ShritejShrikant_Chavan_HW_5_PartA

October 12, 2023

1 HW5 - MultiClass Classification with DistilBert - 14 Points

- **Fill the missing code indicated by # CODE HERE**
- Make sure to run all the cells in the Notebook and try to understand the code.
- ***The submitted notebook shoud have output visible for all the cells**
- **The HW assumes that you have gone though the 2_imdb_bert.ipynb file from Lecture 6**
- **You have to submit two files for this part of the HW** >(1) ipynb (colab notebook) and >(2) pdf file (pdf version of the colab file).**
- **Files should be named as follows**: >FirstName_LastName_HW_5_PartA**

1.1 Outline

1. **Setting up the Environment**: Installing necessary libraries and setting up paths.
2. **Creating Huggingface Dataset for Custom Dataset**: Understanding the structure and content of the dataset.
3. **Data Preprocessing**: Techniques to prepare the data for training, including handling different data splits and tokenization
4. **Training the Model**: Feeding data and adjusting weights.
5. **Inference**: Evaluate model on test set and making predictions.

2 Setting up the Environment

[2]:
```
# CHANGE FOLDERS AS PER YOUR SETUP
from pathlib import Path
if 'google.colab' in str(get_ipython()):
    from google.colab import drive
    drive.mount("/content/drive")
    !pip install datasets transformers evaluate wandb accelerate -U -qq
    base_folder = Path("/content/drive/MyDrive/NLP")
else:
    base_folder = Path("/home/harpreet/Insync/google_drive_shaannoor/data")
```

```python
from transformers import AutoConfig, AutoModelForSequenceClassification,␣
↪AutoTokenizer, Trainer, TrainingArguments
from transformers import AutoTokenizer, DataCollatorWithPadding, pipeline
from datasets import load_dataset, DatasetDict, Dataset, ClassLabel
import evaluate

import torch
from torch.utils.data import DataLoader

import wandb

import numpy as np
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import random

import textwrap
```

```
Mounted at /content/drive
                                         519.6/519.6

kB 6.1 MB/s eta 0:00:00
                                          7.7/7.7 MB
68.8 MB/s eta 0:00:00
                                          81.4/81.4 kB
9.6 MB/s eta 0:00:00
                                          2.1/2.1 MB
68.7 MB/s eta 0:00:00
                                         258.1/258.1 kB
24.5 MB/s eta 0:00:00
                                         115.3/115.3

kB 9.4 MB/s eta 0:00:00
                                         194.1/194.1 kB
19.4 MB/s eta 0:00:00
                                         134.8/134.8 kB
15.3 MB/s eta 0:00:00
                                         302.0/302.0 kB
29.5 MB/s eta 0:00:00
                                          3.8/3.8 MB
94.2 MB/s eta 0:00:00
                                          1.3/1.3 MB
79.6 MB/s eta 0:00:00
                                         190.0/190.0 kB
23.3 MB/s eta 0:00:00
                                         241.0/241.0 kB
28.3 MB/s eta 0:00:00
```

```
  Preparing metadata (setup.py) … done
                                 62.7/62.7 kB
7.5 MB/s eta 0:00:00
                                295.0/295.0 kB
33.9 MB/s eta 0:00:00
  Building wheel for pathtools (setup.py) … done
```

[3]:
```
# CHANGE FOLDERS TO WHERE YOU WANT TO SAVE DATA AND MODELS
data_folder = base_folder/'datasets/Classification_HW/csv_files'
model_folder = base_folder/'models/nlp_spring_2023/HW5'
model_folder.mkdir(exist_ok=True)
```

[4]:
```
def print_wrap(text, d):
    # Wrap the text to limit the width to 'd'
    wrapped_text = textwrap.fill(text, width=d)

    # Print the wrapped text
    print(wrapped_text)
```

3 Exploring and Understanding Dataset

3.1 Stack Exchange MultiClass Dataset

- In this HW, you will identify tags for stack exchange Questions.
- This data is a subset of data available in a Kaggle Competition.
- The given dataset has different questions asked in the StackExchange website for various technical domains.
- We have fetched only those questions that contain the top 10 individual tags.
- **Each question has only one tag. This means that this is a multi-class classification problem.**
- These are the ten categories for tags in the data.

Index	Tag
0	C#
1	java
2	php
3	javascript
4	android
5	jquery
6	c++
7	python
8	iphone
9	asp.net

3.2 Load Data set

[5]:
```
# The file 'multiclass_hw_basic_clean.csv' is availaible on e-Learning␣
↪0_data_Folder
# Make sure that you specify the correct path
# The file name need to be in the string, thaat is why we have used␣
↪str(file_path)
# We loaded imdb dataset from huggingface
# in this case we are creating a hugginmgface dataset from csv file
stack_dataset = load_dataset('csv', data_files= str(data_folder /
↪'multiclass_hw_basic_clean.csv'))
```

```
Downloading data files:   0%|          | 0/1 [00:00<?, ?it/s]

Extracting data files:   0%|          | 0/1 [00:00<?, ?it/s]

Generating train split: 0 examples [00:00, ? examples/s]
```

3.3 Understanding your data

[6]:
```
print(stack_dataset)
```

```
DatasetDict({
    train: Dataset({
        features: ['Unnamed: 0.1', 'Unnamed: 0', 'Title', 'Body',
'cleaned_text', 'Tags', 'Tag_Number_final', 'combined_text',
'basic_cleaned_text'],
        num_rows: 188878
    })
})
```

3.4 Understanding the datatype of columns

[7]:
```
stack_dataset['train'].features
```

[7]:
```
{'Unnamed: 0.1': Value(dtype='int64', id=None),
 'Unnamed: 0': Value(dtype='int64', id=None),
 'Title': Value(dtype='string', id=None),
 'Body': Value(dtype='string', id=None),
 'cleaned_text': Value(dtype='string', id=None),
 'Tags': Value(dtype='string', id=None),
 'Tag_Number_final': Value(dtype='int64', id=None),
 'combined_text': Value(dtype='string', id=None),
 'basic_cleaned_text': Value(dtype='string', id=None)}
```

- As you can see the dataset has lot of faeatures. However they are not all useful.
- Title is the title of the stack exchange post
- Body is the main text of the post

- combined_text is Title and Body combined with no pre-processing
- basic_cleaned_text is Title and Body combined with basic preprocessing (remove html tags, urls, emails).
- cleaned_text - Here we have combined Body and Text and has done some motre pre-propcessing in addition to basic (removing stopwords, lammetization)
- Tags - names of programming language to which the post belongs
- Tag_Number_final - index corresponding to Tags
- **Your goal in this HW is to predict Tags given Body and Title of the post**
- **You will use Tag_Number_final and basic_cleaned_text for this HW**

3.5 Acess indivdual element

[8]:
```
print_wrap(stack_dataset['train']['Body'][0], 80)
```

```
<p>Is there a simple way to place a detail disclosure icon on a UIButton? I'm
using a navigation controller and I want a button press to push a new view on
the stack, so I thought a detail disclosure icon would be appropriate, but I
haven't found a straightforward way to do that yet.</p>  <p>What I have in mind
is something like the "When Timer Ends" button in the Timer subview of the Clock
app.</p>
```

[9]:
```
print_wrap(stack_dataset['train']['Title'][0], 80)
```

```
detail disclosure indicator on UIButton
```

[10]:
```
print_wrap(stack_dataset['train']['basic_cleaned_text'][0], 80)
```

```
detail disclosure indicator on UIButton Is there a simple way to place a detail
disclosure icon on a UIButton? I'm using a navigation controller and I want a
button press to push a new view on the stack, so I thought a detail disclosure
icon would be appropriate, but I haven't found a straightforward way to do that
yet. What I have in mind is something like the "When Timer Ends" button in the
Timer subview of the Clock app.
```

[11]:
```
# get label of last ten examples
stack_dataset['train']['Tag_Number_final'][-10:]
```

[11]:
```
[4, 4, 9, 5, 5, 4, 7, 3, 4, 2]
```

[12]:
```
# Assuming 'stack_dataset' is a huggingface dataset
# Select only the desired columns and rename them
selected_columns = {
    'text': stack_dataset['train']['basic_cleaned_text'],
    'label': stack_dataset['train']['Tag_Number_final']
}
```

```
# Create a new dataset with the selected columns
stack_selected_columns = Dataset.from_dict(selected_columns)
```

[13]:
```
stack_selected_columns
```

[13]:
```
Dataset({
    features: ['text', 'label'],
    num_rows: 188878
})
```

[14]:
```
stack_selected_columns.features
```

[14]:
```
{'text': Value(dtype='string', id=None),
 'label': Value(dtype='int64', id=None)}
```

[15]:
```
stack_selected_columns['label'][:10]
```

[15]:
```
[8, 4, 3, 9, 4, 0, 3, 2, 0, 7]
```

[16]:
```
print_wrap(stack_selected_columns['text'][0], 80)
```

```
detail disclosure indicator on UIButton Is there a simple way to place a detail
disclosure icon on a UIButton? I'm using a navigation controller and I want a
button press to push a new view on the stack, so I thought a detail disclosure
icon would be appropriate, but I haven't found a straightforward way to do that
yet. What I have in mind is something like the "When Timer Ends" button in the
Timer subview of the Clock app.
```

3.6 Exploratory Data Analysis (EDA)

3.6.1 Change dataset format to Pandas

[17]:
```
# Set the format to Pandas
# CODE HERE

stack_selected_columns.set_format(type='pandas')
```

[18]:
```
# get all rows the dataset
df = stack_selected_columns[:]
```

[19]:
```
df.head()
```

[19]:
```
                                                text  label
0  detail disclosure indicator on UIButton Is the…      8
1  hello world fails to show up in emulator I fol…      4
2  Why is JSHint throwing a "possible strict viol…      3
3  Programmatically Make Bound Column Invisible I…      9
```

```
4  More than one EditText - not getting focus, no…      4
```

[ ]:
```
# DO NOT RUN THIS CELL
```

[20]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 188878 entries, 0 to 188877
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   text    188874 non-null  object
 1   label   188878 non-null  int64
dtypes: int64(1), object(1)
memory usage: 2.9+ MB
```

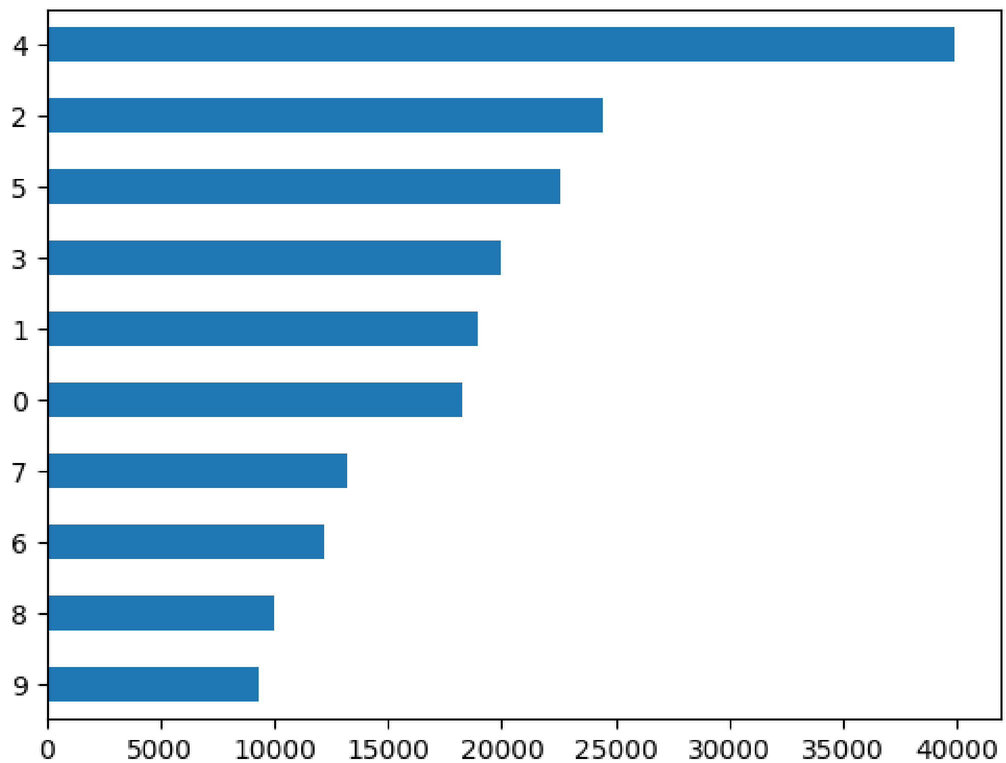
[ ]:
```
# DO NOT RUN THIS CELL
```

3.6.2 Visualize distribution of class labels

It is important to undetrstand the distribution of the class labels to check if there is any imbalance among the categories.

[21]:
```
# Plot a horizontal bar chart showing the count of each unique value in the␣
↪'label' column of the dataframe 'df'.
# The counts are displayed in ascending order for better visualization of the␣
↪distribution.
# CODE HERE
# check distribution of class labels in training dataset
df['label'].value_counts(ascending=True).plot.barh()
```

[21]: `<Axes: >`



[ ]:
```
# DO NOT RUN THIS CELL
```

Conclusions:

From the above figure, we can clearly see the imbalance between labels. The most of the questions are of 'Android' Language and least from 'asp.net'

3.6.3 Check length of the reviews

[22]:
```
# Add empty strings for rows atht do not have any text
df['text'] = df['text'].fillna('')
```

[24]:
```
# Add a new column to the dataframe 'df' named 'words_per_review'.
# This column computes the number of words in each review in the 'text' column␣
↪by splitting the text on spaces and counting the resulting words.
df['words_per_review'] = df['text'].apply(lambda x : len(x.split(' '))) # CODEE␣
↪HERE
```

[25]:
```
df.head()
```

[25]:

	text	label	words_per_review
0	detail disclosure indicator on UIButton Is the…	8	81
1	hello world fails to show up in emulator I fol…	4	266

