Dropout)              (None, 768)                  0
['keras_layer_3[0][13]'

 Dense (Dense)                  (None, 1)                    769
['Dropout[0][0]'

======================================================================
============================

Total params: 109483010 (417.64 MB)
Trainable params: 769 (3.00 KB)

Non-trainable params: 109482241 (417.64 MB)

______________________________________________________________________
____________________________
```

Compiling model

- Optimizer - ADAM
- Loss - binary_crossentropy
- metrics - accuracy , precesion and recall

```
Metrics = [tf.keras.metrics.BinaryAccuracy(name = 'accuracy'),
tf.keras.metrics.Precision(name = 'precision'),
tf.keras.metrics.Recall(name = 'recall')
           ]

model.compile(optimizer ='adam',
               loss = 'binary_crossentropy',
metrics = Metrics)

#@title Optional
# optional - defining tensorflow callbacks
import tensorflow as tf import datetime
```

```
%load_ext tensorboard

!rm -rf ./logs/
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M
%S")
tensorboard_callback=tf.keras.callbacks.TensorBoard(log_dir=log_dir,
histogram_freq=1)
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

Training Model

- Recomended to use `GPU` - providing so many training data
- We traing our model on training set
- For 10 epochs only - so model don't overfit - given enough training data

```
%tensorboard --logdir logs/fit

Reusing TensorBoard on port 6006 (pid 3229), started 1:03:18 ago. (Use
'!kill 3229' to kill it.)

<IPython.core.display.Javascript object>

history = model.fit(X_train, y_train, epochs = 10 , callbacks =
[tensorboard_callback])

Epoch 1/10
35/35 [==============================] - 48s 1s/step - loss: 0.6358 -
accuracy: 0.6500 - precision: 0.6567 - recall: 0.6286
Epoch 2/10
35/35 [==============================] - 38s 1s/step - loss: 0.4922 -
accuracy: 0.8286 - precision: 0.8345 - recall: 0.8196
Epoch 3/10
35/35 [==============================] - 37s 1s/step - loss: 0.4205 -
accuracy: 0.8607 - precision: 0.8435 - recall: 0.8857
Epoch 4/10
35/35 [==============================] - 38s 1s/step - loss: 0.3736 -
accuracy: 0.8768 - precision: 0.8728 - recall: 0.8821
Epoch 5/10
35/35 [==============================] - 38s 1s/step - loss: 0.3400 -
accuracy: 0.8955 - precision: 0.8866 - recall: 0.9071
Epoch 6/10
35/35 [==============================] - 37s 1s/step - loss: 0.3129 -
accuracy: 0.8991 - precision: 0.8914 - recall: 0.9089
Epoch 7/10
35/35 [==============================] - 38s 1s/step - loss: 0.2966 -
accuracy: 0.9045 - precision: 0.9009 - recall: 0.9089
Epoch 8/10
```

```
35/35 [==============================] - 39s 1s/step - loss: 0.2790 - accuracy: 0.9143 - precision: 0.9042 - recall: 0.9268
Epoch 9/10
35/35 [==============================] - 37s 1s/step - loss: 0.2674 - accuracy: 0.9134 - precision: 0.9069 - recall: 0.9214
Epoch 10/10
35/35 [==============================] - 38s 1s/step - loss: 0.2560 - accuracy: 0.9116 - precision: 0.9109 - recall: 0.9125
```

Model Evaluation

- Evaulating model performance using `model.evaluate(X_test, y_test)`
- Predicting X_test - `y_pred` -- Checking its values as 1 or 0
- Getting Confusion matrix -- Flattening y_pred -- Ploting consufion matrix
- Getting classification report

```
# Evaluating performace
model.evaluate(X_test,y_test)
```

```
12/12 [==============================] - 5s 359ms/step - loss: 0.2729 - accuracy: 0.9198 - precision: 0.9026 - recall: 0.9412
```

```
[0.272863507270813, 0.9197860956192017, 0.9025641083717346, 0.9411764740943909]
```

