Artificial intelligence project report

Multi-class Text Classification

Rashin Rahnamoun

Common preprocessing section in all methods:

Google colab environment has been used for implementation . The mount_drive function is in charge of connecting to google drive through which the dev and test data is accessed..

```
def mount_drive(path):
  drive.mount(path)
```

create_new_dataframe function is responsible for reading the input file. The input file uses::: to separate fields . Two columns including description and classification are separated and placed under the titleX column andY column in a Pandas data frame.

```
def create_new_dataframe(input_file):
  dataset = pd.read_csv(input_file, sep = ':::')
  new_df = dataset.iloc[:, [2, 3]].copy()
  new_df.columns = ['X', 'Y']
  return new_df
```

removal function receives the input text and removes their non-numeric characters and similar items . The lemmatize function uses the ready-made spacy ,library and differentiates the roots of words. Under this step, for example plural and singular words will become one. The stopwords function receives the input text and removes the most frequent words from the text . The nltk library .was used to find frequently used English words

```
def removal(text) :
  #print(text)
  soup = BeautifulSoup(text, "html.parser")
  text = soup.get_text()
  text = re.sub('\[[^]]*\]', '', text)
  text = text.translate(text.maketrans("\n\t\r", "   "))
  special_char_pattern = re.compile(r'([{.(-)!}])')
  text = special_char_pattern.sub(" \\1 ", text)
  text = re.sub(r'[^a-zA-Z0-9\s]|\[|\]', '', text)
  return text
```

```
def lemmatize(text):
  text = nlp(text)
  text = ' '.join([word.lemma_ if word.lemma_ != '-PRON-' else word.text for word in text])
  return text
```

```
def stopwords(text):
  tokens = tokenizer.tokenize(text)
  tokens = [token.strip() for token in tokens]
  filtered_tokens = [token for token in tokens if token.lower() not in stop]
  filtered_text = ' '.join(filtered_tokens)
  return filtered_text
```

Finally, reach the text_processing function which receives the path of the input , file and its name, then calls the functions mentioned above as a pipeline to .complete the pre-processing stage

```
def text_preprocessing(path):
  DataFrame = create_new_dataframe(path)
  #DataFrame = DataFrame.apply(removal).apply(lemmatize).apply(stopwords)
  DataFrame['X'] = DataFrame['X'].map(removal)
  DataFrame['X'] = DataFrame['X'].map(lemmatize)
  DataFrame['X'] = DataFrame['X'].map(stopwords)
  return DataFrame
```

Finally, this function is called once for thetrain data and once for thetest data, and the results are saved in the file incsv ,format . As a result, in the next steps . this cleaned file is used as the starting step to save the code execution time

Main part of code to text preprocessing

```
[ ] train_df = text_preprocessing("drive/My Drive/NLP/data/train.txt")
    test_df  = text_preprocessing("drive/My Drive/NLP/data/test.txt")
```

```
[ ] test_df.to_csv('/content/drive/My Drive/NLP/clean_test.csv', index=False)
    train_df.to_csv('/content/drive/My Drive/NLP/clean_train.csv', index=False)
    print(train_df.shape)
    print(test_df.shape)
```

Naive Bayes : method

In the second part of the code, which deals with the implementation of the Naïve Bayes learning method firstly :encounter the following code:

In this area of code, clean training and test loaded into memory

```
[ ] from google.colab import drive
    import pandas as pd
    import numpy as np
```

```
def mount_drive(path):
  drive.mount(path)

mount_drive('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ] TestData = pd.read_csv('/content/drive/My Drive/NLP/clean_test.csv')
    TrainData = pd.read_csv('/content/drive/My Drive/NLP/clean_train.csv')
    print(TestData.shape)
    print(TrainData.shape)
```

```
(7968, 2)
(45149, 2)
```

This part of the code is to load the code from the peripheral memory to the main memory. The drive mounting section has been repeated for it so that the program can be started from this point. Then the two cleaned filestrain andtest . are read from the file in the corresponding path

custom_label_encoder function converts columnY from the input table, which contains text labels, into its numerical equivalent , which is necessary to apply .the algorithm

```
def custom_label_encoder(df, column):
    df[column] = pd.Categorical(df[column]).codes
    return df
```

merge_and_encode function mergesthe test andtrain samples to get a single numeric label in theY column of both data sources. Explanation that the input data must be converted into a single numeric label in the Y column. ,Therefore two tables are merged, a unique label is generated for them, and then they are : again converted into two separate tables

```
[ ] def merge_and_encode(TrainData, TestData, column):
        TrainData['source'] = 'train'
        TestData['source'] = 'test'
        combined_data = pd.concat([TrainData, TestData], ignore_index=True)
        combined_data = custom_label_encoder(combined_data, column)
        TrainData_encoded = combined_data[combined_data['source'] == 'train'].drop(columns=['source'])
        TestData_encoded = combined_data[combined_data['source'] == 'test'].drop(columns=['source']).reset_index(drop=True)
        return TrainData_encoded, TestData_encoded
```

Now this merge_and_encode function has been called, inside which the custom_label_encoder function has been used. For better understanding, the : output of the first two tables are displayed

```
TrainData, TestData = merge_and_encode(TrainData, TestData, 'Y')
#TestData.reset_index(drop=True)
print(TrainData.head())
print(TestData.head())
```

```
                                                   X   Y
0  beginning 21st Century , World War raged stop ...   0
1  Eighteen year-old Alex , 13 year-old Maggie , ...   7
2  Four friends plan camping trip small campgroun...  11
3  series , Ernie Coombes hosts simple formated T...   8
4  According legend , God gave Vincente Ferrer , ...   6
                                                   X   Y
0  Sandro wellknown journalist conduct survey hum...   4
1  young boy life change kidnap sea pirate prison...  18
2  coast Yugoslavia live fisherman Ivo Kralj wife...   7
3  Crime TV show mosaic individual criminal case ...   5
4  Adam lost soul lose girlfriend Amy plum office...  18
```

MyDict tab receives a threshold from the input of the data set, firstly, if punctuation remains in the text, it ignores them, and secondly, it adds words . that have a frequency higher than the threshold to the dictionary

```
import string
from collections import Counter

def MyDict(dataset, threshold):
    token_counter = Counter()
    punctuation_table = str.maketrans('', '', string.punctuation)
    for words in dataset['X'].str.split():
        cleaned_words = [word.translate(punctuation_table) for word in words]
        token_counter.update(cleaned_words)
    filtered_tokens = {word: count for word, count in token_counter.items() if count >= threshold and word}
    vocabulary = {word: idx for idx, word in enumerate(sorted(filtered_tokens))}
    return vocabulary
```

