C1W1_Assignment

December 23, 2020

1 Week 1: Multiple Output Models using the Keras Functional API

Welcome to the first programming assignment of the course! Your task will be to use the Keras functional API to train a model to predict two outputs. For this lab, you will use the **Wine Quality Dataset** from the **UCI machine learning repository**. It has separate datasets for red wine and white wine.

Normally, the wines are classified into one of the quality ratings specified in the attributes. In this exercise, you will combine the two datasets to predict the wine quality and whether the wine is red or white solely from the attributes.

You will model wine quality estimations as a regression problem and wine type detection as a binary classification problem.

Please complete sections that are marked (TODO)

1.1 Imports

[30]:

```
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Input

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import itertools

import utils
```

1.2 Load Dataset

You will now download the dataset from the UCI Machine Learning Repository.

1.2.1 Pre-process the white wine dataset (TODO)

You will add a new column named `is_red` in your dataframe to indicate if the wine is white or red. - In the white wine dataset, you will fill the column `is_red` with zeros (0).

[31]:

```python
## Please uncomment all lines in this cell and replace those marked with `# YOUR CODE HERE`.
## You can select all lines in this code cell with Ctrl+A (Windows/Linux) or Cmd+A (Mac), then press Ctrl+/ (Windows/Linux) or Cmd+/ (Mac) to uncomment.



# # URL of the white wine dataset
URL = 'http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv'

# # load the dataset from the URL
white_df = pd.read_csv(URL, sep=";")

# # fill the `is_red` column with zeros.
white_df["is_red"] = 0

# # keep only the first of duplicate items
white_df = white_df.drop_duplicates(keep='first')
```

[ ]:

```python

```

[32]:

```python
# You can click `File -> Open` in the menu above and open the `utils.py` file
# in case you want to inspect the unit tests being used for each graded function.

utils.test_white_df(white_df)
```

```
All public tests passed
```

[33]:

```python
print(white_df.alcohol[0])
print(white_df.alcohol[100])

# EXPECTED OUTPUT
# 8.8
# 9.1
```

```
8.8
9.1
```

1.2.2 Pre-process the red wine dataset (TODO)

- In the red wine dataset, you will fill in the column `is_red` with ones (1).

[34]:

```
## Please uncomment all lines in this cell and replace those marked with `#␣
↪YOUR CODE HERE`.
## You can select all lines in this code cell with Ctrl+A (Windows/Linux) or␣
↪Cmd+A (Mac), then press Ctrl+/ (Windows/Linux) or Cmd+/ (Mac) to uncomment.



# # URL of the red wine dataset
URL = 'http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/
↪winequality-red.csv'

# # load the dataset from the URL
red_df = pd.read_csv(URL, sep=";")

# # fill the `is_red` column with ones.
red_df["is_red"] = 1

# # keep only the first of duplicate items
red_df = red_df.drop_duplicates(keep='first')
```

[35]:

```
utils.test_red_df(red_df)
```

```
All public tests passed
```

[36]:

```
print(red_df.alcohol[0])
print(red_df.alcohol[100])

# EXPECTED OUTPUT
# 9.4
# 10.2
```

```
9.4
10.2
```

[37]:

```
print(white_df.shape)
print(red_df.shape)
```

```
(3961, 13)
(1359, 13)
```

1.2.3 Concatenate the datasets

Next, concatenate the red and white wine dataframes.

[63]:

```
df = pd.concat([red_df, white_df], ignore_index=True)
```

[64]:

```
print(df.alcohol[0])
print(df.alcohol[100])

# EXPECTED OUTPUT
# 9.4
# 9.5
```

```
9.4
9.5
```

[65]:

