Text Classification with RoBERTa

Entrée [1]:

```
# pip install tokenizers
```

Entrée [2]:

```
# pip install transformers
```

Entrée [88]:

```
import numpy as np
import regex as re
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statistics
import math
import os

from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

import tensorflow as tf
import tensorflow.keras.backend as K
import tokenizers
from transformers import RobertaTokenizer, TFRobertaModel

from collections import Counter

import warnings
warnings.filterwarnings("ignore")
```

Entrée [89]:

```
# Detect hardware, return appropriate distribution strategy (you can see that it is pretty
try:
    # TPU detection. No parameters necessary if TPU_NAME environment variable is set (alway
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
    print('Running on TPU ', tpu.master())
except ValueError:
    # Default distribution strategy in Tensorflow. Works on CPU and single GPU.
    strategy = tf.distribute.get_strategy()

print('Number of replicas:', strategy.num_replicas_in_sync)
```

```
Number of replicas: 1
```

Entrée [90]:

```
MODEL_NAME = 'roberta-base'
MAX_LEN = 256 # 1st value 256
ARTIFACTS_PATH = '../artifacts/'

BATCH_SIZE = 8 * strategy.num_replicas_in_sync # 1st value 8
EPOCHS = 6 # fist value 3

if not os.path.exists(ARTIFACTS_PATH):
    os.makedirs(ARTIFACTS_PATH)
```

Entrée [125]:

```
df = pd.read_excel ("converted-to-excel.xlsx")
```

Entrée [126]:

```
df = df.drop(columns=['Process_Name',"Unnamed: 0" ,'Type'])
```

Entrée [127]:

```
df
```

Out[127]:

	Concept	Definition
0	PROJECT CHARTER	. the project charter provides preapproved fin...
1	PROJECT MANAGEMENT PLAN	. project management plan components include l...
2	ENTERPRISE ENVIRONMENTAL FACTORS	the enterprise environmental factors influence...
3	ORGANIZATIONAL PROCESS ASSETS	the organizational process assets influence co...
4	EXPERT JUDGMENT	. examples expert judgment control costs proce...
...	...	...
191	WORK PERFORMANCE INFORMATION	. work performance information includes inform...
192	COST FORECASTS	either calculated eac value bottomup eac value...
193	CHANGE REQUESTS	. analysis project performance may result chan...
194	PROJECT MANAGEMENT PLAN UPDATES	Any change to the project management plan goes...
195	PROJECT DOCUMENTS UPDATES	Project documents that may be updated as a res...

196 rows × 2 columns

Entrée [129]:

```
for i in range(len(df)):
    df['Concept'][i] = df['Concept'][i].strip()
    if isinstance(df['Definition'][i], float):
        df['Definition'][i] = ""
```

Entrée [130]:

```
X_data = df[['Definition']].to_numpy().reshape(-1)
y_data = df[['Concept']].to_numpy().reshape(-1)
```

Entrée [131]:

