Phase 2 MLP Multilabel Model Training Guideline

ST1506: DSDA FINAL YEAR PROJECT

COMPANY PROFILING TEXT MINING

August 2021

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 our MLP Multi-label model in this specific context and possibly improve from the current work.

Table of Contents

Breakdown of Notebook Contents

Data Importing

```
# suppress future warnings
import warnings
warnings.filterwarnings('ignore')
```

```
# install necessary libraries that might not be found
!pip install tensorflow_addons
!pip install -U spacy
!python -m spacy validate
!pip install -U pip setuptools wheel
!pip install -U spacy[cuda110,transformers,lookups]
!python -m spacy download en_core_web_lg

# 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 spacy
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("SpaCy version={}".format(spacy.__version__))
print("Python version={}".format(platform.python_version()))
```

Here, we install all the libraries that we might need, and import them to check their version numbers.

```
# importing necessary modules for this project
import tensorflow as tf

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import string
import spacy

# activate the GPU to run spaCy with GPU
spacy.prefer_gpu()

%matplotlib inline
```

We import the libraries under their respective short forms for use in other sections of the notebook.

```python
# use pandas to read the excel file and populate it in a pandas dataframe
companies = pd.read_excel('./clean_dataset.xlsx')

# see the top 10 companies that are populated in the dataframe
companies.head(10)
```

	Company_ID	Company	Country	PIC	Sector	Subsector	Archetype	Valu
0	4137190062363536	AKSORN CHAROEN TAT ACT. CO.,LTD.	THAILAND	NaN	TMT	media	media_aggregator/distributor	Midst
1	23248790229909752	DONGGUAN SHENGYA CLEANING APPLIAME CO.LTD	CHINA	NaN	TMT	consumer electronics	consumer electronics_distributor	Dowr

We will then import the `clean_dataset.xlsx` file and preview its first 10 rows.

Exploratory Data Analysis

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

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

Here, we list all the column names present in the dataset.

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

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

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

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

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

```
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
```

Then, general statistics of the dataset are calculated and shown. The numbers seen here may change depending on the dataset used.

```
# get total number of countries
df_totalCountries = companies['Country'].value_counts()

# get list of unique sector
df_sector = companies['Sector'].value_counts()

# get list of unique archetype
df_archetype = companies['Archetype'].value_counts()

# get list of unique valuechain
df_valuechain = companies['Valuechain'].value_counts()

print('List of unique countries:\n{}'.format(df_totalCountries))
print()
print('List of unique sector:\n{}'.format(df_sector))
print()
print('List of unique valuechain:\n{}'.format(df_valuechain))
```

We then print the counts of the unique values in each column.

```
List of unique countries:
SINGAPORE            3459
MALAYSIA             2200
THAILAND             1817
CHINA                 971
HONG KONG             577
INDONESIA             552
VIETNAM                12
MYANMAR                 3
TAIWAN                  2
LABUAN                  2
AUSTRALIA               2
INDONESIANO CIF         1
CANADA                  1
UNITED KINGDOM          1
Name: Country, dtype: int64
```

```
List of unique sector:
OOS           2082
CNI           1744
REH            942
IND            886
CG             862
ONG            667
TMT            637
cni            258
tmt            188
oos             79
ong             66
ind             11
cg               7
reh              6
                 1
auto & mec       1
Name: Sector, dtype: int64
```

```
List of unique valuechain:
MIDSTREAM                 1696
Downstream                1221
Midstream                  702
                           659
DOWNSTREAM                 658
midstream                  309
Manufacturer               296
UPSTREAM                   267
Trader                     184
downstream                 174
Upstream or Downstream     173
Upstream                   146
Across value chain         133
distributor                126
upstream                    38
upstream or downstream      13
manufacturer                 8
steel fabricator             1
Name: Valuechain, dtype: int64
```

Here are some examples of what may be shown in the results. Unique countries, sectors and valuechains are shown.

```
# get list of unique subsector
df_subsector = companies['Subsector'].value_counts()

print('List of unique Subsector:\n{}'.format(df_subsector))
```

```
List of unique Subsector:
others                                  2161
building_material                        831
retail n distribution                    560
buildings & industrial                   542
auto & mec                               482
petrochemical                            421
consumer electronics                     337
metals and mining                        300
residential                              281
cni_service providers                    279
restaurants, catering & services         270
fmcg                                     186
utilities                                178
it_services                              171
commercial                               146
o&g                                      135
mixed                                    129
o&g_service providers/contractors        123
semiconductor                            117
diversified                              114
infrastructure                           107
hotels and accommodation                  88
telecommunication                         88
animal protein                            76
media                                     73
cni_equipment suppliers                   59
agribusiness                              47
digital_business                          39
ong_traders                               37
industrial                                34
ioc/noc                                   15
building material                          6
gas                                        2
auto component dealer                      1
                                           1
buildings & industrial_contractor          1
IND                                        1
Name: Subsector, dtype: int64
```

Unique subsectors are printed here.

```
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
                                      ...
gas and lng                              2
MIDSTREAM                                2
tisp - fiber cable                       2
building material_manufacturer           1
metals and mining                        1
Name: Archetype, Length: 94, dtype: int64
```

Unique archetypes are printed here.

```
# 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 = companies.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...

Unnecessary columns are dropped here so they do not interfere with further data analysis.