In the next step , the MyBagOfWords function generates BOW .from the input The explanation is that the given sample has a large volume. Therefore, in the step of applying the algorithm, if the normal matrix is used, we will have a memory overflow. The compressed version of the array was also tested and faced the same problem. Therefore, the private version is used in this implementation. The explanation of BOW is within the content of the lesson and will not be repeated here. The working method is to count the words in the text and create a vector of the words in the dictionary in the text and convert it into a numerical vector. Pay attention that in the final line and before returning, a .private version is produced

```
def MyBagOfWords(dataset, vocabulary):
    num_documents = len(dataset)
    num_tokens = len(vocabulary)
    bag_of_words = lil_matrix((num_documents, num_tokens), dtype=int)
    for row in dataset.itertuples(index=True):
        index = row.Index
        words = row.X.split()
        for word in words:
            if word in vocabulary:
                bag_of_words[index, vocabulary[word]] += 1
    bag_of_words = bag_of_words.tocsr()

    return bag_of_words
```

: Finally, this function is called along with the creation of the dictionary

```
vocab = MyDict(TrainData, 4)
print(len(vocab))
bow_train = MyBagOfWords(TrainData, vocab)
bow_test = MyBagOfWords(TestData, vocab)
print(bow_train.shape)
print(bow_test.shape)
```

```
43574
(45149, 43574)
(7968, 43574)
```

But the main body of the code is about the implementation of two key functions naive_bayes_train and naive_bayes_test Before describing these two main . functions, we will discuss two preliminary functions .class_probability probability function every Class in Collection data particle for direct object on basis Label Hi presentation done Calculate may slowdown

```python
def class_probability(labels):
    total_examples = len(labels)
    probabilities = dict(Counter(labels))
    for key in probabilities.keys():
        probabilities[key] = probabilities[key] / float(total_examples)
    return probabilities
```

feature_probability function probability one feature (word) . in one Class Moeen on basis number it, number the whole words in class, size Vocabulary and one parameter smoothing alpha Calculate may slowdown

```python
def feature_probability(feature_counts, total_word_count, vocab_size, alpha):
    total_feature_weight = feature_counts.sum()
    probability = (total_feature_weight + alpha) / (total_word_count + alpha * vocab_size)
    return probability
```

The naive_bayes_train function of theNaive Bayes model with Calculate Possibilities Class and Possibilities Feature for every Class Education may give in every Class ring may zand number the whole words Class particle for direct object Calculate may slow and then with use from feature_probability function probability every Feature in it Class particle for direct object Calculate may slow down

```python
def naive_bayes_train(feature_matrix, labels, vocab, alpha=1.0):
    class_probs = class_probability(labels)
    unique_classes = np.unique(labels)
    num_rows, num_features = feature_matrix.shape

    feature_likelihoods = {}
    for cls in unique_classes:
        feature_likelihoods[cls] = np.zeros(num_features)

    for cls in unique_classes:
        row_indices = np.where(labels == cls)[0]
        class_docs = feature_matrix[row_indices, :]
        total_word_count = class_docs.sum()
        num_docs, vector_length = class_docs.shape

        for i in range(vector_length):
            feature_col = class_docs[:, i]
            feature_likelihoods[cls][i] = feature_probability(feature_col, total_word_count, len(vocab), alpha)

    return class_probs, feature_likelihoods
```

function naive_bayes_test with use Naive Bayes on data Hi new before the nose may slow down Possibilities Report every Class and Features data new particle for direct object Calculate does and classy particle for direct object with the highest possibility Report to title Class predicted choice does

```
def naive_bayes_test(feature_matrix, class_probs, feature_likelihoods):
    log_class_probs = {cls: np.log(prob) for cls, prob in class_probs.items()}
    log_feature_likelihoods = {cls: np.log(likelihoods) for cls, likelihoods in feature_likelihoods.items()}

    predictions = []
    for i in range(feature_matrix.shape[0]):
        current_features = feature_matrix[i]
        class_predictions = {}
        for cls, log_cls_prob in log_class_probs.items():
            log_prob = log_cls_prob
            log_prob += (current_features @ log_feature_likelihoods[cls].T).sum()
            class_predictions[cls] = log_prob
        predicted_class = max(class_predictions, key=class_predictions.get)
        predictions.append(predicted_class)
    return predictions
```

As a result Algorithm Naive Bayes particle for direct object for classification text how many class with Calculate Possibilities Class and possibility Features and prediction on basis this Possibilities Implementation does

The training phase by Naïve Bayes algorithm is time consuming. Therefore, after calling it with the help of the pickle tool, the values of p_class and p_features . are saved in the file

This is a save function with pickle library to save the Naive Bayes generated model. Before that, train the model

```
[ ] import pickle
    def save_naive_bayes_model(class_probs, feature_likelihoods, filepath):
        model = {
            'class_probs': class_probs,
            'feature_likelihoods': feature_likelihoods
        }
        with open(filepath, 'wb') as f:
            pickle.dump(model, f)

    p_class, p_features = naive_bayes_train(bow_train, TrainData['Y'], vocab, 0.5)
    save_naive_bayes_model(p_class, p_features, '/content/drive/My Drive/NLP/naive_bayes_model.pkl')
```

Now this file is saved and to save time, we load it first:

This is a function to load the Naive Bayes model

```
[ ] import pickle
    def load_naive_bayes_model(filepath):
        with open(filepath, 'rb') as f:
            model = pickle.load(f)
        return model['class_probs'], model['feature_likelihoods']

    loaded_class_probs, loaded_feature_likelihoods = load_naive_bayes_model('/content/drive/My Drive/NLP/naive_bayes_model.pkl')
    predictions = naive_bayes_test(bow_test, loaded_class_probs, loaded_feature_likelihoods)
```

Pay attention, after loading the training phase, the previously described test .function is called and the prediction classes are returned to the calling function

