Phase 2 BERT Model Training Guideline

ST1506: DSDA Final Year Project

Company Profiling Text Mining

August 2021

SINGAPORE POLYTECHNIC | SP

Synopsis

The purpose of this documentation is to demonstrate a detailed breakdown on the content of the ipython file for model training, so that users will understand the rationale behind the content.

This documentation is for data scientists who want to see the usage of BERT in this specific context and possibly improve from the current work.

Table of Contents

Breakdown Of Notebook Contents

Hardware Rendering

1. When the user chooses to run the code using GPU in Google Colab, the user can run the notebook as per usual so long as the runtime has been set to GPU in Google Colab.

2. However, if the user chooses to run the code using TPU, the user firstly must initialize itself first. An output will be rendered, which will state the number of logical TPU devices the TPU cluster comprises of, which are capable of parallel processing: This process also defines a distribution strategy for distributed training over the number of logical TPU devices.

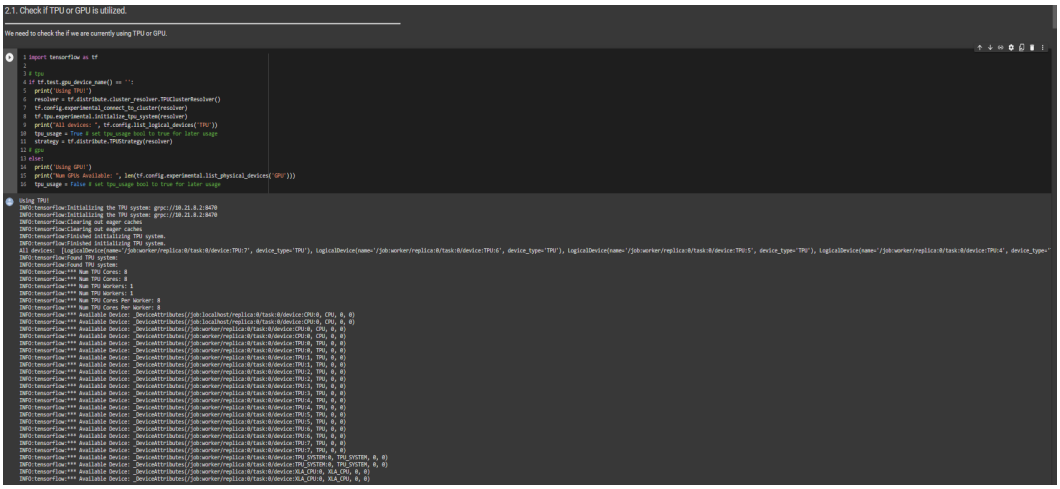


Figure 1: TPU initialization with number of workers

Loading necessary libraries

1. In Section 2.2. of the notebook, we will install the libraries needed for the project. The libraries installed are shown below.

```
# install necessary libraries that might not be found
!pip install transformers --quiet

# check versions of libraries we are going to use
%tensorflow_version 2.x
import os
import tensorflow
import sklearn
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib
import platform

message="        Versions        "
print("*"*len(message))
print(message)
print("*"*len(message))
print("Tensorflow version={}".format(tensorflow.__version__))
print("Keras version={}".format(tensorflow.keras.__version__))
print("Sklearn version={}".format(sklearn.__version__))
print("Numpy version={}".format(np.__version__))
print("Pandas version={}".format(pd.__version__))
print("Seaborn version={}".format(sns.__version__))
print("Matplotlib version={}".format(matplotlib.__version__))
print("Python version={}".format(platform.python_version()))
```

```
**************************
        Versions
**************************
Tensorflow version=2.5.0
Keras version=2.5.0
Sklearn version=0.22.2.post1
Numpy version=1.19.5
Pandas version=1.1.5
Seaborn version=0.11.1
Matplotlib version=3.2.2
Python version=3.7.11
```

Figure 2: Python Libraries Installed and the version output of imported libraries

2. In Section 2.4., we will load the training, and the validation dataset as Pandas Dataframes. The training dataset will be used as the training dataset for the model, while the validation dataset will serve as a test dataset.

```
# load the train and test datasets
df_train = pd.read_excel('./clean_dataset.xlsx')
df_test = pd.read_excel('./val_dataset.xlsx')

# see the top 5 companies that are populated in the dataframe
df_train.head(5)
```

	Company_ID	Company	Country	PIC	Sector	Subsector	Archetype	Valuechain	Websites
0	4137190062363536	AKSORN CHAROEN TAT ACT. CO.,LTD.	THAILAND	NaN	TMT	media	media_aggregator/distributor	Midstream	https://getlinks.co/6831
1	23248790229909752	DONGGUAN SHENGYA CLEANING APPLIAME CO.LTD	CHINA	NaN	TMT	consumer electronics	consumer electronics_distributor	Downstream	https://baike.baidu.com/item/%E4%B8%9C9
2	28486505934571008	EXIS TECH SDN. BHD.	MALAYSIA	NaN	oos	others	others	NaN	http://www.exis-tech.com/
3	38251695094669872	BEI JING ESTRABA IMPORT AND EXPORT	CHINA	NaN	NaN	NaN	NaN	NaN	NaN
4	39910921263510784	Aztech Electronics Pte Ltd	SINGAPORE	NaN	TMT	consumer electronics	consumer electronics_distributor	Downstream	https://www.aztech.com/business/about-us/

Figure 3: Loading the excel files and preview of the training dataset

Exploratory Data Analysis

1. First, we will identify the various columns in the training dataset to get a better understanding of the dataset. As shown below are the columns identified in the Dataframe.

```
[ ]  1 # see the row headers of the entire pandas dataframe first
     2 list(companies.columns)
```

```
['Company_ID',
 'Company',
 'Country',
 'PIC',
 'Sector',
 'Subsector',
 'Archetype',
 'Valuechain',
 'Websites',
 'Company Profile Information',
 'Remarks']
```

Figure 4: List of columns in the training dataset

2. Here, we identify the number of records, as well as the number of unique labels.

```
# get the total number of records in the dataframe
df_count = df_train['Company_ID'].count()

# get count of unique contries where companies are based in
df_countCountry = df_train['Country'].nunique()

# get count of total unique sectors where companies are from
df_countSector = df_train['Sector'].nunique()

# get count of total unique subseector where companies are from
df_countsubSector = df_train['Subsector'].nunique()

# get count of total unique valuechain where companies are from
df_countValuechain = df_train['Valuechain'].nunique()

# get count of total unique archetypes
df_countArchetype = df_train['Archetype'].nunique()

print('Total number of records:', df_count)
print('Total number of countries:', df_countCountry)
print('Total number of sectors:', df_countSector)
print('Total number of subsectors:', df_countsubSector)
print('Total number of valuechain:', df_countValuechain)
print('Total number of archetypes:', df_countArchetype)
```

```
Total number of records: 9600
Total number of countries: 14
Total number of sectors: 16
Total number of subsectors: 37
Total number of valuechain: 18
Total number of archetypes: 94
```