```
2  Why is JSHint throwing a "possible strict viol…      3                  42
3  Programmatically Make Bound Column Invisible I…      9                  62
4  More than one EditText - not getting focus, no…      4                 200
```

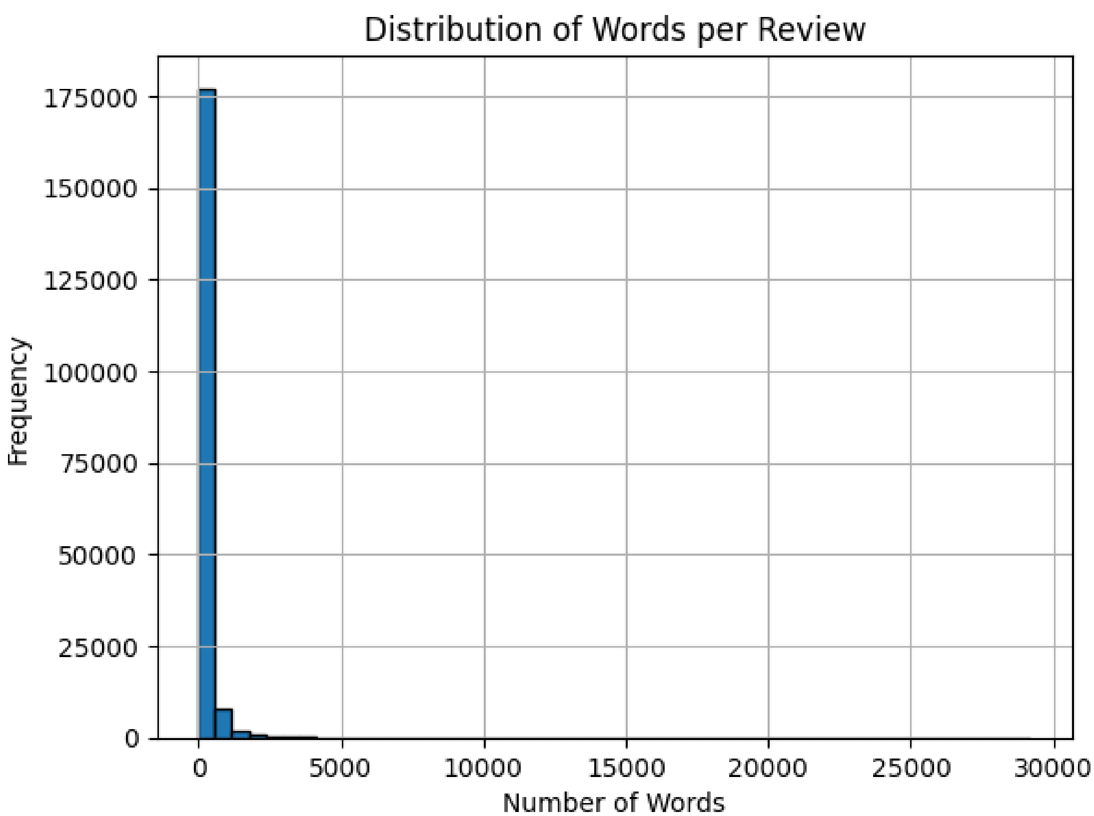
[ ]:
```
# DO NOT RUN THIS CELL
```

Plot the distribution of review length

[26]:
```
# Plot a histogram of the 'words_per_review' column
df['words_per_review'].hist(bins=50, edgecolor='black')

# Adding labels and a title for clarity
plt.xlabel('Number of Words')
plt.ylabel('Frequency')
plt.title('Distribution of Words per Review')

# Display the plot
plt.show()
```



[27]:
```
# The model we are going to use has token (subwords) limit of 512.
# Let us check how many reviews has more than 500 words
```

```
count = (df['words_per_review'] > 500).sum()
print(f"Number of reviews with more than 500 words: {count}")
```

```
Number of reviews with more than 500 words: 14769
```

[28]:
```
# count the rows that do not have any text
count = (df['words_per_review'] ==0).sum()
print(f"Number of reviews with no text words: {count}")
```

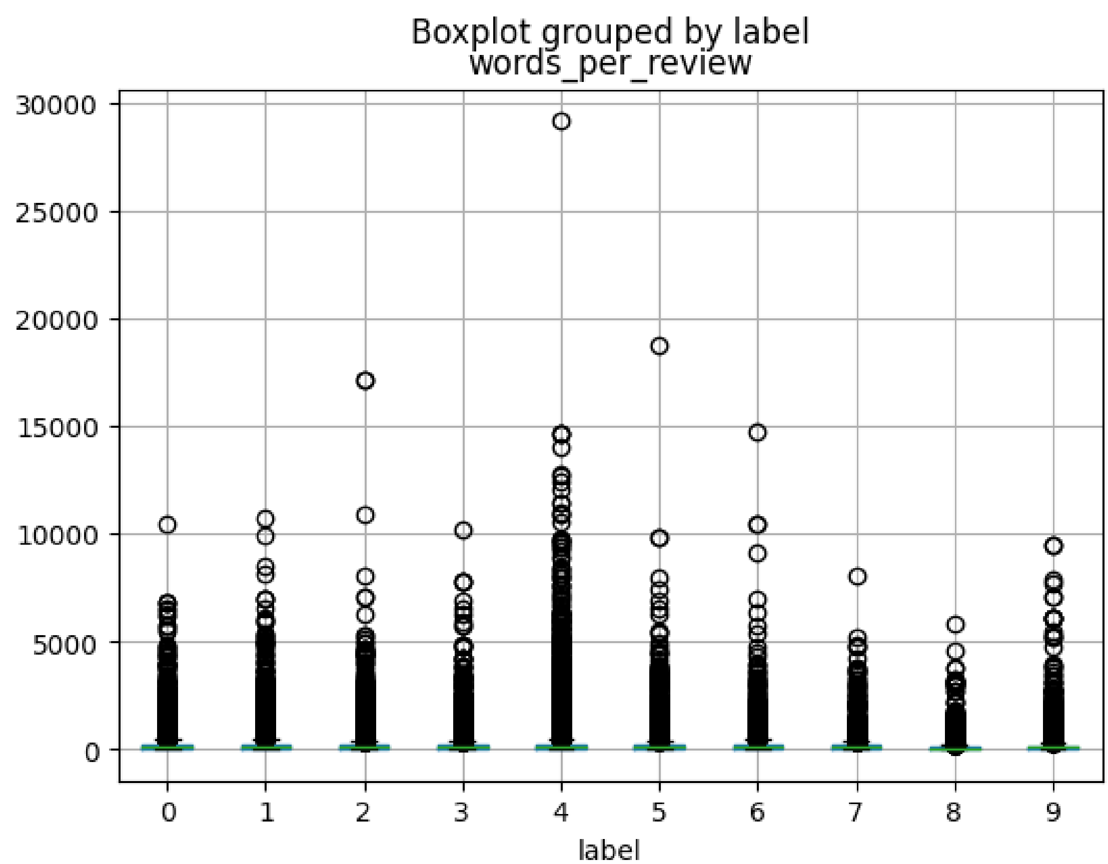
```
Number of reviews with no text words: 0
```

[29]:
```
# check the rows that have less than 20 words
count = (df['words_per_review'] <20).sum()
print(f"Number of reviews with less than 20 words: {count}")
```

```
Number of reviews with less than 20 words: 1608
```

[30]:
```
# distribution of number of words for each class label
df.boxplot('words_per_review', by='label')
```

[30]:
```
<Axes: title={'center': 'words_per_review'}, xlabel='label'>
```



- From the above graph, it seems that the distribution of number of words is similar for all the classes.
- Most models have max sequence length of 512. We have less than 1% observatins that have more than 512 words.

3.6.4 Reset dataset format

[31]:
```
# reset the format back to huggingface dataset
stack_selected_columns.reset_format()
# CODE HERE
```

[32]:
```
stack_selected_columns
```

[32]:
```
Dataset({
    features: ['text', 'label'],
    num_rows: 188878
})
```

4 Data Pre-processing

4.0.1 Create train, valid, test splits

[33]:
```
# We know this information from how we created this dataset
class_names = ['c#', 'java', 'php','javascript', 'android', 'jquery', 'c++', ␣
↪'python', 'iphone', 'asp.net']
```

[34]:
```
# Cast the 'label' column of stack_selected_columns to the ClassLabel type with␣
↪specified class names from class_names.
stack_selected_columns = stack_selected_columns.cast_column('label',␣
↪ClassLabel(names = class_names))
```

```
Casting the dataset:   0%|          | 0/188878 [00:00<?, ? examples/s]
```

The code above modifies the label column of the stack_selected_columns data structure to represent categorical data using the class names provided in class_names. This will help us to keep the index and names mapping together.

[35]:
```
stack_selected_columns.features
```

[35]:
```
{'text': Value(dtype='string', id=None),
 'label': ClassLabel(names=['c#', 'java', 'php', 'javascript', 'android',
'jquery', 'c++', 'python', 'iphone', 'asp.net'], id=None)}
```

[36]:
```
stack_selected_columns
```

[36]:
```
Dataset({
    features: ['text', 'label'],
    num_rows: 188878
})
```

[73]:
```
# Split the 'stack_selected_columns' dataset into training, validation, and
 ↪test sets.
# The aim is to have 60% for training, 20% for validation, and 20% for testing.

# First, split the dataset into a 60% training set and a 40% temporary set (to
 ↪be further split).
# Use stratified sampling based on the 'label' column to ensure that each split
 ↪has a similar distribution of labels.
test_val_splits = stack_selected_columns.train_test_split(test_size= 0.4,
 ↪seed=21, stratify_by_column='label') # CODE HERE

# Extract the 60% training dataset.
train_split = test_val_splits["train"] #CODE HERE


# Split the 40% temporary set into two equal parts: validation (20%) and test
 ↪(20%).
# Again, use stratified sampling based on the 'label' column.
test_val_splits =  test_val_splits["test"].train_test_split(test_size= 0.5,
 ↪seed=21, stratify_by_column='label') #CODE HERE

# Extract the validation and test datasets.
val_split = test_val_splits['train']# CODE HERE
test_split = test_val_splits['test'] # CODE HERE
```

[74]:
```
# combine train, val splits into one dataset
train_val_dataset = DatasetDict({'train': train_split, 'val': val_split})

# create test dataset from test split
test_dataset = DatasetDict({'test': test_split})
```

4.0.2 Create small subset for experimentation

The code below creates a new dataset, train_val_subset, with subsets of the original train_val_dataset. For each split (‘train’ and ‘val’), it ensures an equal representation of 200 samples from each label (0-9). The balanced subsets are then stored in the respective splits of train_val_subset.

[75]:
```
# Objective: Create a balanced subset from the 'train_val_dataset' where each
 ↪label has an equal number of samples.
# The resulting subset is stored in a 'DatasetDict' structure for easy
 ↪management of train and validation splits.
```

```
# Initialize an empty DatasetDict to store the balanced subsets.
train_val_subset = DatasetDict()

# Iterate over the desired splits - 'train' and 'val'.
for split in ['train', 'val']:
    # Lists to accumulate the sampled text data and their corresponding labels.
    texts = []
    labels = []

    # Loop through each label (assuming there are 10 labels numbered from 0 to␣
 ↪9).
    for label in range(10):
        # Filter out the samples in the current split that have the current␣
 ↪label.
        label_texts = train_val_dataset[split].filter(lambda x: x['label'] ==␣
 ↪label)['text']

        # Randomly sample 200 texts from the filtered data.
        label_subset = random.sample(list(label_texts), 200)

        # Append these sampled texts and their corresponding labels to the␣
 ↪accumulation lists.
        texts.extend(label_subset)
        labels.extend([label]*len(label_subset))