Now it's time to display the level of accuracy:

```
from collections import defaultdict

def evaluate_accuracy ( predictions , true_labels ):
correct = sum (pred == true for pred, true in zip (predictions,
true_labels))
    return correct / len (true_labels)

def calculate_precision_recall_f1 ( predictions , true_labels , average =
'macro' ):
class_labels = np.unique(true_labels)
precision_dict = defaultdict( int )
recall_dict = defaultdict( int )
f1_dict = defaultdict( int )

    for label in class_labels:
true_positive = sum ((pred == label) & (true == label) for pred, true in
zip (predictions, true_labels))
predicted_positive = sum (pred == label for pred in predictions)
actual_positive = sum (true == label for true in true_labels)

precision = true_positive / predicted_positive if predicted_positive != 0
otherwise 0
recall = true_positive / actual_positive if actual_positive != 0 otherwise
0
f1_score = 2 * (precision * recall) / (precision + recall) if (precision +
recall) != 0 otherwise 0

precision_dict[label] = precision
recall_dict[label] = recall
f1_dict[label] = f1_score

    if average == 'macro' :
precision = np.mean( list (precision_dict.values()))
recall = np.mean( list (recall_dict.values()))
f1_score = np.mean( list (f1_dict.values()))
    elif average == 'micro' :
true_positive = sum ((pred == true) for pred, true in zip (predictions,
true_labels))
predicted_positive = sum (predictions)
actual_positive = sum (true_labels)
```

```
precision = true_positive / predicted_positive if predicted_positive != 0
otherwise 0
recall = true_positive / actual_positive if actual_positive != 0 otherwise
0
f1_score = 2 * (precision * recall) / (precision + recall) if (precision +
recall) != 0 otherwise 0
    else :
        raise ValueError ( "Unsupported average type. Use 'macro' or
'micro'." )

    return precision, recall, f1_score

# Evaluate accuracy
loaded_accuracy = evaluate_accuracy(predictions, np.array(TestData[ 'Y'
]))
print ( f "Loaded Model Accuracy: {loaded_accuracy * 100:.2f } %" )

# Evaluate precision, recall, and F1 score
precision, recall, f1_score = calculate_precision_recall_f1(predictions,
np.array(TestData[ 'Y' ]), average= 'macro' )

print ( f "Precision: {precision :.2f } " )
print ( f "Recall: {recall :.2f } " )
print ( f "F1 Score: {f1_score :.2f } " )
```

this piece Code including Functions for Calculate criteria Evaluation different for one Model classification is

1 ' The .class_probability probability function every Label Class in Collection ' data particle for direct object Calculate may slow down

2 ' Function .feature_probability probability one Feature particle for direct ' object with attention to number the whole words, size Vocabulary and parameter smoothing Calculate may slow down

3 ' Function .evaluate_accuracy accuracy Model particle for direct object with ' comparison Label Hi before the nose done with Label Hi real Calculate may slow down

Function .4calculate_precision_recall_f1 precision, calling and ScoreF1 particle for direct object for every Label Class Calculate may slow and Values average particle for direct object on basis type average specific done (macro or micro) takes turn

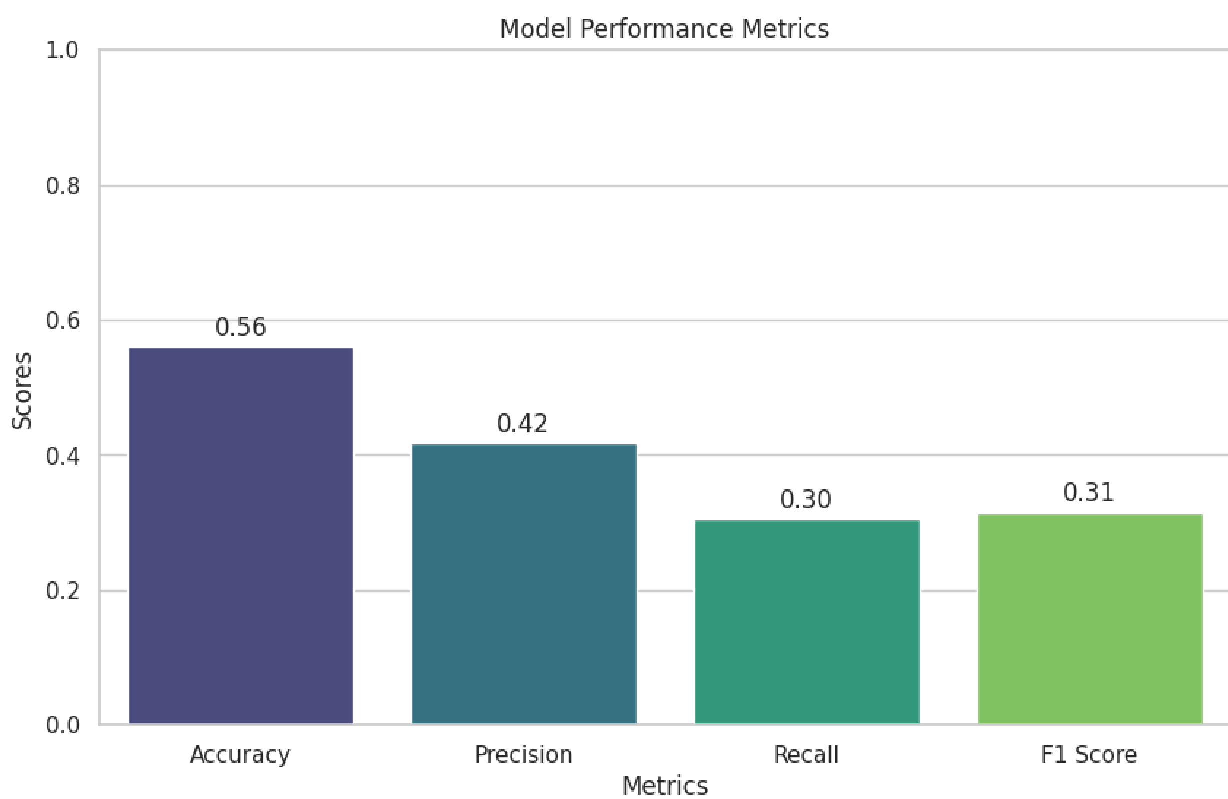
' so from definition this functions, Code with use from Function evaluate_accuracy accuracy Model particle for direct object Evaluation done ' ,and the result particle for direct object Print may slow down then accuracy

calling and ScoreF1 " particle for direct object with use from The calculate_precision_recall_f1 function calculates done and Results particle for " direct object Print does

