

```
In [17]: #####
#
# DEPENDENCIES: Python 3.12.9, Tensorflow 2.18.0
# Python Libraries: numpy, pandas, matplotlib, seaborn, tensorflow, sklearn
#
# sublibraries: matplotlib.pyplot, tensorflow.keras, sklearn
#                tensorflow.keras.layers, tensorflow.keras.layers.TextVectori
#                sklearn.naive_bayes.MultinomialNB
#                sklearn.metrics.confusion_matrix

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import TextVectorization

from HW6utils import load_sms_dataset, naive_bayes_report

from sklearn.metrics import confusion_matrix
```

```
In [18]: #####
#
# TO DO:
# 1) Use load_sms_dataset from HW6utils to load the SMS dataset.
#    This dataset inputs are 5572 text messages and the targets
#    are 1 (spam message) or 0 (ham message). Specify that the
#    function use an 80/20 split in forming the training and testing dataset
# 2) Determine the number of training and testing samples. Print out the fi
#    and print out whether they are HAM or SPAM.

filename = "data/spam.csv"
train_split = 0.8
(train_x, train_y), (test_x, test_y), (avg_words_len, total_words_len) = load_sm

print(f"shape of training samples = {train_x.shape}")
print(f"shape of testing samples = {test_x.shape}\n")

for sample in range(10):
    if train_y[sample]==0:
        print(f"HAM:\n {train_x[sample]}\n")
    else:
        print(f"SPAM:\n {train_x[sample]}\n")
```

Average number of tokens in all sentences = 15  
 Total number of unique words in corpus = 15585  
 shape of training samples = (4457,)  
 shape of testing samples = (1115,)

HAM:

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

HAM:

Ok lar... Joking wif u oni...

SPAM:

Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA  
 to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over1  
 8's

HAM:

U dun say so early hor... U c already then say...

HAM:

Nah I don't think he goes to usf, he lives around here though

SPAM:

FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like  
 some fun you up for it still? Tb ok! XxX std chgs to send, å£1.50 to rcv

HAM:

Even my brother is not like to speak with me. They treat me like aids paten  
 t.

HAM:

As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has  
 been set as your callertune for all Callers. Press \*9 to copy your friends C  
 allertune

SPAM:

WINNER!! As a valued network customer you have been selected to receive a  
 £900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hou  
 rs only.

SPAM:

Had your mobile 11 months or more? U R entitled to Update to the latest col  
 our mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986  
 030

In [19]: #####  
 #  
 # TO DO:

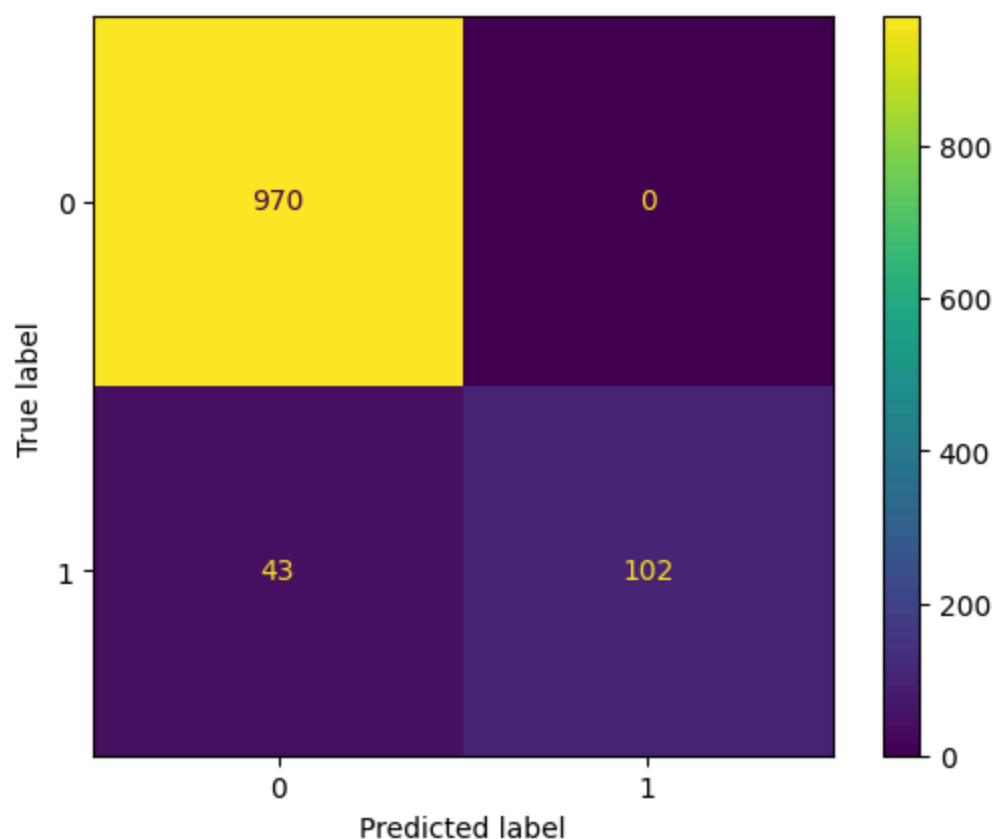
# 1) scikit-learn (a.k.a. sklearn) is a free and open-source  
 # machine learning library for the Python programming language.  
 # One sklearn function of use to us id its multinomial naive  
 # bayes model for text. We've encapsulated the determination  
 # of the sklearn naive bayes model in a HW6utils function.  
 # Describe what this function is doing and use it to print

```
# a report characterize the naive bayes baseline metrics our
# trained LSTM model will need to beat.
# 2) The sklearn NB model report returns three metrics;
# precision, recall and f1-score. Look through the documentation
# and describe what each of these metrics actually measures.
```

```
naive_bayes_report(train_x, train_y, test_x, test_y)
```

NB baseline accuracy = 96.14%

	precision	recall	f1-score	support
0	0.96	1.00	0.98	970
1	1.00	0.70	0.83	145
accuracy			0.96	1115
macro avg	0.98	0.85	0.90	1115
weighted avg	0.96	0.96	0.96	1115



```
In [ ]: #####
#
# TO DO:
# 1) create a text vectorization layer (text_vec) that uses
# the average word length in the corpus (15) as the output_sequence_length
# and assumes an output sequence length equal to the number of unique
# words (tokens) in the corpus. Assume an integer output mode.
# 2) Adapt the textvectorization layer to create a vocabulary.
# 3) Create an embedding layer with input_dim = avg_words_len and output_dim
# 4) Create a bidirectional LSTM layer that takes the text message strings
# as inputs and outputs the probability of the input message being SPAM.
# The model has an input layer with shape (1,) and data type tf.string.
```

```

# This is followed by the textvectorization layer and then the embedding
# We then follow with a bidirectional LSTM of 64 nodes and a tanh activation
# This is followed by another bidirectional LSTM of 64 nodes, then a Flatten
# a dropout layer (0.1 probability) which feeds a dense layer of 32 nodes
# The output layer is another dense layer with a single node and sigmoid
#

# create TextVectorization Layer
text_vec = TextVectorization(
    #max_tokens = avg_words_len,
    standardize = 'lower_and_strip_punctuation',
    #output_mode = 'int',
    output_sequence_length=avg_words_len
)

text_vec.adapt(train_x)

#create Embedding Layer

embedding_layer = layers.Embedding(
    input_dim=total_words_len,
    output_dim=128,
    embeddings_initializer='uniform',
    #input_length=avg_words_len,
)

#####
# instantiate and compile model

inputs = layers.Input(shape=(1,), dtype=tf.string)
x = text_vec(inputs)
x = embedding_layer(x)
x = layers.Bidirectional(layers.LSTM(64, activation='tanh', return_sequences=True))
x = layers.Bidirectional(layers.LSTM(64))(x)
#x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(32, activation='relu')(x)
outputs = layers.Dense(1, activation='sigmoid')(x)
model = keras.Model(inputs, outputs)

model.summary()

model.compile(optimizer = "adam", loss= keras.losses.BinaryCrossentropy(), metrics=