    # After collecting samples for all labels in the current split, create a␣
 ↪dataset with these samples.
    # Then, add this dataset to the appropriate split in the 'train_val_subset'␣
 ↪DatasetDict.
    train_val_subset[split] = Dataset.from_dict({'text': texts, 'label':␣
 ↪labels})
```

```
Filter:   0%|          | 0/113326 [00:00<?, ? examples/s]

Filter:   0%|          | 0/113326 [00:00<?, ? examples/s]

Filter:   0%|          | 0/113326 [00:00<?, ? examples/s]

Filter:   0%|          | 0/113326 [00:00<?, ? examples/s]

Filter:   0%|          | 0/113326 [00:00<?, ? examples/s]

Filter:   0%|          | 0/113326 [00:00<?, ? examples/s]

Filter:   0%|          | 0/113326 [00:00<?, ? examples/s]

Filter:   0%|          | 0/113326 [00:00<?, ? examples/s]

Filter:   0%|          | 0/113326 [00:00<?, ? examples/s]
```

```
Filter:   0%|          | 0/113326 [00:00<?, ? examples/s]
Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]
Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]
Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]
Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]
Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]
Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]
Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]
Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]
Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]
Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]
```

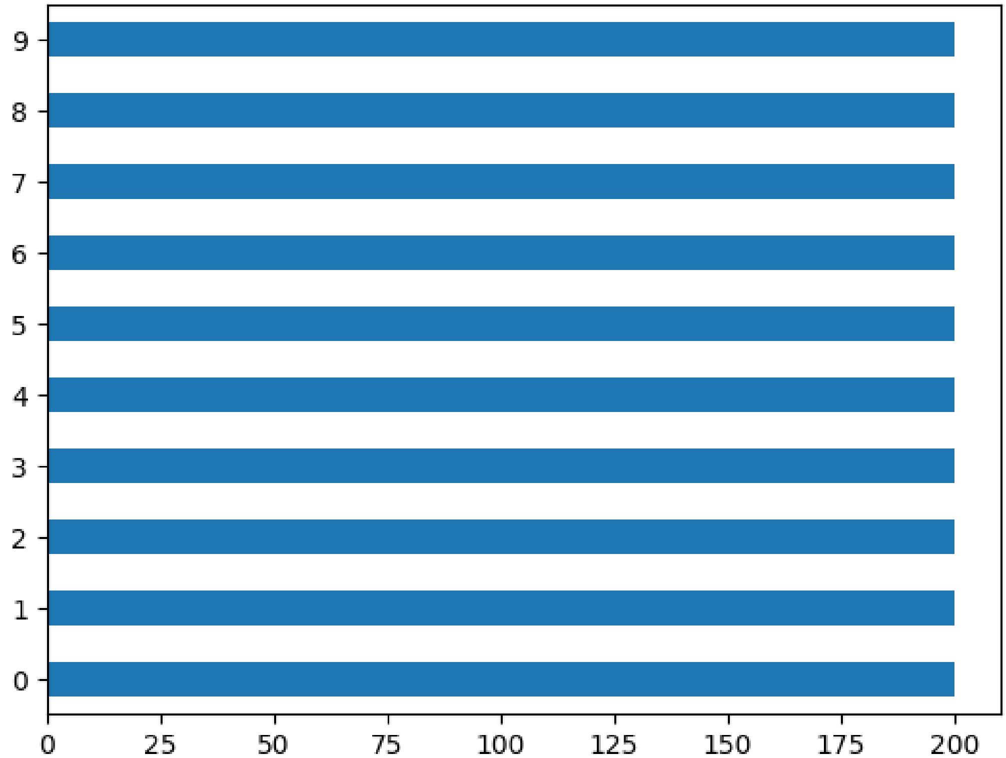
[76]:
```python
# Set the format of the train_val_subset to be a pandas DataFrame for easier
↪data manipulation and analysis.
# CODE HERE
train_val_subset.set_format(type='pandas')
```

[77]:
```python
train_val_subset
```

[77]:
```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 2000
    })
    val: Dataset({
        features: ['text', 'label'],
        num_rows: 2000
    })
})
```

[78]:
```python
# Plot a horizontal bar chart of the count of each label in the 'train' split
↪of train_val_subset, in ascending order.
# CODE HERE
train_val_subset['train']['label'].value_counts(ascending=True).plot.barh()
```

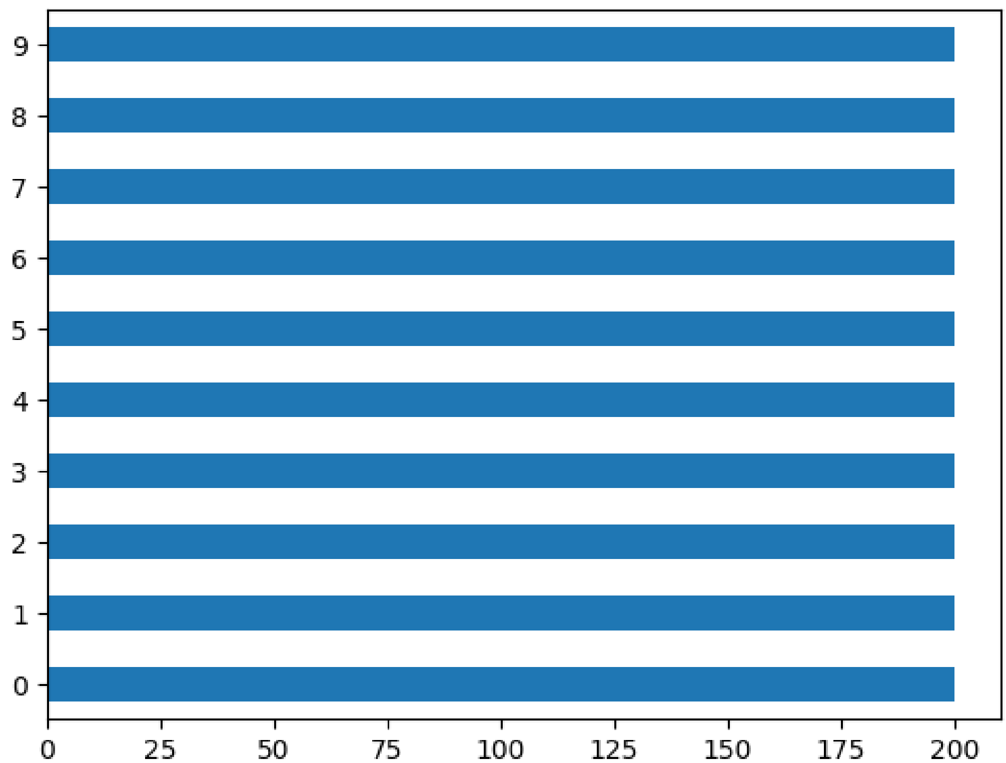
[78]:
```
<Axes: >
```



[43]:
```
# DO NOT RUN THIS CELL
```

[79]:
```
# Plot a horizontal bar chart of the count of each label in the 'val' split of␣
 ↪train_val_subset, in ascending order.
# CODE HERE
train_val_subset['val']['label'].value_counts(ascending=True).plot.barh()
```

[79]: <Axes: >



[ ]:
```
# DO NOT RUN THIS CELL
```

[80]:
```
# Reset the format of train_val_subset to its original huggingface format
# CODE HERE
train_val_subset.reset_format()
```

[81]:
```
# Retrieve the feature structures (data types and associated details) of the␣
↪'train' split from train_val_subset.
train_val_subset['train'].features
```

[81]:
```
{'text': Value(dtype='string', id=None),
 'label': Value(dtype='int64', id=None)}
```

[82]:
```
# Cast the 'label' column of the entire train_val_subset to the ClassLabel type␣
↪using the provided class names from class_names.
train_val_subset = train_val_subset.cast_column('label', ClassLabel(names =␣
↪class_names)) # CODE HERE
```

```
Casting the dataset:   0%|          | 0/2000 [00:00<?, ? examples/s]

Casting the dataset:   0%|          | 0/2000 [00:00<?, ? examples/s]
```

[83]:
```python
train_val_subset['train'].features
```

[83]:
```
{'text': Value(dtype='string', id=None),
 'label': ClassLabel(names=['c#', 'java', 'php', 'javascript', 'android',
'jquery', 'c++', 'python', 'iphone', 'asp.net'], id=None)}
```

4.1 Tokenization

[84]:
```python
# Define a checkpoint for the DistilBERT model with an uncased vocabulary.
# Instantiate the tokenizer for this model using the specified checkpoint.

from transformers import AutoTokenizer

checkpoint = 'distilbert-base-uncased' # CODE HERE
tokenizer = AutoTokenizer.from_pretrained(checkpoint) # CODE HERE
```

4.1.1 Understanding pre-trained Tokenizer

We will now understand how the tokenizer work by feeding one simple example.

[85]:
```python
text = ["Tokenization is the process of splitting sequence to tokens",
        "I like BUAN6482"]
```

[86]:
```python
# get the vocab size
print(f'Pretrained tokenizer vocab size {tokenizer.vocab_size}')
```

```
Pretrained tokenizer vocab size 30522
```

[87]:
```python
encoded_text = tokenizer(
    text, padding=True, truncation=True, return_tensors='pt')
```

[88]:
```python
encoded_text
```

[88]:
```
{'input_ids': tensor([[  101, 19204,  3989,  2003,  1996,  2832,  1997, 14541,
5537,  2000,
         19204,  2015,   102],
        [  101,  1045,  2066, 20934,  2319, 21084,  2620,  2475,   102,     0,
             0,     0,     0]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0]])}
```

What is the difference from `bert-base-uncased` tokenizer?

- `distilbert-base-uncased` do not have token-type-ids

[89]:
```python
tokens_first_sentence = tokenizer.convert_ids_to_tokens(
    encoded_text.input_ids[0])
tokens_second_sentence = tokenizer.convert_ids_to_tokens(
```

```
    encoded_text.input_ids[1])

print(tokens_first_sentence)
print(tokens_second_sentence)
```

```
['[CLS]', 'token', '##ization', 'is', 'the', 'process', 'of', 'splitting',
'sequence', 'to', 'token', '##s', '[SEP]']
['[CLS]', 'i', 'like', 'bu', '##an', '##64', '##8', '##2', '[SEP]', '[PAD]',
'[PAD]', '[PAD]', '[PAD]']
```

[90]:
```
tokenizer.convert_tokens_to_string(tokens_first_sentence)
```

[90]:
```
'[CLS] tokenization is the process of splitting sequence to tokens [SEP]'
```

[91]:
```
tokenizer.convert_tokens_to_string(tokens_second_sentence)
```

[91]:
```
'[CLS] i like buan6482 [SEP] [PAD] [PAD] [PAD] [PAD]'
```

[92]:
```
special_tokens = tokenizer.all_special_tokens
special_tokens_ids = tokenizer.all_special_ids
print(special_tokens, special_tokens_ids)
```

```
['[UNK]', '[SEP]', '[PAD]', '[CLS]', '[MASK]'] [100, 102, 0, 101, 103]
```

4.1.2 Create function for Tokenizer

[93]:
```
# Define a function to tokenize the text in a batch using the predefined␣
↪tokenizer.
# The text data is extracted from the "text" key of the batch.
# The function will truncate the tokenized data if it exceeds the tokenizer's␣
↪maximum length.

def tokenize_fn(batch):
    return tokenizer(batch["text"], truncation=True)
```

4.1.3 Use map function to apply tokenization to all splits

[94]:
```
train_val_subset
```

[94]:
```
DatasetDict({
    train: Dataset({
        features: ['text', 'label'],
        num_rows: 2000
    })
    val: Dataset({
        features: ['text', 'label'],
        num_rows: 2000
```

```
    })
})
```

[96]:
```
# Map the tokenize_fn function over the entire train_val_subset dataset in
 ↪batches.
# This will tokenize the text data in each batch and return a new dataset with
 ↪tokenized data.
tokenized_dataset = train_val_subset.map(tokenize_fn, batched=True

                                          #, batch_size=2
                                          )# CODE HERE
```

```
Map:   0%|          | 0/2000 [00:00<?, ? examples/s]

Map:   0%|          | 0/2000 [00:00<?, ? examples/s]
```

[97]:
```
tokenized_dataset
```

[97]:
```
DatasetDict({
    train: Dataset({
        features: ['text', 'label', 'input_ids', 'attention_mask'],
        num_rows: 2000
    })
    val: Dataset({
        features: ['text', 'label', 'input_ids', 'attention_mask'],
        num_rows: 2000
    })
})
```

[ ]:
```
# DO NOT RUN THIS CELL
```

We can see that tokenization step has added three new columns ('input_ids', 'token_type_ids', 'attention_mask') to the dataset

[98]:
```
tokenized_dataset = tokenized_dataset.remove_columns(
    ['text']
)
```

[99]:
```
tokenized_dataset.set_format(type='torch')
```

[100]:
```
tokenized_dataset
```

[100]:
```
DatasetDict({
    train: Dataset({
        features: ['label', 'input_ids', 'attention_mask'],
        num_rows: 2000
    })
    val: Dataset({
```