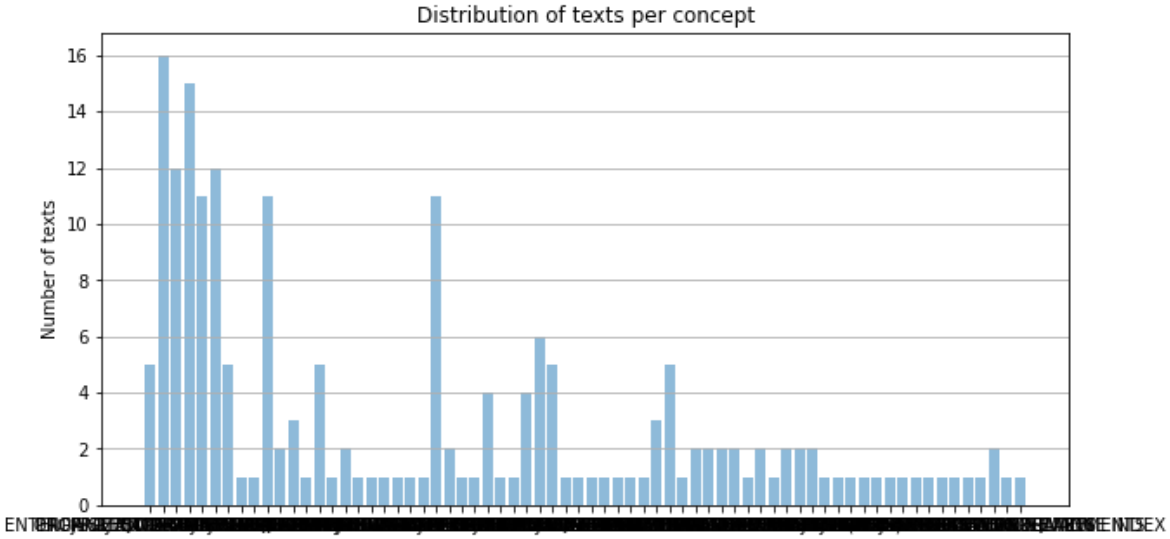
```
categories = df[['Concept']].values.reshape(-1)

counter_categories = Counter(categories)
category_names = counter_categories.keys()
category_values = counter_categories.values()

y_pos = np.arange(len(category_names))

plt.figure(1, figsize=(10, 5))
plt.bar(y_pos, category_values, align='center', alpha=0.5)
plt.xticks(y_pos, category_names)
plt.ylabel('Number of texts')
plt.title('Distribution of texts per concept')
plt.gca().yaxis.grid(True)
plt.show()

print(counter_categories)
```




```
Counter({'PROJECT MANAGEMENT PLAN': 16, 'ORGANIZATIONAL PROCESS ASSETS': 1
5, 'ENTERPRISE ENVIRONMENTAL FACTORS': 12, 'DATA ANALYSIS': 12, 'EXPERT JU
DGMENT': 11, 'PROJECT DOCUMENTS': 11, 'PROJECT DOCUMENTS UPDATES': 11, 'CH
ANGE REQUESTS': 6, 'PROJECT CHARTER': 5, 'MEETINGS': 5, 'DECISION MAKING':
5, 'PROJECT MANAGEMENT PLAN UPDATES': 5, 'PROJECT MANAGEMENT INFORMATION S
YSTEM (PMIS)': 5, 'WORK PERFORMANCE DATA': 4, 'WORK PERFORMANCE INFORMATIO
N': 4, 'AGREEMENTS': 3, 'LEADS AND LAGS': 3, 'BUSINESS DOCUMENTS': 2, 'INT
ERPERSONAL AND TEAM SKILLS': 2, 'DECOMPOSITION': 2, 'ANALOGOUS ESTIMATIN
G': 2, 'PARAMETRIC ESTIMATING': 2, 'THREE-POINT ESTIMATING': 2, 'BOTTOM-UP
ESTIMATING': 2, 'BASIS OF ESTIMATES': 2, 'CRITICAL PATH METHOD': 2, 'RESOU
RCE OPTIMIZATION': 2, 'SCHEDULE COMPRESSION': 2, 'PROJECT FUNDING REQUIREM
ENTS': 2, 'SCOPE MANAGEMENT PLAN': 1, 'REQUIREMENTS MANAGEMENT PLAN': 1,
'DATA GATHERING': 1, 'DATA REPRESENTATION': 1, 'CONTEXT DIAGRAM': 1, 'PROT
OTYPES': 1, 'REQUIREMENTS DOCUMENTATION': 1, 'REQUIREMENTS TRACEABILITY MA
TRIX': 1, 'PRODUCT ANALYSIS': 1, 'PROJECT SCOPE STATEMENT': 1, 'SCOPE BASE
LINE': 1, 'VERIFIED DELIVERABLES': 1, 'INSPECTION': 1, 'ACCEPTED DELIVERAB
LES': 1, 'SCHEDULE MANAGEMENT PLAN': 1, 'ROLLING WAVE PLANNING': 1, 'ACTIV
ITY LIST': 1, 'ACTIVITY ATTRIBUTES': 1, 'MILESTONE LIST': 1, 'PRECEDENCE D
IAGRAMMING METHOD': 1, 'DEPENDENCY DETERMINATION AND INTEGRATION': 1, 'PRO
JECT SCHEDULE NETWORK DIAGRAMS': 1, 'DURATION ESTIMATES': 1, 'SCHEDULE NET
WORK ANALYSIS': 1, 'AGILE RELEASE PLANNING': 1, 'SCHEDULE BASELINE': 1, 'P
ROJECT SCHEDULE': 1, 'SCHEDULE DATA': 1, 'PROJECT CALENDARS': 1, 'SCHEDULE
FORECASTS': 1, 'COST MANAGEMENT PLAN': 1, 'COST ESTIMATES': 1, 'COST AGGRE
```

```
GATION': 1, 'HISTORICAL INFORMATION REVIEW': 1, 'FUNDING LIMIT RECONCILIAT
ION': 1, 'FINANCING': 1, 'COST BASELINE': 1, 'TO-COMPLETE PERFORMANCE INDE
X': 1, 'COST FORECASTS': 1})
```

Entrée [132]:

```
n_texts = len(X_data)
print('Texts in dataset: %d' % n_texts)

Concepts = df['Concept'].unique()
n_Concepts = len(Concepts)
print('Number of Concepts: %d' % n_Concepts)

print('Done!')
```

```
Texts in dataset: 196
Number of Concepts: 68
Done!
```

Tokenize & encode

Entrée [133]:

```
def roberta_encode(texts, tokenizer):
    ct = len(texts)
    input_ids = np.ones((ct, MAX_LEN), dtype='int32')
    attention_mask = np.zeros((ct, MAX_LEN), dtype='int32')
    token_type_ids = np.zeros((ct, MAX_LEN), dtype='int32') # Not used in text classificati

    for k, text in enumerate(texts):
        # Tokenize
        tok_text = tokenizer.tokenize(text)

        # Truncate and convert tokens to numerical IDs
        enc_text = tokenizer.convert_tokens_to_ids(tok_text[:(MAX_LEN-2)])

        input_length = len(enc_text) + 2
        input_length = input_length if input_length < MAX_LEN else MAX_LEN

        # Add tokens [CLS] and [SEP] at the beginning and the end
        input_ids[k,:input_length] = np.asarray([0] + enc_text + [2], dtype='int32')

        # Set to 1s in the attention input
        attention_mask[k,:input_length] = 1

    return {
        'input_word_ids': input_ids,
        'input_mask': attention_mask,
        'input_type_ids': token_type_ids
    }
```