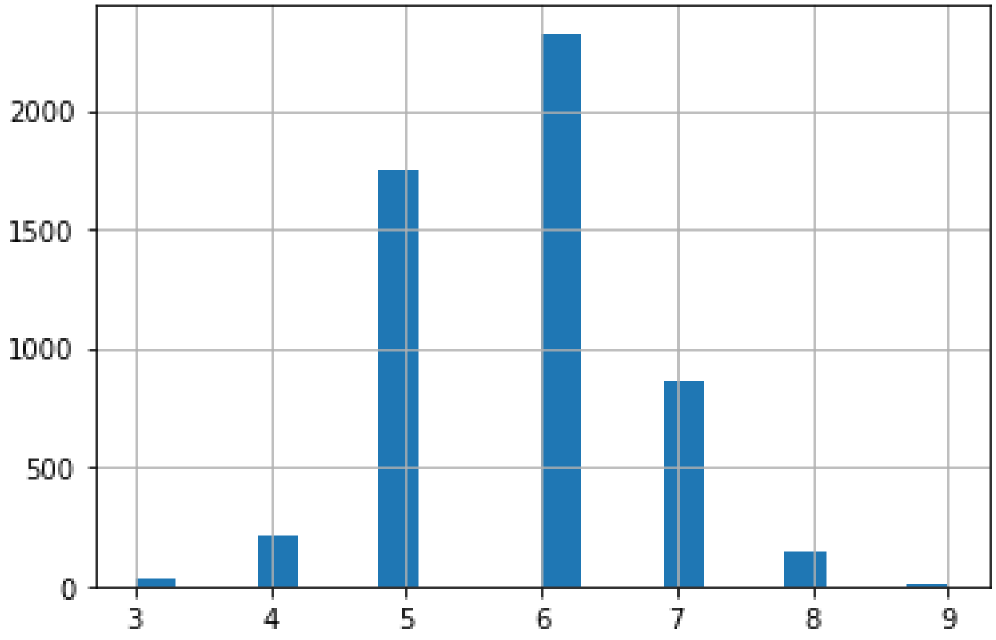
```
# NOTE: In a real-world scenario, you should shuffle the data.
# YOU ARE NOT going to do that here because we want to test
# with deterministic data. But if you want the code to do it,
# it's in the commented line below:

#df = df.iloc[np.random.permutation(len(df))]
```

This will chart the quality of the wines.

[66]:

```
df['quality'].hist(bins=20);
```



[67]:

```
np.unique(df['quality'],return_counts=True)
```

```
[67]: (array([3, 4, 5, 6, 7, 8, 9]),
       array([  30,  206, 1752, 2323,  856,  148,    5]))
```

1.2.4 Imbalanced data (TODO)

You can see from the plot above that the wine quality dataset is imbalanced. - Since there are very few observations with quality equal to 3, 4, 8 and 9, you can drop these observations from your dataset. - You can do this by removing data belonging to all classes except those > 4 and < 8.

```
[68]: ## Please uncomment all lines in this cell and replace those marked with `#␣
       ↪YOUR CODE HERE`.
      ## You can select all lines in this code cell with Ctrl+A (Windows/Linux) or␣
       ↪Cmd+A (Mac), then press Ctrl+/ (Windows/Linux) or Cmd+/ (Mac) to uncomment.




      # # get data with wine quality greater than 4 and less than 8
      df = df[(df['quality'] > 4) & (df['quality'] <8 )]

      # # reset index and drop the old one
      df = df.reset_index(drop=True)
```

```
[69]: np.unique(df['quality'],return_counts=True)
```

```
[69]: (array([5, 6, 7]), array([1752, 2323,  856]))
```

```
[70]: df.alcohol[0]
```

```
[70]: 9.4
```

```
[71]: utils.test_df_drop(df)
```

```
 All public tests passed
```

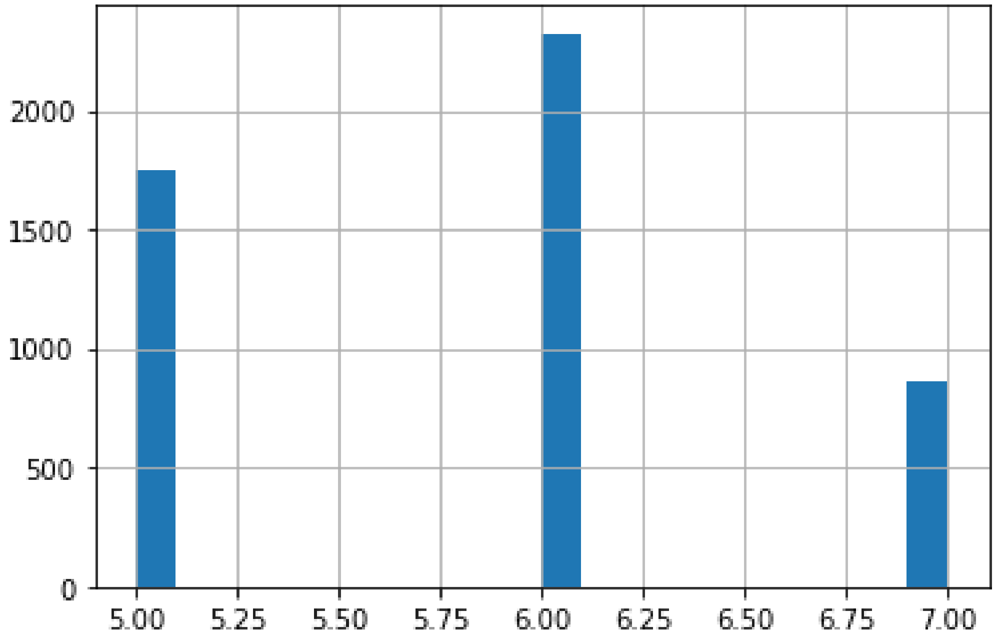
```
[72]: print(df.alcohol[0])
      print(df.alcohol[100])

      # EXPECTED OUTPUT
      # 9.4
      # 10.9
```

```
9.4
10.9
```