Figure 5: Number of records, and labels in interest

3. We proceed to identify the unique Archetype labels present in the dataset.

```
print('List of unique archetype:\n{}'.format(df_archetype))
```

```
List of unique archetype:
others                                2161
building_material_manufacturer         573
buildings & industrial_contractor       496
consumer discretionary distributor      358
cni_service providers                   279
                                       ...
MIDSTREAM                                 2
tisp - tower                              2
tisp - fiber cable                        2
building material_manufacturer            1
metals and mining                         1
Name: Archetype, Length: 94, dtype: int64
```

Figure 6: Unique archetype breakdown

4. Here, we drop the unnecessary columns from the dataset, so that they do not interfere with the EDA process.

```
# declare the list of the row names that are redundant
rows_to_drop = ['Company_ID', 'PIC', 'Websites', 'Remarks']

# use a conditional expression to filter out those rows
df_filteredCompanies = df_train.drop(labels=rows_to_drop, axis=1)

df_filteredCompanies
```

	Company	Country	Sector	Subsector	Archetype	Valuechain	Company Profile Information
0	AKSORN CHAROEN TAT ACT. CO.,LTD.	THAILAND	TMT	media	media_aggregator/distributor	Midstream	For over 80 years of experience in creating an...
1	DONGGUAN SHENGYA CLEANING APPLIAME CO.LTD	CHINA	TMT	consumer electronics	consumer electronics_distributor	Downstream	Yatai's main products cover various cleaning m...
2	EXIS TECH SDN. BHD.	MALAYSIA	oos	others	others	NaN	In the beginning, it started off by providing ...
3	BEI JING ESTRABA IMPORT AND EXPORT	CHINA	NaN	NaN	NaN	NaN	NaN
4	Aztech Electronics Pte Ltd	SINGAPORE	TMT	consumer electronics	consumer electronics_distributor	Downstream	Being a turnkey, one-stop integrated solutions...
...	...	...	...	...	...	...	...
9595	SOONBEE INVESTMENT HOLDINGS PTE. LT D	SINGAPORE	OOS	others	others	NaN	SOONBEE INVESTMENT HOLDINGS is an ACRA-registe...
9596	SICHUAN CHINA RAILWAY WENRUI REAL E	CHINA	NaN	NaN	NaN	NaN	NaN
9597	GROUP INDUSTRIES PTE LTD	SINGAPORE	CNI	building_material	building_material_manufacturer	MIDSTREAM	Group Industries Pte Ltd was founded in 1986. ...
9598	SIAM NISSAN EASTERN CO.,LTD.	THAILAND	IND	auto & mec	auto dealer	Trader	Auto Dealership - Retail
9599	FAT INVESTMENT HONG KONG LIMITED	HONG KONG	NaN	NaN	NaN	NaN	NaN

Figure 7: New training dataset after dropping columns

5. Records with all NaN values are extracted, shown, and discarded, as they cannot be used in the training process.

```
# find all the rows with nan data in sector, subsector, archetype and valuechain
cols_to_check = ['Sector', 'Subsector', 'Archetype', 'Valuechain', 'Company Profile Information']
empty = df_filteredCompanies[df_filteredCompanies[cols_to_check].isnull().all(1)]

empty
```

	Company	Country	Sector	Subsector	Archetype	Valuechain	Company Profile Information
3	BEI JING ESTRABA IMPORT AND EXPORT	CHINA	NaN	NaN	NaN	NaN	NaN
12	CHANGTU COUNTRY LONGXING FERTILIZER CO.,LTD	CHINA	NaN	NaN	NaN	NaN	NaN
28	ZIBOBOSHANHONGLIWEI MOTOR CO.,LTD	CHINA	NaN	NaN	NaN	NaN	NaN
48	BEIJING DUO MEIDUO SHIYOU PRODUCTS SALES CO., ...	CHINA	NaN	NaN	NaN	NaN	NaN
63	TRUSVEST SDN. BHD.	MALAYSIA	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...
9576	REAL CHARM INVESTMENT LIMITED	HONG KONG	NaN	NaN	NaN	NaN	NaN
9585	JIJIN INTERNATIONAL TRADE (SHANGHAI)	CHINA	NaN	NaN	NaN	NaN	NaN
9592	QITAI TIANSHAN CEMENT CO., LTD	CHINA	NaN	NaN	NaN	NaN	NaN
9596	SICHUAN CHINA RAILWAY WENRUI REAL E	CHINA	NaN	NaN	NaN	NaN	NaN
9599	FAT INVESTMENT HONG KONG LIMITED	HONG KONG	NaN	NaN	NaN	NaN	NaN

1149 rows × 7 columns

Figure 8: Records with NaN values

6. The records with valid data are then kept.

```
# now we get the dataset that are valid
df_valid = pd.concat([df_filteredCompanies, empty, empty]).drop_duplicates(keep=False)

df_valid
```

	Company	Country	Sector	Subsector	Archetype	Valuechain	Company Profile Information
0	AKSORN CHAROEN TAT ACT. CO.,LTD.	THAILAND	TMT	media	media_aggregator/distributor	Midstream	For over 80 years of experience in creating an...
1	DONGGUAN SHENGYA CLEANING APPLIAME CO.LTD	CHINA	TMT	consumer electronics	consumer electronics_distributor	Downstream	Yatai's main products cover various cleaning m...
2	EXIS TECH SDN. BHD.	MALAYSIA	oos	others	others	NaN	In the beginning, it started off by providing ...
4	Aztech Electronics Pte Ltd	SINGAPORE	TMT	consumer electronics	consumer electronics_distributor	Downstream	Being a turnkey, one-stop integrated solutions...
5	TONGDUN INTERNATIONAL PTE LTD	SINGAPORE	tmt	it_services	it_services	midstream	Tongdun Technology is a professional third-par...
...	...	...	...	...	...	...	...
9593	BANGKOK PATANA SCHOOL	THAILAND	CG	retail n distribution	consumer discretionary distributor	Downstream	Bangkok Patana School is Thailand's original B...
9594	INDRATI LINES PTE LTD	SINGAPORE	OOS	others	others	NaN	Indra is one of the leading global technology ...
9595	SOONBEE INVESTMENT HOLDINGS PTE. LT D	SINGAPORE	OOS	others	others	NaN	SOONBEE INVESTMENT HOLDINGS is an ACRA-registe...
9597	GROUP INDUSTRIES PTE LTD	SINGAPORE	CNI	building_material	building_material_manufacturer	MIDSTREAM	Group Industries Pte Ltd was founded in 1986. ...
9598	SIAM NISSAN EASTERN CO.,LTD.	THAILAND	IND	auto & mec	auto dealer	Trader	Auto Dealership - Retail

8447 rows × 7 columns

Figure 9: Dataframe with the valid results

7. Here, we will turn the numbers seen in the EDA into graphs, so that it is easier to spot the trend. Shown below is a bar chart of the breakdown of labels in Sector. It seems that OOS, CNI and REH make up the majority of the dataset.