Entrée [134]:

```
# Transform categories into numbers
Concept_to_id = {}
Concept_to_name = {}

for index, c in enumerate(y_data):
    if c in Concept_to_id:
        Concept_id = Concept_to_id[c]
    else:
        Concept_id = len(Concept_to_id)
        Concept_to_id[c] = Concept_id
        Concept_to_name[Concept_id] = c

    y_data[index] = Concept_id

# Display dictionary
Concept_to_name
```

Out[134]:

```
{0: 'PROJECT CHARTER',
 1: 'PROJECT MANAGEMENT PLAN',
 2: 'ENTERPRISE ENVIRONMENTAL FACTORS',
 3: 'ORGANIZATIONAL PROCESS ASSETS',
 4: 'EXPERT JUDGMENT',
 5: 'DATA ANALYSIS',
 6: 'MEETINGS',
 7: 'SCOPE MANAGEMENT PLAN',
 8: 'REQUIREMENTS MANAGEMENT PLAN',
 9: 'PROJECT DOCUMENTS',
 10: 'BUSINESS DOCUMENTS',
 11: 'AGREEMENTS',
 12: 'DATA GATHERING',
 13: 'DECISION MAKING',
 14: 'DATA REPRESENTATION',
 15: 'INTERPERSONAL AND TEAM SKILLS',
 16: 'CONTEXT DIAGRAM',
 17: 'PROTOTYPES',
 18: 'REQUIREMENTS DOCUMENTATION',
 19: 'REQUIREMENTS TRACEABILITY MATRIX',
 20: 'PRODUCT ANALYSIS',
 21: 'PROJECT SCOPE STATEMENT',
 22: 'PROJECT DOCUMENTS UPDATES',
 23: 'DECOMPOSITION',
 24: 'SCOPE BASELINE',
 25: 'VERIFIED DELIVERABLES',
 26: 'WORK PERFORMANCE DATA',
 27: 'INSPECTION',
 28: 'ACCEPTED DELIVERABLES',
 29: 'WORK PERFORMANCE INFORMATION',
 30: 'CHANGE REQUESTS',
 31: 'PROJECT MANAGEMENT PLAN UPDATES',
 32: 'SCHEDULE MANAGEMENT PLAN',
 33: 'ROLLING WAVE PLANNING',
 34: 'ACTIVITY LIST',
 35: 'ACTIVITY ATTRIBUTES',
 36: 'MILESTONE LIST',
 37: 'PRECEDENCE DIAGRAMMING METHOD',
 38: 'DEPENDENCY DETERMINATION AND INTEGRATION',
```

```
 39: 'LEADS AND LAGS',
 40: 'PROJECT MANAGEMENT INFORMATION SYSTEM (PMIS)',
 41: 'PROJECT SCHEDULE NETWORK DIAGRAMS',
 42: 'ANALOGOUS ESTIMATING',
 43: 'PARAMETRIC ESTIMATING',
 44: 'THREE-POINT ESTIMATING',
 45: 'BOTTOM-UP ESTIMATING',
 46: 'DURATION ESTIMATES',
 47: 'BASIS OF ESTIMATES',
 48: 'SCHEDULE NETWORK ANALYSIS',
 49: 'CRITICAL PATH METHOD',
 50: 'RESOURCE OPTIMIZATION',
 51: 'SCHEDULE COMPRESSION',
 52: 'AGILE RELEASE PLANNING',
 53: 'SCHEDULE BASELINE',
 54: 'PROJECT SCHEDULE',
 55: 'SCHEDULE DATA',
 56: 'PROJECT CALENDARS',
 57: 'SCHEDULE FORECASTS',
 58: 'COST MANAGEMENT PLAN',
 59: 'COST ESTIMATES',
 60: 'COST AGGREGATION',
 61: 'HISTORICAL INFORMATION REVIEW',
 62: 'FUNDING LIMIT RECONCILIATION',
 63: 'FINANCING',
 64: 'COST BASELINE',
 65: 'PROJECT FUNDING REQUIREMENTS',
 66: 'TO-COMPLETE PERFORMANCE INDEX',
 67: 'COST FORECASTS'}
```

Entrée [135]:

```
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.3, random_s
```

Entrée [136]:

```
tokenizer = RobertaTokenizer.from_pretrained(MODEL_NAME)
```

Encoding

Entrée [137]:

```
X_train = roberta_encode(X_train, tokenizer)
X_test = roberta_encode(X_test, tokenizer)

y_train = np.asarray(y_train, dtype='int32')
y_test = np.asarray(y_test, dtype='int32')
```