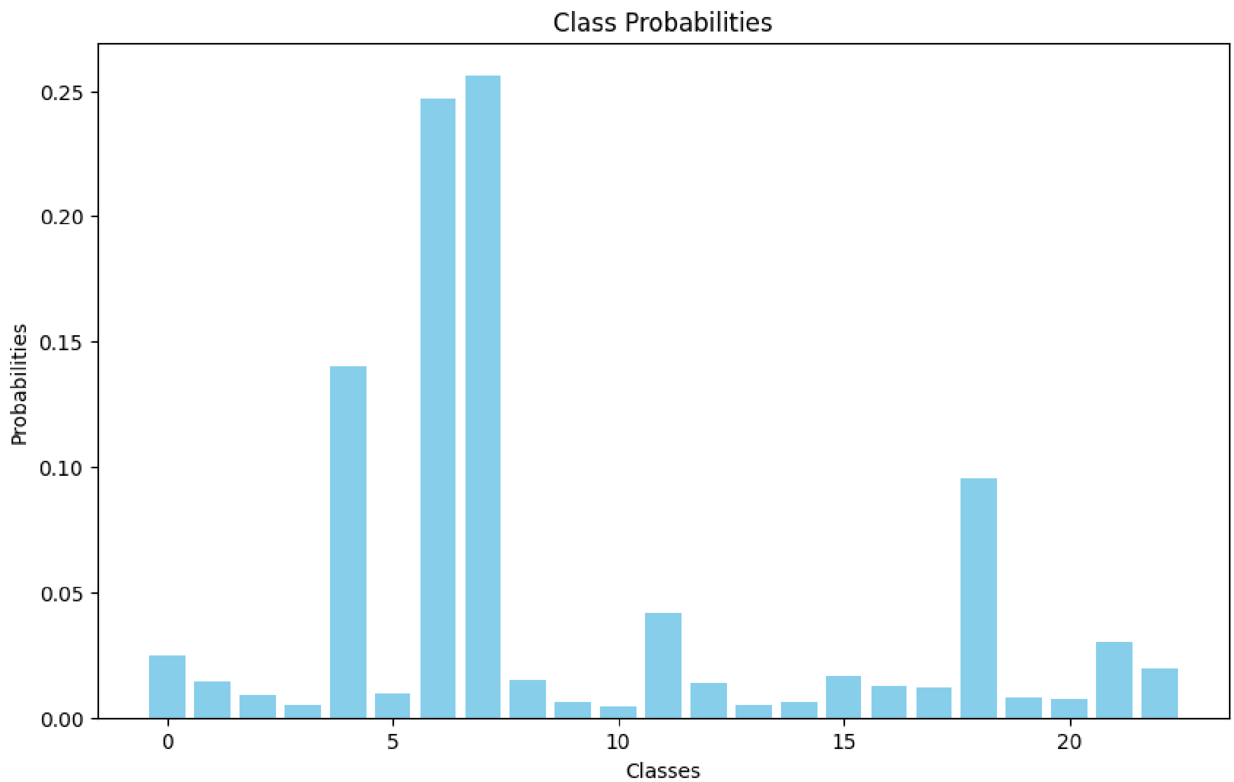
:Regarding the data example of this project, the results are as follows

```
Loaded Model Accuracy: 55.65%
Precision: 0.42
Recall: 0.29
F1 Score: 0.30
```

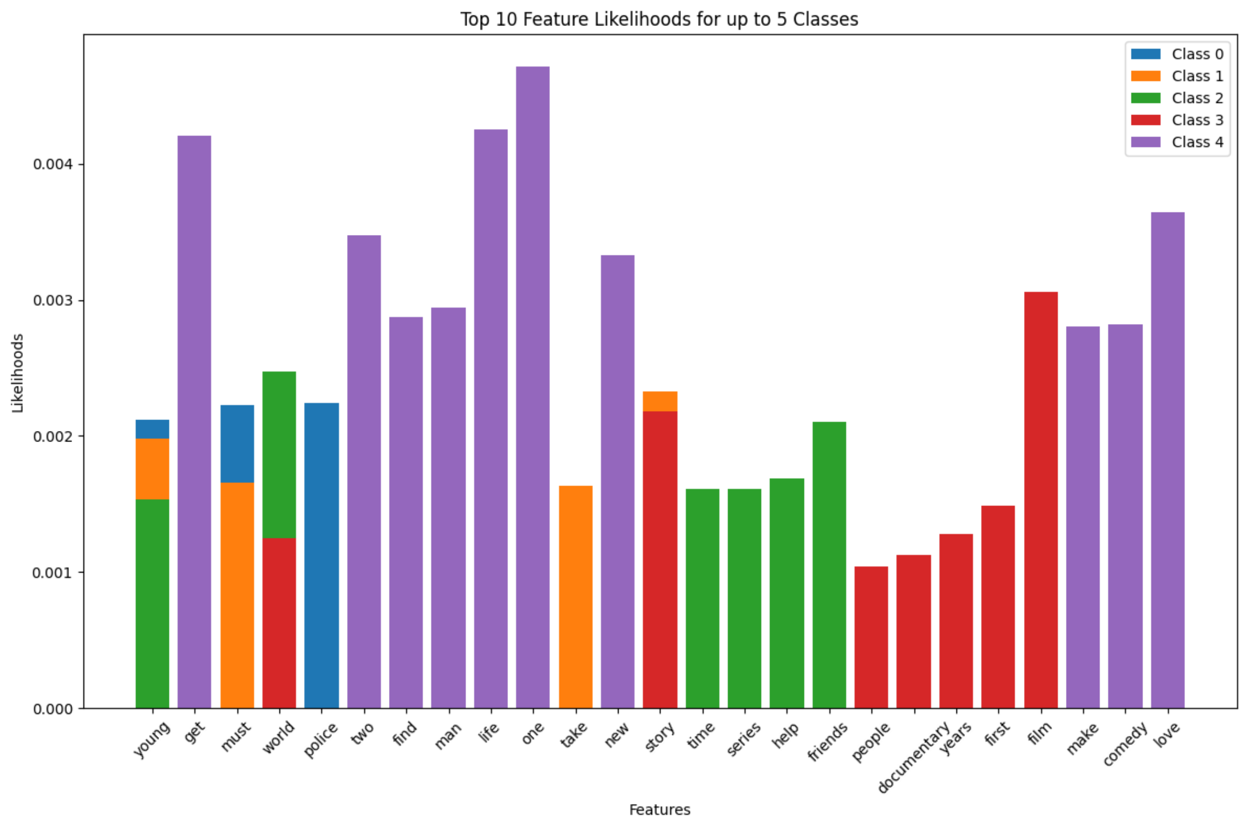
In the text of the program, the same results are displayed as a diagram, which is shown here without the description of the result code:



p-class values can also be seen in the following diagram, of course, the code is available in the program:



On the other hand ,p_features for ten important features with 5 sample classes is as follows:



LSTM neural network method

At first, similar to the previous steps, the cleaned data is uploaded and is not ,explained due to its duplication. In the first stepone-hot encoding is done , that is, a vector of zero and one will be created on theY column or the same labels as the number of classes in this example, meaning the presence or absence of a value for each class, which is required to apply the output to theLSTM network. :with several classes

```
labels = TrainData[ 'Y' ].unique()
label_map = {label: idx for idx, label in enumerate (labels)}
TrainData[ 'Y' ] = TrainData[ 'Y' ]. map (label_map)
# one-hot encoding
cat_labels = to_categorical(TrainData[ 'Y' ], num_classes= len (labels))
```