   We also plotted a pie chart of the countries found in the dataset. We can see that most of the portions comes from Thai companies, followed by China and Malaysia.

```
sns.set_style('darkgrid')
plt_dims = (30, 13)
fig, ax = plt.subplots(figsize=plt_dims)

# plot a barplot to see number of companies that belongs to specific sectors
sns.countplot(x="Sector", data=df_valid, ax=ax)
plt.show()
```

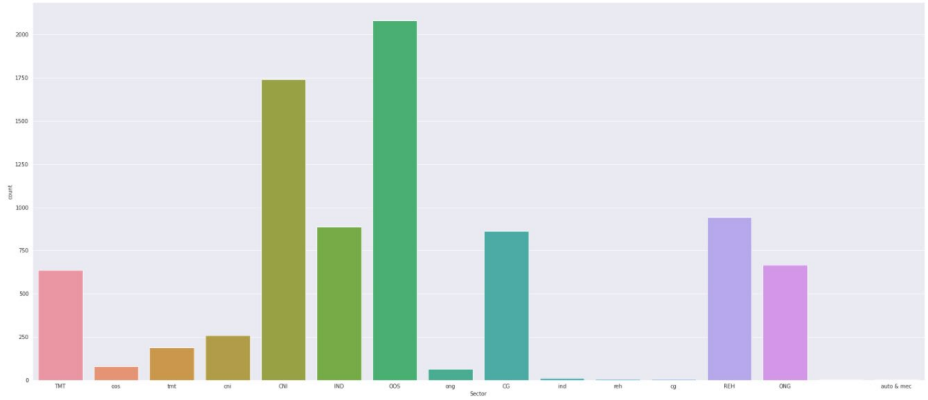


Figure 10: Bar Chart of the various Sector counts

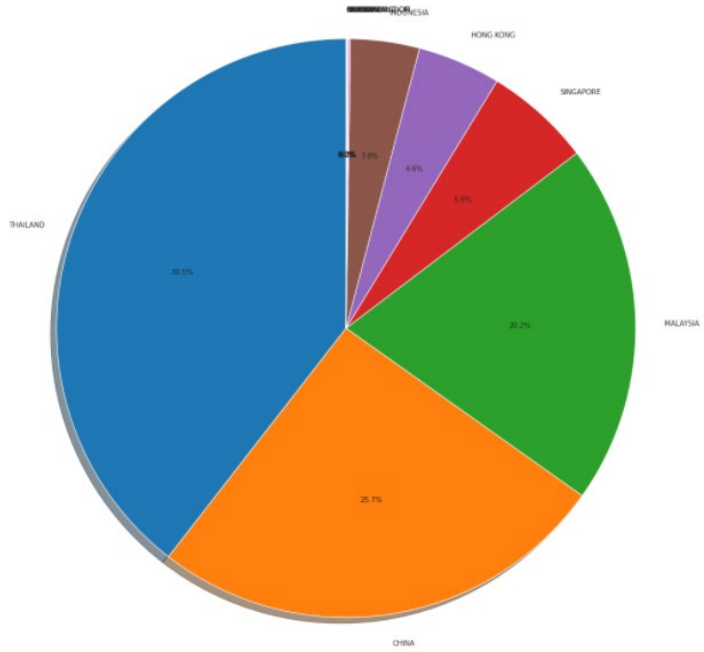


Figure 11: Pie chart breakdown of the countries the companies originate from

8. We would like to see examples of how the worded company profile description looks like. Therefore, we printed the 1st 10 companies text description for analysis.

```
# configurate pandas dataframe to let us see the entire company description IN FULL
pd.set_option('display.max_colwidth', None)

# get the 1st 10 results and observe
df_valid.loc[0:10,'Company Profile Information']
```

```
0
For over 80 years of experience in creating and developing high-quality learning materials has enabled us to provide world-class educatio
nal innovation to meet the needs of all teachers, students, institutions and educational authorities.
1
Yatai's main products cover various cleaning machinery, cleaning agents, cleaning tools, stone maintenance and other cleaning products; c
leaning solutions and services include product technical consultation, product customization, employee training, maintenance and so on.\n
2
In the beginning, it started off by providing technical support for test handlers, then moving into module design and production. Its fir
st, in-house designed, full-fledged handler was introduced in 2008.\nSince then, the company has designed and produced a wide range of tu
rret and pick-and-place solutions for its customers all over the world.
4                                                    Being a turnkey, one-stop integrated solutions provider based
in Singapore, Aztech is equipped with state-of-the-art equipment, R&D, design, manufacturing and packaging capabilities to deliver a seam
less, unified experience. Each and every time. Always striving towards the edge of technology for more than 34 years, we have been buildi
ng capabilities to serve clients' manufacturing needs, including the consumer electronics, telecommunications, healthtech, LED lighting,
automotive and technology start-up market segments.
5     Tongdun Technology is a professional third-party intelligent risk management and decision-making service provider headquartered in
Hangzhou, Zhejiang. By integrating artificial intelligence into business scenarios, Tongdun Technology offers solutions in intelligent us
er analysis, intelligent risk management, intelligent antifraud and intelligent operation to clients from various industries including fi
nancial industry, internet business, logistics, healthcare, retail, smart cities and government bodies. Over 10,000 corporate clients hav
e chosen Tongdun Technology's products and services
6
For over 30 years, EMASTIGA has been servicing the construction and the oil and gas industries with distinction.We try our hardest to cre
ate win-win situations and value-add to our clients with every transaction.
7
Sales of hydraulic equipment and parts
8
For Excelkos, it all started with a far-sighted vision to deliver quality industrial products and solutions to meet the increasing demand
s of various industries.  The vision became a reality in 1988 when the Excelkos Group was established to distribute and provide quality r
ubber industrial chemicals.
9
Kum Eng Huat is the authorised dealer for Osram and Philips, as well as a trusted supplier of lighting solutions for major projects with
the local government.
10
Wholesale of other building_materials, not elsewhere classified
Name: Company Profile Information, dtype: object
```

Figure 12: First 10 company text descriptions

Data Pre-processing

1. To ensure that there are no escape characters such as (‘\n’) and (‘\t’), we will first replace those with a space (‘ ‘) to ensure that it does not interfere with the model learning process later on.

   We also used a distribution plot to see the most frequent word length among the entire dataset. From the frequency distribution plot, we can assume that most company descriptions have around 700 to 800 word descriptions.