Create RoBERTa model

Entrée [139]:

```
def build_model(n_Concepts):
    with strategy.scope():
        input_word_ids = tf.keras.Input(shape=(MAX_LEN,), dtype=tf.int32, name='input_word_
        input_mask = tf.keras.Input(shape=(MAX_LEN,), dtype=tf.int32, name='input_mask')
        input_type_ids = tf.keras.Input(shape=(MAX_LEN,), dtype=tf.int32, name='input_type_

        # Import RoBERTa model from HuggingFace
        roberta_model = TFRobertaModel.from_pretrained(MODEL_NAME)
        x = roberta_model(input_word_ids, attention_mask=input_mask, token_type_ids=input_t

        # Huggingface transformers have multiple outputs, embeddings are the first one,
        # so let's slice out the first position
        x = x[0]

        x = tf.keras.layers.Dropout(0.1)(x)
        x = tf.keras.layers.Flatten()(x)
        x = tf.keras.layers.Dense(344, activation='tanh')(x)
        x = tf.keras.layers.Dense(172, activation='tanh')(x)
        x = tf.keras.layers.Dense(n_Concepts, activation='softmax')(x)

        model = tf.keras.Model(inputs=[input_word_ids, input_mask, input_type_ids], outputs
        model.compile(
            optimizer=tf.keras.optimizers.Adam(lr=1e-5),
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

        return model
```

Entrée [140]:

```
Concepts = df['Concept'].unique()
n_concepts = len(Concepts)

with strategy.scope():
    model = build_model(n_concepts)
    model.summary()
```

```
Downloading:   0%|          | 0.00/657M [00:00<?, ?B/s]

Some layers from the model checkpoint at roberta-base were not used when ini
tializing TFRobertaModel: ['lm_head']
- This IS expected if you are initializing TFRobertaModel from the checkpoin
t of a model trained on another task or with another architecture (e.g. init
ializing a BertForSequenceClassification model from a BertForPreTraining mod
el).
- This IS NOT expected if you are initializing TFRobertaModel from the check
point of a model that you expect to be exactly identical (initializing a Ber
tForSequenceClassification model from a BertForSequenceClassification mode
l).
All the layers of TFRobertaModel were initialized from the model checkpoint
at roberta-base.
If your task is similar to the task the model of the checkpoint was trained
on, you can already use TFRobertaModel for predictions without further train
ing.

Model: "model"
_________________________________________________________________________
_______________________
 Layer (type)                Output Shape              Param #     Connected t
o
=========================================================================
======================
 input_word_ids (InputLayer)    [(None, 256)]        0           []

 input_mask (InputLayer)        [(None, 256)]        0           []

 input_type_ids (InputLayer)    [(None, 256)]        0           []

 tf_roberta_model (TFRobertaMod  TFBaseModelOutputWi  124645632  ['input_wor
d_ids[0][0]',
 el)                            thPoolingAndCrossAt               'input_mas
k[0][0]',
                                tentions(last_hidde               'input_typ
e_ids[0][0]']
                                n_state=(None, 256,
                                 768),
                                 pooler_output=(Non
                                e, 768),
                                 past_key_values=No
                                ne, hidden_states=N
                                one, attentions=Non
                                e, cross_attentions
                                =None)

 dropout_37 (Dropout)           (None, 256, 768)     0           ['tf_robert
a_model[0][0]']

 flatten (Flatten)              (None, 196608)       0           ['dropout_3
7[0][0]']
```

```
 dense (Dense)                  (None, 344)          67633496    ['flatten
[0][0]']

 dense_1 (Dense)                (None, 172)          59340       ['dense[0]
[0]']

 dense_2 (Dense)                (None, 68)           11764       ['dense_1
[0][0]']

==================================================================================
======================
Total params: 192,350,232
Trainable params: 192,350,232
Non-trainable params: 0
__________________________________________________________________________________
_____________________
```

Train model

Entrée [141]:

```
with strategy.scope():
    print('Training...')
    history = model.fit(X_train,
                        y_train,
                        epochs=EPOCHS,
                        batch_size=BATCH_SIZE,
                        verbose=1,
                        validation_data=(X_test, y_test))
```

```
Training...
Epoch 1/6
WARNING:tensorflow:Gradients do not exist for variables ['tf_roberta_model/r
oberta/pooler/dense/kernel:0', 'tf_roberta_model/roberta/pooler/dense/bias:
0'] when minimizing the loss. If you're using `model.compile()`, did you for
get to provide a `loss`argument?
WARNING:tensorflow:Gradients do not exist for variables ['tf_roberta_model/r
oberta/pooler/dense/kernel:0', 'tf_roberta_model/roberta/pooler/dense/bias:
0'] when minimizing the loss. If you're using `model.compile()`, did you for
get to provide a `loss`argument?
18/18 [==============================] - 291s 15s/step - loss: 3.5163 - accu
racy: 0.2409 - val_loss: 2.8176 - val_accuracy: 0.4576
Epoch 2/6
18/18 [==============================] - 255s 14s/step - loss: 1.6050 - accu
racy: 0.6861 - val_loss: 1.8569 - val_accuracy: 0.6441
Epoch 3/6
18/18 [==============================] - 256s 14s/step - loss: 0.6841 - accu
racy: 0.9051 - val_loss: 1.4298 - val_accuracy: 0.7119
Epoch 4/6
18/18 [==============================] - 273s 15s/step - loss: 0.3534 - accu
racy: 0.9562 - val_loss: 1.3002 - val_accuracy: 0.7458
Epoch 5/6
18/18 [==============================] - 265s 15s/step - loss: 0.2038 - accu
racy: 0.9635 - val_loss: 1.2377 - val_accuracy: 0.7458
Epoch 6/6
18/18 [==============================] - 259s 14s/step - loss: 0.1520 - accu
racy: 0.9781 - val_loss: 1.2436 - val_accuracy: 0.7458
```