After this step, theLSTM network :is defined

```
texts = np.array(TrainData[ 'X' ])
train_sentences, val_sentences, train_labels, val_labels =
train_test_split(texts, cat_labels, test_size= 0.25 , random_state= 42 )
# Calculate max length of sequences
```

```
max_len = round ( sum ([ len (i.split()) for i in train_sentences]) / len
(train_sentences))

#TextVectorization layer
max_vocab_len = 10000
text_vector = layers.experimental.preprocessing.TextVectorization(
max_tokens=max_vocab_len,
output_mode= 'int' ,
output_sequence_length=max_len
)
text_vector.adapt(train_sentences)

#Embedding layer
embedding = layers.Embedding(
input_dim=max_vocab_len,
output_dim= 256 ,
input_length=max_len
)

#main model of LSTM network
inputs = layers.Input(shape=( 1 ,), dtype= 'string' )
x = text_vector(inputs)
x = embedding(x)
x = layers.SpatialDropout1D( 0.8 )(x)
x = layers.Bidirectional(layers.LSTM( 300 ,
kernel_regularizer=regularizers.l2( 0.001 )))(x)
x = layers.Dropout( 0.5 )(x)
x = layers.Flatten()(x)
x = layers.Dense( 64 , activation= 'relu' ,
kernel_regularizer=regularizers.l2( 0.001 ))(x)
x = layers.Dropout( 0.5 )(x)
#x = layers.Dense(32, activation='relu',
kernel_regularizer=regularizers.l2(0.001))(x)
outputs = layers.Dense( len (labels), activation= 'softmax' )(x)
```

This part is not so easy to get. Various architectures are tested with various parameter settings until the best result is produced. For example, theLSTM network was tested with two layers, but the results were not better, or increasing the capacity of individual layers, and even removing some layers that are included in the above code . As a result, according to the limited possibilities in terms of GPU access .the above model obtained the best results ,

:In the next section, model execution and trained model storage are done

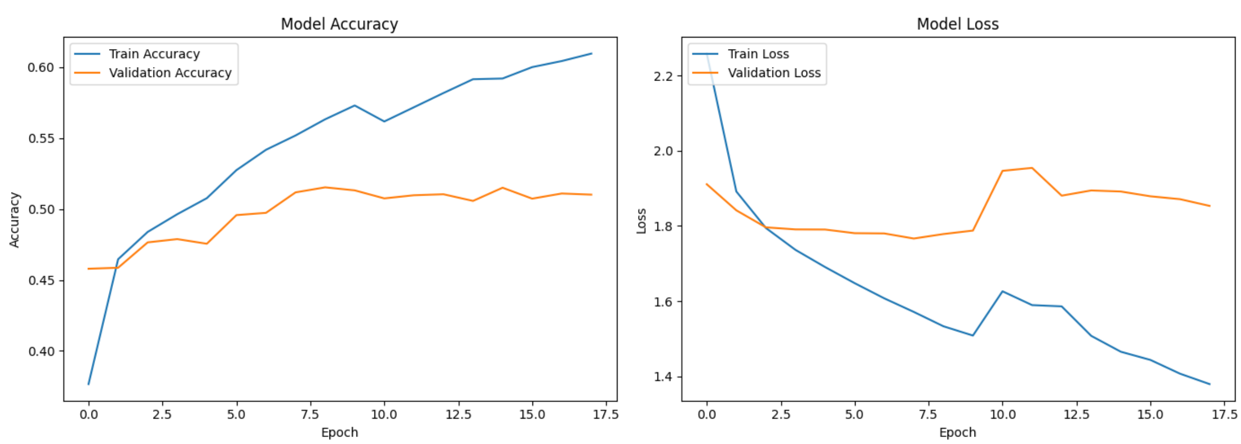
```
lstm = tf.keras.Model(inputs, outputs, name= 'LSTM_MODEL' )
lstm. compile (loss= 'categorical_crossentropy' , optimizer= 'adam' ,
metrics=[ 'accuracy' ])
early_stopping = EarlyStopping(monitor= 'val_loss' , patience= 10 ,
restore_best_weights= True )

# Fit the model
lstm_history = lstm.fit(train_sentences, train_labels, epochs= 20 ,
batch_size= 32 , validation_data=(val_sentences, val_labels),
callbacks=[early_stopping])
# Save the model
lstm.save( '/content/drive/MyDrive/NLP/lstm_model' , save_format= 'tf' )
with open ( '/content/drive/MyDrive/NLP/lstm_history.pkl' , 'wb' ) as file
:
pickle.dump(lstm_history.history, file )
```

As you can see, this learning phase took place in ۲۰ epochs, and this is the output :of each epoch

```
Epoch 1/20
1059/1059 [==============================] - 52s 41ms/step - loss: 2.2581 -
accuracy: 0.3766 - val_loss : 1.9108 - val_accuracy: 0.4579
Epoch 2/20
1059/1059 [==============================] - 24s 23ms/step - loss: 1.8915 -
accuracy: 0.4646 - val_loss : 1.8414 - val_accuracy: 0.4586
Epoch 3/20
1059/1059 [==============================] - 21s 20ms/step - loss: 1.7944 -
accuracy: 0.4839 - val_loss : 1.7962 - val_accuracy: 0.4765
Epoch 4/20
1059/1059 [==============================] - 20s 19ms/step - loss: 1.7363 -
accuracy: 0.4964 - val_loss : 1.7907 - val_accuracy: 0.4788
Epoch 5/20
1059/1059 [==============================] - 19s 18ms/step - loss: 1.6907 -
accuracy: 0.5077 - val_loss : 1.7904 - val_accuracy: 0.4755
Epoch 6/20
1059/1059 [==============================] - 19s 18ms/step - loss: 1.6478 -
accuracy: 0.5275 - val_loss : 1.7807 - val_accuracy: 0.4957
Epoch 7/20
1059/1059 [==============================] - 19s 18ms/step - loss: 1.6075 -
accuracy: 0.5418 - val_loss : 1.7800 - val_accuracy: 0.4973
Epoch 8/20
1059/1059 [==============================] - 21s 20ms/step - loss: 1.5714 -
accuracy: 0.5519 - val_loss : 1.7664 - val_accuracy: 0.5118
Epoch 9/20
1059/1059 [==============================] - 19s 18ms/step - loss: 1.5333 -
accuracy: 0.5633 - val_loss : 1.7784 - val_accuracy: 0.5153
Epoch 10/20
1059/1059 [==============================] - 19s 18ms/step - loss: 1.5085 -
accuracy: 0.5730 - val_loss : 1.7876 - val_accuracy: 0.5132
```

```
Epoch 11/20
1059/1059 [==============================] - 19s 18ms/step - loss: 1.6262 -
accuracy: 0.5618 - val_loss : 1.9464 - val_accuracy: 0.5075
Epoch 12/20
1059/1059 [==============================] - 19s 18ms/step - loss: 1.5895 -
accuracy: 0.5718 - val_loss : 1.9542 - val_accuracy: 0.5097
Epoch 13/20
1059/1059 [==============================] - 19s 18ms/step - loss: 1.5861 -
accuracy: 0.5818 - val_loss : 1.8804 - val_accuracy: 0.5105
Epoch 14/20
1059/1059 [==============================] - 19s 18ms/step - loss: 1.5077 -
accuracy: 0.5916 - val_loss : 1.8943 - val_accuracy: 0.5058
Epoch 15/20
1059/1059 [==============================] - 19s 18ms/step - loss: 1.4655 -
accuracy: 0.5921 - val_loss : 1.8915 - val_accuracy: 0.5151
Epoch 16/20
1059/1059 [==============================] - 18s 17ms/step - loss: 1.4438 -
accuracy: 0.6001 - val_loss : 1.8787 - val_accuracy: 0.5074
Epoch 17/20
1059/1059 [==============================] - 19s 18ms/step - loss: 1.4072 -
accuracy: 0.6044 - val_loss : 1.8711 - val_accuracy: 0.5110
Epoch 18/20
1059/1059 [==============================] - 19s 18ms/step - loss: 1.3795 -
accuracy: 0.6096 - val_loss : 1.8533 - val_accuracy: 0.5102
```

