CMP6228 – Deep Neural Networks

Deep learning dataset description and solution

Coursework Assignment Report

Zan Zver
18133498

Word count: 2012

BIRMINGHAM CITY University

Faculty of Computing, Engineering and the Built Environment
Birmingham City University

Contents

List of Tables

List of Figures

Listings

1 Introduction

This report covers text prediction of chocolate ratings. The report describes dataset, its properties and a bit of cleaning as well. Since text prediction is not new, some related work is presented highlighting the inspiration for this project. In the last section more information about the model can be found including what data is going to be used and what the construction of the model is going to be.

2 Problem statement

The company (FlavorsOfCacao n.d.) has build modest dataset that describes chocolate rating. Their goal is for users to see the rating and decide to buy or not chocolate based on the rating.

So, what would the perfect chocolate be? It takes time, effort and finance to produce chocolate. To mitigate that, we can use neural networks to predict chocolate bar ratings and potentially discover the perfect conditions for chocolate bars.

3 Proposed method

To overcome the problem, we have chosen Python as programming language with Keras (Keras n.d.) being main NN library in use. Since we need computation power, Google Colab (GoogleColab n.d.) is being used as development platform. With that in mind, we can split the project into 11 main subsections.

3.1 Import libraries

As noted from before, main library used for our project is Keras. But, we need to do some data processing and for that we need other libraries (sklearn, numpy, pandas) and drawing the charts (matplolib).

```
import pandas as pd
from sklearn.model_selection import KFold
import keras
import tensorflow as tf
from sklearn.preprocessing import OneHotEncoder
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from keras import layers
from keras import models
from keras import regularizers
from keras import optimizers
from sklearn.preprocessing import LabelEncoder
from keras.utils import np_utils
from sklearn.metrics import multilabel_confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix,
    ConfusionMatrixDisplay
import warnings
```

Listing 1: Import libraries

3.2 Import the data

Data is being directly imported from Kaggle to Google Colab. This is done with the help of their API (Kaggle n.d.). So, first step is to create local (to Colab) dictionary (with the name of kaggle). Next, we move our API keys from Google Drive to our new folder. Now we change permissions of the keys so they can be read. After that, we download the dataset and we can see it saved locally. More information on the dataset can be found on appendix A1: About the data.

```
! mkdir ~/.kaggle #create local kaggle folder
! cp /content/drive/MyDrive/Kaggle/kaggle.json ~/.kaggle/ #move
    auth key from one drive to temp folder
! chmod 600 ~/.kaggle/kaggle.json #use auth key
! kaggle datasets download rtatman/chocolate-bar-ratings -f
    flavors_of_cacao.csv #download the file from kaggle and unzip
    it while it is downloading
```

Listing 2: Import the data

3.3 Save data to pandas

Now that we have the data saved locally (as CSV), we can access it. With Python, we save it to Pandas dataframe so we can modify it later.

```
cacao_dataframe_full=pd.read_csv('/content/flavors_of_cacao.csv')
```

Listing 3: Save data to pandas

3.4 Filter the data

There are some attributes in the dataset that are not useful for our case. Due to that, we are going to remove them.

```
cacao_dataframe = cacao_dataframe_full
del cacao_dataframe["Company\xa0\n(Maker-if known)"]
del cacao_dataframe["REF"]
del cacao_dataframe["Review\nDate"]
```

Listing 4: Remove attributes

Replace new lines (\n) with spaces.

```
cacao_dataframe.columns = cacao_dataframe.columns.str.replace
                              ("\n", ' ', regex=True)
```

Listing 5: Remove spaces

Data in attribute "Cocoa Percent" has % symbol next to the numeric value. This is something that needs to be changed, therefore all of the % symbols are going to be removed from the attribute.

```
cacao_dataframe2 = pd.DataFrame(
             cacao_dataframe["Cocoa Percent"].str.replace("%",""))
cacao_dataframe.drop(["Cocoa Percent"], axis=1, inplace=True)
cacao_dataframe = cacao_dataframe.join(cacao_dataframe2)
```