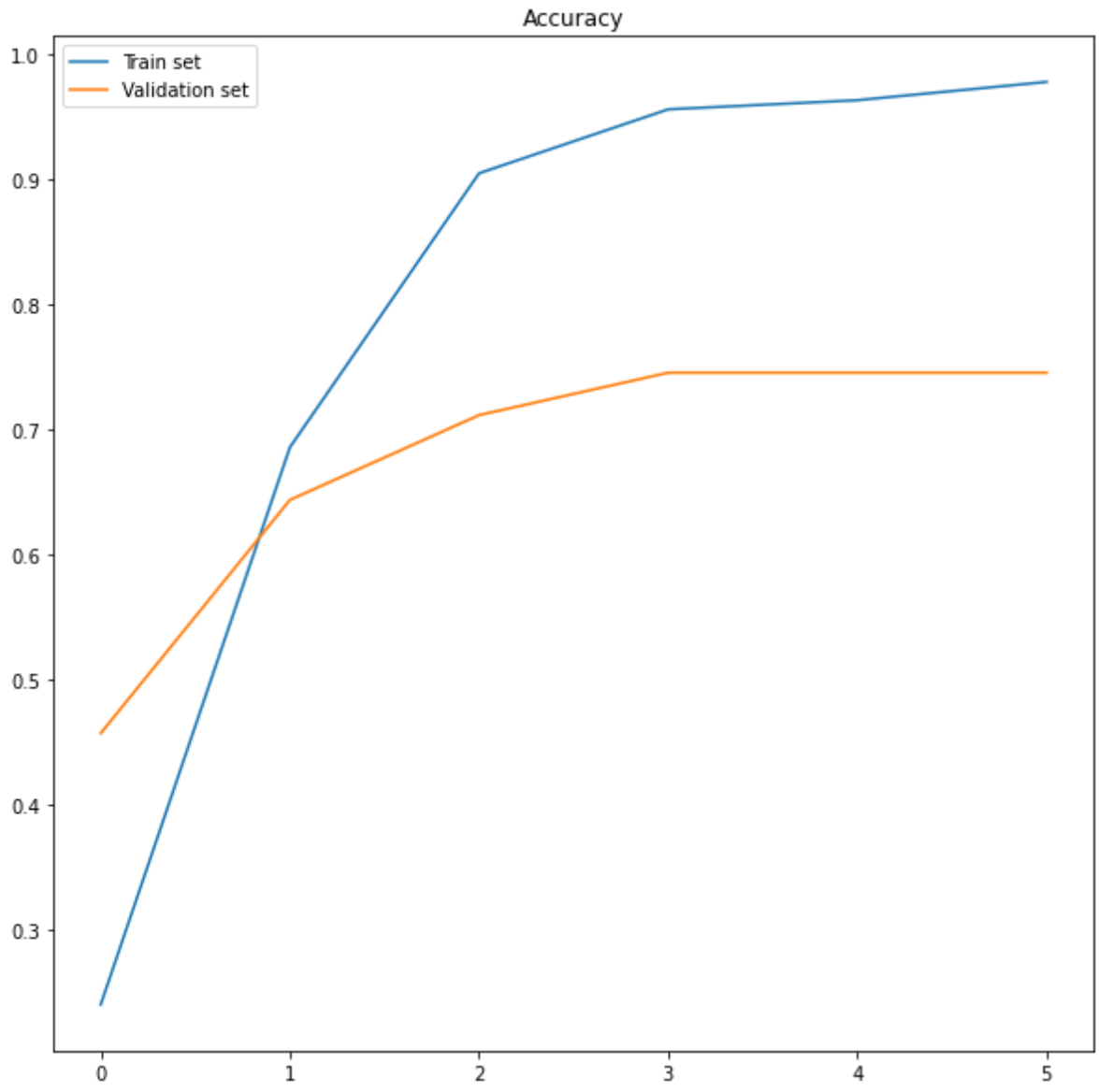
Entrée [142]:

```
# This plot will look much better if we train models with more epochs
plt.figure(figsize=(10, 10))
plt.title('Accuracy')

xaxis = np.arange(len(history.history['accuracy']))
plt.plot(xaxis, history.history['accuracy'], label='Train set')
plt.plot(xaxis, history.history['val_accuracy'], label='Validation set')
plt.legend()
```