```
        features: ['label', 'input_ids', 'attention_mask'],
        num_rows: 2000
    })
})
```

[101]:
```
tokenized_dataset['train'].features
```

[101]:
```
{'label': ClassLabel(names=['c#', 'java', 'php', 'javascript', 'android',
'jquery', 'c++', 'python', 'iphone', 'asp.net'], id=None),
 'input_ids': Sequence(feature=Value(dtype='int32', id=None), length=-1,
id=None),
 'attention_mask': Sequence(feature=Value(dtype='int8', id=None), length=-1,
id=None)}
```

[102]:
```
print(len(tokenized_dataset["train"]["input_ids"][2]))
print(len(tokenized_dataset["train"]["input_ids"][1]))
```

```
288
353
```

The varying lengths in the dataset indicate that padding has not been applied yet. Instead of padding the entire dataset, we prefer processing small batches during training. Padding is done selectively for each batch based on the maximum length in the batch. We will discuss this in more detail in a later section of this notebook.

5 Model Training

5.1 Model Config File

5.1.1 Download config file of pre-trained Model

[104]:
```
# Load the configuration associated with the specified checkpoint (e.g.,␣
 ↪DistilBERT model configuration).
# This configuration contains details about the model architecture and settings.
# use Autoconfig class
from transformers import AutoConfig

config = AutoConfig.from_pretrained(checkpoint) # CODE HERE
```

[105]:
```
config
```

[105]:
```
DistilBertConfig {
  "_name_or_path": "distilbert-base-uncased",
  "activation": "gelu",
  "architectures": [
    "DistilBertForMaskedLM"
  ],
  "attention_dropout": 0.1,
```

```
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "initializer_range": 0.02,
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "pad_token_id": 0,
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embds": false,
  "tie_weights_": true,
  "transformers_version": "4.34.0",
  "vocab_size": 30522
}
```

5.1.2 Modify Configuration File

- We need to modify configuration fie to add ids to label and label to ids mapping
- Adding id2label and label2id to the configuration file provides a consistent, interpretable, and user-friendly way to handle model outputs.

[106]:
```python
class_names = tokenized_dataset["train"].features["label"].names
class_names
```

[106]:
```
['c#',
 'java',
 'php',
 'javascript',
 'android',
 'jquery',
 'c++',
 'python',
 'iphone',
 'asp.net']
```

[107]:
```python
id2label = {}
for id_, label_ in enumerate(class_names):
    id2label[str(id_)] = label_
id2label
```

[107]:
```
{'0': 'c#',
 '1': 'java',
 '2': 'php',
 '3': 'javascript',
 '4': 'android',
```

```
 '5': 'jquery',
 '6': 'c++',
 '7': 'python',
 '8': 'iphone',
 '9': 'asp.net'}
```

[108]:
```
label2id = {}
for id_, label_ in enumerate(class_names):
    label2id[label_] = id_
label2id
```

[108]:
```
{'c#': 0,
 'java': 1,
 'php': 2,
 'javascript': 3,
 'android': 4,
 'jquery': 5,
 'c++': 6,
 'python': 7,
 'iphone': 8,
 'asp.net': 9}
```

[109]:
```
config.id2label = id2label
config.label2id = label2id
```

[110]:
```
config
```

[110]:
```
DistilBertConfig {
  "_name_or_path": "distilbert-base-uncased",
  "activation": "gelu",
  "architectures": [
    "DistilBertForMaskedLM"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "id2label": {
    "0": "c#",
    "1": "java",
    "2": "php",
    "3": "javascript",
    "4": "android",
    "5": "jquery",
    "6": "c++",
    "7": "python",
    "8": "iphone",
```

```
    "9": "asp.net"
  },
  "initializer_range": 0.02,
  "label2id": {
    "android": 4,
    "asp.net": 9,
    "c#": 0,
    "c++": 6,
    "iphone": 8,
    "java": 1,
    "javascript": 3,
    "jquery": 5,
    "php": 2,
    "python": 7
  },
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "pad_token_id": 0,
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embds": false,
  "tie_weights_": true,
  "transformers_version": "4.34.0",
  "vocab_size": 30522
}
```

5.2 Download pre-trained model

[111]:
```
# Instantiate a model for sequence classification using the specified␣
↪checkpoint.
# The provided configuration (config) ensures the model aligns with the␣
↪structure and settings of the original checkpoint.
# Use AutoModelForSequenceClassification
# Pass the checkpoint and config

from transformers import AutoModelForSequenceClassification

model =  AutoModelForSequenceClassification.from_pretrained(checkpoint,␣
↪config=config) # CODE HERE
```

```
Downloading model.safetensors:   0%|          | 0.00/268M [00:00<?, ?B/s]

Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'pre_classifier.weight', 'pre_classifier.bias',
```

```
'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

5.3 Model Input/Collate Function

[112]:
```python
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

[113]:
```python
features = [tokenized_dataset["train"][i] for i in range(2)]
```

[114]:
```python
features
```

[114]:
```
[{'label': tensor(0),
  'input_ids': tensor([  101,  3853,  4677,  2075,  2003,  2036,  2641,  2004,
2655, 12221,
           1029,  2043,  1045, 10408,  1037,  4677,  2075,  2107,  2004,  1999,
           1024,  2000, 29547,  2099,  1006,  1007,  1012,  2000,  3367,  4892,
           1006,  1007,  1012,  1012,  1012,  1012,  1012,  2052,  2008,  4677,
           2075,  2036,  2022,  2641,  2004,  1037,  2655,  5963,  1029,
102]),
  'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
          1, 1])},
 {'label': tensor(0),
  'input_ids': tensor([  101,  3180,  3670,  2000,  2131,  1996,  5371, 18442,
2013, 24471,
           4877,  2065,  1045,  2031,  1037,  3793,  8758,  2007,  2195,  2015,
          24471,  4877, 12775,  8167,  2063,  1012,  4012,  1013,  6764,  1013,
           4261, 21057, 22407, 17914,  2487,  1013, 29379,  1012,  3347,  1012,
          10751,  9189,  1012, 16855,  2094,  1011,  8840,  2140,  1012, 20704,
           2072, 12775,  8167,  2063,  1012,  4012,  1013,  6764,  1013,  4261,
           2683, 15136, 23833, 22203,  1013, 29379,  1012,  3347,  1012, 22857,
           2361,  1012, 10751,  9189,  1012,  1060, 23833,  2549,  1011,  9812,
           1012, 12395,  2615, 12775,  8167,  2063,  1012,  4012,  1013,  6764,
           1013,  4261,  2683, 15136,  8889,  2692,  2549,  1013, 29379,  1012,
           3347,  1012, 22857,  2361,  1012, 10751,  9189,  1012,  1060, 23833,
           2549,  1011,  9812,  1012,  2112,  2487,  1012, 10958,  2099, 12775,
           8167,  2063,  1012,  4012,  1013,  6764,  1013,  4261,  2683, 15136,
           2692, 22932,  2549,  1013, 29379,  1012,  3347,  1012, 22857,  2361,
           1012, 10751,  9189,  1012,  1060, 23833,  2549,  1011,  9812,  1012,
           2112,  2475,  1012, 10958,  2099, 12775,  8167,  2063,  1012,  4012,
           1013,  6764,  1013,  4261,  2683, 15136, 12521, 22203,  1013, 29379,
           1012,  3347,  1012, 22857,  2361,  1012, 10751,  9189,  1012,  1060,
          23833,  2549,  1011,  9812,  1012,  2112,  2509,  1012, 10958,  2099,
          12775,  8167,  2063,  1012,  4012,  1013,  6764,  1013,  4261,  2683,
```

```
          15136, 16576, 23777,  1013, 29379,  1012,  3347,  1012, 22857,  2361,
           1012, 10751,  9189,  1012,  1060, 23833,  2549,  1011,  9812,  1012,
           2112,  2549,  1012, 10958,  2099,  1045,  2342,  1037,  1996,  6764,
           2171,  1998,  2049, 24471,  4877,  3793,  8758,  2475,  1012,  3793,
           1027, 29379,  1012,  3347,  1012, 10751,  9189,  1012, 16855,  2094,
           1011,  8840,  2140,  1012, 20704,  2072,  2013,  2182, 12775,  8167,
           2063,  1012,  4012,  1013,  6764,  1013,  4261, 21057, 22407, 17914,
           2487,  1013, 29379,  1012,  3347,  1012, 10751,  9189,  1012, 16855,
           2094,  1011,  8840,  2140,  1012, 20704,  2072, 29379,  1012,  3347,
           1012, 22857,  2361,  1012, 10751,  9189,  1012,  1060, 23833,  2549,
           1011,  9812,  1012,  2112,  2487,  1012, 10958,  2099,  2013,  2182,
          12775,  8167,  2063,  1012,  4012,  1013,  6764,  1013,  4261,  2683,
          15136,  8889,  2692,  2549,  1013, 29379,  1012,  3347,  1012, 22857,
           2361,  1012, 10751,  9189,  1012,  1060, 23833,  2549,  1011,  9812,
           1012,  2112,  2487,  1012, 10958,  2099,  1998,  2061,  2006,  4283,
           1024,  1007,   102]),
  'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
          1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])}]
```

[115]:
```
model_input = data_collator(features)
model_input.keys()
```

```
You're using a DistilBertTokenizerFast tokenizer. Please note that with a fast
tokenizer, using the `__call__` method is faster than using a method to encode
the text followed by a call to the `pad` method to get a padded encoding.
```

[115]:
```
dict_keys(['input_ids', 'attention_mask', 'labels'])
```

[116]:
```
print(model_input.input_ids[0][0:10])
print(model_input.input_ids[0][-20:])
print(model_input.input_ids[1][0:10])
print(model_input.input_ids[1][-20:])
```

```
tensor([  101,  3853,  4677,  2075,  2003,  2036,  2641,  2004,  2655, 12221])
tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
tensor([  101,  3180,  3670,  2000,  2131,  1996,  5371, 18442,  2013, 24471])
tensor([ 9189,  1012,  1060, 23833,  2549,  1011,  9812,  1012,  2112,  2487,
         1012, 10958,  2099,  1998,  2061,  2006,  4283,  1024,  1007,   102])
```

[117]:
```
print(model_input.attention_mask[0][-20:])
print(model_input.attention_mask[1][-20:])
```

```
tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

[118]:
```
print(tokenizer.convert_ids_to_tokens(model_input.input_ids[0][0:10]))
```

```
['[CLS]', 'function', 'chain', '##ing', 'is', 'also', 'considered', 'as',
'call', '##backs']
```

[119]:
```
print(tokenizer.convert_ids_to_tokens(model_input.input_ids[0][-10:]))
```

```
['[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]']
```

[120]:
```
print(tokenizer.convert_ids_to_tokens(model_input.input_ids[1][0:10]))
```

```
['[CLS]', 'regular', 'expression', 'to', 'get', 'the', 'file', '##name', 'from',
'ur']
```

[121]:
```
print(tokenizer.convert_ids_to_tokens(model_input.input_ids[1][-10:]))
```

```
['.', 'ra', '##r', 'and', 'so', 'on', 'thanks', ':', ')', '[SEP]']
```

5.4 Understanding Model Output

[122]:
```
# model output
model=model.to(device=0)
model_input= model_input.to(device=0)
model.train()
model_output = model(**model_input)
```

[123]:
```
# keys in model output
model_output.keys()
```

[123]:
```
odict_keys(['loss', 'logits'])
```

[124]:
```
# let us look at logits
model_output.logits
```

[124]:
```
tensor([[-0.0030,  0.1295,  0.0664, -0.1238,  0.2144,  0.1278, -0.0482, -0.1120,
          0.0344, -0.1262],
        [ 0.0898, -0.0235,  0.0124, -0.1586,  0.2566,  0.0514, -0.0394, -0.0743,
         -0.0307, -0.0310]], device='cuda:0', grad_fn=<AddmmBackward0>)
```

[125]:
```
model_output.logits.shape
```

[125]:
```
torch.Size([2, 10])
```

[126]:
```
model_output.loss
```

[126]:
```
tensor(2.2759, device='cuda:0', grad_fn=<NllLossBackward0>)
```

5.5 Evaluation metric(s)

5.5.1 Function to compute metric

5.6 NOTE The DIFFERENCE BETWEEN THIS FUNCTION AND THE SAME FUNBCTION WE CREATED FOR IMDB DATASET

- evaluate.combine and combined_metrics.compute does not work for multiclass classification. This is a known bug that might get resolved in future versions of evaluate