```
# 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

Empty columns are collected, then displayed here.

```python
# 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 Informatio
0	AKSORN CHAROEN TAT ACT. CO.,LTD.	THAILAND	TMT	media	media_aggregator/distributor	Midstream	For over 80 years of experience creating an
1	DONGGUAN SHENGYA CLEANING APPLIAME CO.LTD	CHINA	TMT	consumer electronics	consumer electronics_distributor	Downstream	Yatai's mai products cover vario cleaning m.

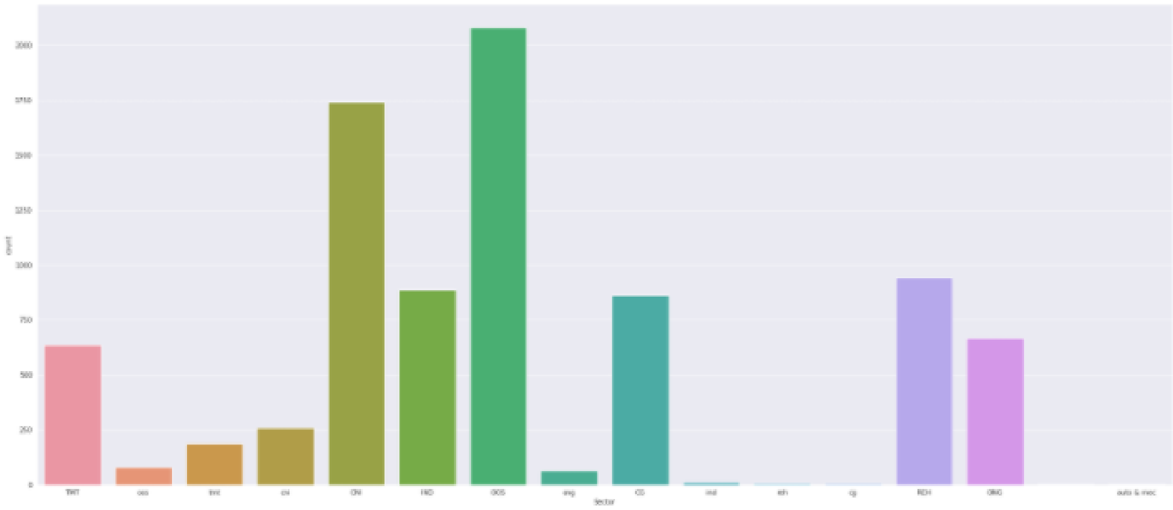
The empty rows are then dropped here.

```python
# now we check the count of the total filtered dataset again
df_filterCount = df_valid['Company'].count()

print('Total number of filtered records:', df_filterCount)
```

```
Total number of filtered records: 8447
```

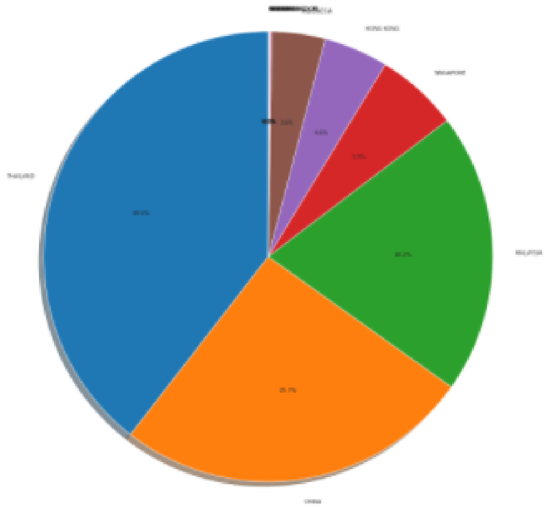
The new total number of records is calculated and shown here.

```python
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()
```



The statistics of the dataset are now plotted. Here, a unique count of each sector is plotted.

```
# Pie chart
labels = list(df_valid['Country'].unique())
sizes = list(df_valid['Country'].value_counts())

plt_dims = (30, 13)
fig1, ax1 = plt.subplots(figsize=plt_dims)
ax1.pie(sizes, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
# Equal aspect ratio ensures that pie is drawn as a circle
ax1.axis('equal')
plt.tight_layout()
plt.show()
```



Here, a pie chart of the unique country counts are plotted.

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

# get the 1st 50 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 educational innovation to meet the needs of all teachers, students, institut
ions and educational authorities.
1
Yatai's main products cover various cleaning machinery, cleaning agents, cleaning tools, stone mainten
ance and other cleaning products; cleaning solutions and services include product technical consultati
on, product customization, employee training, maintenance and so on.\n
```

Here, the first 10 company profile descriptions are previewed.