Listing 6: Remove symbols

While developing the code, we found out that our dataset is not big enough (in terms of records). This was clearly marked when we figured out that we only have 2 ratings that are marked as 5. To combat this, we have multiplied the whole dataset by 10. With that in mind, we still have the same ratio between results, it just makes it simpler to avoid facing wrong sizes upon fitting data to the model. Alternative would be to do data augmentation.

```python
cacao_dataframe_full = cacao_dataframe_full.loc
                            [cacao_dataframe_full.index.repeat(10)]
cacao_dataframe_full = cacao_dataframe_full.reset_index(drop=True)
```

Listing 7: Expend data

While we did some data inspection, there seems to be data that is not unified. To combat this, all of the ratings are "rounded" the same way.

```python
def fix_ratings(rating):
    if rating < 0.5:
        return 0.0
    elif rating < 1.5:
        return 1.0
    elif rating < 2.5:
        return 2.0
    elif rating < 3.5:
        return 3.0
    elif rating < 4.5:
        return 4.0
    elif rating < 5.5:
        return 5.0
    else:
        return 0.0

cacao_dataframe_full["Rating"] =
    cacao_dataframe_full["Rating"].apply(lambda x: fix_ratings(x))
```

Listing 8: Fix ratings

3.5 One hot encoding of the data

In our dataset, we have 6 attributes. 2 of them are of numeric type (Cocoa Percent and Rating), the other 4 are string types. With that in mind, we cannot do neural network operations on such dataset without some modifications.

To overcome the problem, we use one hot encoding for encoding the data(Jie et al. 2019). What one hot encoding does, it binary encodes words into the dataframe. For example, if chocolate would have origin in UK, attribute would be created with the name of UK and all of chocolates with origin of UK would have 1 as a value and others would have 0.

In the code block bellow, we can see how we are encoding the data. First empty dataframe is build that will store the data and then the function is build that will encode the data. Next, we pass in our dataframe to be encoded and save it as encoded dataframe. In last step, we remove original attributes since they are not needed anymore.

```python
encoded_dataframe = pd.DataFrame()
encoder = OneHotEncoder(handle_unknown='ignore')
```

```
def encodeRow(df, columnName):
    encoder_df = pd.DataFrame(encoder.fit_transform(df[columnName
   ]).toarray())
    newNames= []
    for x in encoder.get_feature_names_out():
       x = x.replace("_", " ")
       x = x.rstrip()
       x = x.replace('\n', "_")
       x = x.replace(' ', "_")
       newNames.append(x)
    encoder_df.columns = list(newNames)
    return encoder_df

encoded_dataframe = cacao_dataframe_full.join(
                        encodeRow(cacao_dataframe_full,[
                            "Specific Bean Origin or Bar Name",
                            "Company Location",
                            "Bean Type",
                            "Broad Bean Origin"]
                        ))

encoded_dataframe.drop([
                "Specific Bean Origin or Bar Name",
                "Company Location",
                "Bean Type",
                "Broad Bean Origin"],
                axis=1, inplace=True)
```

Listing 9: One hot encoding of the data

3.6 Split the data - X, y

Now that we have the dataframe encoded we can split it up to X (features) and y (target). In the code block bellow, we can see X is collection of attributes from encoded dataframe that are indexed 1 and above. Opposite to that, y is single attribute from encoded dataframe that has index 0.

```
X = encoded_dataframe.iloc[:,1:len(encoded_dataframe.columns)]
y = encoded_dataframe.iloc[:,0:1]
```

Listing 10: Split the data - X and y

3.7 Split the data - k-fold

Next step is to split the data again but into 3 sections: train, test and validate. This can be done in a number of different ways, but one of the most efficient ways is k-fold cross validation (Rodriguez, Perez, and Lozano 2009). What k-fold does, is it splits the data into k (number) of sections. With this, we can have the best ratio between train, test and validate.