[127]:
```
# Define a function to compute evaluation metrics for sequence classification.
# The function takes in the evaluation predictions which consist of logits and␣
↪true labels.
# The function calculates the macro F1 score and accuracy, and returns them as␣
↪a dictionary.

def compute_metrics(eval_pred):
    # Split the evaluation predictions into logits (model predictions) and␣
↪actual labels.
```

```
    logits, labels = eval_pred
    # Convert logits to class predictions by picking the class with the highest␣
↪logit for each input.
    predictions = np.argmax(logits, axis=-1)

    # Load the macro F1 score metric.
    f1_metric = evaluate.load("f1", average="macro")
    # Load the accuracy metric.
    accuracy = evaluate.load("accuracy")

    # Initialize an empty dictionary to store computed metric results.
    evaluations = {}
    # Compute and store the macro F1 score.
    evaluations.update(f1_metric.compute(predictions=predictions,␣
↪references=labels, average="macro"))
    # Compute and store the accuracy.
    evaluations.update(accuracy.compute(predictions=predictions,␣
↪references=labels))

    return evaluations
```

5.7 Set up Logger for experiments

[128]:
```
# YOU WILL NEED TO CREATE AN ACCOUNT FOR WANDB
# It may provide a link for token , copy paste the token following instructions
# setup wandb
wandb.login()  # you will need to craete wandb account first
# Set project name for logging
%env WANDB_PROJECT = nlp_course_fall_2023-HW5-PartA
```

```
<IPython.core.display.Javascript object>
```

```
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server
locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here:
https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to
quit:

 ..........

wandb: Appending key for api.wandb.ai to your netrc file:
/root/.netrc
```

```
env: WANDB_PROJECT=nlp_course_fall_2023-HW5-PartA
```

5.8 Hyperparameters and Checkpointing

[129]:
```
# Define the directory where model checkpoints will be saved
model_folder = base_folder / "models"/"nlp_spring_2023/imdb/bert"
# Create the directory if it doesn't exist
model_folder.mkdir(exist_ok=True, parents=True)

# Configure training parameters
training_args = TrainingArguments(
    # Training-specific configurations
    num_train_epochs=2,  # Total number of training epochs
    # Number of samples per training batch for each device
    per_device_train_batch_size=16,
    # Number of samples per evaluation batch for each device
    per_device_eval_batch_size=16,
    weight_decay=0.01,  # Apply L2 regularization to prevent overfitting
    learning_rate=2e-5,  # Step size for the optimizer during training
    optim='adamw_torch',  # Optimizer,

    # Checkpoint saving and model evaluation settings
    output_dir=str(model_folder),  # Directory to save model checkpoints
    evaluation_strategy='steps',  # Evaluate model at specified step intervals
    eval_steps=20,  # Perform evaluation every 10 training steps
    save_strategy="steps",  # Save model checkpoint at specified step intervals
    save_steps=20,  # Save a model checkpoint every 10 training steps
    load_best_model_at_end=True,  # Reload the best model at the end of training
    save_total_limit=2,  # Retain only the best and the most recent model␣
↪checkpoints
    # Use 'accuracy' as the metric to determine the best model
    metric_for_best_model="accuracy",
    greater_is_better=True,  # A model is 'better' if its accuracy is higher


    # Experiment logging configurations (commented out in this example)
    logging_strategy='steps',
    logging_steps=20,
    report_to='wandb',  # Log metrics and results to Weights & Biases platform
    run_name= 'stack_exp1',  # Experiment name for Weights & Biases
)
```

5.9 Initialize Trainer

[130]:
```
# initialize trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
```

```
    eval_dataset=tokenized_dataset["val"],
    compute_metrics=compute_metrics,
    tokenizer=tokenizer,
)
```

5.10 Start Training

[131]:
```
trainer.data_collator
```

```
Using bos_token, but it is not set yet.
Using eos_token, but it is not set yet.
```

[131]:
```
DataCollatorWithPadding(tokenizer=DistilBertTokenizerFast(name_or_path='distilbe
rt-base-uncased', vocab_size=30522, model_max_length=512, is_fast=True,
padding_side='right', truncation_side='right', special_tokens={'unk_token':
'[UNK]', 'sep_token': '[SEP]', 'pad_token': '[PAD]', 'cls_token': '[CLS]',
'mask_token': '[MASK]'}, clean_up_tokenization_spaces=True),
added_tokens_decoder={
        0: AddedToken("[PAD]", rstrip=False, lstrip=False, single_word=False,
normalized=False, special=True),
        100: AddedToken("[UNK]", rstrip=False, lstrip=False, single_word=False,
normalized=False, special=True),
        101: AddedToken("[CLS]", rstrip=False, lstrip=False, single_word=False,
normalized=False, special=True),
        102: AddedToken("[SEP]", rstrip=False, lstrip=False, single_word=False,
normalized=False, special=True),
        103: AddedToken("[MASK]", rstrip=False, lstrip=False, single_word=False,
normalized=False, special=True),
}, padding=True, max_length=None, pad_to_multiple_of=None, return_tensors='pt')
```

[132]:
```
trainer.train()  # start training
```

```
<IPython.core.display.HTML object>
```

```
wandb: Currently logged in as: shritej24c
(redeem_team). Use `wandb login --relogin` to force relogin
```

```
VBox(children=(Label(value='Waiting for wandb.init()…\r'), FloatProgress(value=0.
↪011112714022222766, max=1.0…
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

```
Downloading builder script:   0%|          | 0.00/6.77k [00:00<?, ?B/s]

Downloading builder script:   0%|          | 0.00/4.20k [00:00<?, ?B/s]
```

[132]:
```
TrainOutput(global_step=250, training_loss=1.4311842498779297,
metrics={'train_runtime': 659.8772, 'train_samples_per_second': 6.062,
'train_steps_per_second': 0.379, 'total_flos': 513914348390400.0, 'train_loss':
1.4311842498779297, 'epoch': 2.0})
```

5.11 Evaluation

5.11.1 Check performance on validation set

[133]:
```
# Evaluate the trained model on the tokenized validation dataset.
# This will provide metrics like loss, accuracy, etc. based on the model's␣
↪performance on the validation set.
trainer.evaluate(tokenized_dataset["val"])
```

```
<IPython.core.display.HTML object>
```

[133]:
```
{'eval_loss': 0.9137545228004456,
 'eval_f1': 0.7951475208357947,
 'eval_accuracy': 0.792,
 'eval_runtime': 35.451,
 'eval_samples_per_second': 56.416,
 'eval_steps_per_second': 3.526,
 'epoch': 2.0}
```

5.11.2 Check Confusion Matrix

[134]:
```
# Use the trainer to generate predictions on the tokenized validation dataset.
# The resulting object, valid_output, will contain the model's logits (raw␣
↪prediction scores) for each input in the validation set.
valid_output = trainer.predict(tokenized_dataset["val"])
```

```
<IPython.core.display.HTML object>
```

[135]:
```
# Retrieve the named fields (attributes) of the valid_output object.
# This helps understand the structure of the prediction output and the␣
↪available information it contains.
valid_output._fields
```

[135]:
```
('predictions', 'label_ids', 'metrics')
```

[136]:
```
# Check and print the shape of the predictions and label_ids from the␣
↪valid_output object.
# This provides insight into the dimensions of the predicted outputs and the␣
↪true labels for the validation set.
```

```
print(valid_output.predictions.shape)
print(valid_output.label_ids.shape)
```

```
(2000, 10)
(2000,)
```

[137]:
```
# Convert the logits (raw prediction scores) from the valid_output object into␣
↪class predictions.
# For each input, pick the class with the highest logit as the predicted class.
# Also, extract the true label IDs from valid_output and store them as an array␣
↪for further analysis.
valid_preds = np.argmax(valid_output.predictions, axis=1) # CODE HERE
valid_labels = np.array(valid_output.label_ids)  # CODE HERE
```

[138]:
```
class_names = train_val_subset["val"].features["label"].names
class_names
```

[138]:
```
['c#',
 'java',
 'php',
 'javascript',
 'android',
 'jquery',
 'c++',
 'python',
 'iphone',
 'asp.net']
```

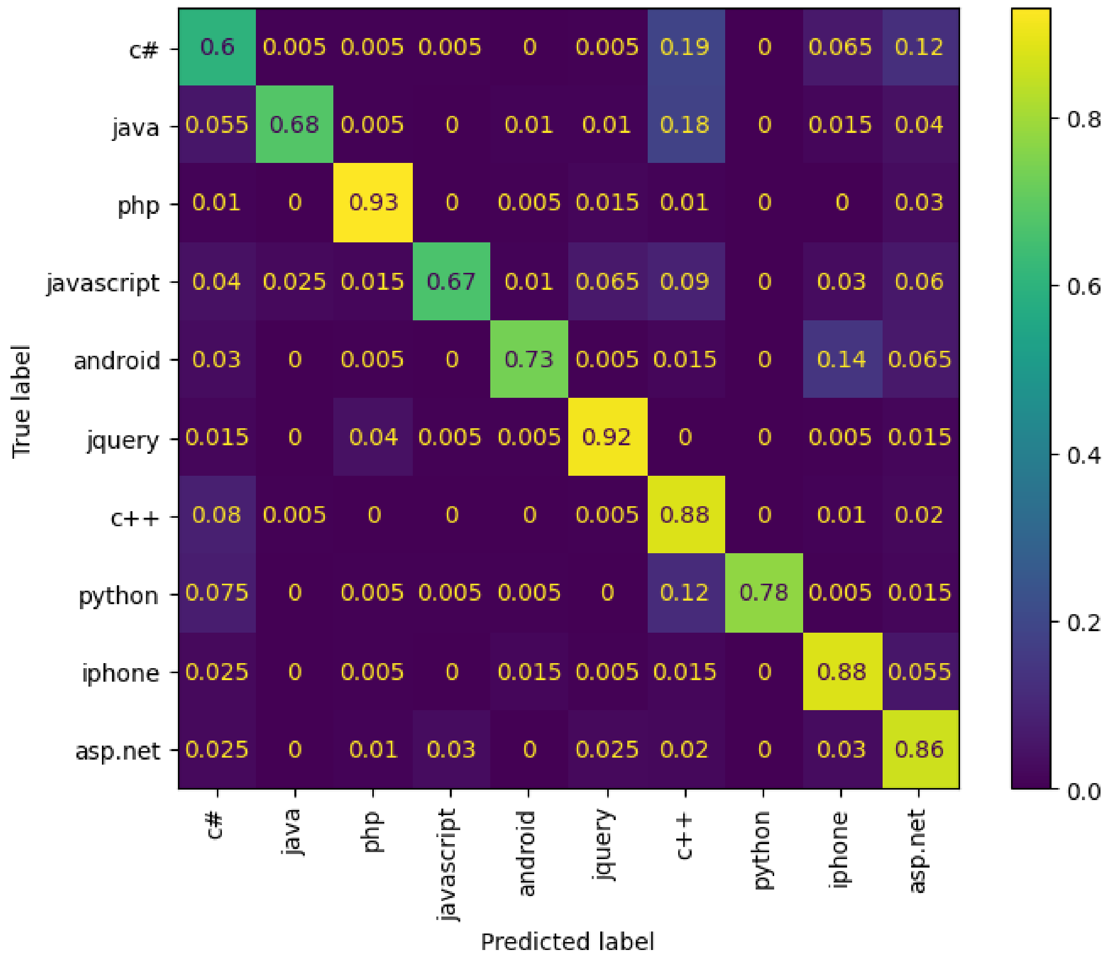
[139]:
```
# Plot a confusion matrix to visualize the model's classification performance␣
↪on the validation set.
# The matrix shows the true labels versus the predicted labels.
# The matrix is normalized by the number of true samples per class, making it␣
↪easier to identify misclassifications.

fig, ax = plt.subplots(figsize=(8, 6))  # Initialize a plotting figure with a␣
↪specified size.

# Generate and display the confusion matrix using true labels and predicted␣
↪labels.
# The matrix is normalized, and custom display labels and x-axis tick rotation␣
↪are applied for better visualization.
ConfusionMatrixDisplay.from_predictions(
    y_true=valid_labels,        # Actual labels from the validation set.
    y_pred=valid_preds,         # Predicted class labels by the model.
    ax=ax,                      # Plotting axis.
    normalize="true",           # Normalize by true class counts.
    display_labels=class_names, # Custom class names for display.
```

```
    xticks_rotation=90           # Rotate x-axis ticks for better readability.
)
```

[139]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7983544c3970>



[140]:
```
# Log the confusion matrix to the Weights & Biases (Wandb) platform for monitoring and visualization.
# This allows for tracking the model's classification performance across different runs or iterations.

