**EXPERIMENT - 6**

**AIM**: To implement spam detection on relevant dataset

**Spam Detection**

Messaging spam, sometimes called SPIM, is a type of spam targeting users of instant messaging (IM) services, SMS, or private messages within websites. Spam detection can be performed using neural networks or transformer based deep learning networks.

Steps for detecting spam:

1. Categorically encode input dataset to 0 and 1 for binary classification.

2. Use TF AutoTokenizer to convert the input data into standard tokens. This also generates the attention mask used by BERT.

3. Load Distilled BERT (base and uncased) pretrained from huggingface

4. Setup accuracy as monitoring metric and Adam as the optimizer

5. Compile the model

6. Evaluate batch\_size, no of epochs and the learing rate for the model hyperparameters.

7. Train the models on train data and plot loss/accuracy curves.

8. Calculate precision, recall and F1 scores on test data

9. Perform Hyperparameter tuning to improve metrics

10. Plot confusion matrix and verify on unseen data

**SMS Spam Collection Dataset**

The SMS Spam Collection v.1 is a public set of SMS labeled messages that have been collected for mobile phone spam research. It has one collection composed by 5,574 English, real and non-encoded messages, tagged according being legitimate (ham) or spam. It has a total of 4,827 SMS legitimate messages (86.6%) and a total of 747 (13.4%) spam messages.

**CODE:**

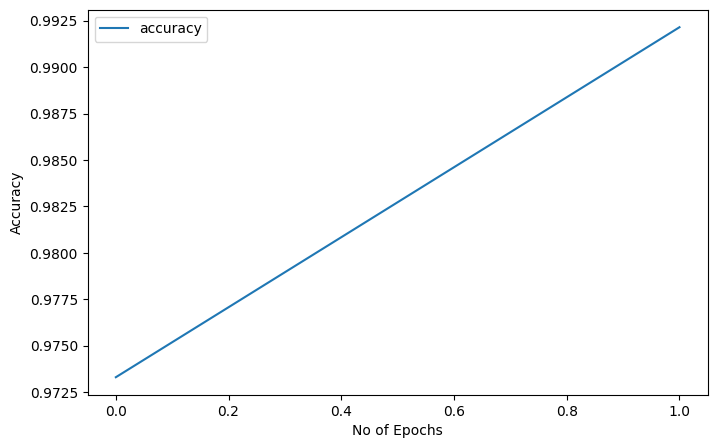
| import os  import numpy as np  import pandas as pd  import seaborn as sns  import tensorflow as tf  import matplotlib.pyplot as plt  from tensorflow.keras.optimizers import Adam  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import confusion\_matrix, classification\_report  from transformers import AutoTokenizer, TFAutoModelForSequenceClassification  for dirname, \_, filenames in os.walk("/content/input"):  for filename in filenames:  print(os.path.join(dirname, filename))  df = pd.read\_csv("/content/input/SPAM text message 20170820 - Data.csv")  df.head()  df.info()  df.isnull().sum()  df["Category"]  df["Category"].value\_counts()  df["Category"].value\_counts().plot(kind="bar")  df["Category"] = pd.get\_dummies(df["Category"], drop\_first=True)  pred\_dict = {0: "Ham", 1: "Spam"}  df.head()  X = df["Message"]  y = df["Category"]  X = list(X)  y = list(y)  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  X, y, test\_size=0.2, random\_state=42  )  tokenizer = AutoTokenizer.from\_pretrained("distilbert-base-uncased")  tokenized\_train\_data = tokenizer(X\_train, return\_tensors="np", padding=True)  tokenized\_test\_data = tokenizer(X\_test, return\_tensors="np", padding=True)  labels = np.array(y\_train)  model = TFAutoModelForSequenceClassification.from\_pretrained("distilbert-base-uncased")  model.summary()  model.compile(optimizer=Adam(3e-5), metrics=["accuracy"])  history = model.fit(  dict(tokenized\_train\_data), labels, batch\_size=16, epochs=2, shuffle=True  )  pd.DataFrame(history.history["loss"]).plot(figsize=(8, 5))  plt.xlabel("No of Epochs")  plt.ylabel("Loss")  plt.legend(["loss"])  plt.show()  pd.DataFrame(history.history["accuracy"]).plot(figsize=(8, 5))  plt.xlabel("No of Epochs")  plt.ylabel("Accuracy")  plt.legend(["accuracy"])  plt.show()  dict(tokenized\_train\_data)  # Predicting our values  y\_pred = model.predict(dict(tokenized\_test\_data))  y\_pred  # Converting our values intoı 0 and 1 labels  logits = y\_pred.logits  softmax = tf.nn.softmax(logits)  predictions = np.argmax(softmax.numpy(), axis=1)  y\_test = np.array(y\_test)  predictions  y\_test  # Evaluating our results  cm = confusion\_matrix(y\_test, predictions)  cr = classification\_report(y\_test, predictions)  print("Confusion Matrix:\n", cm)  print("\nClassification Report:\n", cr)  sns.heatmap(cm, annot=True, cmap="BuPu", fmt="g")  """## Unseen Data Test"""  def run(input: str):  tokenized\_irl = tokenizer([input], return\_tensors="np", padding=True)  y\_irl = model.predict(dict(tokenized\_irl))  new\_logits = y\_irl.logits  new\_softmax = tf.nn.softmax(new\_logits)  new\_predictions = np.argmax(new\_softmax.numpy(), axis=1)  print("Input: ", input)  print("Prediction: ", pred\_dict[new\_predictions[0]])  run("Hi I am Jarvis")  run("Special Offer for you from bank") |
| --- |