The code block bellow is enabling us to do k-fold. First, we define parameters of k-fold into the variable of kf. Next, the function (kFoldValidation) is written that will split the data upon being called.

```
kf = KFold(n_splits = 10, shuffle = True, random_state = 6)
def kFoldValidation(df):
  result = next(kf.split(df), None)
  train = df.iloc[result[0]]
  test =  df.iloc[result[1]]
  return train, test
```

Listing 11: Create k-fold function

Now that we have our k-fold function sorted, we can use it. First, we split the data into two sections: 1) train and 2) validation and testing. Next step is to split second section apart into 2.1) validation and 2.2) testing.

```
X_train, X_val_and_test = kFoldValidation(X)
y_train, y_val_and_test = kFoldValidation(y)
X_val, X_test = kFoldValidation(X_val_and_test)
y_val, y_test = kFoldValidation(y_val_and_test)
```

Listing 12: Split the data to train, test and validate

The data is in the format of Pandas dataframe until now. This is something that needs to be changed in order for model to process the data. That is why, we have converted the data to numpy array. As seen in the code block bellow, data is also being transformed. Y (target) data is being encoded with label encoder since ratings (1-5) are potential labels. With X (input) data, this is being scaled based on the encoded data.

```
sc = StandardScaler(with_mean=False)
sc.fit(X_train)

le = LabelEncoder()
le.fit(y_train)

X_train_transf = sc.transform(X_train)
X_test_transf = sc.transform(X_test)
X_val_transf = sc.transform(X_val)

y_train_cat = np_utils.to_categorical(le.transform(y_train))
y_test_cat = np_utils.to_categorical(le.transform(y_test))
y_val_cat = np_utils.to_categorical(le.transform(y_val))
```

Listing 13: Reshape the data to fit the model

3.8 Build Neural Network

3.8.1 Model theory

Plan for the model is to have 5 types of inputs (represented in *Figure 1*). Although final (input) number is not going to be 5, rather it is going to be based on encoded

cases (1242 in our case). The output layer is expected to predict ratings. This should be a number in the range of 1 (the worse) to 5 (the best).
Inputs (X) for the model are:

- $I_1 \Rightarrow$ Specific Bean Originor Bar Name,
- $I_2 \Rightarrow$ CocoaPercent,
- $I_3 \Rightarrow$ CompanyLocation,
- $I_4 \Rightarrow$ BeanType,
- $I_5 \Rightarrow$ Broad BeanOrigin.

Output (y) for the model is:

- $O_1 \Rightarrow$ Rating 1
- $O_2 \Rightarrow$ Rating 2
- $O_3 \Rightarrow$ Rating 3
- $O_4 \Rightarrow$ Rating 4
- $O_5 \Rightarrow$ Rating 5

3.8.2 Constructing the model

In the model construction, different parameters were tested. At the end, the parameters (and hyper parameters) that yielded the best results are:

- number of units: 200, 100 and 50
  - In the first layer, we do start with 200 units. As we progress, number of units is decreased in each hidden layer. This results in H1 being 200, H2 100 and H3 has only 50 units. In the last (output) layer, we only have 5 units (since ratings are 1-5).
- batch size: 128
  - Batch size of 64 was the starter and it was tested up to 256. After the batch size of 158, difference was minimal therefore this was the chosen number.
- number of epochs: 100
  - For testing purposes, we have started with 20 epochs to save the time, but went with 100 for final product since the results are better with it.
- learning rate: 0.00001

  - Learning rates of 0.001, 0.0001, 0.00001 and 0.000001 were tested. At the end, 0.00001 gave the best results.