Data Pre-processing

```
# get rid of the \n found in the respective descriptions
df_valid = df_valid.replace('\n',' ', regex=True)

# now we validate to see if theye are really gone
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 educational innovation to meet the needs of all teachers, students, institut
ions and educational authorities.
1
Yatai's main products cover various cleaning machinery, cleaning agents, cleaning tools, stone mainten
ance and other cleaning products; cleaning solutions and services include product technical consultati
on, product customization, employee training, maintenance and so on.
```

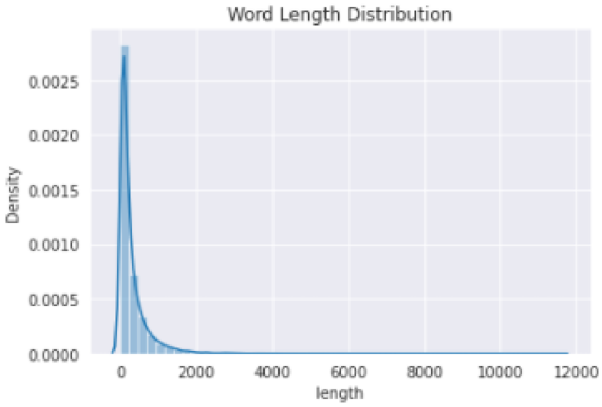
Here, newline \n characters are replaced with a blank space, and the results are previewed.

```
# first, add in a new column that tabluates the length of the respecive company description
df_valid["length"] = df_valid["Company Profile Information"].str.len()
df_valid.head()
```

Company	Country	Sector	Subsector	Archetype	Valuechain	Company Profile Information	length
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.	238.0

Company profile description lengths are calculated here, then added to the dataframe for easy access later.

```python
# 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()
```



The previously calculated word lengths are then plotted into the histogram shown. This may be different if other datasets were used.

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

df_valid
```

	Company	Country	Sector	Subsector	Archetype	Valuechain	Company I Informatio
0	AKSORN CHAROEN TAT ACT. CO.,LTD.	THAILAND	TMT	media	media_aggregator/distributor	Midstream	For over 80 experience creating an developing quality lean materials h enabled us provide wo educational innovation t the needs c teachers, s institutions educational authorities.

All cells containing NaN are filled with a space here, so that the row they are part of can be used for model training.

```python
# 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())

# save counts for use in model

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
```

Here, we programmatically generate the IDs for each target in each of the 4 labels. We then sort the tags for consistency across runs.

```
# 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
        print(row, '\n')
        df_valid.drop(index, inplace=True)


df_valid['list_tag'] = taglist_df

df_valid.shape
```

```
Company                        TOP FOUNTAIN LIMITED
Country                                   HONG KONG
Sector
Subsector
Archetype
Valuechain
Company Profile Information
length
Name: 1552, dtype: object
```

The correct prediction labels generated in the previous cell will be appended to each record in our dataframe here, for easy access later during training. Each invalid record will be printed out here and dropped.

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

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

Here, we convert all the column datatypes to string, to prevent any datatype issues during tokenisation.

```python
# import required libraries
from spacy.language import Language
from spacy.tokens import Doc
from spacy.lang.char_classes import ALPHA, ALPHA_LOWER, ALPHA_UPPER, CONCAT_QUOTES, LIST_ELLIPSES, LIST_ICONS
from spacy.util import compile_infix_regex

# initialise nlp engine
nlp = spacy.load("en_core_web_lg")

# declare custom properties
Doc.set_extension('processed', default=True, force=True)

# Modify tokenizer infix patterns
infixes = (
    LIST_ELLIPSES
    + LIST_ICONS
    + [
        r"(?<=[0-9])[+\-\*^](?=[0-9-])",
        r"(?<=[{al}{q}])\.(?=[{au}{q}])".format(
            al=ALPHA_LOWER, au=ALPHA_UPPER, q=CONCAT_QUOTES
        ),
        r"(?<=[{a}]),(?=[{a}])".format(a=ALPHA),
        r"(?<=[{a}0-9])[:<>=/](?=[{a}])".format(a=ALPHA),
    ]
)

infix_re = compile_infix_regex(infixes)
nlp.tokenizer.infix_finditer = infix_re.finditer