# log the Confusion Matrix to Wandb
wandb.log({
    "conf_mat": wandb.plot.confusion_matrix(
        preds=valid_preds,        # Model's predicted class labels.
        y_true=valid_labels,      # Actual labels from the validation set.
```

```
        class_names=class_names   # Custom class names for display in the␣
↪confusion matrix.
    )
})
```

[141]:
```
wandb.finish()
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

5.11.3 Check the best saved model

[142]:
```
# After training, let us check the best checkpoint
# We need this for Predictions and Evaluations
best_model_checkpoint_step = trainer.state.best_model_checkpoint.split('-')[-1]
print(f"The best model was saved at step {best_model_checkpoint_step}.")
```

```
The best model was saved at step 240.
```

6 Inference

6.1 Test Set Evaluation

[143]:
```
# Create a subset of the test dataset ensuring equal representation from each␣
↪label.
# This can be useful for testing a model's performance on a balanced dataset.

test_subset = DatasetDict()  # Initialize an empty DatasetDict to store the␣
↪subsetted data.

texts = []  # List to accumulate sampled text data.
labels = [] # List to accumulate corresponding labels for the sampled texts.

# Iterate over each label from 0 to 9 (10 labels in total).
for label in range(10):
    # Filter out data samples in the 'test' split of test_dataset where the␣
↪'label' matches the current label.
    # This provides all texts for the specific label.
    label_texts = test_dataset['test'].filter(lambda x: x['label'] ==␣
↪label)['text']

    # Randomly sample 200 texts from the filtered set.
    label_subset = random.sample(list(label_texts), 200)
```

```
    # Append the sampled texts to the 'texts' list.
    texts.extend(label_subset)

    # Append the corresponding label for each of the sampled texts to the␣
↪'labels' list.
    labels.extend([label]*len(label_subset))

# Construct a new dataset from the accumulated texts and labels and assign it␣
↪to the 'test' split in test_subset.
test_subset['test'] = Dataset.from_dict({'text': texts, 'label': labels})
```

```
Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]

Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]

Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]

Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]

Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]

Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]

Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]

Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]

Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]

Filter:   0%|          | 0/37776 [00:00<?, ? examples/s]
```

[144]:
```
test_subset
```

[144]:
```
DatasetDict({
    test: Dataset({
        features: ['text', 'label'],
        num_rows: 2000
    })
})
```

NOTE we used `from evaluate import evaluator` in imdb dataset. Again this is currently not working for multiclass classification. Hence we will craete our own evaluator.

[145]:
```
def evaluator(model, dataset, tokenizer, compute_metrics, batch_size=16):
    """
    Evaluates a model's performance on a given dataset.

    Parameters:
    - model: The trained model to evaluate.
    - dataset: The dataset on which the model will be evaluated.
```

```
    - tokenizer: The tokenizer used to preprocess the text data.
    - compute_metrics: A function to compute evaluation metrics.
    - batch_size: Size of batches for evaluation. Default is 16.

    Returns:
    - evaluations: A dictionary containing computed evaluation metrics.
    """

    # Tokenize the dataset and truncate if the tokenized sequence is longer␣
↪than the model's maximum input length.
    tokenized_dataset= dataset.map(lambda batch: tokenizer(batch["text"],␣
↪truncation=True), batched=True)

    # Set the format of the tokenized dataset to be compatible with PyTorch.
    tokenized_dataset.set_format(type='torch')

    # Remove the 'text' column from the tokenized dataset as it's no longer␣
↪needed post-tokenization.
    tokenized_dataset= tokenized_dataset.remove_columns(['text'])

    # Define a collation function that pads tokenized sequences to the same␣
↪length for batching.
    collate_fn = DataCollatorWithPadding(tokenizer=tokenizer)

    # Initialize a DataLoader to iterate over the tokenized dataset in batches.
    datalaoder = DataLoader(tokenized_dataset, batch_size=batch_size,␣
↪collate_fn=collate_fn)

    # Put the model in evaluation mode and move it to the GPU.
    model.eval()
    model.to('cuda')

    # Initialize variables to store the model's logits (raw prediction scores)␣
↪and true labels.
    eval_logits = None
    eval_labels = None

    # Disable gradient calculations for efficient memory usage during␣
↪evaluation.
    with torch.inference_mode():
        # Iterate over batches from the DataLoader.
        for batch in datalaoder:
            # Prepare the inputs and move them to the GPU.
            inputs = {k: v.to('cuda') for k, v in batch.items() if k !=␣
↪'labels'}
```

```
            # Get the model's predictions for the current batch.
            outputs = model(**inputs)
            logits = outputs.logits
            labels = batch['labels'].to('cuda')

            # Append the logits and labels of the current batch to the␣
↪accumulating variables.
            if eval_logits is None:
                eval_logits = logits.cpu().numpy()
                eval_labels = labels.cpu().numpy()
            else:
                eval_logits = np.append(eval_logits, logits.cpu().numpy(),␣
↪axis=0)
                eval_labels = np.append(eval_labels, labels.cpu().numpy(),␣
↪axis=0)

    # Compute evaluation metrics using the provided compute_metrics function.
    evaluations = compute_metrics((eval_logits, eval_labels))

    return evaluations
```

[146]:
```
evaluations = evaluator(model, test_subset['test'], tokenizer, compute_metrics,␣
↪batch_size=16)
```

```
Map:   0%|          | 0/2000 [00:00<?, ? examples/s]
```

[147]:
```
evaluations
```

[147]:
```
{'f1': 0.7939556747432626, 'accuracy': 0.791}
```

6.2 Pipeline for Predictions

6.3 Create pipelne for inference

[148]:
```
# Convert the path to the 'checkpoint-220' inside the 'model_folder' to a␣
↪string format.
# SEE THE step number for best moel from the section -- Check the best saved␣
↪model
# This was 220 for me, you might get a different numnber
checkpoint = str(model_folder/f'checkpoint-{best_model_checkpoint_step}')

# Create a text classification pipeline using the Hugging Face's pipeline␣
↪method.
# The pipeline is initialized with:
# - The task set to "text-classification".
# - Model and tokenizer both loaded from the specified checkpoint path.
# - Execution set to the primary device (typically the first GPU).
```

```
custom_pipeline = pipeline(
    task="text-classification",
    model=checkpoint,
    tokenizer=checkpoint,
    device=0)
```

```
Special tokens have been added in the vocabulary, make sure the associated word
embeddings are fine-tuned or trained.
```

[149]:
```
checkpoint
```

[149]:
```
'/content/drive/MyDrive/NLP/models/nlp_spring_2023/imdb/bert/checkpoint-240'
```

6.4 Prediction for individual or small list of examples

[150]:
```
sample = test_subset['test']['text'][0]
sample
```

[150]:
```
'C# & Windows Update Api (WUApiLib) I am using Windows Update API (WUApiLib) in
a C# .NET 2.0 project. I get the following error on Windows XP (in Windows 7 it
works alright):  "System.MissingMethodException: Method not found:
\'WUApiLib.UpdateSearcher WUApiLib.UpdateSessionClass.CreateUpdateSearcher()\'."
This is my code: WUApiLib.UpdateSessionClass session = new
WUApiLib.UpdateSessionClass(); WUApiLib.IUpdateSearcher searcher =
session.CreateUpdateSearcher(); WUApiLib.ISearchResult result =
searcher.Search("Type=\'Software\'");  if (result.Updates.Count > 0) {     //do
stuff }  The error occurs at runtime, the compiler shows no errors… Does
anybody know why I get this error? '
```

[151]:
```
preds = custom_pipeline(sample)
preds
```

[151]:
```
[{'label': 'c#', 'score': 0.3751984238624573}]
```

[152]:
```
sample = test_split['text'][12]
sample
```

[152]:
```
'"Explicit" preventing automatic type conversion?  Possible Duplicate: What does
the explicit keyword in C++ mean?   I do not understand the following. If I
have: class Stack{     explicit Stack(int size); }  without the keyword explicit
I would be allowed to do: Stack s; s = 40;  Why would I be allowed to do the
above if explicit wasn\'t provided?? Is it because this is stack-allocation (no
constructor) and C++ allows anything to be assigned to the variable unless
explicit is used? '
```

[153]:
```python
preds = custom_pipeline(sample)
preds
```

[153]:
```
[{'label': 'c++', 'score': 0.5075118541717529}]
```

6.5 Prediction for large dataset

[154]:
```python
predictions = custom_pipeline(test_subset['test']['text'], truncation=True)
```

[155]:
```python
predictions
```

[155]:
```
[{'label': 'c#', 'score': 0.3751984238624573},
 {'label': 'iphone', 'score': 0.26100224256515503},
 {'label': 'c#', 'score': 0.36539867520332336},
 {'label': 'c#', 'score': 0.35040923953056335},
 {'label': 'asp.net', 'score': 0.15593917667865753},
 {'label': 'c#', 'score': 0.2736811339855194},
 {'label': 'c#', 'score': 0.3634810745716095},
 {'label': 'c++', 'score': 0.3895452618598938},
 {'label': 'c#', 'score': 0.3348501920700073},
 {'label': 'c++', 'score': 0.25139185786247253},
 {'label': 'c#', 'score': 0.24365904927253723},
 {'label': 'c#', 'score': 0.23755154013633728},
 {'label': 'c++', 'score': 0.32494401931762695},
 {'label': 'c++', 'score': 0.31954726576805115},
 {'label': 'c#', 'score': 0.3455428183078766},
 {'label': 'asp.net', 'score': 0.22062942385673523},
 {'label': 'c#', 'score': 0.17923590540885925},
 {'label': 'c#', 'score': 0.24061596393585205},
 {'label': 'asp.net', 'score': 0.398060142993927},
 {'label': 'c++', 'score': 0.2887530028820038},
 {'label': 'asp.net', 'score': 0.2483542561531067},
 {'label': 'c#', 'score': 0.34768274426460266},
 {'label': 'c#', 'score': 0.36391183733940125},
 {'label': 'c#', 'score': 0.33997467160224915},
 {'label': 'c++', 'score': 0.3222533166408539},
 {'label': 'asp.net', 'score': 0.3051544427871704},
 {'label': 'c++', 'score': 0.4248054325580597},
 {'label': 'c#', 'score': 0.3667224049568176},
 {'label': 'c#', 'score': 0.36500081419944763},
 {'label': 'asp.net', 'score': 0.6044130921363831},
 {'label': 'asp.net', 'score': 0.2713795602321625},
 {'label': 'c#', 'score': 0.36078205704689026},
 {'label': 'asp.net', 'score': 0.27326780557632446},
 {'label': 'c#', 'score': 0.2757297456264496},
 {'label': 'c#', 'score': 0.3774144649505615},
 {'label': 'c#', 'score': 0.17990702390670776},
```

{'label': 'c#', 'score': 0.2541741728782654},
{'label': 'c#', 'score': 0.35771235823631287},
{'label': 'c#', 'score': 0.3651278614997864},
{'label': 'c#', 'score': 0.3317638337612152},
{'label': 'c++', 'score': 0.35707589983940125},
{'label': 'c#', 'score': 0.37834665179252625},
{'label': 'c#', 'score': 0.3658619523048401},
{'label': 'asp.net', 'score': 0.21058589220046997},
{'label': 'c#', 'score': 0.35171306133270264},
{'label': 'c#', 'score': 0.3288207948207855},
{'label': 'c#', 'score': 0.349677175283432},
{'label': 'c#', 'score': 0.34794220328330994},
{'label': 'c#', 'score': 0.23129898309707642},
{'label': 'c++', 'score': 0.3147343695163727},
{'label': 'c#', 'score': 0.32662054896354675},
{'label': 'c++', 'score': 0.35580718517303467},
{'label': 'asp.net', 'score': 0.33791324496269226},
{'label': 'c++', 'score': 0.2648288607597351},
{'label': 'c++', 'score': 0.15033745765686035},
{'label': 'jquery', 'score': 0.19554740190505981},
{'label': 'c#', 'score': 0.332001268863678},
{'label': 'c#', 'score': 0.33766046166419983},
{'label': 'c#', 'score': 0.3577427268028259},
{'label': 'c#', 'score': 0.34025639295578003},
{'label': 'c#', 'score': 0.24752797186374664},
{'label': 'c#', 'score': 0.3203703761100769},
{'label': 'c#', 'score': 0.27649572491645813},
{'label': 'c#', 'score': 0.34390610456466675},
{'label': 'asp.net', 'score': 0.23828838765621185},
{'label': 'c#', 'score': 0.3464420437812805},
{'label': 'c++', 'score': 0.28555232286453247},
{'label': 'c#', 'score': 0.38088011741638184},
{'label': 'c#', 'score': 0.23315005004405975},
{'label': 'java', 'score': 0.3175959885120392},
{'label': 'c#', 'score': 0.3546580672264099},
{'label': 'c#', 'score': 0.3574037253856659},
{'label': 'c#', 'score': 0.22493767738342285},
{'label': 'asp.net', 'score': 0.2015601396560669},
{'label': 'c++', 'score': 0.3798215389251709},
{'label': 'asp.net', 'score': 0.49440258741378784},
{'label': 'c#', 'score': 0.36785924434661865},
{'label': 'c#', 'score': 0.29751089215278625},
{'label': 'c#', 'score': 0.38691356778144836},
{'label': 'c#', 'score': 0.35501012206077576},
{'label': 'iphone', 'score': 0.29841554164886475},
{'label': 'c#', 'score': 0.3478191196918487},
{'label': 'c#', 'score': 0.25258558988571167},