- momentum
  - This function is left disabled. After testing it, momentum of 0.2 and 0.3 brings the best results but if it is disabled (set to 0.0), results are even better.
- dropout: 0.3 and 0.2
  - First layer is using dropout of 0.3. After that, we have decreased dropout to 0.2 for the rest of hidden layers (H2 and H3).
- activation function: LeakyReLU and softplus in last
  - Input function and hidden layers 1-3 (H1, H2 and H3) are using linear activation function. In the last layer, softplus converts vector to a distribution that can be used for predicting Ratings.
- number of layers: 5
  - Input layer
  - Hidden layer 1
  - Hidden layer 2
  - Hidden layer 3
  - Output layer
- optimizer: RMSprop
  - Diferent optimises were tested along the way. At the end, there were two candidates: RMSprop and SGD. After some testing RMSprop delivered slightly better results therefore it was chosen.
- loss function: categorical crossentropy
  - Loss function of choice is categorical crossentropy. This is due to our problem being predicting Ratings, therefore we have chosen that.
- metrics: accuracy
  - accuracy is not the only metrics used in industry, but it is the most well known. This is the reason for implementing it. Besides that, confusion matrix is being used as well (that is showcased at the end).

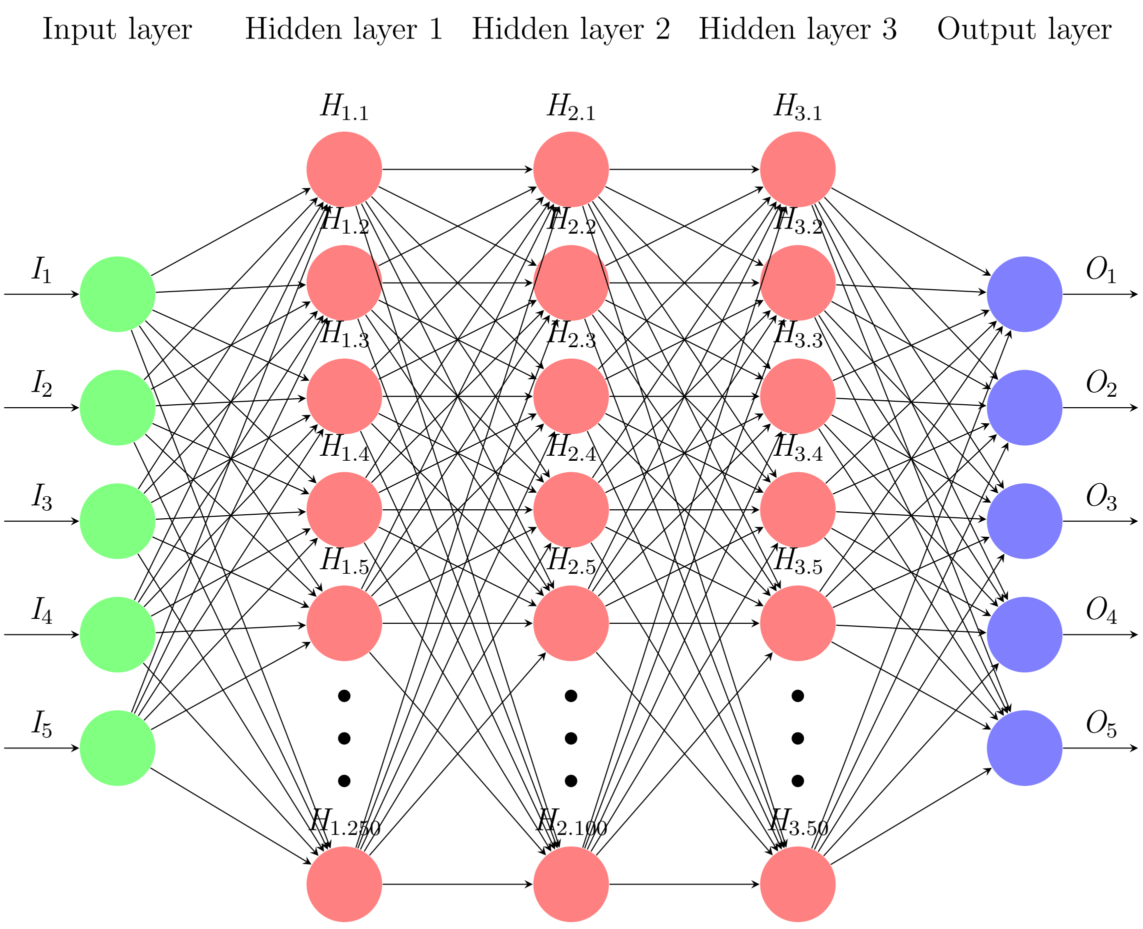


Figure 1: Visualisation of the model

Code block representing how model is build together.

```python
model = models.Sequential()
# hidden layer 1
model.add(layers.Dense(250,
                       activation='LeakyReLU',
                       input_dim=X_train.shape[1],
                       kernel_regularizer=regularizers.l2(0.00001)
    ))
model.add(layers.Dropout(0.3))
# hidden layer 2
model.add(layers.Dense(100, activation='LeakyReLU'))
model.add(layers.Dropout(0.2))
# hidden layer 3
model.add(layers.Dense(50, activation='LeakyReLU'))
model.add(layers.Dropout(0.2))
# output layer, 5 untis since ratings are 1,2,3,4,5
model.add(layers.Dense(5, activation='softplus'))
```

Listing 14: Build model

Representation of compiling the model.

```
model.compile(optimizer='RMSprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Listing 15: Compile model

Fitting the model and saving results into local variable named history.

```
history = model.fit(X_train_transf,
                    y_train_cat,
                    validation_data=(X_val_transf, y_val_cat),
                    batch_size=158,
                    epochs=100,
                    verbose=2)