You can plot again to see the new range of data and quality

```
[73]: df['quality'].hist(bins=20);
```



1.2.5 Train Test Split (TODO)

Next, you can split the datasets into training, test and validation datasets. - The data frame should be split 80:20 into `train` and `test` sets. - The resulting `train` should then be split 80:20 into `train` and `val` sets. - The `train_test_split` parameter `test_size` takes a float value that ranges between 0. and 1, and represents the proportion of the dataset that is allocated to the test set. The rest of the data is allocated to the training set.

[74]:
```
## Please uncomment all lines in this cell and replace those marked with `#␣
↪YOUR CODE HERE`.
## You can select all lines in this code cell with Ctrl+A (Windows/Linux) or␣
↪Cmd+A (Mac), then press Ctrl+/ (Windows/Linux) or Cmd+/ (Mac) to uncomment.



## Please do not change the random_state parameter. This is needed for grading.

# # split df into 80:20 train and test sets
train, test = train_test_split(df, test_size=0.2, random_state = 1)

# # split train into 80:20 train and val sets
train, val = train_test_split(train, test_size=0.2, random_state = 1)
```

[76]:
```
utils.test_data_sizes(train.size, test.size, val.size)
```

```
All public tests passed
```

Here's where you can explore the training stats. You can pop the labels 'is_red' and 'quality' from the data as these will be used as the labels

[77]:

```
train_stats = train.describe()
train_stats.pop('is_red')
train_stats.pop('quality')
train_stats = train_stats.transpose()
```

Explore the training stats!

[78]:

```
train_stats
```

[78]:

```
                       count        mean        std      min       25%  \
fixed acidity         3155.0    7.221616   1.325297  3.80000   6.40000
volatile acidity      3155.0    0.338929   0.162476  0.08000   0.23000
citric acid           3155.0    0.321569   0.147970  0.00000   0.25000
residual sugar        3155.0    5.155911   4.639632  0.60000   1.80000
chlorides             3155.0    0.056976   0.036802  0.01200   0.03800
free sulfur dioxide   3155.0   30.388590  17.236784  1.00000  17.00000
total sulfur dioxide  3155.0  115.062282  56.706617  6.00000  75.00000
density               3155.0    0.994633   0.003005  0.98711   0.99232
pH                    3155.0    3.223201   0.161272  2.72000   3.11000
sulphates             3155.0    0.534051   0.149149  0.22000   0.43000
alcohol               3155.0   10.504466   1.154654  8.50000   9.50000

                            50%       75%        max
fixed acidity           7.00000    7.7000   15.60000
volatile acidity        0.29000    0.4000    1.24000
citric acid             0.31000    0.4000    1.66000
residual sugar          2.80000    7.6500   65.80000
chlorides               0.04700    0.0660    0.61100
free sulfur dioxide    28.00000   41.0000  131.00000
total sulfur dioxide  117.00000  156.0000  344.00000
density                 0.99481    0.9968    1.03898
pH                      3.21000    3.3300    4.01000
sulphates               0.51000    0.6000    1.95000
alcohol                10.30000   11.3000   14.00000
```

1.2.6 Get the labels (TODO)

The features and labels are currently in the same dataframe. - You will want to store the label columns `is_red` and `quality` separately from the feature columns.
- The following function, `format_output`, gets these two columns from the dataframe (it's given to you). - `format_output` also formats the data into numpy arrays. - Please use the `format_output` and apply it to the `train`, `val` and `test` sets to get dataframes for the labels.