Out[142]:

```
<matplotlib.legend.Legend at 0x28e8214a5b0>
```



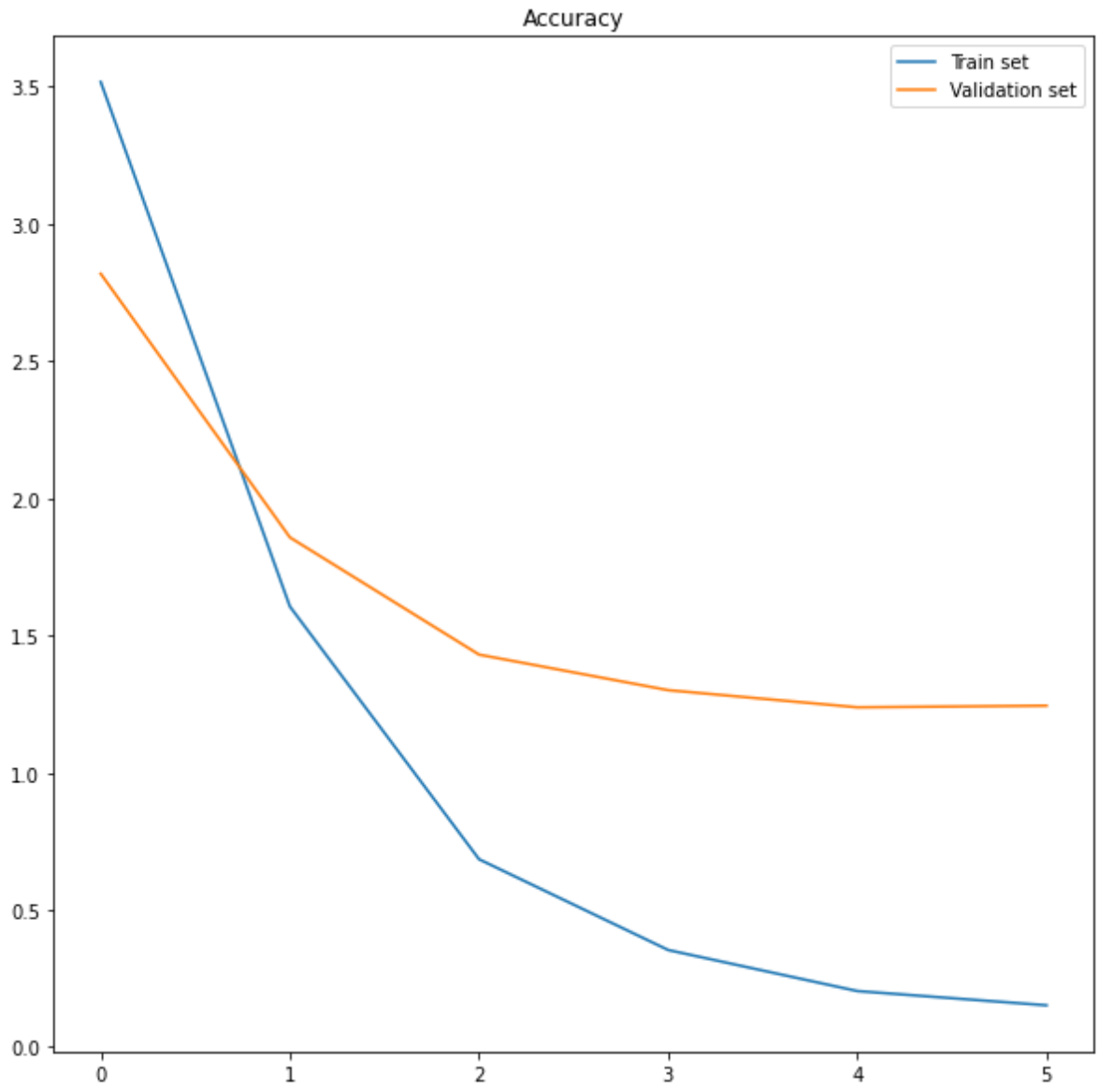
Entrée [143]:

```
plt.figure(figsize=(10, 10))
plt.title('Accuracy')

xaxis = np.arange(len(history.history['loss']))
plt.plot(xaxis, history.history['loss'], label='Train set')
plt.plot(xaxis, history.history['val_loss'], label='Validation set')
plt.legend()
```