```

Listing 16: Fit model

3.9 Test Neural Network

To test the NN, we put the last batch of never seen data to it. This is test data. With that in mind, we can see our evaluation and determine if this is matching to the results from model training.

```
scores = model.evaluate(X_test_transf, y_test_cat, batch_size=128)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

Listing 17: Evaluate the model

3.10 Calculate and display model results

To calculate and display results, we first need to grab the data. We do this by getting history data that we have created from above.

```
history_dict = history.history
history_dict.keys()
```

Listing 18: Create dictionary of history values

This step is not necessary, but it is convenient. What we do is, extract values based on keys and save it to local variables.

```
training_acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']

training_loss = history_dict["loss"]
val_loss = history_dict["val_loss"]
```

Listing 19: Save data from dict to local variables

In the next step, we plot our first graph as training and validation accuracy.

```
epochs = range(1, len(val_acc) + 1)

plt.plot(epochs, training_acc, 'bo', label="Training Accuracy")
```

```python
plt.plot(epochs, val_acc, 'b', label='Validation Accuracy')

plt.title('Plot training and validation accuracy')

plt.legend()
plt.show()
```

Listing 20: Plot training and validation accuracy

Finaly, we plot the last graph as training and validation loss.

```python
epochs = range(1, len(training_loss) + 1)

plt.plot(epochs, training_loss, 'bo', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')

plt.title('Plot training and validation loss')

plt.legend()
plt.show()
```

Listing 21: Plot training and validation loss

3.11 Confusion matrix

To evaluate the data (further), we are using confusion matrix (Visa et al. 2011). This standard is popular in the industry. In the first code block, we are calculating multi label confusion matrix and displaying the results.

```python
warnings.filterwarnings('ignore')
preds = model.predict(X_test_transf)
preds = np.where(preds < 0.5, 0, 1)

target_names = ['Rating 1', 'Rating 2', 'Rating 3', 'Rating 4', '
    Rating 5']
cm = multilabel_confusion_matrix(y_test_cat, preds)
print(classification_report(y_test_cat, preds, target_names=
    target_names))
```

Listing 22: Calculate confusion matrix

Second code block is for visualise confusion matrix. Since we have multi label confusion matrix, each label (class) is visualised separately.