# custom retokenizer
@Language.component('custom_retokenizer')
def custom_retoken(doc):
    doc_text = doc.text.upper()
    doc_split = [i.text.upper() for i in doc]
    temp_kw = [i.lstrip().rstrip() for i in keywords_masterlist if len(i.lstrip().rstrip().split(' ')) > 1]
    for token in temp_kw:
        token_length = len(token.split(' '))
        token_split = token.split(' ')
        if token in doc_text and token_split[0] in doc_split:
            merge_pos = doc_split.index(token_split[0])
            with doc.retokenize() as retokenizer:
                try:
                    retokenizer.merge(doc[merge_pos:merge_pos + token_length], attrs={'LEMMA' : token.lower()})
                except:
                    print(merge_pos, merge_pos+token_length)

    return doc

# custom lemmatizer
@Language.component("custom_preprocess")
def custom_preprocess(doc):
    temp = []
    # filter through each token and add to preprocessed text if requirements #
    # met.                                                                    #
    for t in doc:
        if (not t.is_punct and not t.like_num and not t.is_stop and not t.is_digit and not (t.ent_type == 396 or t.ent_type == 397)):
            temp.append(t.lemma_.upper())

    doc._.processed = temp

    return doc

# add custom pipeline components to default pipeline
nlp.add_pipe('custom_retokenizer')
nlp.add_pipe('custom_preprocess', last=True)
```

Here, we prepare spaCy for the tokenisation of our dataset. We modified spaCy's tokenisation rules to treat hyphenated words as one single word, and added custom pipelines to get spaCy to merge any special phrases that would otherwise be separated, e.g., company names. Another custom pipeline manages the lemmatisation of the dataset.

```python
# run the pipeline on data
processed_doc = list(nlp.pipe(df_valid['Company Profile Information']))
```

spaCy is run on the dataset here, which will output to a variable called `processed_doc`.

```python
# add lemmatised words to dataframe
df_valid['processed'] = [doc._.processed for doc in processed_doc]

# add tok2vec vectors to dataframe
df_valid['tok2vec_vectors'] = [doc.vector for doc in processed_doc]

df_valid
```

Lemmatised words, as well as spaCy's generated tok2vec vectors will be added into the dataframe here. A preview will be printed out.

```python
# test out doc2vec
from gensim.models.doc2vec import Doc2Vec, TaggedDocument

# train doc2vec
documents = [TaggedDocument(doc._.processed, [i]) for i, doc in enumerate(processed_doc)]
d2v_model = Doc2Vec(documents, vector_size=700, window=2, workers=4)

# add doc2vec vectors to dataframe
df_valid['doc2vec_vectors'] = [d2v_model.infer_vector(doc._.processed) for doc in processed_doc]

df_valid.head(3)
```

Here, the libraries needed for doc2vec tokenisation are imported, set up, and performed here. Results are automatically added to the dataframe, and a preview of the dataframe will be output.

```python
# import validation dataset
df_evaluate = pd.read_excel('./val_dataset.xlsx')
df_evaluate.replace('NAN', np.NaN, inplace=True)

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

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

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

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

# process validation dataset
df_evaluate['Valuechain'] = df_evaluate['Valuechain'].str.split().str.join(' ')
df_evaluate['Valuechain'] = df_evaluate['Valuechain'].str.upper()
df_evaluate['Sector'] = df_evaluate['Sector'].str.upper()
df_evaluate['Valuechain'].replace('', ' ', inplace=True)

# add tags to validation data
taglist_df = []
# process tags for records
for index, row in df_evaluate.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_evaluate.drop(index, inplace=True)

df_evaluate['list_tag'] = taglist_df

# run spacy on validation data
evaluate_doc = list(nlp.pipe(df_evaluate['Company Profile Information']))

# add spacy processed words and vectors to validation dataframe
df_evaluate['processed'] = [doc._.processed for doc in evaluate_doc]
df_evaluate['tok2vec_vectors'] = [doc.vector for doc in evaluate_doc]

# do doc2vec processing
df_evaluate['doc2vec_vectors'] = [d2v_model.infer_vector(doc._.processed) for doc in evaluate_doc]

df_evaluate.head(3)
```

All the previously mentioned pre-processing steps are done to the test dataset here. A preview will show the test dataframe after all the pre-processing is complete.

Model Training

```
import tensorflow.keras as Keras

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

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

The TensorFlow Keras library is imported and checked here.

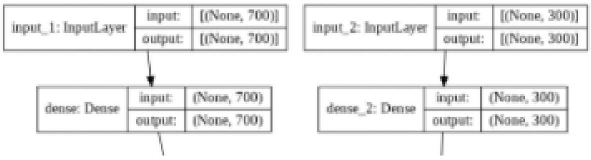
```
X_train = [np.array(list(df_valid['doc2vec_vectors'])), np.array(list(df_valid['tok2vec_vectors']))]
y_train = np.array(list(df_valid['list_tag']))

X_test = [np.array(list(df_evaluate['doc2vec_vectors'])), np.array(list(df_evaluate['tok2vec_vector
s']))]
y_test = np.array(list(df_evaluate['list_tag']))
```