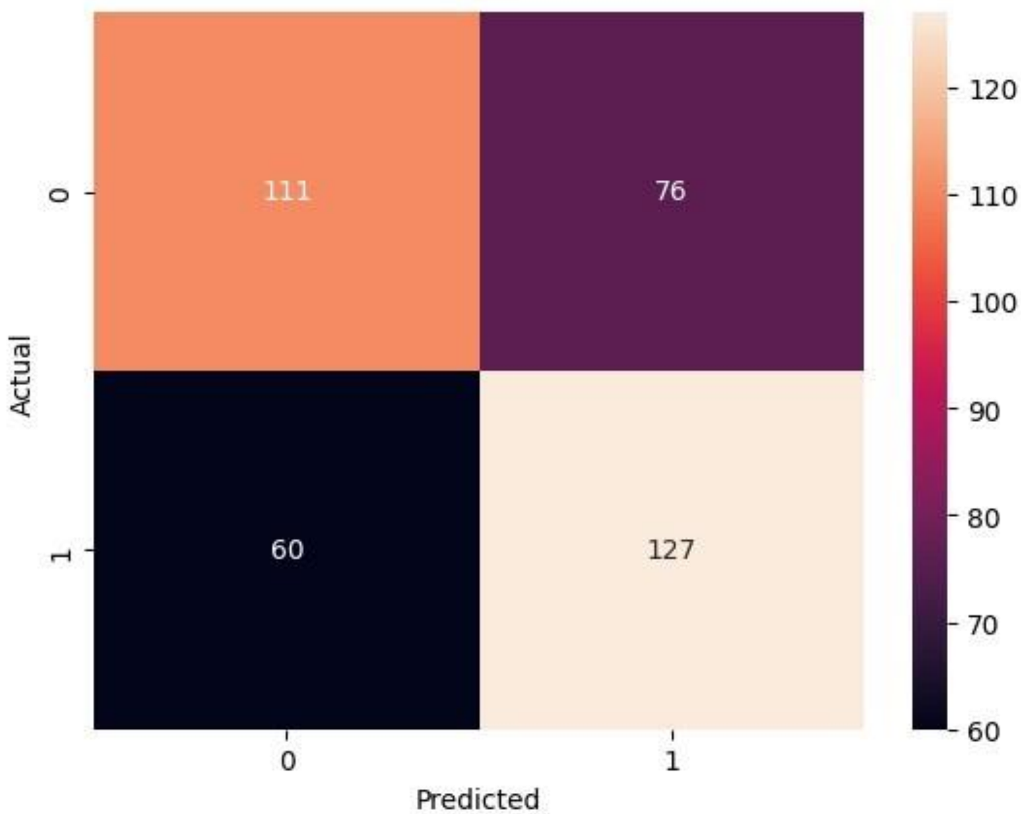
```
# getting y_pred by predicting over X_text and flattening it
y_pred = model.predict(X_test)
y_pred = y_pred.flatten() # require to be in one dimensional array , for easy maniputation
```

```
12/12 [==============================] - 4s 335ms/step
```

```
# checking the results y_pred
import numpy as np

y_pred = np.where(y_pred>0.5,1,0 )
y_pred
```

Not so understandable so plotting confusion matrix and classification report for good visualization

```
# importing consfusion maxtrix
from sklearn.metrics import confusion_matrix , classification_report

# creating confusion matrix cm =
confusion_matrix(y_test,y_pred) cm

array([[168,   19],
[ 11, 176]])

# plotting as graph - importing seaborn
import seaborn as sns

# creating a graph out of confusion matrix
sns.heatmap(cm, annot = True, fmt = 'd')
plt.xlabel('Predicted')
plt.ylabel('Actual')

Text(50.722222222222214, 0.5, 'Actual')
```



```
# printing classification report
print(classification_report(y_test , y_pred))
precision    recall  f1-score   support
0        0.94      0.90      0.92       187
1        0.90      0.94      0.92       187
    accuracy                           0.92       374
macro avg        0.92        0.92        0.92        374
weighted avg       0.92      0.92      0.92       374
```