4.1. Removing \n and \t

Now, we will like to standardize all the paragraphs such that the are homogenous, before we tokenize the paragraph

```
# get rid of the \n found in the respective descriptions
df_valid = df_valid.replace(["\n", "\t"], ' ', regex=True)
```

```
# now plot a distribution plot to see the word length distribution
sns.distplot(df_valid["length"], kde=True)
plt.title('Word Length Distribution')
plt.show()
```

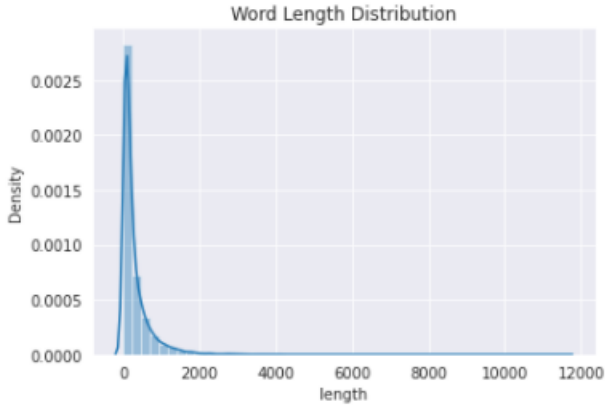


Figure 13: Removing special characters and getting the word length distribution

2. We then proceed on to populate NaN cells with spaces to proceed on with text tokenization later on.

```
# fill na with space instead of others
df_valid.fillna(" " ,inplace=True)
df_valid
```

	Company	Country	Sector	Subsector	Archetype	Valuechain	Company Profile Information	length
0	AKSORN CHAROEN TAT ACT. CO.,LTD.	THAILAND	TMT	media	media_aggregator/distributor	Midstream	For over 80 years of experience in creating and developing high-quality learning materials has enabled us to provide world-class educational innovation to meet the needs of all teachers, students, institutions and educational authorities.	238
1	DONGGUAN SHENGYA CLEANING APPLIAME CO.LTD	CHINA	TMT	consumer electronics	consumer electronics_distributor	Downstream	Yatai's main products cover various cleaning machinery, cleaning agents, cleaning tools, stone maintenance and other cleaning products; cleaning solutions and services include product technical consultation, product customization, employee training, maintenance and so on.	273
2	EXIS TECH SDN. BHD.	MALAYSIA	oos	others	others		In the beginning, it started off by providing technical support for test handlers, then moving into module design and production. Its first, in-house designed, full-fledged handler was introduced in 2008. Since then, the company has designed and produced a wide range of turret and pick-and-place solutions for its customers all over the world.	344

Figure 14: A snippet off populating NaN cells to space

3. Section 4.5. of the notebook retrieves the 4 labels, namely sector, subsector, archetype and valuechain, and assign unique labels to the respective company description. These unique labels will be used later as the training labels when training the model later.

4.5. Assigning tags

In this section, we will be assigning tags to every row, so that we can make use of the given keywords as the training dataset labels.

```
# Programmatically assign tags to each definition
sector_keywords = pd.read_excel('./sector_master_definition.xlsx')
df_keywords = sector_keywords[['Sector', 'Subsector', 'Archetype', 'Value Chain', 'Sector Keywords']]

# capitalise all tags
df_keywords['Value Chain'] = df_keywords['Value Chain'].str.upper()
df_keywords.fillna(' ', inplace=True)
df_keywords['Sector Keywords'] = df_keywords['Sector Keywords'].str.upper()
df_keywords['Sector Keywords'].replace(' ', '[]', inplace=True)

# save unique tags, sorted for consistency across runs
sector = np.sort(df_keywords['Sector'].unique())
subsector = np.sort(df_keywords['Subsector'].unique())
archetype = np.sort(df_keywords['Archetype'].unique())
valuechain = np.sort(df_keywords['Value Chain'].unique())
print(len(sector), len(subsector), len(archetype), len(valuechain))
tag_counts = [len(sector), len(subsector), len(archetype), len(valuechain)]

# assign number tag list to each row
taglist = []
for index, row in df_keywords.iterrows():
    temp = []

    temp.append(np.where(sector == row['Sector'])[0][0])
    temp.append(np.where(subsector == row['Subsector'])[0][0])
    temp.append(np.where(archetype == row['Archetype'])[0][0])
    temp.append(np.where(valuechain == row['Value Chain'])[0][0])

    taglist.append(temp)

# assign completed taglist to column in dataframe
df_keywords['list_tag'] = taglist
```

```
7 32 92 9
```

Figure 15: Assigning tags to each company description

4. We redid data pre-processing for the training dataset for homogeneity.

```
# process data for homogenity
df_valid['Valuechain'] = df_valid['Valuechain'].str.split().str.join(' ')
df_valid['Valuechain'] = df_valid['Valuechain'].str.upper()
df_valid['Sector'] = df_valid['Sector'].str.upper()
df_valid['Valuechain'].replace('', ' ', inplace=True)

taglist_df = []
# process tags for records
for index, row in df_valid.iterrows():
    temp = []

    try: # for error handling
        temp.append(np.where(sector == row['Sector'])[0][0])
        temp.append(np.where(subsector == row['Subsector'])[0][0])
        temp.append(np.where(archetype == row['Archetype'])[0][0])
        temp.append(np.where(valuechain == row['Valuechain'])[0][0])

        taglist_df.append(temp)
    except:
        # drop data if not valid
        df_valid.drop(index, inplace=True)


df_valid['list_tag'] = taglist_df

df_valid.shape
```

```
(8423, 9)
```

Figure 16: Pre-Processing df_valid

5. Similar to point 4, we also did a similar data pre-processing to the test dataset under Section 4.7. in the notebook.

4.7. Preprocessing data for validation testset.

We will perform data preprocessing for the validation testset.

```
df_test.replace('NAN', np.NaN, inplace=True)

# drop unnecessary columns
df_test.drop(rows_to_drop, axis=1, inplace=True)

# replace newline characters in validation data
df_test = df_test = df_test.replace('\n', ' ', regex=True)

# fill in NAN values in validation data
df_test.fillna(' ', inplace=True)

# change dtype of validation data columns
for i in columns_to_convert:
    df_valid[i] = df_valid[i].astype(str)

# add tags to validation data
taglist_df = []
# process tags for records
for index, row in df_test.iterrows():
    temp = []

    try: # for error handling
        temp.append(np.where(sector == row['Sector'])[0][0])
        temp.append(np.where(subsector == row['Subsector'])[0][0])
        temp.append(np.where(archetype == row['Archetype'])[0][0])
        temp.append(np.where(valuechain == row['Valuechain'])[0][0])

        taglist_df.append(temp)
    except Exception as e:
        # drop data if not valid
        print(row.name, e, '\n')
        df_test.drop(index, inplace=True)

df_test['list_tag'] = taglist_df
```

Figure 17: Data pre-processing for the test dataset

6. We went on to ensure that the dtypes for the respective columns are in string and not other dtypes.

   We made use of HuggingFace tokenizer function. We first load BertConfig, which is the configuration of the Bert model, to bert-base-uncased to instantiate a BERT model according to the specified arguments. We then modify one of the parameter, changing the output hidden states to be False. We then proceed on to load the BERT tokenizer Fast, which essentially tokenizer faster than its BERT tokenizer due to the code being implemented as a “fast” BERT tokenizer.

```
# we will have to ensure all the dtype of the respective columns are in string and not float for spacy to handle properly, so now we will
attempt to convert all into strings
columns_to_convert = ['Sector', 'Subsector', 'Archetype', 'Valuechain', 'Company Profile Information']

for i in columns_to_convert:
  df_valid[i] = df_valid[i].astype(str)
```

```
# instead of using spacy tokenizer, we will make use of the preprocessing and tokenizer in the python package from huggingface
from transformers import TFBertModel, BertConfig, BertTokenizerFast

# use the uncased bert model
config = BertConfig.from_pretrained('bert-base-uncased')

# set the output_hidden_state to false since we have 4 different classes with their respective labels AND we want to define our own hidde
n layers
config.output_hidden_states = False

# load the BERT tokenizer first. BertTokenizerFast tokenize faster than BertTokenizer as it uses Rust
tokenizer = BertTokenizerFast.from_pretrained(pretrained_model_name_or_path='bert-base-uncased', config=config)
```