{'label': 'c++', 'score': 0.33267074823379517},
{'label': 'c++', 'score': 0.32484984397888184},
{'label': 'c#', 'score': 0.2332613170146942},
{'label': 'iphone', 'score': 0.24832108616828918},
{'label': 'c#', 'score': 0.35635989904403687},
{'label': 'c++', 'score': 0.330030232667923},
{'label': 'c++', 'score': 0.31787919998168945},
{'label': 'c#', 'score': 0.3504035770893097},
{'label': 'c#', 'score': 0.3535408675670624},
{'label': 'c#', 'score': 0.35372766852378845},
{'label': 'c++', 'score': 0.33929187059402466},
{'label': 'c#', 'score': 0.3545624017715454},
{'label': 'c++', 'score': 0.2874908745288849},
{'label': 'asp.net', 'score': 0.2998979985713959},
{'label': 'asp.net', 'score': 0.5507053136825562},
{'label': 'c#', 'score': 0.3418256342411041},
{'label': 'c#', 'score': 0.36701878905296326},
{'label': 'asp.net', 'score': 0.2631703019142151},
{'label': 'c#', 'score': 0.3503167927265167},
{'label': 'java', 'score': 0.5063978433609009},
{'label': 'iphone', 'score': 0.30351340770721436},
{'label': 'c#', 'score': 0.34593531489372253},
{'label': 'c#', 'score': 0.25445953011512756},
{'label': 'c#', 'score': 0.3575623631477356},
{'label': 'c#', 'score': 0.36443987488746643},
{'label': 'c#', 'score': 0.37545347213745117},
{'label': 'c#', 'score': 0.24264515936374664},
{'label': 'c++', 'score': 0.37398388981819153},
{'label': 'c#', 'score': 0.20102821290493011},
{'label': 'c++', 'score': 0.2536715567111969},
{'label': 'c++', 'score': 0.40315085649490356},
{'label': 'c#', 'score': 0.24871191382408142},
{'label': 'asp.net', 'score': 0.5605983138084412},
{'label': 'c#', 'score': 0.35987284779548645},
{'label': 'iphone', 'score': 0.24357359111309052},
{'label': 'c++', 'score': 0.3254084587097168},
{'label': 'c++', 'score': 0.30935072898864746},
{'label': 'asp.net', 'score': 0.23263660073280334},
{'label': 'c++', 'score': 0.34573283791542053},
{'label': 'iphone', 'score': 0.34578245878219604},
{'label': 'iphone', 'score': 0.33908191323280334},
{'label': 'c#', 'score': 0.33509257435798645},
{'label': 'c#', 'score': 0.35128358006477356},
{'label': 'asp.net', 'score': 0.35267138481140137},
{'label': 'c#', 'score': 0.36161187291145325},
{'label': 'asp.net', 'score': 0.23629887402057648},
{'label': 'c++', 'score': 0.32861319184303284},

```
{'label': 'asp.net', 'score': 0.26452916860580444},
{'label': 'c#', 'score': 0.3404276669025421},
{'label': 'c#', 'score': 0.3556191921234131},
{'label': 'c#', 'score': 0.34946972131729126},
{'label': 'c#', 'score': 0.34538349509239197},
{'label': 'c#', 'score': 0.3521118760108948},
{'label': 'c++', 'score': 0.26338326930999756},
{'label': 'c#', 'score': 0.37912917137145996},
{'label': 'asp.net', 'score': 0.5767567753791809},
{'label': 'c#', 'score': 0.3792703151702881},
{'label': 'c#', 'score': 0.35378068685531616},
{'label': 'asp.net', 'score': 0.3333076536655426},
{'label': 'c#', 'score': 0.2035377472639084},
{'label': 'c#', 'score': 0.35550636053085327},
{'label': 'c++', 'score': 0.4170159697532654},
{'label': 'c#', 'score': 0.356682151556015},
{'label': 'c#', 'score': 0.20514199137687683},
{'label': 'c#', 'score': 0.24736589193344116},
{'label': 'c#', 'score': 0.3342791795730591},
{'label': 'c#', 'score': 0.20260266959667206},
{'label': 'python', 'score': 0.5549541115760803},
{'label': 'c#', 'score': 0.3402210772037506},
{'label': 'c#', 'score': 0.3534471094608307},
{'label': 'asp.net', 'score': 0.29003041982650757},
{'label': 'asp.net', 'score': 0.45511990785598755},
{'label': 'c#', 'score': 0.3433423936367035},
{'label': 'c#', 'score': 0.361013263463974},
{'label': 'asp.net', 'score': 0.2497199922800064},
{'label': 'c#', 'score': 0.35841912031173706},
{'label': 'c++', 'score': 0.3936580419540405},
{'label': 'c#', 'score': 0.35771608352661133},
{'label': 'c#', 'score': 0.3599867522716522},
{'label': 'c#', 'score': 0.33569303154945374},
{'label': 'c#', 'score': 0.3763948976993561},
{'label': 'c#', 'score': 0.3612290024757385},
{'label': 'c#', 'score': 0.37404945492744446},
{'label': 'asp.net', 'score': 0.17536962032318115},
{'label': 'c++', 'score': 0.3057312071323395},
{'label': 'c++', 'score': 0.27482667565345764},
{'label': 'iphone', 'score': 0.3177536427974701},
{'label': 'java', 'score': 0.6233880519866943},
{'label': 'asp.net', 'score': 0.34108924865722656},
{'label': 'c#', 'score': 0.34918442368507385},
{'label': 'c#', 'score': 0.3598320484161377},
{'label': 'c#', 'score': 0.35226884484291077},
{'label': 'c++', 'score': 0.30181124806404114},
{'label': 'c++', 'score': 0.34323957562446594},
```

{'label': 'c#', 'score': 0.3322962820529938},
{'label': 'c++', 'score': 0.32707884907722473},
{'label': 'asp.net', 'score': 0.6204645037651062},
{'label': 'c#', 'score': 0.2571370601654053},
{'label': 'c++', 'score': 0.23119716346263885},
{'label': 'c++', 'score': 0.3175566792488098},
{'label': 'c#', 'score': 0.37908676266670227},
{'label': 'c#', 'score': 0.23729413747787476},
{'label': 'c#', 'score': 0.2433936595916748},
{'label': 'c#', 'score': 0.35587844252586365},
{'label': 'c#', 'score': 0.34429681301116943},
{'label': 'c++', 'score': 0.3113579750061035},
{'label': 'c#', 'score': 0.35490480065345764},
{'label': 'c++', 'score': 0.26715314388275146},
{'label': 'c++', 'score': 0.28345900774002075},
{'label': 'c#', 'score': 0.24188902974128723},
{'label': 'c#', 'score': 0.33764955401420593},
{'label': 'c#', 'score': 0.36769986152648926},
{'label': 'c#', 'score': 0.3497064709663391},
{'label': 'c#', 'score': 0.386740118265152},
{'label': 'c#', 'score': 0.37528857588768005},
{'label': 'c#', 'score': 0.3583312928676605},
{'label': 'asp.net', 'score': 0.306080162525177},
{'label': 'java', 'score': 0.6282139420509338},
{'label': 'java', 'score': 0.6031397581100464},
{'label': 'c++', 'score': 0.3540189862251282},
{'label': 'java', 'score': 0.6016428470611572},
{'label': 'java', 'score': 0.6127496361732483},
{'label': 'java', 'score': 0.6225875616073608},
{'label': 'java', 'score': 0.5795493125915527},
{'label': 'java', 'score': 0.6025856137275696},
{'label': 'java', 'score': 0.5602813959121704},
{'label': 'asp.net', 'score': 0.2137666791677475},
{'label': 'java', 'score': 0.5885897874832153},
{'label': 'java', 'score': 0.6202579736709595},
{'label': 'c++', 'score': 0.2398836314678192},
{'label': 'java', 'score': 0.6222434639930725},
{'label': 'java', 'score': 0.6070467829704285},
{'label': 'java', 'score': 0.6105993390083313},
{'label': 'java', 'score': 0.6088469624519348},
{'label': 'java', 'score': 0.5884371399879456},
{'label': 'java', 'score': 0.6227880120277405},
{'label': 'java', 'score': 0.28344470262527466},
{'label': 'java', 'score': 0.5789116024971008},
{'label': 'java', 'score': 0.6138261556625366},
{'label': 'java', 'score': 0.5840141773223877},
{'label': 'java', 'score': 0.6005536317825317},

{'label': 'java', 'score': 0.6345862746238708},
{'label': 'java', 'score': 0.6203948855400085},
{'label': 'c++', 'score': 0.3118289113044739},
{'label': 'java', 'score': 0.37573176622390747},
{'label': 'java', 'score': 0.609987199306488},
{'label': 'java', 'score': 0.5858430862426758},
{'label': 'c++', 'score': 0.25364044308662415},
{'label': 'java', 'score': 0.6108651161193848},
{'label': 'java', 'score': 0.579169750213623},
{'label': 'c++', 'score': 0.3394230008125305},
{'label': 'java', 'score': 0.6160876750946045},
{'label': 'jquery', 'score': 0.2182166576385498},
{'label': 'java', 'score': 0.6199662685394287},
{'label': 'java', 'score': 0.6155897378921509},
{'label': 'c++', 'score': 0.3161337673664093},
{'label': 'java', 'score': 0.451907753944397},
{'label': 'java', 'score': 0.6208351850509644},
{'label': 'java', 'score': 0.5752308964729309},
{'label': 'asp.net', 'score': 0.1583862602710724},
{'label': 'java', 'score': 0.5758549571037292},
{'label': 'java', 'score': 0.5784118175506592},
{'label': 'java', 'score': 0.5626246333122253},
{'label': 'php', 'score': 0.20559829473495483},
{'label': 'iphone', 'score': 0.21563640236854553},
{'label': 'java', 'score': 0.6211493611335754},
{'label': 'android', 'score': 0.7427754998207092},
{'label': 'java', 'score': 0.5102331042289734},
{'label': 'java', 'score': 0.6052942276000977},
{'label': 'java', 'score': 0.5583535432815552},
{'label': 'java', 'score': 0.5983929634094238},
{'label': 'java', 'score': 0.595815896987915},
{'label': 'java', 'score': 0.6226882338523865},
{'label': 'c++', 'score': 0.30436787009239197},
{'label': 'java', 'score': 0.6057022213935852},
{'label': 'c++', 'score': 0.32735976576805115},
{'label': 'jquery', 'score': 0.31881266832351685},
{'label': 'java', 'score': 0.6197628378868103},
{'label': 'c++', 'score': 0.2977607548236847},
{'label': 'c#', 'score': 0.19319207966327667},
{'label': 'java', 'score': 0.5959317684173584},
{'label': 'java', 'score': 0.630544126033783},
{'label': 'java', 'score': 0.6128865480422974},
{'label': 'java', 'score': 0.6213439702987671},
{'label': 'c#', 'score': 0.21888992190361023},
{'label': 'iphone', 'score': 0.3331228196620941},
{'label': 'c++', 'score': 0.29651689529418945},
{'label': 'java', 'score': 0.6021669507026672},

{'label': 'java', 'score': 0.6295087933540344},
{'label': 'java', 'score': 0.5437619090080261},
{'label': 'iphone', 'score': 0.31129592657089233},
{'label': 'java', 'score': 0.637323796749115},
{'label': 'c++', 'score': 0.25114870071411133},
{'label': 'c++', 'score': 0.2959915101528168},
{'label': 'java', 'score': 0.6221556067466736},
{'label': 'java', 'score': 0.6162229776382446},
{'label': 'java', 'score': 0.6182222962379456},
{'label': 'java', 'score': 0.6001720428466797},
{'label': 'c++', 'score': 0.23817090690135956},
{'label': 'c++', 'score': 0.29256054759025574},
{'label': 'java', 'score': 0.516783595085144},
{'label': 'java', 'score': 0.589652955532074},
{'label': 'java', 'score': 0.5987550616264343},
{'label': 'c++', 'score': 0.3023630380630493},
{'label': 'java', 'score': 0.6252198815345764},
{'label': 'java', 'score': 0.6300654411315918},
{'label': 'java', 'score': 0.6103970408439636},
{'label': 'java', 'score': 0.5874303579330444},
{'label': 'java', 'score': 0.6242458820343018},
{'label': 'java', 'score': 0.586571455001831},
{'label': 'java', 'score': 0.5875054597854614},
{'label': 'c++', 'score': 0.27189260721206665},
{'label': 'asp.net', 'score': 0.26775890588760376},
{'label': 'java', 'score': 0.6140320301055908},
{'label': 'java', 'score': 0.5866297483444214},
{'label': 'java', 'score': 0.5849863886833191},
{'label': 'c#', 'score': 0.22616082429885864},
{'label': 'java', 'score': 0.6343089938163757},
{'label': 'java', 'score': 0.5859867930412292},
{'label': 'java', 'score': 0.6107364296913147},
{'label': 'java', 'score': 0.6314464807510376},
{'label': 'java', 'score': 0.6221138834953308},
{'label': 'java', 'score': 0.6108351349830627},
{'label': 'c++', 'score': 0.2695170044898987},
{'label': 'java', 'score': 0.6094903945922852},
{'label': 'java', 'score': 0.5877007246017456},
{'label': 'java', 'score': 0.5937656164169312},
{'label': 'java', 'score': 0.6205621957778931},
{'label': 'java', 'score': 0.5912801027297974},
{'label': 'c++', 'score': 0.30800727009773254},
{'label': 'java', 'score': 0.6213082075119019},
{'label': 'c#', 'score': 0.25046902894973755},
{'label': 'jquery', 'score': 0.45568928122520447},
{'label': 'java', 'score': 0.6197907328605652},
{'label': 'java', 'score': 0.6241872906684875},