The relevant model training and test inputs and outputs are extracted here and saved to standardised names.

```python
# create multi-output model
from tensorflow.keras.layers import Dense, Input, Dropout, Concatenate
from tensorflow.keras import Model

# function to build model branches
def multi_branch(x, name, input_dim, output_dim, dropout_rate):
    x = Dense(input_dim // 2, activation='selu')(x)
    x = Dropout(dropout_rate)(x)

    if name == 'sector' or name == 'subsector' or name == 'archetype':
        x = Dense(input_dim // 3, activation='selu')(x)
        x = Dropout(dropout_rate)(x)

    x = Dense(input_dim // 4, activation='tanh')(x)
    x = Dropout(dropout_rate)(x)

    if name == 'sector' or name == 'subsector' or name == 'archetype':
        x = Dense(input_dim // 4, activation='tanh')(x)
        x = Dropout(dropout_rate)(x)

    x = Dense(input_dim // 8, activation='swish')(x)

    # output
    output = Dense(output_dim, name=name, activation='softmax')(x)

    return output

def create_multilabel(labels, labels_output_dim, bow_dim, t2v_dim, dropout_rate=0.3):
    # check labels
    assert len(labels) == len(labels_output_dim)

    input_layer_1 = Input(bow_dim)
    input_layer_2 = Input(t2v_dim)

    # group 1 dense layers
    group_1 = Dense(int(bow_dim), activation='tanh')(input_layer_1)
    group_1 = Dense(bow_dim // 1.5, activation='tanh')(group_1)

    # group 2 dense layers
    group_2 = Dense(t2v_dim, activation='tanh')(input_layer_2)
    group_2 = Dense(t2v_dim // 1.5, activation='tanh')(group_2)

    # concatenate layers
    merge_layer = Concatenate()([group_1, group_2])
    merge_layer = Dense((bow_dim // 1.5) + (t2v_dim // 1.5), activation='swish')(merge_layer)
    merge_layer = Dropout(dropout_rate)(merge_layer)
    merge_layer = Dense((bow_dim // 2) + (t2v_dim // 1.5), activation='swish')(merge_layer)
    merge_layer = Dropout(dropout_rate)(merge_layer)

    # multilabel branches
    branches = []
    for i in range(len(labels)):
        branches.append(multi_branch(merge_layer, labels[i], bow_dim, labels_output_dim[i], dropout_rate))

    # put model together
    model = Model(inputs=[input_layer_1, input_layer_2], outputs=branches, name='company_classification_model')

    return model

# one hot
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)
```

Here, functions necessary for model training are implemented. `multi_branch` is to create the 4 branches in the model, `create_multilabel` is to create the main model body, and one_hot is to convert the target labels into one-hot vectors.

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

model = create_multilabel(label_names, tag_counts, 700, 300)

tf.keras.utils.plot_model(model, show_shapes=True)
```



Here, the model is initialised, and a model graph is plotted to show the structure of the entire model.

```
# preprocess labels before training
y_train_multi = {label_names[i] : one_hot(y_train[:, i], tag_counts[i]) for i in range(4)}
y_test_multi = {label_names[i] : one_hot(y_test[:, i], tag_counts[i]) for i in range(4)}

losses = {i : 'categorical_crossentropy' for i in label_names}
```

Here, the target labels for training and testing are one-hot encoded and saved into a format suitable for the model to train on. The loss functions are also defined here.

```
# import metrics
from tensorflow.keras.metrics import Precision, Recall, AUC
from tensorflow.keras.optimizers import SGD
from timeit import default_timer as timer

model.compile(optimizer=SGD(nesterov=True), loss=losses, metrics=['accuracy', Precision(), Recall(),
AUC(name='auc_precision_recall', num_thresholds=10000)])
start = timer()
history = model.fit(X_train, y_train_multi, epochs=50, batch_size=20)
end = timer()
print("Total time taken for execution: ", end-start, "s")
```