Figure 18: Loading BERT tokenizer from HuggingFace

7. We will proceed to randomise our dataset for more randomness.

```
# randomise dataset first here
df_train_rand = df_valid.sample(frac=1).reset_index(drop=True)
df_test_rand = df_test.sample(frac=1).reset_index(drop=True)
```

```
# drop the length column section from the train_rand df since its not necessary
df_train_rand.drop(['length'], axis=1, inplace=True)

# define one hot encode function
def one_hot(arr, n_cat):
    output = []
    for n in arr:
        result = np.zeros(n_cat)
        result[n] = 1

        output.append(result)

    return np.array(output, dtype=int)
```

Figure 19: Randomise dataset and further dropping unnecessary columns

8. We will now split the training dataset (df_train_rand) and test dataset (df_test_rand) into X_train and y_train, X_test and y_test respectively. The y labels are the label for the respective company descriptions.

```
# split into data and labels
X_train = tokenizer(text=df_train_rand['Company Profile Information'].to_list(),
                    add_special_tokens=True, # add special tokens like [SEP] and others
                    max_length=100,  # this is the max length of the sentence-to-be-token
                    truncation=True,
                    padding=True,
                    return_tensors='tf', # to return it as tf tensors to feed into keras API
                    return_token_type_ids=False,
                    return_attention_mask=True, # generate attention mask
                    verbose=True)

X_test = tokenizer(text=df_test_rand['Company Profile Information'].to_list(),
                   add_special_tokens=True,
                   max_length=100,  # we use the max length that BERT can handle
                   truncation=True,
                   padding=True,
                   return_tensors='tf',
                   return_token_type_ids=False,
                   return_attention_mask=True,
                   verbose=True)

y_train = np.array(list(df_train_rand['list_tag']))
y_test = np.array(list(df_test_rand['list_tag']))
```

Figure 20: Splitting train and test dataset into training data and the labels

- The add_special_tokens param is set to True in order to add special BERT tokens like [CLS], [SEP] and [PAD] to the 'tokenized' encodings.
- The max_length defines the length of the tokenized text.
- Truncation ensure that the max_length is strictly adhered, whereby longer sentence are truncated to max_length.
- Padding is added such that, say a length of a description is less than 100 words. After doing tokenizing, it adds 0 at the back such that the entire length is eventually 100.

9. To have a better understanding of what happens after tokenizing, we can have a look at the input_id, how it looks like when the sentence is tokenized, and the attention_mask, which tells the model which tokens should be attended to and which tokens should not.

```
X_train["input_ids"][0]
```

```
<tf.Tensor: shape=(100,), dtype=int32, numpy=
array([  101, 17712,  2121,  7300,  4021,  3534, 17371,  2078,  1012,
        1038, 14945,  1012,  2001,  2631,  1999,  2384,  1012,  1996,
        2194,  1005,  1055,  2240,  1997,  2449,  2950,  4346,  2968,
        2578,  2006,  1037,  3206,  2030,  7408,  3978,  1012,   102,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0],     dtype=int32)>
```

```
# see how the 1st sentence of the training dataset is being encoded
tokenizer.decode(X_train["input_ids"][0])
```

```
"[CLS] aker solutions asia pacific sdn. bhd. was founded in 2005. the company's line of business includes providing management services o
n a contract or fee basis. [SEP] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [P
AD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD]"
```

```
X_train['attention_mask']
```

```
<tf.Tensor: shape=(8423, 100), dtype=int32, numpy=
array([[1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 1, 1, 1],
       ...,
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 1, 1, 1]], dtype=int32)>
```

Figure 21: (From top to bottom) The first company description input_id, tokenized form and overall atention mask for X_train

10. Finally, we will one hot encode the labels.

```
label_names = ['sector', 'subsector', 'archetype', 'valuechain']

y_train_multi = {label_names[i] : tf.convert_to_tensor(one_hot(y_train[:, i], tag_counts[i]), dtype=tf.int16) for i in range(4)}
y_test_multi = {label_names[i] : tf.convert_to_tensor(one_hot(y_test[:, i], tag_counts[i]), dtype=tf.int16) for i in range(4)}
```

Figure 22: One-hot encoding

Model Training

1. We will now do a quick check on the Keras version we are using, and then create our multi ouput branches for namely Sector, Subsector, Archetype and Valuechain.

   In our multi_branch output, we will load the first layer from the BERT transformer model as we only need the MainLayer. Then we will define the pooled layers for the respective branches. We set training to False so that we can freeze the pretrained layers. We eventually used a softmax activation function. We will slot in the input_ids and the attention mask. The attention_mask essentially tells the model that we should not focus attention on [PAD] tokens, which is essential for classification problems.

```
import keras

print('--- Version Checking ---')
print("Keras:", keras.__version__)
```

```
--- Version Checking ---
Keras: 2.5.0
```

```
from tensorflow.keras.regularizers import l2
from tensorflow.keras.layers import Conv1D, Dense, Dropout, MaxPooling1D, GlobalAveragePooling1D, Input, Lambda, Embedding
from tensorflow.keras.layers import ReLU, BatchNormalization, Activation
from tensorflow.keras import Sequential
from tensorflow.keras.models import Model
from tensorflow.keras.initializers import TruncatedNormal

# create mulit output model
def multi_branch(inputs, name, output_dim, counter, transformer_model):

  # Load the Transformers BERT model as a layer in a Keras model
  bert = transformer_model.layers[0] # get only the top head
  bert_model = bert(inputs)[1]
  dropout = Dropout(config.hidden_dropout_prob, name="pooled_output_{}".format(counter+1))
  pooled_output = dropout(bert_model, training=False) # freeze pretrained layers

  # build the model output
  output = Dense(output_dim, kernel_initializer=TruncatedNormal(stddev=config.initializer_range), name=name, activation='softmax')(pooled
_output)

  return output
```

```
# create the multi output branches
def create_model(labels, output_dim):
  # input
  input_ids = Input(shape=(100,), name='input_ids', dtype='int32')
  attention_mask = Input(shape=(100,), name='attention_mask', dtype='int32')
  inputs = {'input_ids': input_ids, 'attention_mask': attention_mask}

  # preload the BERT transformer model
  transformer_model = TFBertModel.from_pretrained('bert-base-uncased', config=config)

  # create the respective branches for the outputs
  branches = []

  for i in range(len(labels)):
    branches.append(multi_branch(inputs, labels[i], output_dim[i], i, transformer_model))

  # fit the entire model into keras Model class
  model = Model(inputs=inputs, outputs=branches, name='company_classification_model')

  return model
```

Figure 23: Creating multi-branches for the categories classification

2. Once we have created the branches, we will create the model, and define categorical_crossentrophy as our loss, and Adam as the optimizer. We will be using metrics such as accuracy, precision, recall, area under curve to observe our model performance.