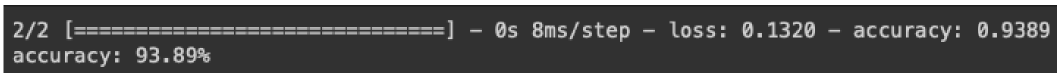
```python
f, axes = plt.subplots(1, 5, figsize=(20, 10))
axes = axes.ravel()
for i in range(5):
    disp = ConfusionMatrixDisplay(confusion_matrix(y_test_cat[:, i
    ],
                                                            preds[:, i]),
                                              display_labels=[0, i+1])
    disp.plot(ax=axes[i], values_format='.4g')
    disp.ax_.set_title(f'Rating {i+1}')
    if i<10:
        disp.ax_.set_xlabel('')
    if i%5!=0:
        disp.ax_.set_ylabel('')
    disp.im_.colorbar.remove()

plt.subplots_adjust(wspace=0.10, hspace=0.1)
f.colorbar(disp.im_, ax=axes)
plt.show()
```

Listing 23: Visualise confusion matrix

4 Experimental Results

Model evaluation shows us accuracy of 93% with test data.



```
2/2 [==============================] - 0s 8ms/step - loss: 0.1320 - accuracy: 0.9389
accuracy: 93.89%
```

Figure 2: Evaluation model results

Graph that represent accuracy over the time of epochs. As seen at the begging graph stars in 85% and then raises to 90% to 92%. There is a bit of overfitting in the represented graph. Different methods were tried to reduce that but difference of 1% to 2% between training and validation is the best result that came out.

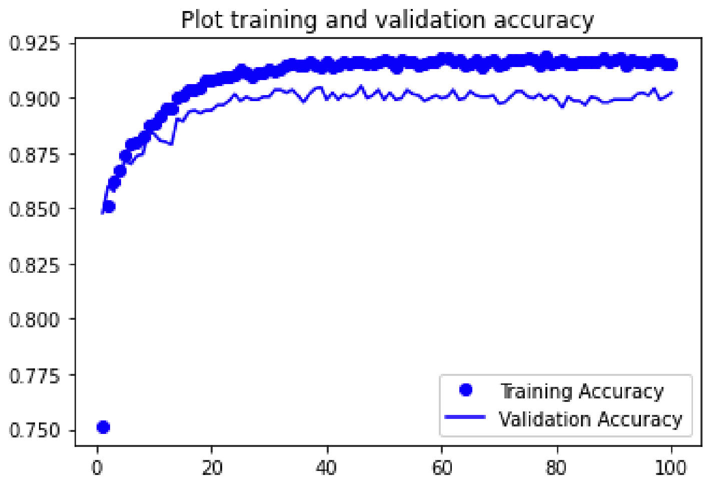


Figure 3: Representation of training and validation accuracy

Representation of loss graph is showcasing training and validation difference. As the graph is showcasing, training and validation are on point.

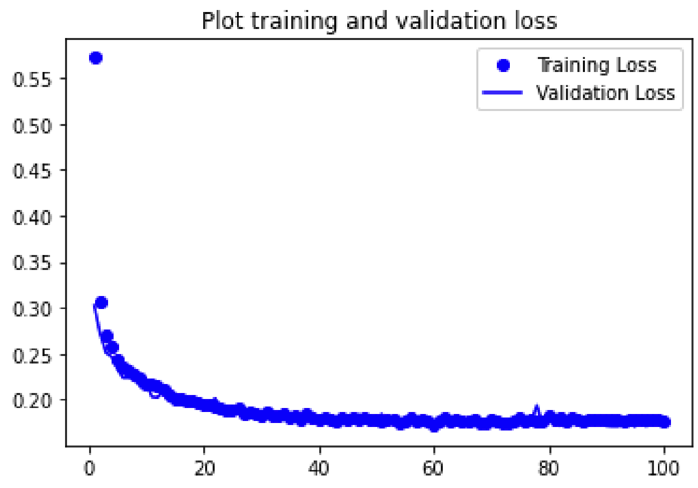
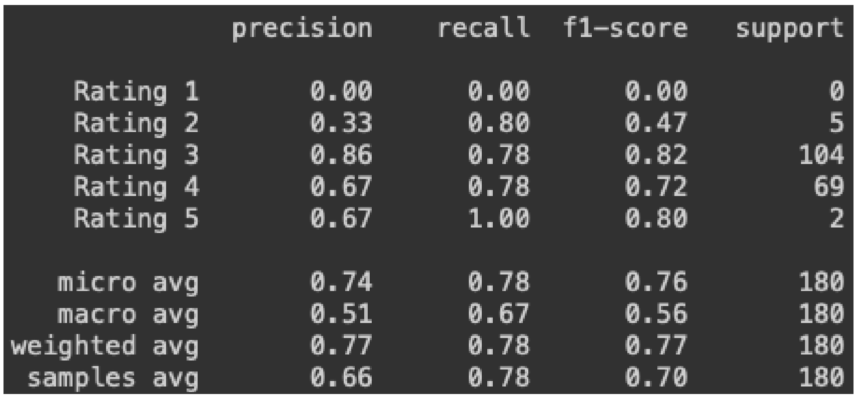