[79]:

```python
def format_output(data):
    is_red = data.pop('is_red')
    is_red = np.array(is_red)
    quality = data.pop('quality')
    quality = np.array(quality)
    return (quality, is_red)
```

[80]:

```python
## Please uncomment all lines in this cell and replace those marked with `# YOUR CODE HERE`.
## You can select all lines in this code cell with Ctrl+A (Windows/Linux) or Cmd+A (Mac), then press Ctrl+/ (Windows/Linux) or Cmd+/ (Mac) to uncomment.



# # format the output of the train set
train_Y = format_output(train)

# # format the output of the val set
val_Y = format_output(val)

# # format the output of the test set
test_Y = format_output(test)
```

[81]:

```python
utils.test_format_output(df, train_Y, val_Y, test_Y)
```

```
All public tests passed
```

[ ]:

```python
```

Notice that after you get the labels, the `train`, `val` and `test` dataframes no longer contain the label columns, and contain just the feature columns. - This is because you used `.pop` in the `format_output` function.

[ ]:

```python
train.head()
```

1.2.7 Normalize the data (TODO)

Next, you can normalize the data, x, using the formula:

$$x_{norm} = \frac{x - \mu}{\sigma}$$

- The `norm` function is defined for you. - Please apply the `norm` function to normalize the dataframes that contains the feature columns of `train`, `val` and `test` sets.

[85]:

```python
def norm(x):
    return (x - train_stats['mean']) / train_stats['std']
```

[86]:
```
## Please uncomment all lines in this cell and replace those marked with `#␣
↪YOUR CODE HERE`.
## You can select all lines in this code cell with Ctrl+A (Windows/Linux) or␣
↪Cmd+A (Mac), then press Ctrl+/ (Windows/Linux) or Cmd+/ (Mac) to uncomment.




# # normalize the train set
norm_train_X = norm(train)

# # normalize the val set
norm_val_X = norm(val)

# # normalize the test set
norm_test_X = norm(test)
```

[87]:
```
utils.test_norm(norm_train_X, norm_val_X, norm_test_X, train, val, test)
```

```
All public tests passed
```

1.3 Define the Model (TODO)

Define the model using the functional API. The base model will be 2 `Dense` layers of 128 neurons each, and have the `'relu'` activation. - Check out the documentation for tf.keras.layers.Dense

[135]:
```
## Please uncomment all lines in this cell and replace those marked with `#␣
↪YOUR CODE HERE`.
## You can select all lines in this code cell with Ctrl+A (Windows/Linux) or␣
↪Cmd+A (Mac), then press Ctrl+/ (Windows/Linux) or Cmd+/ (Mac) to uncomment.




def base_model(inputs):
   # connect a Dense layer with 128 neurons and a relu activation
    x = Dense(128 , activation = "relu")(inputs)

    # connect another Dense layer with 128 neurons and a relu activation
    x = Dense(128 , activation = "relu")(x)
    return x
```

[136]:
```
utils.test_base_model(base_model)
```

```
All public tests passed
```

2 Define output layers of the model (TODO)

You will add output layers to the base model. - The model will need two outputs.

One output layer will predict wine quality, which is a numeric value. - Define a `Dense` layer with 1 neuron. - Since this is a regression output, the activation can be left as its default value `None`.

The other output layer will predict the wine type, which is either red `1` or not red `0` (white). - Define a `Dense` layer with 1 neuron. - Since there are two possible categories, you can use a sigmoid activation for binary classification.

Define the `Model` - Define the `Model` object, and set the following parameters: - `inputs`: pass in the inputs to the model as a list. - `outputs`: pass in a list of the outputs that you just defined: wine quality, then wine type. - **Note**: please list the wine quality before wine type in the outputs, as this will affect the calculated loss if you choose the other order.

[139]:

```python
## Please uncomment all lines in this cell and replace those marked with `# YOUR CODE HERE`.
## You can select all lines in this code cell with Ctrl+A (Windows/Linux) or Cmd+A (Mac), then press Ctrl+/ (Windows/Linux) or Cmd+/ (Mac) to uncomment.



def final_model(inputs):
    x = base_model(inputs)

#     # connect the output Dense layer for regression
    wine_quality = Dense(units='1', name='wine_quality')(x)

#    # connect the output Dense layer for classification. this will use a sigmoid activation.
    wine_type = Dense(units='1', activation="sigmoid", name='wine_type')(x)

#     # define the model using the input and output layers
    model = Model(inputs=inputs, outputs=[wine_quality,wine_type])

    return model
```

[140]:

```python
utils.test_final_model(final_model)
```

```
All public tests passed
```

2.1 Compiling the Model

Next, compile the model. When setting the loss parameter of `model.compile`, you're setting the loss for each of the two outputs (wine quality and wine type).