```

**Model: "functional\_5"**

Layer (type)	Output Shape	Par
input_layer_5 (InputLayer)	(None, 1)	
text_vectorization_5 (TextVectorization)	(None, 15)	
embedding_5 (Embedding)	(None, 15, 128)	1,994
bidirectional_10 (Bidirectional)	(None, 15, 128)	98
bidirectional_11 (Bidirectional)	(None, 128)	98
dropout_5 (Dropout)	(None, 128)	
dense_10 (Dense)	(None, 32)	4
dense_11 (Dense)	(None, 1)	

**Total params:** 2,196,673 (8.38 MB)

**Trainable params:** 2,196,673 (8.38 MB)

**Non-trainable params:** 0 (0.00 B)

```
In [ ]: # #####
#
# TO DO:
# 1) Create a call back that saves the model with the smallest validation loss
# using the training data with the testing data being used for validation.
# 5 epochs. Note that this model can take a very long time to train, about
```

```
In [21]: callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="HW6-best-model-2a.keras",
        save_best_only = True,
        monitor = "val_loss"
    )
]
num_epochs = 10
history = model.fit(train_x, train_y, epochs = num_epochs,
                    validation_data = (test_x, test_y),
                    batch_size = 128,
                    callbacks = callbacks)
```

```

Epoch 1/10
35/35 ————— 5s 43ms/step - accuracy: 0.8648 - loss: 0.4231 -
val_accuracy: 0.9767 - val_loss: 0.1036
Epoch 2/10
35/35 ————— 1s 36ms/step - accuracy: 0.9791 - loss: 0.0833 -
val_accuracy: 0.9767 - val_loss: 0.0798
Epoch 3/10
35/35 ————— 2s 43ms/step - accuracy: 0.9941 - loss: 0.0294 -
val_accuracy: 0.9803 - val_loss: 0.0768
Epoch 4/10
35/35 ————— 2s 53ms/step - accuracy: 0.9974 - loss: 0.0147 -
val_accuracy: 0.9740 - val_loss: 0.1047
Epoch 5/10
35/35 ————— 2s 54ms/step - accuracy: 0.9974 - loss: 0.0070 -
val_accuracy: 0.9794 - val_loss: 0.0914
Epoch 6/10
35/35 ————— 2s 53ms/step - accuracy: 0.9995 - loss: 0.0026 -
val_accuracy: 0.9767 - val_loss: 0.1146
Epoch 7/10
35/35 ————— 2s 51ms/step - accuracy: 0.9998 - loss: 8.4459e-0
4 - val_accuracy: 0.9776 - val_loss: 0.1262
Epoch 8/10
35/35 ————— 2s 54ms/step - accuracy: 0.9981 - loss: 0.0061 -
val_accuracy: 0.9785 - val_loss: 0.1030
Epoch 9/10
35/35 ————— 2s 55ms/step - accuracy: 0.9987 - loss: 0.0021 -
val_accuracy: 0.9803 - val_loss: 0.1036
Epoch 10/10
35/35 ————— 2s 53ms/step - accuracy: 1.0000 - loss: 1.8712e-0
4 - val_accuracy: 0.9785 - val_loss: 0.1100

```

```

In [24]: #####
#
# TO DO:
# 1) For the best model, use the sklearn functions classification_report
# and confusion_matrix to evaluate your model using a 0.5 prediction thre
# your model's accuracy and compare to NB baseline accuracy. Print out t
# classification report for your model and compare to NB model's report.
# confusion matrix for your model, display its heatmap and compare to the
# matrix of the NB baseline.

best_model = keras.models.load_model("HW6-best-model-2a.keras")

from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

threshold = 0.5
predictions = best_model.predict(test_x)
predictions = (predictions >= threshold).astype('int32')
accuracy = accuracy_score(test_y, predictions)
print(f"test accuracy = {accuracy*100:.2f}%")
print(classification_report(test_y, predictions))

cm = confusion_matrix(test_y, predictions)

```

```
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
```

35/35 ————— 1s 12ms/step

test accuracy = 98.03%

	precision	recall	f1-score	support
0	0.99	0.99	0.99	970
1	0.91	0.94	0.93	145
accuracy			0.98	1115
macro avg	0.95	0.97	0.96	1115
weighted avg	0.98	0.98	0.98	1115

Out[24]: <sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x1625000e0>