Out[143]:

```
<matplotlib.legend.Legend at 0x28e8043e280>
```



Evaluation

Entrée [144]:

```
def plot_confusion_matrix(X_test, y_test, model):
    y_pred = model.predict(X_test)
    y_pred = [np.argmax(i) for i in model.predict(X_test)]

    con_mat = tf.math.confusion_matrix(labels=y_test, predictions=y_pred).numpy()

    con_mat_norm = np.around(con_mat.astype('float') / con_mat.sum(axis=1)[:, np.newaxis],
    label_names = list(range(len(con_mat_norm)))

    con_mat_df = pd.DataFrame(con_mat_norm,
                              index=label_names,
                              columns=label_names)

    figure = plt.figure(figsize=(10, 10))
    sns.heatmap(con_mat_df, cmap=plt.cm.Blues, annot=True)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

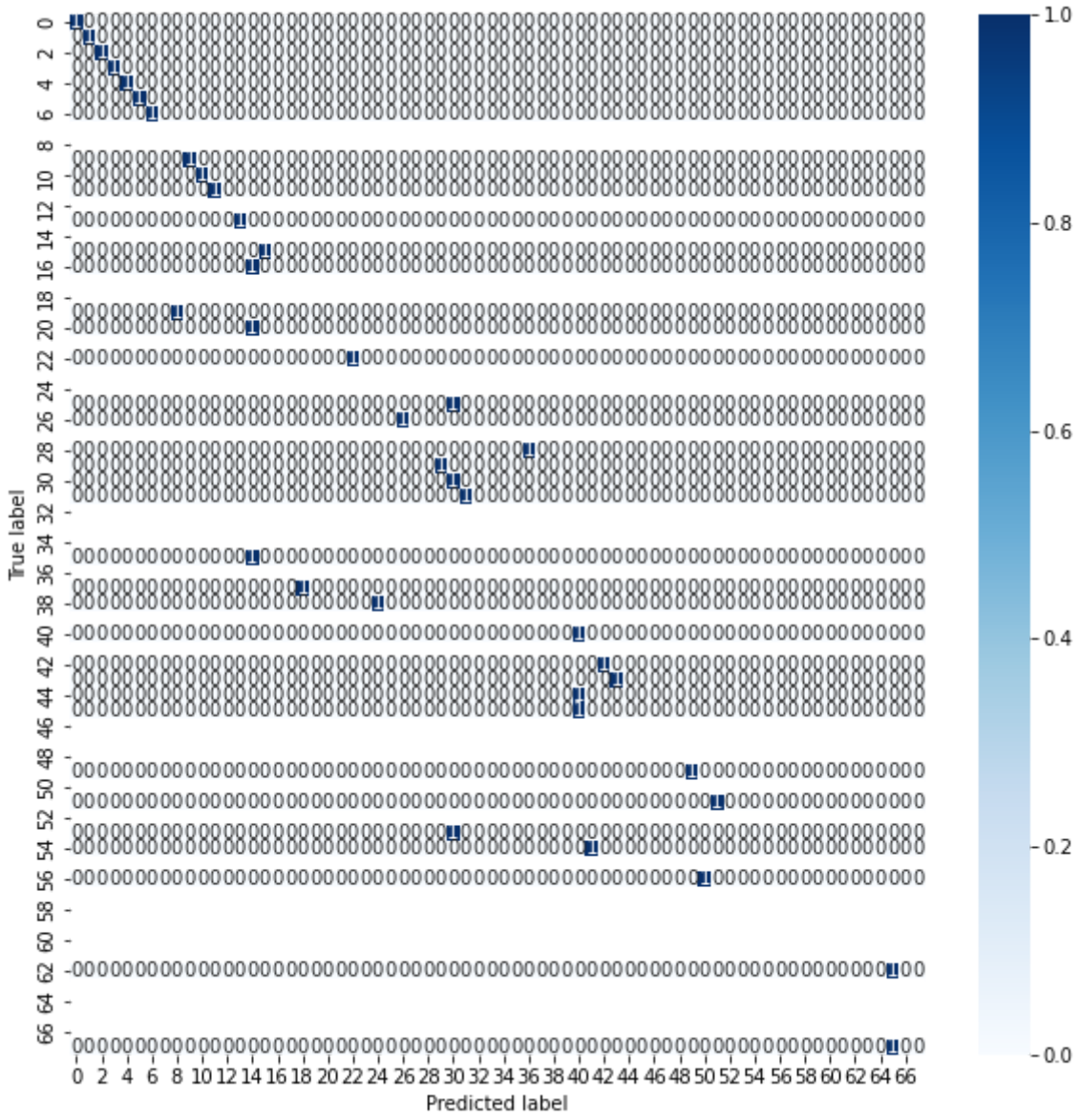
Entrée [145]:

```
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1] * 100))
```

```
Accuracy: 74.58%
```

Entrée [146]:

```
plot_confusion_matrix(X_test, y_test, model)
```



Exporting model

Entrée [150]:

```
model.save("WORKING_MODEL")
```

```
WARNING:absl:Found untraced functions such as encoder_layer_call_fn, encoder
_layer_call_and_return_conditional_losses, pooler_layer_call_fn, pooler_laye
r_call_and_return_conditional_losses, embeddings_layer_call_fn while saving
(showing 5 of 420). These functions will not be directly callable after load
ing.