To set more than one loss, use a dictionary of key-value pairs. - You can look at the docs for the losses here. - **Note**: For the desired spelling, please look at the "Functions" section of the

documentation and not the "classes" section on that same page. - wine_type: Since you will be performing binary classification on wine type, you should use the binary crossentropy loss function for it. Please pass this in as a string.
- **Hint**, this should be all lowercase. In the documentation, you'll see this under the "Functions" section, not the "Classes" section. - wine_quality: since this is a regression output, use the mean squared error. Please pass it in as a string, all lowercase. - **Hint**: You may notice that there are two aliases for mean squared error. Please use the shorter name.

You will also set the metric for each of the two outputs. Again, to set metrics for two or more outputs, use a dictionary with key value pairs. - The metrics documentation is linked here. - For the wine type, please set it to accuracy as a string, all lowercase. - For wine quality, please use the root mean squared error. Instead of a string, you'll set it to an instance of the class RootMeanSquaredError, which belongs to the tf.keras.metrics module.

Note: If you see the error message >Exception: wine quality loss function is incorrect.

- Please also check your other losses and metrics, as the error may be caused by the other three key-value pairs and not the wine quality loss.

[145]:
```
## Please uncomment all lines in this cell and replace those marked with `#␣
↪YOUR CODE HERE`.
## You can select all lines in this code cell with Ctrl+A (Windows/Linux) or␣
↪Cmd+A (Mac), then press Ctrl+/ (Windows/Linux) or Cmd+/ (Mac) to uncomment.



inputs = tf.keras.layers.Input(shape=(11,))
rms = tf.keras.optimizers.RMSprop(lr=0.0001)
model = final_model(inputs)

model.compile(optimizer=rms,
              loss = {'wine_type' : "binary_crossentropy",
                      'wine_quality' : tf.keras.losses.mean_squared_error
                     },
              metrics = {'wine_type' : "accuracy",
                         'wine_quality': tf.keras.metrics.
↪RootMeanSquaredError()                        }
             )
```

[146]:
```
utils.test_model_compile(model)
```

```
All public tests passed
```

2.2 Training the Model

Fit the model to the training inputs and outputs. - Check the documentation for model.fit. - Remember to use the normalized training set as inputs. - For the validation data, please use the normalized validation set.

[151]:

```
## Please uncomment all lines in this cell and replace those marked with `#␣
↪YOUR CODE HERE`.
## You can select all lines in this code cell with Ctrl+A (Windows/Linux) or␣
↪Cmd+A (Mac), then press Ctrl+/ (Windows/Linux) or Cmd+/ (Mac) to uncomment.



history = model.fit(norm_train_X, train_Y,
                    epochs = 180, validation_data=(norm_val_X ,val_Y))
```

```
Train on 3155 samples, validate on 789 samples
Epoch 1/3
3155/3155 [==============================] - 1s 404us/sample - loss: 27.0776 -
wine_quality_loss: 26.3686 - wine_type_loss: 0.6713 -
wine_quality_root_mean_squared_error: 5.1387 - wine_type_accuracy: 0.6181 -
val_loss: 19.3812 - val_wine_quality_loss: 18.7434 - val_wine_type_loss: 0.6397
- val_wine_quality_root_mean_squared_error: 4.3291 - val_wine_type_accuracy:
0.7529
Epoch 2/3
3155/3155 [==============================] - 0s 105us/sample - loss: 12.9828 -
wine_quality_loss: 12.3607 - wine_type_loss: 0.5958 -
wine_quality_root_mean_squared_error: 3.5195 - wine_type_accuracy: 0.7696 -
val_loss: 7.3712 - val_wine_quality_loss: 6.8441 - val_wine_type_loss: 0.5642 -
val_wine_quality_root_mean_squared_error: 2.6089 - val_wine_type_accuracy:
0.7427
Epoch 3/3
3155/3155 [==============================] - 0s 119us/sample - loss: 4.6182 -
wine_quality_loss: 4.0951 - wine_type_loss: 0.5209 -
wine_quality_root_mean_squared_error: 2.0241 - wine_type_accuracy: 0.7493 -
val_loss: 3.0449 - val_wine_quality_loss: 2.6203 - val_wine_type_loss: 0.4853 -
val_wine_quality_root_mean_squared_error: 1.5997 - val_wine_type_accuracy:
0.7427
```