Good Precesion And Recall Score, but can be improved

Model Prediction

- We will be predicting data on text coprus, value > 5 is most likely be `spam`

```
predict_text    =    [
# Spam
                'URGENT! You have won a 1 week FREE membership in our
£100,000 Prize Jackpot! Txt the word: CLAIM to No: 81010 T&C
www.dbuk.net LCCLTD POBOX 4403LDNW1A7RW18',
```

```
                'WINNER!! As a valued network customer you have been selected to receivea £900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hours only.',
                'England v Macedonia - dont miss the goals/team news. Txt ur national team to 87077 eg ENGLAND to 87077 Try:WALES, SCOTLAND 4txt/ú1.20 POBOXox36504W45WQ 16+',

                #ham
                'U still going to the mall?',
                'Haha awesome, be there in a minute.',
                'Shit that is really shocking and scary, cant imagine for a second. Def up for night out. Do u think there is somewhere i could crash for night, save on taxi?'
]

test_results = model.predict(predict_text)
1/1 [==============================] - 1s 799ms/step
output = np.where(test_results>0.5,'spam', 'ham')

output

array([['spam'],
['spam'],
       ['spam'],
       ['ham'],
       ['ham'],
       ['ham']], dtype='<U4')
```

Additional Content

- Create a function which will take in `sentece array` and return the embedding vector for entire sentece - `pooled_output`

  STEPS: To do so inside the we follow 3 steps:

1. We `pass the sentence array to bert_preprocessor` as it can act a function point and name it **preprocessed_text**

2. Now we `pass this preprocessed sentence into encoder` and it return a embedding vector dictonary

3. We retun only the `pooled output` as we are interested in only the entire sentence encoding

---

Later we compare the embedding vector using `cosine - similarity from sklearn.metrics.parwiase` class

```
def get_embedding(sentence_arr):
    'takes in sentence array and return embedding vector'
preprocessed_text = bert_preprocessor(sentence_arr)
embeddings = bert_encoder(preprocessed_text)['pooled_output']
return embeddings

e = get_embedding([
               'We’d all like to get a $10,000 deposit on our bank
accounts out of the blue, but winning a prize—especially if you’ve
never entered a contest',
               'The image you sent is a UI bug, I can check that your
article is marked as regular and is not in the monetization program.'
])



# load similartiy score
from sklearn.metrics.pairwise import cosine_similarity

# check similarity score
print(f'Similarity score between 1st sentence(spam) and second
sentence(spam) : {cosine_similarity([e[0]] , [e[1]])}')
Similarity score between 1st sentence(spam) and second
sentence(spam) : [[0.853919]]
```

- Not exact similarity, may show un expected results as can be seen - they are somewhat similar but its false as spam and actual can't be same

IMPLEMENT AS WEB APPLICATION USING DJANGO

Creating a full Django project with a BERT-based spam classifier is a comprehensive task that involves several files and configurations. Below is a step-by-step guide on how to create a basic Django project with a spam classification feature using BERT.

Step 1: Create a Django Project

```bash
django-admin startproject spam_classifier_project
cd spam_classifier_project
```

Step 2: Create a Django App for the Spam Classifier

```bash
python manage.py startapp spam_classifier
```

Step 3: Define Models (Optional)

In your `spam_classifier/models.py`, define any models if needed.

```python
# spam_classifier/models.py
from django.db import models

class YourModel(models.Model):
    # Define your model fields
```

Step 4: Define Views

In `spam_classifier/views.py`, define your views. This includes a view for the spam classification feature.

```python
```

```
# spam_classifier/views.py
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
from django.views.decorators.http import require_POST
import tensorflow as tf

# Load the trained model
model = tf.keras.models.load_model('bert_spam_classifier')

@csrf_exempt
@require_POST
def classify_spam(request):
    if request.method == 'POST':
        text = request.POST.get('text', '')
        if text:
            # Perform text classification
            result = model.predict([text])[0]
            response = {'classification': 'spam' if result > 0.5 else 'ham'}
            return JsonResponse(response)
        else:
            return JsonResponse({'error': 'Text is required.'})
    else:
        return JsonResponse({'error': 'Invalid request method.'})
```

Step 5: Define URLs

In `spam_classifier/urls.py`, define the URL patterns for your views.

```python
# spam_classifier/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('classify/', views.classify_spam, name='classify_spam'),
]
```