   There are 2 sets of codes, one catered for TPU users, and another for GPU users. For those who have earlier ran using GPU as the runtime, the model will directly be created in the else loop. For TPU users, the code will land in the first conditional action, whereby we will create the model all in the strategy_scope, so that we can distribute the workload equally to the TPU devices.

```
from tensorflow.keras.metrics import Precision, Recall, AUC
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.callbacks import EarlyStopping

# for tpu
if tpu_usage:
  with strategy.scope():
    model = create_model(label_names, tag_counts)
    opt = Adam(learning_rate=5e-05, epsilon=1e-08, decay=0.01, clipnorm=1.0)
    losses = {i : 'categorical_crossentropy' for i in label_names}
    model.compile(loss=losses, optimizer=opt, metrics=['accuracy', Precision(), Recall(), AUC(name='auc_precision_recall', num_thresholds
=10000)])

# for gpu
else:
  # create the model
  model = create_model(label_names, tag_counts)
  opt = Adam(learning_rate=5e-05, epsilon=1e-08, decay=0.01, clipnorm=1.0)
  losses = {i : 'categorical_crossentropy' for i in label_names}
  model.compile(loss=losses, optimizer=opt, metrics=['accuracy', Precision(), Recall(), AUC(name='auc_precision_recall', num_thresholds=1
0000)])

# print model summary
model.summary()
```

Figure 24: Code to create the model

3. Once done creating the model, we see the summary of the model, and plot out the model as a diagram.

```
Model: "company_classification_model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
attention_mask (InputLayer)     [(None, 100)]        0
__________________________________________________________________________________________________
input_ids (InputLayer)          [(None, 100)]        0
__________________________________________________________________________________________________
bert (TFBertMainLayer)          TFBaseModelOutputWit 109482240   attention_mask[0][0]
                                                                 input_ids[0][0]
                                                                 attention_mask[0][0]
                                                                 input_ids[0][0]
                                                                 attention_mask[0][0]
                                                                 input_ids[0][0]
                                                                 attention_mask[0][0]
                                                                 input_ids[0][0]
__________________________________________________________________________________________________
pooled_output_1 (Dropout)       (None, 768)          0           bert[0][1]
__________________________________________________________________________________________________
pooled_output_2 (Dropout)       (None, 768)          0           bert[1][1]
__________________________________________________________________________________________________
pooled_output_3 (Dropout)       (None, 768)          0           bert[2][1]
__________________________________________________________________________________________________
pooled_output_4 (Dropout)       (None, 768)          0           bert[3][1]
__________________________________________________________________________________________________
sector (Dense)                  (None, 7)            5383        pooled_output_1[0][0]
__________________________________________________________________________________________________
subsector (Dense)               (None, 32)           24608       pooled_output_2[0][0]
__________________________________________________________________________________________________
archetype (Dense)               (None, 92)           70748       pooled_output_3[0][0]
__________________________________________________________________________________________________
valuechain (Dense)              (None, 9)            6921        pooled_output_4[0][0]
==================================================================================================
Total params: 109,589,900
Trainable params: 109,589,900
Non-trainable params: 0
```

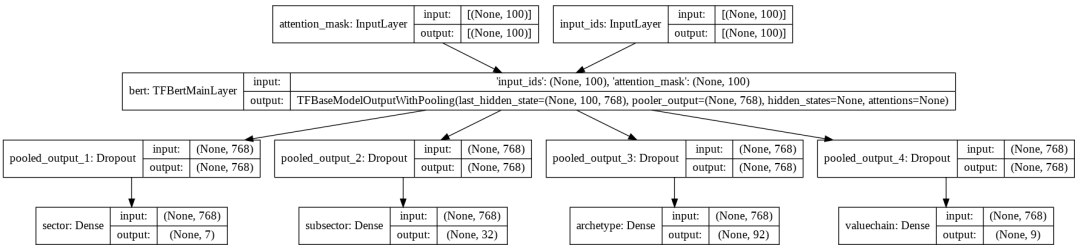


Figure 25: Model summary and diagram

4. We then allow the training to proceed. Here, we have a validation split of 0.2, where we partition 20% data from the train dataset for validation. This is done to monitor overfitting.

```
from tensorflow.keras.metrics import Precision, Recall, AUC, CategoricalAccuracy
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.callbacks import EarlyStopping
from timeit import default_timer as timer

# setting optimizer - based on recommended vals
opt = Adam(learning_rate=5e-05, epsilon=1e-08, decay=0.01, clipnorm=1.0)

# defining loss
losses = {i : 'categorical_crossentropy' for i in label_names}

# train model
start = timer()
history = model.fit({'input_ids': X_train['input_ids'], 'attention_mask': X_train['attention_mask']}, y_train_multi, validation_split=0.
2, epochs=15, batch_size=16)
end = timer()
print("Total time taken for execution: ", end-start, "s")
```

Figure 26: Start of model training

Model Evaluation

1. When our model training has been completed, we will use our X_test dataset to evaluate the model. This is a breakdown of the loss, accuracy, precision, recall, PR AUC and F1 Score of the model for the 4 categories.

```
44/44 [==============================] - 37s 837ms/step - loss: 4.0287 - sector_loss: 0.3630 - subsector_loss: 0.9883 - archetype_loss:
1.8817 - valuechain_loss: 0.7958 - sector_accuracy: 0.8936 - sector_precision: 0.9037 - sector_recall: 0.8850 - sector_auc_precision_reca
ll: 0.9851 - subsector_accuracy: 0.7336 - subsector_precision: 0.8395 - subsector_recall: 0.6014 - subsector_auc_precision_recall: 0.9782
- archetype_accuracy: 0.5179 - archetype_precision: 0.8336 - archetype_recall: 0.3150 - archetype_auc_precision_recall: 0.9704 - valuecha
in_accuracy: 0.7429 - valuechain_precision: 0.8201 - valuechain_recall: 0.6250 - valuechain_auc_precision_recall: 0.9592
```

Label	Loss	Accuracy	Precision	Recall	PR AUC	F1 Score
sector	0.363	89.357%	0.734	0.518	0.743	0.304
subsector	0.988	90.372%	0.839	0.834	0.820	0.418
archetype	1.882	88.500%	0.601	0.315	0.625	0.207
valuechain	0.796	98.505%	0.978	0.970	0.959	0.487

Figure 27: Evaluated score of the model

2. We decided to further analyse the details of the result, and created a custom metric analysis based on prediction made using X_test datasets. Here, we create a breakdown of the accuracy of the Sector and Subsector. The correct rows are the number of correctly classified rows. This means that for those classification with 0 correctly classified rows, all the results are wrongly classified.