[152]:

```
utils.test_history(history)
```

```
 All public tests passed
```

[153]:

```
# Gather the training metrics
loss, wine_quality_loss, wine_type_loss, wine_quality_rmse, wine_type_accuracy␣
↪= model.evaluate(x=norm_val_X, y=val_Y)

print()
print(f'loss: {loss}')
print(f'wine_quality_loss: {wine_quality_loss}')
print(f'wine_type_loss: {wine_type_loss}')
print(f'wine_quality_rmse: {wine_quality_rmse}')
print(f'wine_type_accuracy: {wine_type_accuracy}')
```

```
# EXPECTED VALUES
# ~ 0.30 - 0.38
# ~ 0.30 - 0.38
# ~ 0.018 - 0.030
# ~ 0.50 - 0.62
# ~ 0.97 - 1.0

# Example:
#0.3657050132751465
#0.3463745415210724
#0.019330406561493874
#0.5885359048843384
#0.9974651336669922
```

```
789/789 [==============================] - 0s 27us/sample - loss: 3.0449 -
wine_quality_loss: 2.6203 - wine_type_loss: 0.4853 -
wine_quality_root_mean_squared_error: 1.5997 - wine_type_accuracy: 0.7427

loss: 3.0448804530202964
wine_quality_loss: 2.6203370094299316
wine_type_loss: 0.48525404930114746
wine_quality_rmse: 1.599674105644226
wine_type_accuracy: 0.7427123188972473
```

2.3 Analyze the Model Performance

Note that the model has two outputs. The output at index 0 is quality and index 1 is wine type

So, round the quality predictions to the nearest integer.

[ ]:

```
predictions = model.predict(norm_test_X)
quality_pred = predictions[0]
type_pred = predictions[1]
```

[ ]:

```
print(quality_pred[0])

# EXPECTED OUTPUT
# 5.6 - 6.0
```

[ ]:

```
print(type_pred[0])
print(type_pred[944])

# EXPECTED OUTPUT
# A number close to zero
# A number close to or equal to 1
```

2.3.1 Plot Utilities

We define a few utilities to visualize the model performance.

[ ]:

```
def plot_metrics(metric_name, title, ylim=5):
    plt.title(title)
    plt.ylim(0,ylim)
    plt.plot(history.history[metric_name],color='blue',label=metric_name)
    plt.plot(history.history['val_' + metric_name],color='green',label='val_' + metric_name)
```

[ ]:

```
def plot_confusion_matrix(y_true, y_pred, title='', labels=[0,1]):
    cm = confusion_matrix(y_true, y_pred)
    fig = plt.figure()
    ax = fig.add_subplot(111)
    cax = ax.matshow(cm)
    plt.title('Confusion matrix of the classifier')
    fig.colorbar(cax)
    ax.set_xticklabels([''] + labels)
    ax.set_yticklabels([''] + labels)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    fmt = 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
          plt.text(j, i, format(cm[i, j], fmt),
                  horizontalalignment="center",
                  color="black" if cm[i, j] > thresh else "white")
    plt.show()
```

[ ]:

```
def plot_diff(y_true, y_pred, title = '' ):
    plt.scatter(y_true, y_pred)
    plt.title(title)
    plt.xlabel('True Values')
    plt.ylabel('Predictions')
    plt.axis('equal')
    plt.axis('square')
    plt.plot([-100, 100], [-100, 100])
    return plt
```

2.3.2 Plots for Metrics

[ ]:

```
plot_metrics('wine_quality_root_mean_squared_error', 'RMSE', ylim=2)
```

[ ]:

```
plot_metrics('wine_type_loss', 'Wine Type Loss', ylim=0.2)
```

2.3.3 Plots for Confusion Matrix

Plot the confusion matrices for wine type. You can see that the model performs well for prediction of wine type from the confusion matrix and the loss metrics.

```
[ ]: plot_confusion_matrix(test_Y[1], np.round(type_pred), title='Wine Type', labels␣
      ↪= [0, 1])
```

```
[ ]: scatter_plot = plot_diff(test_Y[0], quality_pred, title='Type')
```