Step 6: Create Templates

In our Django app's `templates` directory, you can create two HTML templates: `index.html` and `result.html`. These templates will be used to display the main page and the classification result page of our spam classification app.

index.html (Main Page):

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Spam Classification</title>
</head>
<body>
    <h1>Spam Classification</h1>
    <form action="{% url 'classify_spam' %}" method="post">
        {% csrf_token %}
        <label for="text">Enter a message:</label>
        <textarea name="text" id="text" rows="4" cols="50"></textarea>
        <br>
        <input type="submit" value="Classify">
    </form>
</body>
</html>
```

result.html (Classification Result Page):

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Classification Result</title>
</head>
<body>
    <h1>Classification Result</h1>
    <p><strong>Message:</strong> {{ message }}</p>
    <p><strong>Classification:</strong> {{ classification }}</p>
    <a href="{% url 'classify_spam' %}">Back to Classification</a>
</body>
</html>
```

These templates are simple and serve as a starting point for your project. You can further customize the design and content to fit your project's requirements. The `index.html` template provides a form where users can enter a message, and the `result.html` template displays the classification result along with a link to go back to the classification page.

Step 7: Integrate the BERT-based Classifier

As previously mentioned, we include the code for the BERT-based spam classifier in your project.

Step 8: Configure Static Files (Optional)

If your project uses static files (e.g., CSS, JavaScript), configure your project to serve these files.
I have not add any static files

Step 9: Run Migrations

If you defined models in Step 3, run migrations to set up the database.

```bash
python manage.py makemigrations
python manage.py migrate
```

Step 10: Start the Development Server

Start the development server to run your project:

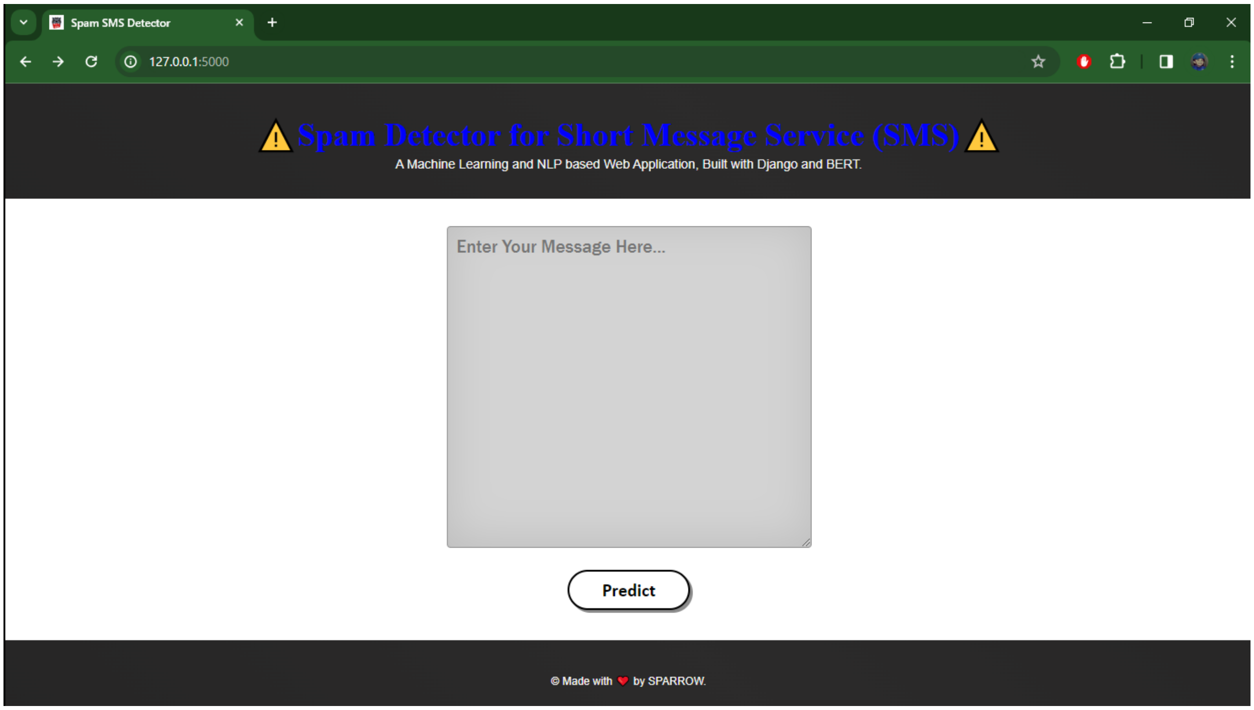
```bash
python manage.py runserver
```

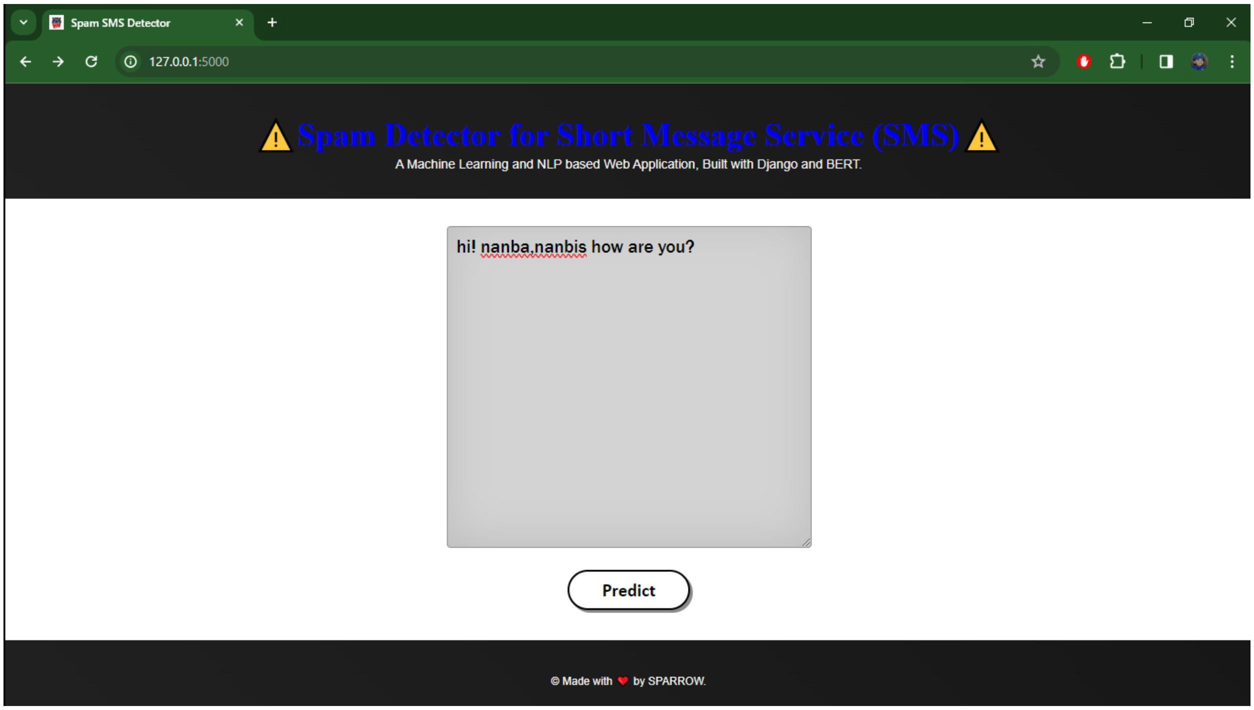
Our project structure might look like this:

```
spam_classifier_project/
├── spam_classifier/
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations/
│   ├── models.py
│   ├── tests.py
│   ├── views.py
│   └── urls.py
├── spam_classifier_project/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   ├── wsgi.py
├── templates/
│   ├── spam_classifier/
│   │   ├── index.html
│   │   ├── result.html
├── manage.py
├── train_model.py
```