Figure 4: Representation of training and validation loss

With the help of SkLearn library, multi label confusion matrix is calculated and shown. There are 4 things that are represented:

- precision
- recall
- f1-score
- support

Calculations are done of micro, macro, weighted and samples. With that, we can see different calculations based on different scenarios.



```
              precision    recall  f1-score   support

    Rating 1       0.00      0.00      0.00         0
    Rating 2       0.33      0.80      0.47         5
    Rating 3       0.86      0.78      0.82       104
    Rating 4       0.67      0.78      0.72        69
    Rating 5       0.67      1.00      0.80         2

   micro avg       0.74      0.78      0.76       180
   macro avg       0.51      0.67      0.56       180
weighted avg       0.77      0.78      0.77       180
 samples avg       0.66      0.78      0.70       180
```

Figure 5: Representation of calculation confusion matrix

Last figure represents multi label confusion matrix. Each label is represented separately.

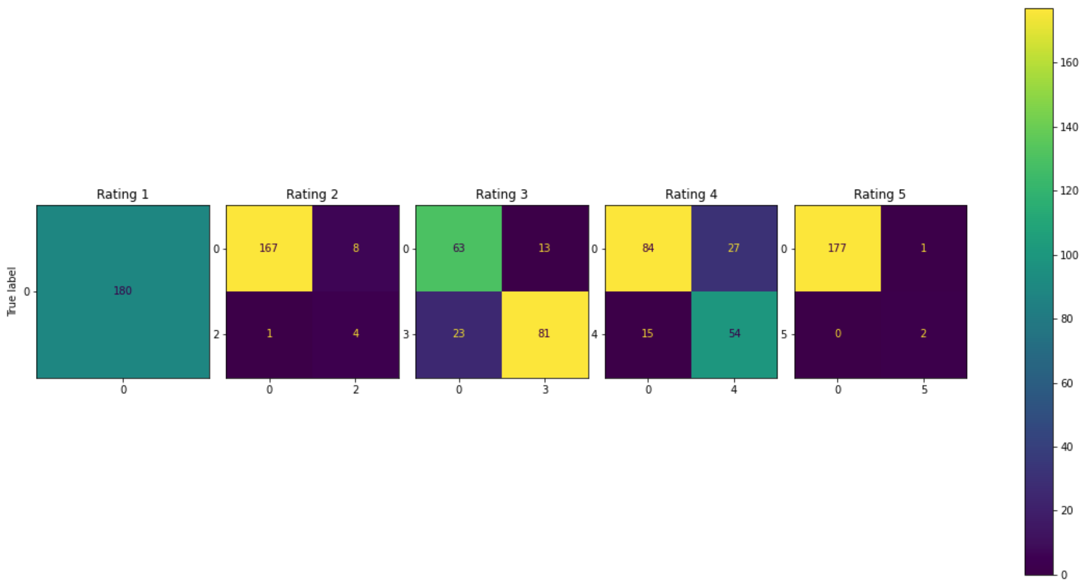


Figure 6: Representation of visualise confusion matrix

5 Summary

At the end of the report, we have successfully created a model that can predict chocolate ratings. With that, concept companies could base their new operations based on the predictions. The model is doing predictions in the code at the moment, this would not be user friendly in the industry. Due to this, we would need to implement some sort of GUI for better access. Hypothetically, if the model would be used and efficient in the industry, if would need to be maintained. With that in mind, data drift can occur. For example, if a country X turns out to have the best conditions and a lot of chocolate industry moves to this, we would have our dataset filled with this country being the best. This can overflow the new locations therefore predictions can be offset.

Overall, there is still room for improvement but that is on the data side as well. For example, if we could get air pollution data, weather data, etc... quality of results would be better. In this report, data was representing kind of a bottleneck since results could be better if we had a bit more diverse dataset. From the model set, at the moment the simple sequential model is being used. If we would build some custom hidden layers, that could result in better accuracy since it would be tailored for our use case.

6 Appendixes

6.1 A1: About the data