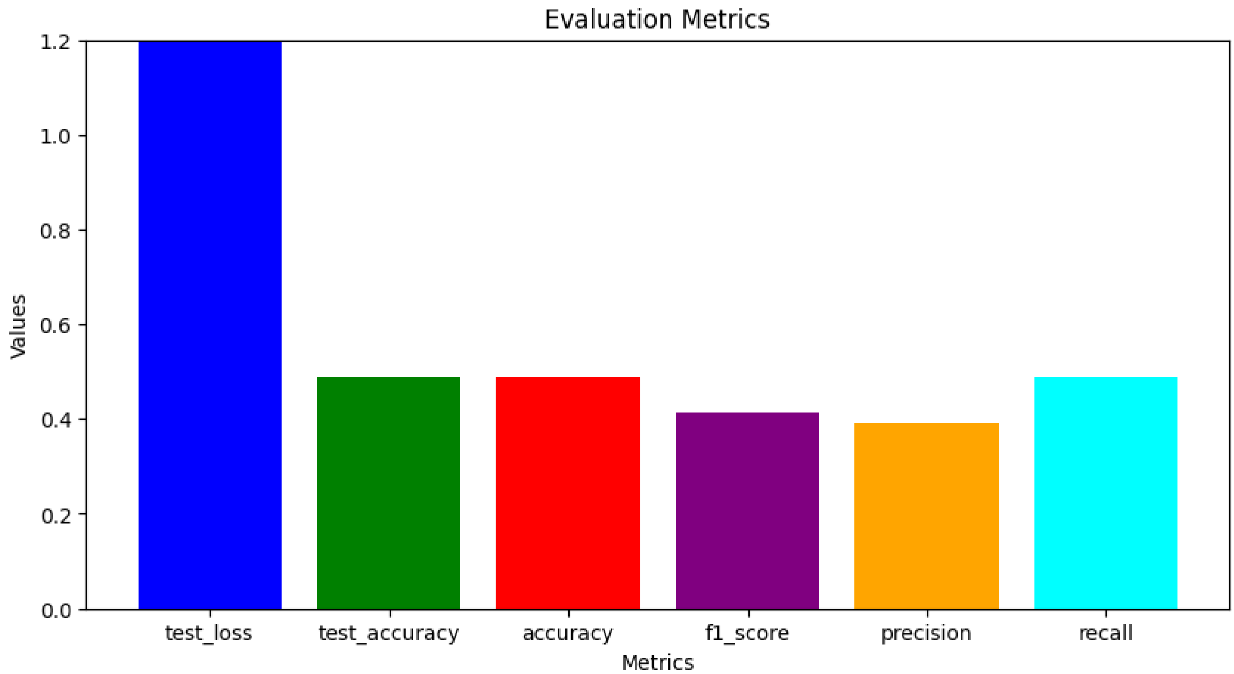
Again, the code related to the comparison of training steps from the model
:memory is not mentioned, but the results are shown in the diagram below



The last step is to display the evaluation results on the test data. The graph and
:results are as follows and the code is in the text

```
249/249 [==============================] - 2s 7ms/step - loss: 1.8224 -
accuracy: 0.4874
Test Loss: 1.8223505020141602
Test Accuracy: 0.4874497950077057
249/249 [==============================] - 2s 6ms/step
/usr/local/lib/python3.10/dist-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
```

```
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
_warn_prf(average, modifier, msg_start, len(result))
Accuracy: 0.48744979919678716
F1 Score: 0.41216515097282114
Precision: 0.3909976864758746
Recall: 0.48744979919678716
```



Fine-tune method onBERT large language model

An important part of this code is the problems of preparing the input data for the pre-trainedBERT model The training of this model is very slow and . .therefore it is possible to change it with difficulty and spending a lot of time

First, columnY : is coded and tokenization is done based on Brett model

```
unique_labels = TrainData[ 'Y' ].unique()
label2id = {label: idx for idx, label in enumerate (unique_labels) }
id2label = {idx: label for label, idx in label2id.items()}

tokenizer = AutoTokenizer.from_pretrained( "distillbert-base-uncased" )
```

Then the data is divided into three parts: training, testing andvalidation of , .course, the test data is generated from the given sample

```
def preprocess_function ( example ):
```

```
    return tokenizer(example[ "X" ], truncation= True , padding= True ,
max_length= 512 )

train_df, val_df = train_test_split(TrainData, test_size= 0.25 ,
random_state= 42 )

tokenized_train = Dataset.from_pandas(train_df). map (preprocess_function,
batched= True )
tokenized_val = Dataset.from_pandas(val_df). map (preprocess_function,
batched= True )
tokenized_test = Dataset.from_pandas(TestData). map (preprocess_function,
batched= True )

tokenized_train = tokenized_train.add_column( "labels" , [label2id[label]
for label in train_df[ 'Y' ]])
tokenized_val = tokenized_val.add_column( "labels" , [label2id[label] for
label in val_df[ 'Y' ]])
tokenized_test = tokenized_test.add_column( "labels" , [label2id[label]
for label in TestData[ 'Y' ]])

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

Then, on the pre-trained BERT model, a new model is defined , the description of which is detailed and requires a review of theTransformer model and BERT :structure and much more details