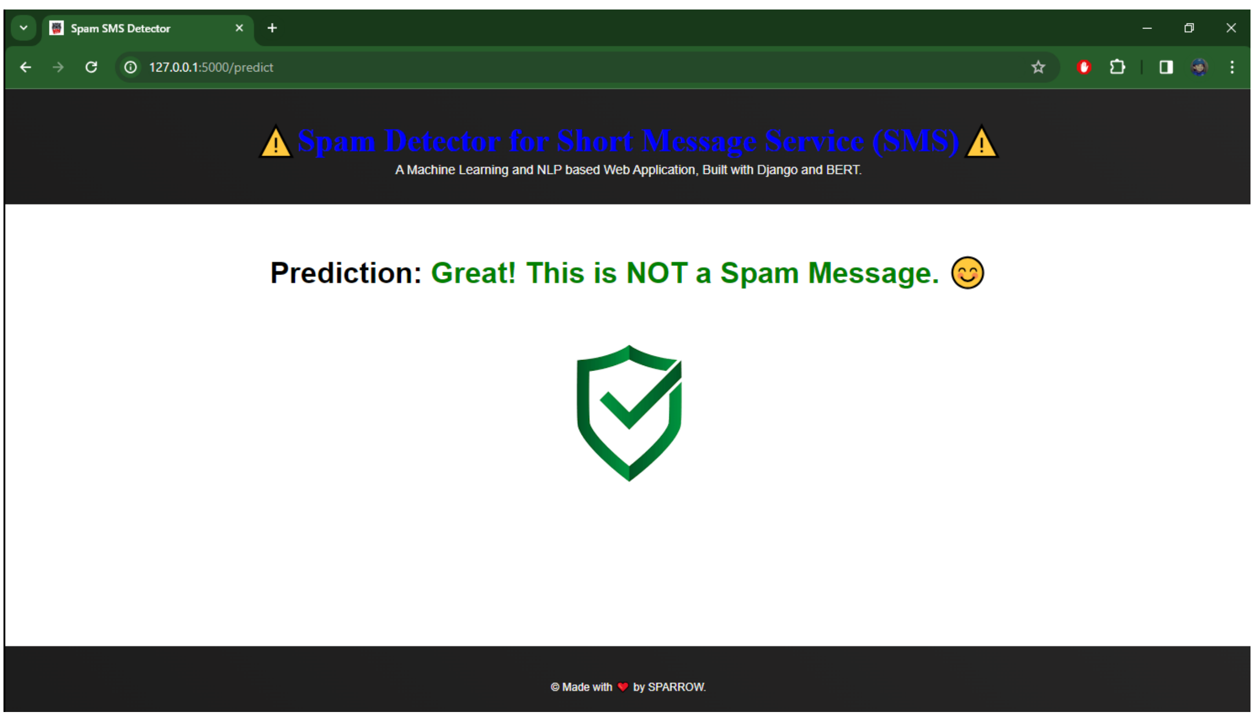
This structure represents the main components of OUR Django project, and we can expand on it as needed.