```
Epoch 1/50
422/422 [==============================] - 19s 29ms/step - loss: 10.1920 - sector_loss: 1.8189 - subse
ctor_loss: 2.9547 - archetype_loss: 3.7350 - valuechain_loss: 1.6834 - sector_accuracy: 0.2881 - secto
r_precision: 0.9231 - sector_recall: 0.0057 - sector_auc_precision_recall: 0.6606 - subsector_accurac
y: 0.2467 - subsector_precision: 0.6552 - subsector_recall: 0.0023 - subsector_auc_precision_recall:
0.7630 - archetype_accuracy: 0.2441 - archetype_precision: 0.3333 - archetype_recall: 0.0031 - archety
pe_auc_precision_recall: 0.7763 - valuechain_accuracy: 0.3395 - valuechain_precision: 0.7500 - valuech
ain_recall: 0.0021 - valuechain_auc_precision_recall: 0.8078
```

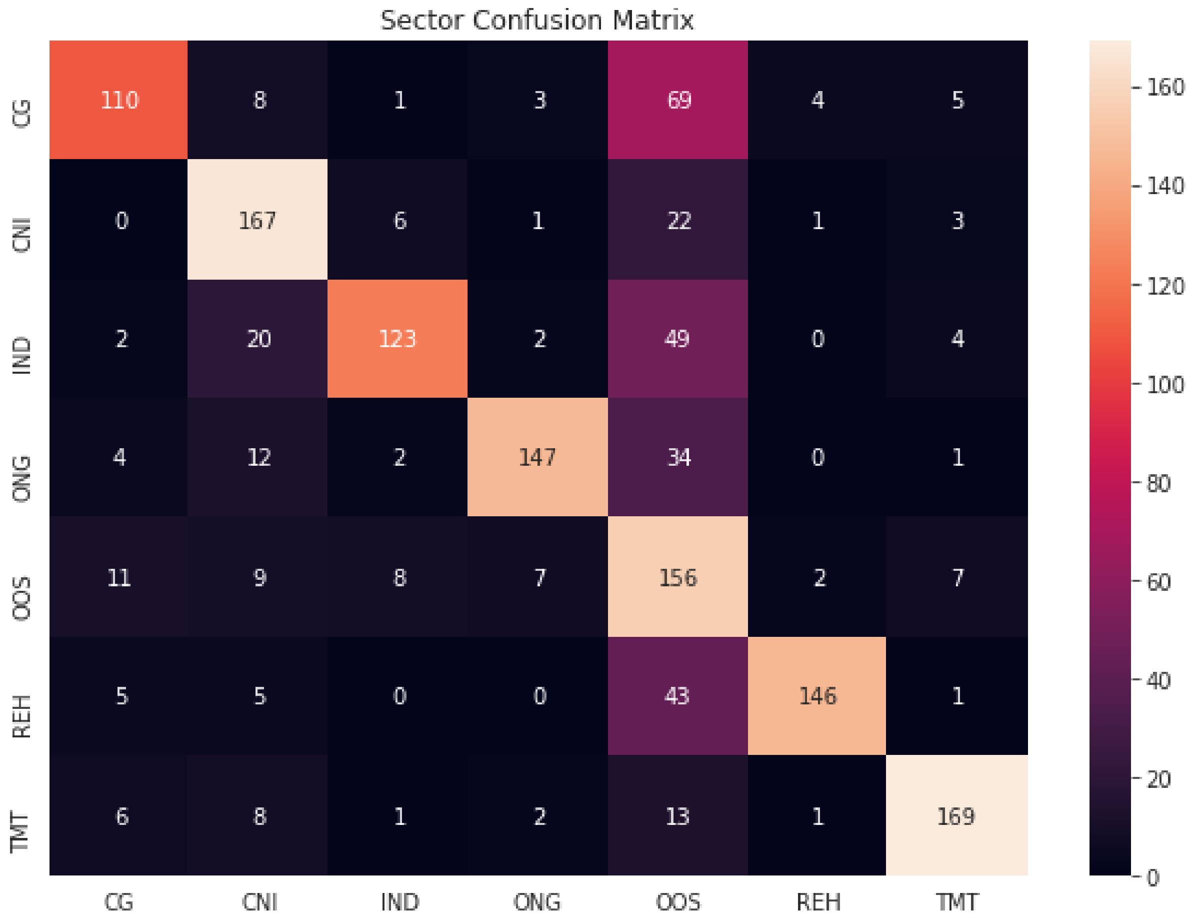
Here, the model is compiled with desired metrics, and trained while being timed. The output will consist of training progress messages and time taken to train in seconds after completion.

```python
# import libs
import seaborn as sns

# custom metric analysis
y_test_multi = [one_hot(np.array(list(df_evaluate['list_tag']))[:, i], tag_counts[i]) for i in range(4)]
y_actual = np.array(df_evaluate['list_tag'].to_list())

# obtain predictions
y_pred = model.predict(X_test)

# process predictions
processed = []
for label in y_pred:
    temp = []
    # turn all softmax preds to numbers
    for row in label:
        temp.append(np.argmax(row))

    processed.append(np.array(temp))

# convert processed preds to correct format
y_pred = np.array(processed).T

# do analysis
# format: [correctly predicted, sec and sub predicted correctly, false negative, false positive]
sector_metrics = {i : [0, 0, 0, 0] for i in range(tag_counts[0])}
subsector_metrics = {i : [0, 0, 0, 0] for i in range(tag_counts[1])}
for i, row in enumerate(y_pred):
    flag = 0
    # check sector
    if row[0] == y_actual[i][0]:
        sector_metrics[row[0]][0] += 1
        flag += 1
    else:
        sector_metrics[row[0]][3] += 1
        sector_metrics[y_actual[i][0]][2] += 1

    # check subsector
    if row[1] == y_actual[i][1]:
        subsector_metrics[row[1]][0] += 1
        flag += 1
    else:
        subsector_metrics[row[1]][3] += 1
        subsector_metrics[y_actual[i][1]][2] += 1

    if flag == 2:
        sector_metrics[row[0]][1] += 1
        subsector_metrics[row[1]][1] += 1