Label	Accuracy	Correct Rows	% Correct Rows	Precision	Recall
CG	78.0%	137	68.5%	0.872	0.780
> agribusiness	88.89%	8	4.0%	1.000	0.889
> animal protein	78.95%	15	7.5%	1.000	0.789
> fmcg	73.68%	28	14.0%	0.622	0.737
> retail n distribution	66.42%	86	43.0%	0.795	0.664
CNI	95.5%	142	71.0%	0.864	0.955
> building_material	93.48%	86	43.0%	0.835	0.935
> buildings & industrial	91.11%	41	20.5%	0.477	0.911
> cni_equipment suppliers	0.00%	0	0.0%	0.000	0.000
> cni_service providers	11.43%	4	2.0%	0.667	0.114
> infrastructure	0.00%	0	0.0%	0.000	0.000
> utilities	91.67%	11	5.5%	0.550	0.917
IND	88.0%	169	84.5%	0.926	0.880
> auto & mec	90.65%	97	48.5%	0.907	0.907
> diversified	69.57%	16	8.0%	0.842	0.696
> metals and mining	80.00%	56	28.0%	0.889	0.800
ONG	94.5%	159	79.5%	0.917	0.945
> ioc/noc	0.00%	0	0.0%	0.000	0.000
> o&g	87.80%	36	18.0%	0.706	0.878
> o&g_service providers/contractors	80.56%	28	14.0%	0.659	0.806
> ong_traders	0.00%	0	0.0%	0.000	0.000
> petrochemical	93.14%	95	47.5%	0.848	0.931
OOS	87.0%	173	86.5%	0.773	0.870
> others	88.00%	173	86.5%	0.755	0.880
REH	91.5%	118	59.0%	0.953	0.915
> commercial	0.00%	0	0.0%	0.000	0.000
> hotels and accommodation	83.33%	15	7.5%	0.789	0.833
> industrial	0.00%	0	0.0%	0.000	0.000
> mixed	0.00%	0	0.0%	0.000	0.000
> residential	88.24%	60	30.0%	0.550	0.882
> restaurants, catering & services	91.67%	43	21.5%	0.759	0.917
TMT	91.0%	119	59.5%	0.973	0.910
> consumer electronics	96.15%	73	36.5%	0.652	0.962
> digital_business	0.00%	0	0.0%	0.000	0.000
> it_services	96.77%	30	15.0%	0.612	0.968
> media	0.00%	0	0.0%	0.000	0.000
> semiconductor	3.45%	1	0.5%	1.000	0.034
> telecommunication	65.22%	15	7.5%	0.600	0.652

Figure 28: Detailed breakdown of classification results for Sector and Sub-sector

3. We also plotted a heatmap for sector and subsector for a easier identification of the sectors and subsectors mainly classified correctly and wrongly.

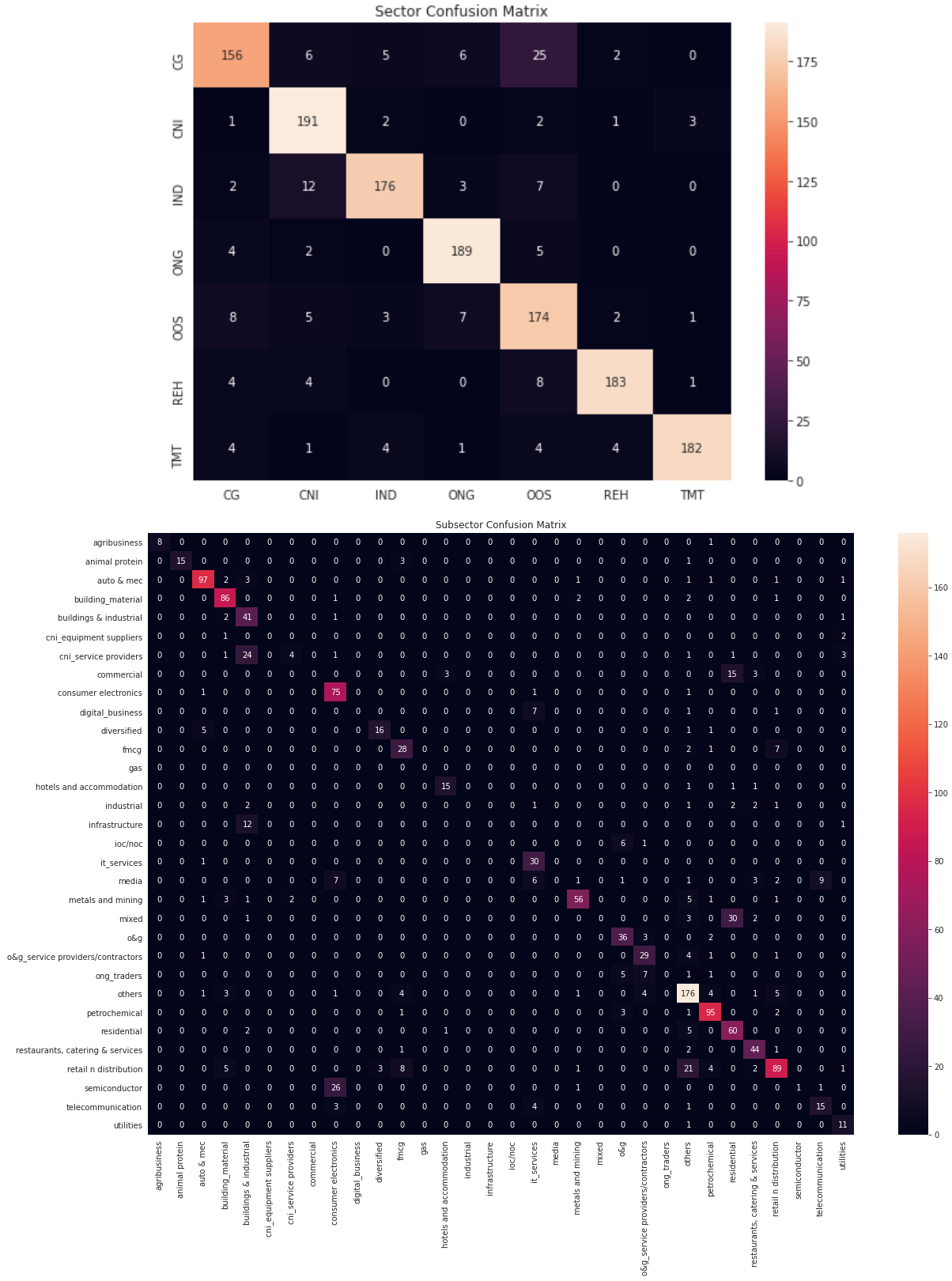
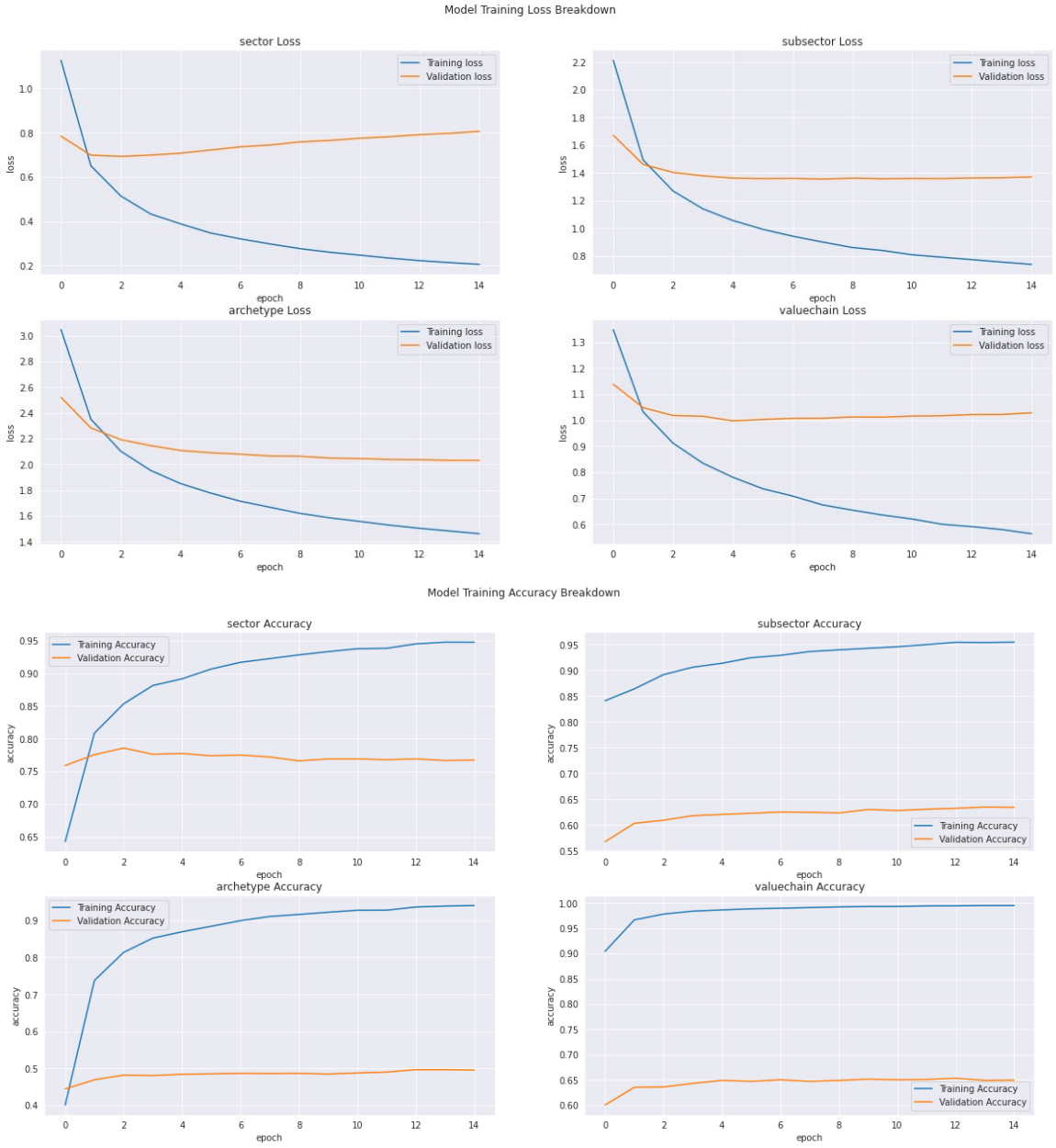