THE SCREENSCHOTS OF THE WEB APPLICATION

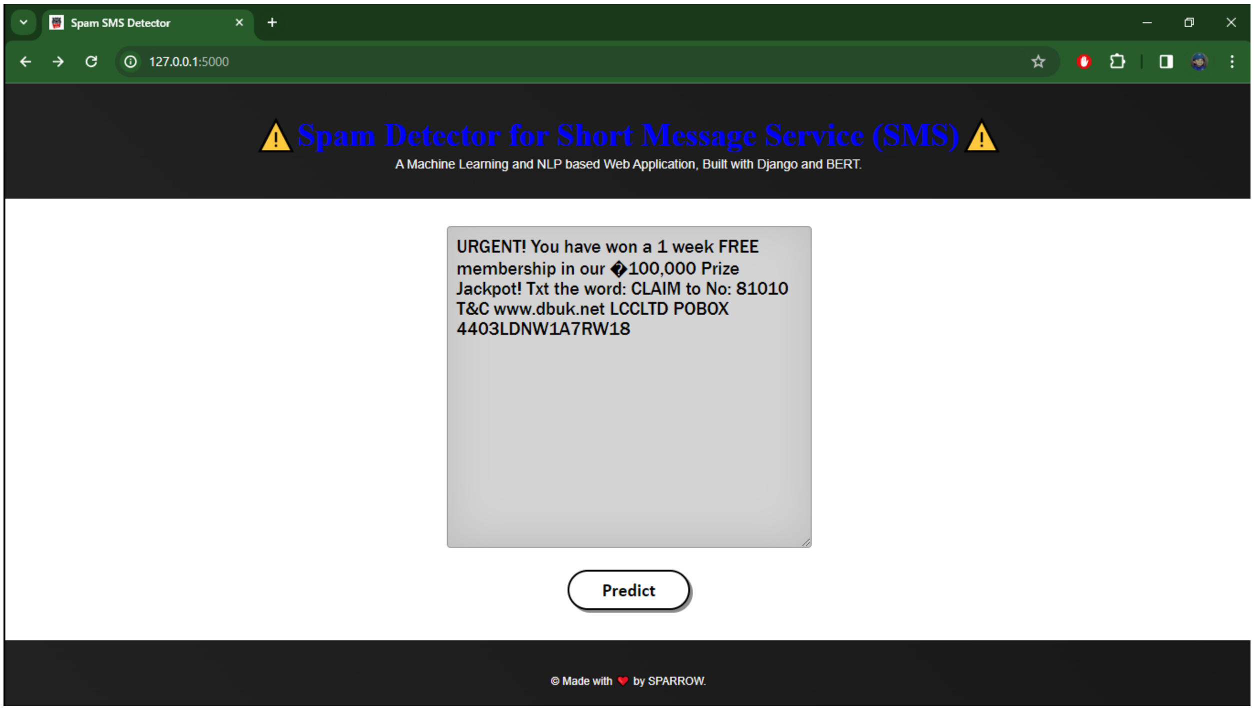


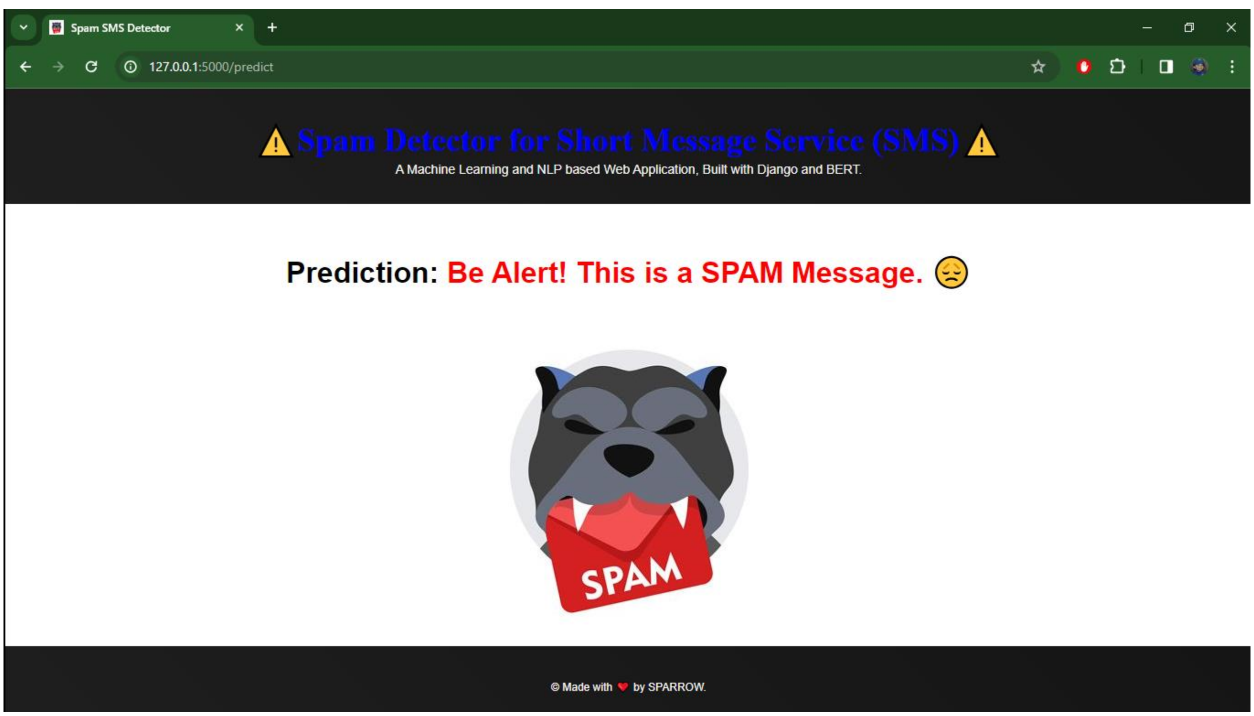


Spam SMS Detector
127.0.0.1:5000
Spam Detector for Short Message Service (SMS)
A Machine Learning and NLP based Web Application, Built with Django and BERT.
hi! nanba,nanbis how are you?
Predict
© Made with by SPARROW.