```
accuracy = evaluate.load( "accuracy" )

def compute_metrics ( eval_pred ):
predictions, labels = eval_pred
predictions = np.argmax(predictions, axis= 1 )
    return accuracy.compute(predictions=predictions, references=labels)

model = AutoModelForSequenceClassification.from_pretrained(
    "distillbert-base-uncased" ,
num_labels= len (unique_labels),
id2label=id2label,
label2id=label2id
)

training_args = TrainingArguments(
output_dir = "my_model" ,
learning_rate= 2e-5 ,
per_device_train_batch_size= 16 ,
per_device_eval_batch_size= 16 ,
```

```
num_train_epochs= 3 ,
weight_decay= 0.01 ,
evaluation_strategy= "epoch" ,
save_strategy= "epoch" ,
load_best_model_at_end= True ,
)

trainer = Trainer(
model=model,
args=training_args,
train_dataset=tokenized_train,
eval_dataset=tokenized_val,
tokenizer=tokenizer,
data_collator=data_collator,
compute_metrics=compute_metrics,
)
```

:Finally, the training is started and the resulting model is saved

```
trainer.train()

trainer.save_model( "/content/drive/MyDrive/NLP/fine_tuned_bert" )
```

Pay attention that the training time is long and the results are as follows:

[6351/6351 1:33:54, Epoch 3/3]

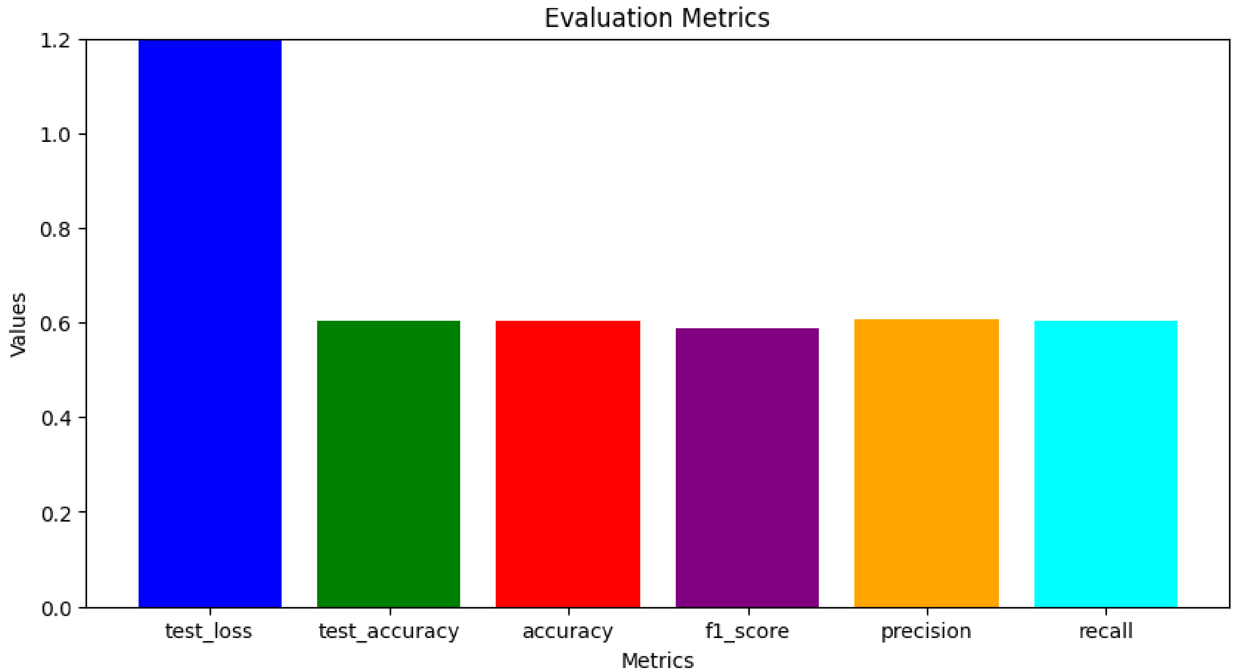
Epoch	Training Loss	Validation Loss	Accuracy
1	1.225200	1.169779	0.647147
2	0.978300	1.129979	0.654235
3	0.804100	1.140417	0.655652

In the following, similar to the above, the code related to the measurement of :the results compared to the test data, the results are as follows