Figure 29: Heatmap for Sector and Subsector classification

4. We also plotted graphs for the sector loss, accuracy and the respective metrics as defined earlier on.



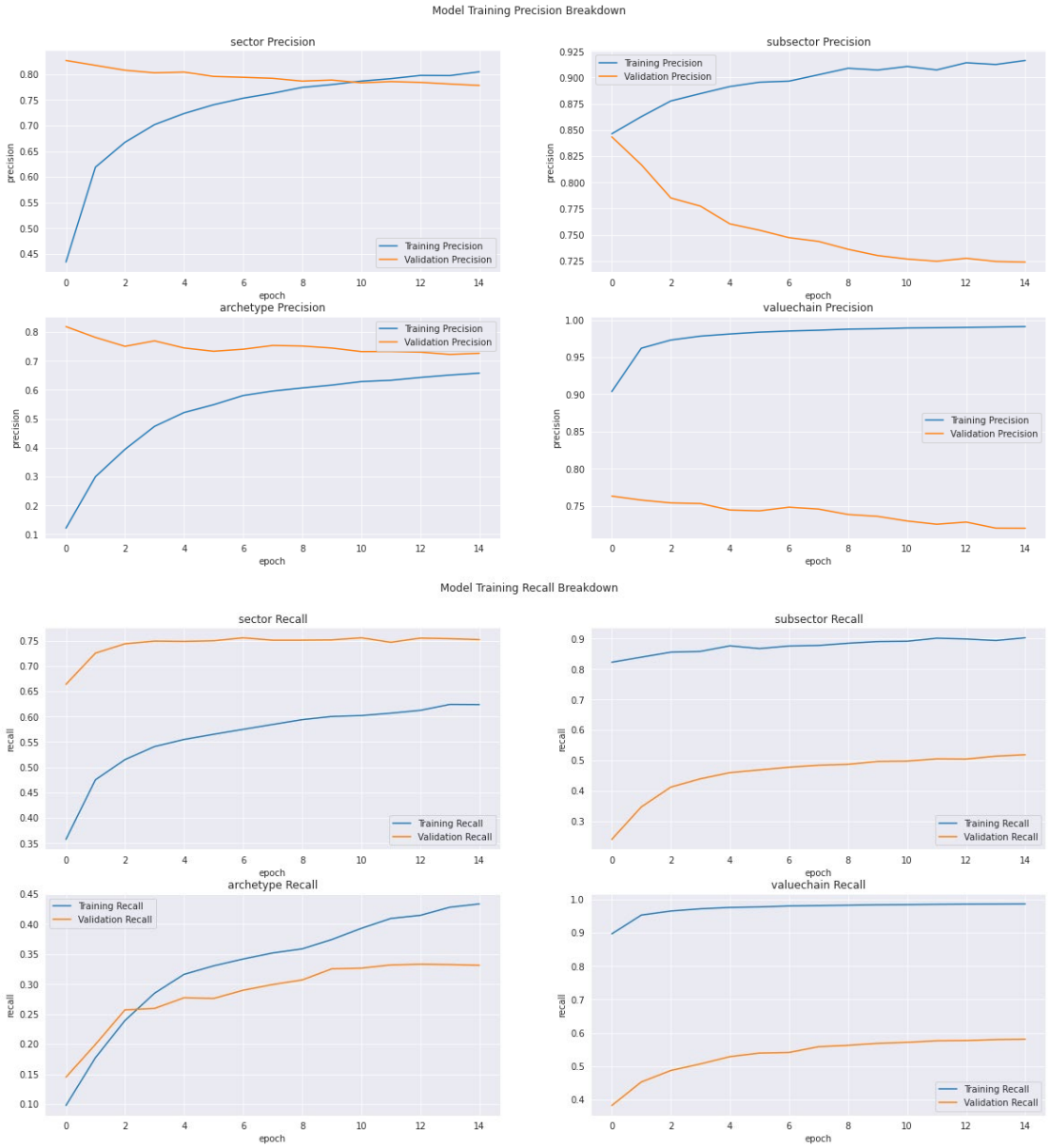


Figure 30: Graphs for the model metrics