Dataset is created by the company FlavorsOfCacao and can be found on Kaggle (FlavorsOfCacaoKaggle n.d.). The company goal is to provide independent chocolate ratings which could be displayed on chocolate bar. Idea they have is for the customer to compare two chocolates not on the price but on the rating. For example, customer would pick up chocolate A and chocolate B in the store, based on the rating customer can decide if chocolate is worth his money. Dataset itself is build on 9 attributes and 1795 records (described in the table bellow).

Dataset attributes	Dataset description
Company (Maker-if known)	• Name of the company that is producing the chocolate (e.g.: Cadbury) • Data type: string
Specific Bean Originor Bar Name	• Where chocolate bar was created • Data type: string
REF	• Value when review was entered in the database • Could be useful if we would be comparing chocolate over time, and how companies have evolved, other than that this attribute will be skipped • Data type: int
ReviewDate	• When the review has been released/added to the website • Data type: int
CocoaPercent	• What is the percentage of cocoa (darkness) in the chocolate • Data type: string
CompanyLocation	• Where is the company based from • Data type: string
Rating	• The mark from 1 (lowest) to 5 (highest) representing quality of the chocolate • Data type: float
BeanType	• Some chocolates do use bean combination, and this can be found here • Data type: string
Broad BeanOrigin	• Where is the broad geo-region of bean origin • Data type: string

Table 1: Dataset attributes described

7 References

References

FlavorsOfCacao (n.d.). *Flavors of cacao*. Last accessed 22 May 2022. URL: http://flavorsofcacao.com/index.html.

FlavorsOfCacaoKaggle (n.d.). *Flavors of cacao Dataset*. Last accessed 22 May 2022. URL: https://www.kaggle.com/datasets/rtatman/chocolate-bar-ratings?datasetId=1919&sortBy=voteCount&searchQuery=neural.

GoogleColab (n.d.). *About Google Colab*. Last accessed 22 May 2022. URL: https://colab.research.google.com.

Jie, Liang et al. (2019). "One-hot encoding and convolutional neural network based anomaly detection". In: *Journal of Tsinghua University (Science and Technology)* 59.7, pp. 523–529.

Kaggle (n.d.). *Kaggle API*. Last accessed 22 May 2022. URL: https://www.kaggle.com/docs/api.

Keras (n.d.). *About Keras*. Last accessed 22 May 2022. URL: https://keras.io.

Rodriguez, Juan D, Aritz Perez, and Jose A Lozano (2009). "Sensitivity analysis of k-fold cross validation in prediction error estimation". In: *IEEE transactions on pattern analysis and machine intelligence* 32.3, pp. 569–575.

Visa, Sofia et al. (2011). "Confusion matrix-based feature selection." In: *MAICS* 710, pp. 120–127.