INFO:tensorflow:Assets written to: WORKING_MODEL\assets

INFO:tensorflow:Assets written to: WORKING_MODEL\assets
```

Entrée [156]:

```
x = roberta_encode(["Project management plan"], tokenizer)
```

Entrée [157]:

```
y= model.predict(x)
y
```

Out[157]:

```
array([[0.0086939 , 0.00407735, 0.00294205, 0.003266  , 0.02949057,
        0.00485756, 0.49417192, 0.01406344, 0.01004485, 0.00319013,
        0.01335767, 0.00350768, 0.00240783, 0.0048913 , 0.00120643,
        0.00489485, 0.00394487, 0.00560273, 0.00285139, 0.00129144,
        0.00557028, 0.00554202, 0.01361904, 0.00276908, 0.00298229,
        0.00187181, 0.00366615, 0.00434398, 0.00330553, 0.00539217,
        0.03115373, 0.01777408, 0.0041836 , 0.0096928 , 0.00925456,
        0.00973937, 0.01197103, 0.0021792 , 0.00402963, 0.00369854,
        0.02422201, 0.0145164 , 0.00269233, 0.00571396, 0.00284135,
        0.01219873, 0.01373511, 0.00400681, 0.00252343, 0.00491598,
        0.00249691, 0.00472584, 0.0126842 , 0.0058393 , 0.00501179,
        0.00869709, 0.00195407, 0.00577125, 0.00374981, 0.00815804,
        0.00319221, 0.02628224, 0.00419917, 0.00365518, 0.00178775,
        0.03251747, 0.0039422 , 0.00647654]], dtype=float32)
```

Entrée [158]:

```
np.argmax(y)
```

Out[158]:

6

Entrée [159]:

```
Concept_to_name
```

Out[159]:

```
{0: 'PROJECT CHARTER',
 1: 'PROJECT MANAGEMENT PLAN',
 2: 'ENTERPRISE ENVIRONMENTAL FACTORS',
 3: 'ORGANIZATIONAL PROCESS ASSETS',
 4: 'EXPERT JUDGMENT',
 5: 'DATA ANALYSIS',
 6: 'MEETINGS',
 7: 'SCOPE MANAGEMENT PLAN',
 8: 'REQUIREMENTS MANAGEMENT PLAN',
 9: 'PROJECT DOCUMENTS',
 10: 'BUSINESS DOCUMENTS',
 11: 'AGREEMENTS',
 12: 'DATA GATHERING',
 13: 'DECISION MAKING',
 14: 'DATA REPRESENTATION',
 15: 'INTERPERSONAL AND TEAM SKILLS',
 16: 'CONTEXT DIAGRAM',
 17: 'PROTOTYPES',
 18: 'REQUIREMENTS DOCUMENTATION',
 19: 'REQUIREMENTS TRACEABILITY MATRIX',
 20: 'PRODUCT ANALYSIS',
 21: 'PROJECT SCOPE STATEMENT',
 22: 'PROJECT DOCUMENTS UPDATES',
 23: 'DECOMPOSITION',
 24: 'SCOPE BASELINE',
 25: 'VERIFIED DELIVERABLES',
 26: 'WORK PERFORMANCE DATA',
 27: 'INSPECTION',
 28: 'ACCEPTED DELIVERABLES',
 29: 'WORK PERFORMANCE INFORMATION',
 30: 'CHANGE REQUESTS',
 31: 'PROJECT MANAGEMENT PLAN UPDATES',
 32: 'SCHEDULE MANAGEMENT PLAN',
 33: 'ROLLING WAVE PLANNING',
 34: 'ACTIVITY LIST',
 35: 'ACTIVITY ATTRIBUTES',
 36: 'MILESTONE LIST',
 37: 'PRECEDENCE DIAGRAMMING METHOD',
 38: 'DEPENDENCY DETERMINATION AND INTEGRATION',
 39: 'LEADS AND LAGS',
 40: 'PROJECT MANAGEMENT INFORMATION SYSTEM (PMIS)',
 41: 'PROJECT SCHEDULE NETWORK DIAGRAMS',
 42: 'ANALOGOUS ESTIMATING',
 43: 'PARAMETRIC ESTIMATING',
 44: 'THREE-POINT ESTIMATING',
 45: 'BOTTOM-UP ESTIMATING',
 46: 'DURATION ESTIMATES',
 47: 'BASIS OF ESTIMATES',
 48: 'SCHEDULE NETWORK ANALYSIS',
 49: 'CRITICAL PATH METHOD',
 50: 'RESOURCE OPTIMIZATION',
 51: 'SCHEDULE COMPRESSION',
 52: 'AGILE RELEASE PLANNING',
 53: 'SCHEDULE BASELINE',
 54: 'PROJECT SCHEDULE',
```

```
55: 'SCHEDULE DATA',
56: 'PROJECT CALENDARS',
57: 'SCHEDULE FORECASTS',
58: 'COST MANAGEMENT PLAN',
59: 'COST ESTIMATES',
60: 'COST AGGREGATION',
61: 'HISTORICAL INFORMATION REVIEW',
62: 'FUNDING LIMIT RECONCILIATION',
63: 'FINANCING',
64: 'COST BASELINE',
65: 'PROJECT FUNDING REQUIREMENTS',
66: 'TO-COMPLETE PERFORMANCE INDEX',
67: 'COST FORECASTS'}
```