# print out stats
```

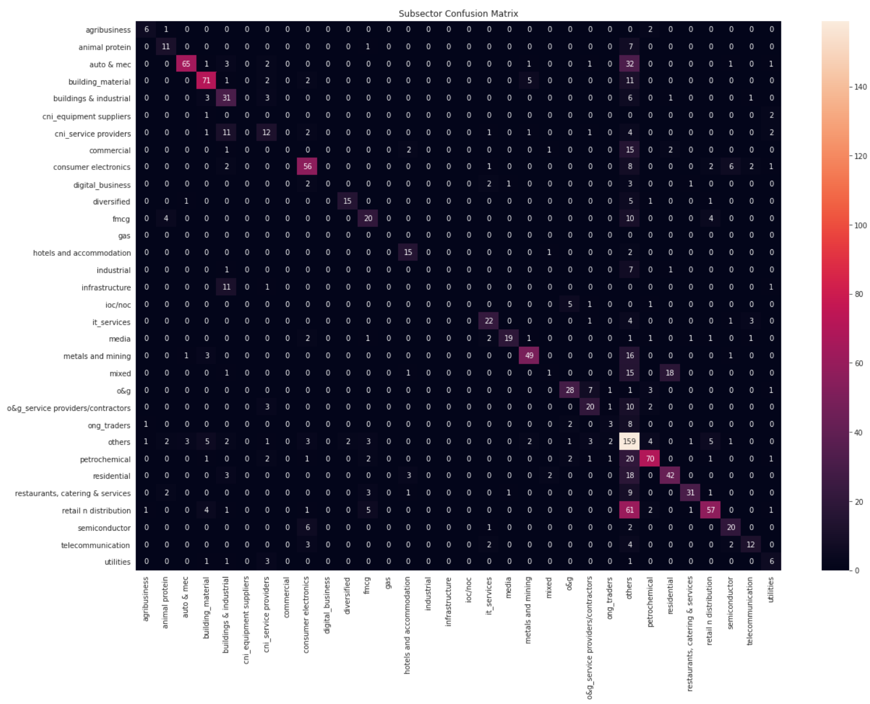
```python
print(f'Label\t\t\t\t\tAccuracy\tCorrect Rows\t% Correct Rows\tPrecision\tRecall')
for sec_label, sec_metrics in sector_metrics.items():
    # generate list of subsectors under a sector
    sub_unique, sub_counts = np.unique(y_actual[y_actual[:, 0] == sec_label][:, 1], return_counts=True)
    sec_subsector = dict(zip(sub_unique, sub_counts))

    # print sector stats
    print('-'*110)
    print(f'{sector[sec_label]:<40.39}{sec_metrics[0] / 200:<16.1%}{sec_metrics[1]:<16}{sec_metrics[1] / 200:<16.1%}{sec_metrics[0] / (se
c_metrics[0] + sec_metrics[3]):<16.3f}{sec_metrics[0] / (sec_metrics[0] + sec_metrics[2]):<6.3f}')

    for sub_label, sub_count in sec_subsector.items():
        sub_metrics = subsector_metrics[sub_label]

        # catch divide-by-zero errors
        try:
            sub_acc = sub_metrics[0] / sub_count
            sub_corr = sub_metrics[1] / 200
            sub_pre = sub_metrics[0] / (sub_metrics[0] + sub_metrics[3])
            sub_rec = sub_metrics[0] / (sub_metrics[0] + sub_metrics[2])
        except ZeroDivisionError:
            sub_pre = sub_rec = sub_corr = sub_acc = 0


        print(f'> {subsector[sub_label]:<38.37}{sub_acc:<16.2%}{sub_metrics[1]:<16}{sub_corr:<16.1%}{sub_pre:<16.3f}{sub_rec:<6.3f}')

# sector confusion matrix
df_sector_cm = pd.DataFrame(tf.math.confusion_matrix(y_actual[:, 0], y_pred[:, 0]).numpy(), index=sector, columns=sector)

plt.figure(1, figsize=(10, 7))
plt.title('Sector Confusion Matrix')
sns.heatmap(df_sector_cm, annot=True, fmt='d')

# subsector confusion matrix
df_subsector_cm = pd.DataFrame(tf.math.confusion_matrix(y_actual[:, 1], y_pred[:, 1]).numpy(), index=subsector, columns=subsector)

plt.figure(2, figsize=(20, 14))
plt.title('Subsector Confusion Matrix')
sns.heatmap(df_subsector_cm, annot=True, fmt='d')
```