```
Test Results: {'eval_loss': 1.3145439624786377, 'eval_accuracy':
0.6036646586345381, 'eval_runtime': 126.6543, 'eval_samples_per_second':
62.911, 'eval_steps_per_second': 3.932}
/usr/local/lib/python3.10/dist-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
_warn_prf(average, modifier, msg_start, len(result))
Accuracy: 0.6036646586345381
F1 Score: 0.5880167395748513
Precision: 0.606679099669115
```

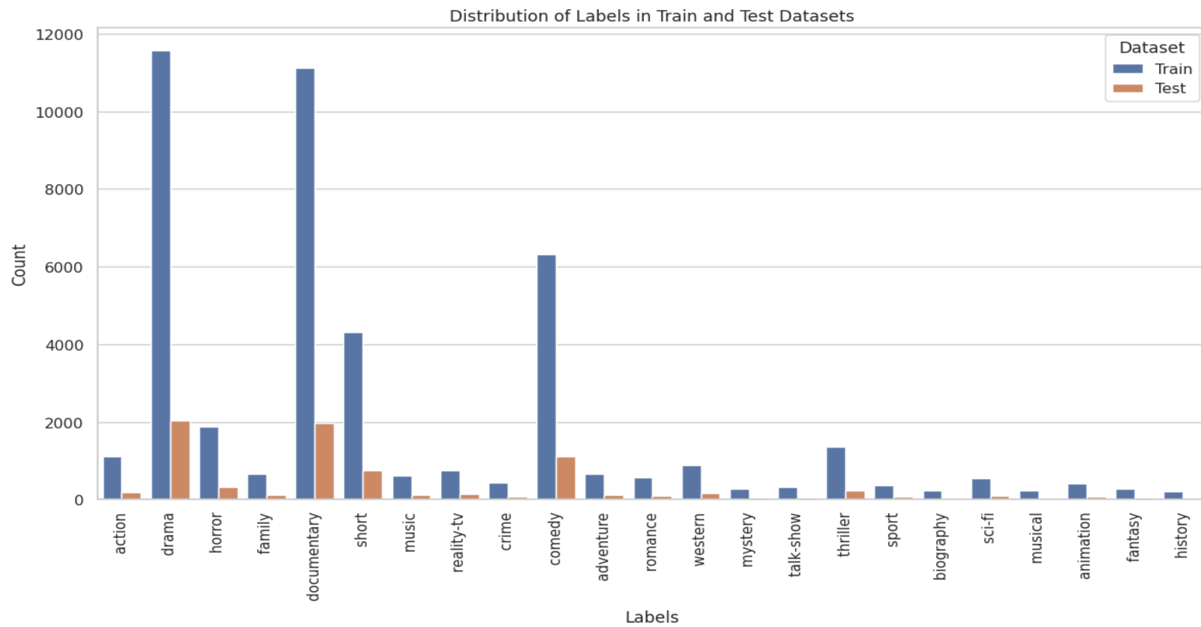
Recall: 0.6036646586345381

:The following figure shows the same results graphically



The problem of data set imbalance

The preliminary investigation on the given training and test sample data shows that according to the given labels, the data is unbalanced with respect to the labels. The following graph shows theTrain andTest data in a graph relative to the distribution of labels:

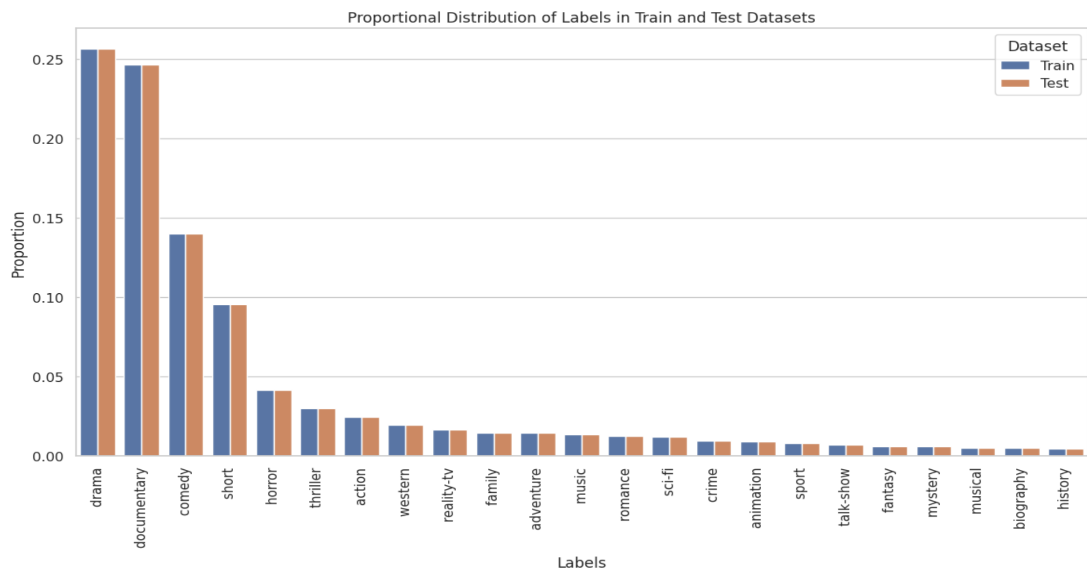


This chart above has a serious problem. Although the difference in the distribution of labels can be seen in it, the ratio of Train and Test data is ambiguous due to the unequal number. Therefore, the following code was written to balance the data

```
# Calculate the proportions of each label within each dataset
train_counts = TrainData[ 'Y' ].value_counts(normalize= True
).reset_index()
test_counts = TestData[ 'Y' ].value_counts(normalize= True ).reset_index()
train_counts.columns = [ 'Y' , 'Proportion' ]
test_counts.columns = [ 'Y' , 'Proportion' ]
train_counts[ 'Dataset' ] = 'Train'
test_counts[ 'Dataset' ] = 'Test'

# Combine the proportion dataframes
proportions_data = pd.concat([train_counts, test_counts], ignore_index=
True )

sns.set_theme(style= "whitegrid" )
plt.figure(figsize=( 14 , 7 ))
sns.barplot(x= 'Y' , y= 'Proportion' , hue= 'Dataset' ,
data=proportions_data)
plt.xticks(rotation= 90 )
plt.title( 'Proportional Distribution of Labels in Train and Test
Datasets' )
plt.xlabel( 'Labels' )
```

```
plt.ylabel( 'Proportion' )
plt.legend(title= 'Dataset' )
plt.show()
```

:The result of the above code is as follows



scaled graph clearly shows that although the distribution of data labels is not equal, but in the two given samples, training and testing follow the same .distribution shape

The lack of balance in the input data can have a serious negative effect on the classification results . Therefore, in the next step, I went to study common methods to deal with unbalanced data. Among these methods, with two methodsof oversampling and undersampling and using the imblearn ready library and again with the sklearn library, testing was done on several basic methods such as Naïve Bayes, KNN, logistic regression . In the undersampling method the total , .distribution of samples was equal to the minimum number of labeled samples Therefore, the training data was greatly reduced. The result was basically useless and was worse than the initial state, which is unbalanced data. The reason was clear. The last tags in the graph above have very few samples, and reducing all the tag samples to such a low number did not produce useful results. In the case of oversampling, only theNaïve Bayes method was tested because there was a practical problem. By repeatedly adding the sample to the number of labels with a small sample, the sample volume became very large, and as a result, the training phase was too slow or even encountered the problem of memory overflow . But .in the same tested case, there was no significant improvement in the result

Maybe the reason for this is the balance betweenTest and Train samples , which . did not have much effect on the result, at least with these common methods

Summary

In this lesson project, due to the time limit, three methods were tested. The traditionalNaïve Bayes method using the ,LSTM network and using theBERT network on which the training data wasfine-tuned Unfortunately, due to time . constraints, I did not manage to implement the neural network or the development of a neural network based on the BERT model .

The results obtained from Naïve Bayes were very good due to the simplicity of the method and the higher speed of implementation . The details of the results are .mentioned in the report

LSTM method had weaker results than the Naïve Bayes method after several different architectures of one or two layers and changes in its parameters, which is worthy of attention.

The pre-trained BERT network method on whichFine-tuning ,was performed although it produced better results than the other methods, it was not far from Naïve Bayes . but the training time was much longer ,

As a result, the use of Naïve Bayes was practically the best implementation method , and of course, it is also related to the execution time , but if we only look at the test result, the BERT- based method produced the best result on the test data.