**OUTPUT:**

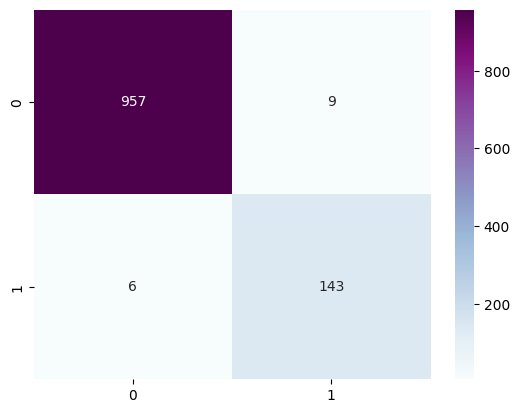
| Model: "tf\_distil\_bert\_for\_sequence\_classification"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  distilbert (TFDistilBertMai multiple 66362880  nLayer)    pre\_classifier (Dense) multiple 590592    classifier (Dense) multiple 1538    dropout\_19 (Dropout) multiple 0    =================================================================  Total params: 66,955,010  Trainable params: 66,955,010  Non-trainable params: 0  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| --- |

| Epoch 1/2  279/279 [==============================] - 150s 399ms/step - loss: 0.0833 - accuracy: 0.9733  Epoch 2/2 |
| --- |

279/279 [==============================] - 104s 372ms/step - loss: 0.0287 - accuracy: 0.9921



| Confusion Matrix:  [[957 9]  [ 6 143]]  Classification Report:  precision recall f1-score support  0 0.99 0.99 0.99 966  1 0.94 0.96 0.95 149  accuracy 0.99 1115  macro avg 0.97 0.98 0.97 1115  weighted avg 0.99 0.99 0.99 1115 |
| --- |



| 1/1 [==============================] - 2s 2s/step  Input: Hi I am Jarvis  Prediction: Ham |
| --- |

| 1/1 [==============================] - 0s 50ms/step  Input: Special Offer for you from bank  Prediction: Spam |
| --- |

**CONCLUSION**: Transformer networks allow for context-based spam classification. Unlike majority of the methods transformer models like BERT model complex semantic relationships in the language and accurately classify sequences. In this experiment, we achieved a maximum accuracy of 99.21%.