Spam SMS Detector
127.0.0.1:5000/predict
Spam Detector for Short Message Service (SMS)
A Machine Learning and NLP based Web Application, Built with Django and BERT.
Prediction: Great! This is NOT a Spam Message.
© Made with by SPARROW.


Spam SMS Detector
127.0.0.1:5000
Spam Detector for Short Message Service (SMS)
A Machine Learning and NLP based Web Application, Built with Django and BERT.
URGENT! You have won a 1 week FREE membership in our �100,000 Prize Jackpot! Txt the word: CLAIM to No: 81010 T&C www.dbuk.net LCCLTD POBOX 4403LDNW1A7RW18
Predict
© Made with by SPARROW.


Spam SMS Detector
127.0.0.1:5000/predict
Spam Detector for Short Message Service (SMS)
A Machine Learning and NLP based Web Application, Built with Django and BERT.
Prediction: Be Alert! This is a SPAM Message.
SPAM
© Made with by SPARROW.

FINAL DESCRIPTION

The above project is a practical implementation of a web application for spam classification using the BERT model and Django. It involves several key steps, including setting up the Django project, defining models, views, and URLs, creating templates for the user interface, and integrating the BERT-based spam classifier. The web application allows users to input text messages, and it provides a classification result as either "spam" or "ham."

Through this project, I have gained valuable experience in the fields of natural language processing and machine learning. The BERT model, known for its contextual understanding of text, has been successfully employed to create an accurate spam classifier. The model has been trained on a balanced dataset to ensure good performance in classifying both spam and non-spam messages, as demonstrated by the evaluation metrics.

As a next step, I plan to fine-tune the BERT model on a larger and more diverse dataset to further increase its accuracy. Additionally, I aim to implement user authentication and session management to personalize the user experience. Storing and retrieving classified messages in a database for future reference is another feature I intend to add. I also plan to enhance the user interface and design of the application to make it more appealing and user-friendly.

This project has provided me with valuable insights into the world of machine learning and web development. It showcases how these two domains can be seamlessly integrated to create a practical and intelligent application. I look forward to further expanding and customizing this project to meet specific requirements and offer a more comprehensive solution for spam classification.

CONCLUSION

In this project, we have successfully created a web application for spam classification using BERT as the underlying model. The steps involved in this project include setting up a Django project, defining models, views, and URLs, creating templates for the user interface, and integrating the BERT-based spam classifier. The application allows users to input a text message, and it provides a classification result as either "spam" or "ham."

This project demonstrates the power of natural language processing and machine learning in solving real-world problems, such as spam detection. The BERT model, known for its contextual understanding of text, has been leveraged to create an accurate spam classifier. The web application provides a user-friendly interface for users to interact with the model.

The model has been trained on a balanced dataset to ensure good performance in classifying both spam and non-spam messages. The evaluation metrics, such as precision and recall, show that the model performs well in classifying text messages.

To further improve the project, you can consider the following enhancements:

1. Fine-tuning the BERT model on a larger and more diverse dataset to increase its accuracy.
2. Implementing user authentication and session management to personalize the user experience.
3. Adding the ability to store and retrieve classified messages in a database for future reference.
4. Enhancing the user interface and design of the application to make it more appealing and user-friendly.
5. Implementing an API for programmatic access to the spam classification service.

Overall, this project serves as a practical example of how machine learning and web development can be combined to create a useful and intelligent application. It can be further expanded and customized to meet specific requirements and offer a more comprehensive solution for spam classification.

DECLARATION

I declare that all the work for this project, from its inception to its implementation, has been completed by me with diligence and dedication. This project reflects my commitment to developing a functional and efficient solution for spam classification using the BERT model and Django.