```
Label                                   Accuracy        Correct Rows    % Correct Rows  Precision       Recall
--------------------------------------------------------------------------------------------------------------
CG                                      55.0%           89              44.5%           0.797           0.550
> agribusiness                          66.67%          6               3.0%            0.667           0.667
> animal protein                        57.89%          10              5.0%            0.550           0.579
> fmcg                                  52.63%          20              10.0%           0.571           0.526
> retail n distribution                 42.54%          53              26.5%           0.792           0.425
--------------------------------------------------------------------------------------------------------------
```

Here, metrics in finer detail are generated and printed, as well as confusion matrices for Sector and Subsector.



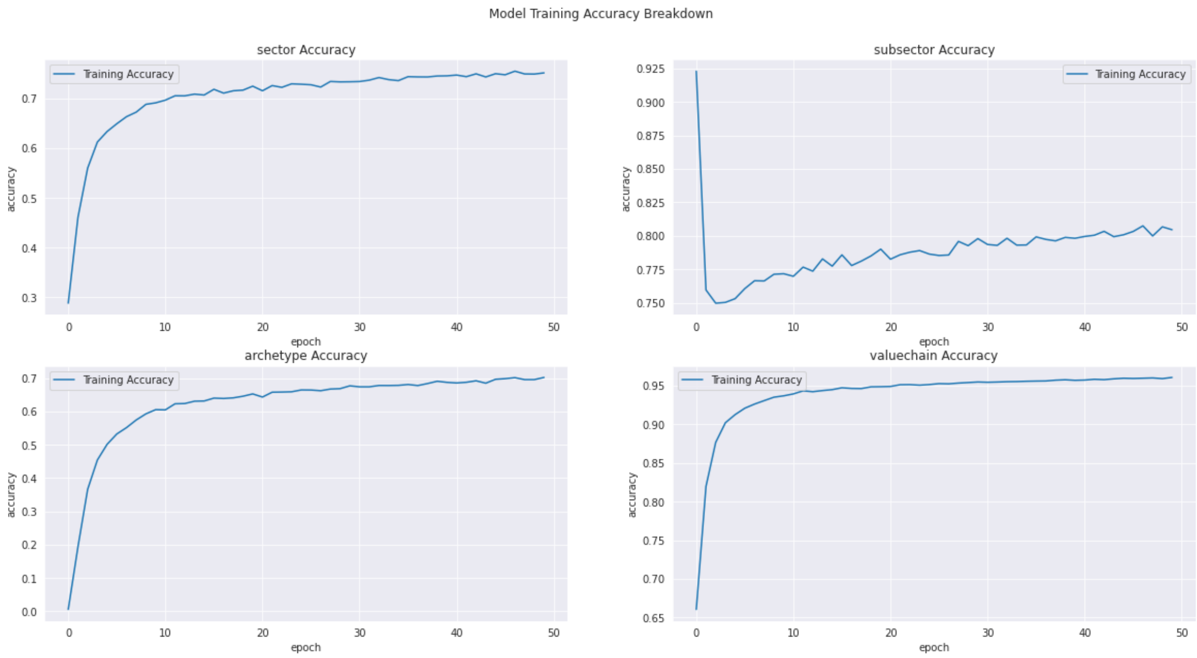
An example of a Sector confusion matrix.



An example of a Subsector confusion matrix.

```python
# plot metric graphs for all labels
metric_names = model.metrics_names

fig = plt.figure(1, figsize=(20,10))
plt.suptitle('Model Training Loss Breakdown', y=0.95)

for i, name in enumerate(label_names):
    plt.subplot(2, 2, i+1)
    plt.plot(history.history[metric_names[i+1]])
    plt.title(f'{name} Loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['Training loss'])

fig = plt.figure(2, figsize=(20,10))
plt.suptitle('Model Training Accuracy Breakdown', y=0.95)

for i, name in enumerate(label_names):
    plt.subplot(2, 2, i+1)
    plt.plot(history.history[metric_names[i+5]])
    plt.title(f'{name} Accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['Training Accuracy'])

fig = plt.figure(3, figsize=(20,10))
plt.suptitle('Model Training Precision Breakdown', y=0.95)

for i, name in enumerate(label_names):
    plt.subplot(2, 2, i+1)
    plt.plot(history.history[metric_names[i+9]])
    plt.title(f'{name} Precision')
    plt.ylabel('precision')
    plt.xlabel('epoch')
    plt.legend(['Training Loss'])

fig = plt.figure(4, figsize=(20,10))
plt.suptitle('Model Training Recall Breakdown', y=0.95)

for i, name in enumerate(label_names):
    plt.subplot(2, 2, i+1)
    plt.plot(history.history[metric_names[i+13]])
    plt.title(f'{name} Recall')
    plt.ylabel('recall')
    plt.xlabel('epoch')
    plt.legend(['Training Loss'])

plt.show()
```

Model training graphs are plotted here and displayed. The metrics plotted are Loss, Accuracy, Precision, and Recall.



An example Accuracy graph plotted by the cell.

```
# save models to file
model.save('./multilabel_model')
d2v_model.save('./doc2vec_model')
```

```
INFO:tensorflow:Assets written to: ./multilabel_model/assets
```

```
!zip -r multilabel_model.zip ./multilabel_model
!rm -rf ./multilabel_model
```

The model is finally saved using TensorFlow's built in model saving function and format, and then compressed into a zip for download.