{'label': 'java', 'score': 0.6211544871330261},
{'label': 'java', 'score': 0.6125881671905518},
{'label': 'java', 'score': 0.6199507713317871},
{'label': 'c#', 'score': 0.14638465642929077},
{'label': 'c++', 'score': 0.330958753824234},
{'label': 'java', 'score': 0.6077225208282471},
{'label': 'java', 'score': 0.6273096799850464},
{'label': 'java', 'score': 0.6097200512886047},
{'label': 'java', 'score': 0.6023251414299011},
{'label': 'c++', 'score': 0.2986281216144562},
{'label': 'java', 'score': 0.5905212163925171},
{'label': 'java', 'score': 0.6180544495582581},
{'label': 'java', 'score': 0.6104788780212402},
{'label': 'c++', 'score': 0.3818559944629669},
{'label': 'java', 'score': 0.6155804991722107},
{'label': 'c#', 'score': 0.23458530008792877},
{'label': 'java', 'score': 0.5889872312545776},
{'label': 'java', 'score': 0.6192531585693359},
{'label': 'java', 'score': 0.5792081356048584},
{'label': 'c++', 'score': 0.28922024369239807},
{'label': 'java', 'score': 0.5979853272438049},
{'label': 'java', 'score': 0.6052347421646118},
{'label': 'c#', 'score': 0.27593573927879333},
{'label': 'c++', 'score': 0.2691136300563812},
{'label': 'java', 'score': 0.5201727747917175},
{'label': 'java', 'score': 0.5772281289100647},
{'label': 'c#', 'score': 0.1525435596704483},
{'label': 'java', 'score': 0.6245246529579163},
{'label': 'c++', 'score': 0.3095436692237854},
{'label': 'java', 'score': 0.5750678777694702},
{'label': 'java', 'score': 0.6249900460243225},
{'label': 'c#', 'score': 0.2715013921260834},
{'label': 'c++', 'score': 0.3808421790599823},
{'label': 'c++', 'score': 0.2992475628852844},
{'label': 'java', 'score': 0.6057494282722473},
{'label': 'java', 'score': 0.6201701164245605},
{'label': 'java', 'score': 0.5956318974494934},
{'label': 'java', 'score': 0.5910103917121887},
{'label': 'java', 'score': 0.6202825307846069},
{'label': 'java', 'score': 0.6224082112312317},
{'label': 'c#', 'score': 0.17795120179653168},
{'label': 'java', 'score': 0.5800893902778625},
{'label': 'java', 'score': 0.5931767821311951},
{'label': 'java', 'score': 0.6178629994392395},
{'label': 'java', 'score': 0.5876515507698059},
{'label': 'c++', 'score': 0.2637854516506195},
{'label': 'java', 'score': 0.6040130257606506},

{'label': 'java', 'score': 0.6108051538467407},
{'label': 'java', 'score': 0.6217148303985596},
{'label': 'c++', 'score': 0.2950921058654785},
{'label': 'asp.net', 'score': 0.1702422797679901},
{'label': 'java', 'score': 0.609817624092102},
{'label': 'asp.net', 'score': 0.16766558587551117},
{'label': 'c++', 'score': 0.305831640958786},
{'label': 'java', 'score': 0.6206262111663818},
{'label': 'java', 'score': 0.5976452231407166},
{'label': 'java', 'score': 0.6122750043869019},
{'label': 'java', 'score': 0.6293336749076843},
{'label': 'java', 'score': 0.6280782222747803},
{'label': 'c#', 'score': 0.20797179639339447},
{'label': 'c++', 'score': 0.359432578086853},
{'label': 'java', 'score': 0.6296573281288147},
{'label': 'c++', 'score': 0.1912107765674591},
{'label': 'java', 'score': 0.602291464805603},
{'label': 'java', 'score': 0.6219275593757629},
{'label': 'java', 'score': 0.6307472586631775},
{'label': 'java', 'score': 0.6181362867355347},
{'label': 'java', 'score': 0.6167314648628235},
{'label': 'c++', 'score': 0.27266818284988403},
{'label': 'java', 'score': 0.6144368052482605},
{'label': 'c++', 'score': 0.31364184617996216},
{'label': 'java', 'score': 0.6111721396446228},
{'label': 'java', 'score': 0.6171788573265076},
{'label': 'java', 'score': 0.5707933306694031},
{'label': 'java', 'score': 0.6155248880386353},
{'label': 'c++', 'score': 0.27410465478897095},
{'label': 'asp.net', 'score': 0.24918678402900696},
{'label': 'java', 'score': 0.62513667345047},
{'label': 'java', 'score': 0.6212679743766785},
{'label': 'asp.net', 'score': 0.2561657428741455},
{'label': 'java', 'score': 0.6246972680091858},
{'label': 'java', 'score': 0.6244049668312073},
{'label': 'php', 'score': 0.6991672515869141},
{'label': 'php', 'score': 0.690876305103302},
{'label': 'php', 'score': 0.668354868888855},
{'label': 'php', 'score': 0.7024375796318054},
{'label': 'php', 'score': 0.6767611503601074},
{'label': 'php', 'score': 0.7099590301513672},
{'label': 'php', 'score': 0.7102322578430176},
{'label': 'php', 'score': 0.6928644180297852},
{'label': 'php', 'score': 0.6843623518943787},
{'label': 'php', 'score': 0.7030240893363953},
{'label': 'php', 'score': 0.6936288475990295},
{'label': 'php', 'score': 0.34417441487312317},

{'label': 'php', 'score': 0.7066322565078735},
{'label': 'php', 'score': 0.693909764289856},
{'label': 'php', 'score': 0.6631758809089661},
{'label': 'php', 'score': 0.7050667405128479},
{'label': 'php', 'score': 0.6611891984939575},
{'label': 'php', 'score': 0.6861642003059387},
{'label': 'php', 'score': 0.5780496597290039},
{'label': 'c++', 'score': 0.26391422748565674},
{'label': 'php', 'score': 0.69761723279953},
{'label': 'php', 'score': 0.2822829782962799},
{'label': 'php', 'score': 0.7030948400497437},
{'label': 'python', 'score': 0.15431095659732819},
{'label': 'php', 'score': 0.7088868021965027},
{'label': 'php', 'score': 0.7104899883270264},
{'label': 'php', 'score': 0.7134771943092346},
{'label': 'php', 'score': 0.6967569589614868},
{'label': 'php', 'score': 0.6973082423210144},
{'label': 'php', 'score': 0.7057332396507263},
{'label': 'php', 'score': 0.6923381090164185},
{'label': 'php', 'score': 0.6828703284263611},
{'label': 'php', 'score': 0.6874122023582458},
{'label': 'php', 'score': 0.7104606032371521},
{'label': 'php', 'score': 0.6587396860122681},
{'label': 'php', 'score': 0.711276113986969},
{'label': 'php', 'score': 0.7024233341217041},
{'label': 'iphone', 'score': 0.3473317325115204},
{'label': 'php', 'score': 0.6725053787231445},
{'label': 'php', 'score': 0.6617135405540466},
{'label': 'php', 'score': 0.708720326423645},
{'label': 'php', 'score': 0.6936309337615967},
{'label': 'php', 'score': 0.5075425505638123},
{'label': 'iphone', 'score': 0.17510958015918732},
{'label': 'php', 'score': 0.6829413175582886},
{'label': 'php', 'score': 0.49439379572868347},
{'label': 'php', 'score': 0.6593475341796875},
{'label': 'php', 'score': 0.6950780153274536},
{'label': 'php', 'score': 0.6867836713790894},
{'label': 'php', 'score': 0.7000406384468079},
{'label': 'php', 'score': 0.6900033950805664},
{'label': 'php', 'score': 0.6866505146026611},
{'label': 'php', 'score': 0.5047606825828552},
{'label': 'php', 'score': 0.7136744260787964},
{'label': 'php', 'score': 0.6960402727127075},
{'label': 'php', 'score': 0.6379117965698242},
{'label': 'php', 'score': 0.6612301468849182},
{'label': 'php', 'score': 0.7002129554748535},
{'label': 'php', 'score': 0.589939296245575},

```
{'label': 'php', 'score': 0.7057697772979736},
{'label': 'asp.net', 'score': 0.29387399554252625},
{'label': 'php', 'score': 0.629019021987915},
{'label': 'php', 'score': 0.691981315612793},
{'label': 'php', 'score': 0.6865869164466858},
{'label': 'php', 'score': 0.6913138031959534},
{'label': 'php', 'score': 0.7065842747688293},
{'label': 'php', 'score': 0.40721389651298523},
{'label': 'php', 'score': 0.49445703625679016},
{'label': 'php', 'score': 0.7088462710380554},
{'label': 'php', 'score': 0.698185920715332},
{'label': 'php', 'score': 0.6776195168495178},
{'label': 'php', 'score': 0.6776436567306519},
{'label': 'php', 'score': 0.7059316039085388},
{'label': 'php', 'score': 0.6654528379440308},
{'label': 'php', 'score': 0.6655356884002686},
{'label': 'php', 'score': 0.7040420770645142},
{'label': 'php', 'score': 0.667953610420227},
{'label': 'php', 'score': 0.6789735555648804},
{'label': 'php', 'score': 0.685849666595459},
{'label': 'php', 'score': 0.6385822892189026},
{'label': 'php', 'score': 0.7087535262107849},
{'label': 'php', 'score': 0.7013763189315796},
{'label': 'php', 'score': 0.5088586807250977},
{'label': 'php', 'score': 0.70463627576828},
{'label': 'php', 'score': 0.7074404358863831},
{'label': 'php', 'score': 0.6886810660362244},
{'label': 'php', 'score': 0.6946655511856079},
{'label': 'php', 'score': 0.46614181995391846},
{'label': 'jquery', 'score': 0.5908973813056946},
{'label': 'php', 'score': 0.6965422034263611},
{'label': 'php', 'score': 0.7073785066604614},
{'label': 'php', 'score': 0.7026752829551697},
{'label': 'php', 'score': 0.6955185532569885},
{'label': 'php', 'score': 0.6844291090965271},
{'label': 'php', 'score': 0.68308025598526},
{'label': 'php', 'score': 0.6990556120872498},
{'label': 'php', 'score': 0.7071481943130493},
{'label': 'php', 'score': 0.6974250674247742},
{'label': 'php', 'score': 0.6995997428894043},
{'label': 'php', 'score': 0.6905179619789124},
{'label': 'php', 'score': 0.6624079346656799},
{'label': 'php', 'score': 0.6936211585998535},
{'label': 'php', 'score': 0.7055285573005676},
{'label': 'php', 'score': 0.5005691647529602},
{'label': 'php', 'score': 0.7134020924568176},
{'label': 'php', 'score': 0.695520281791687},
```

{'label': 'php', 'score': 0.47526443004608154},
{'label': 'asp.net', 'score': 0.23099318146705627},
{'label': 'php', 'score': 0.6617562770843506},
{'label': 'php', 'score': 0.6565548181533813},
{'label': 'php', 'score': 0.6571404337882996},
{'label': 'php', 'score': 0.683904230594635},
{'label': 'php', 'score': 0.7018505334854126},
{'label': 'php', 'score': 0.7077620625495911},
{'label': 'php', 'score': 0.7037099599838257},
{'label': 'php', 'score': 0.7094395160675049},
{'label': 'php', 'score': 0.6838193535804749},
{'label': 'php', 'score': 0.6956809163093567},
{'label': 'asp.net', 'score': 0.1993534415960312},
{'label': 'php', 'score': 0.7109918594360352},
{'label': 'c#', 'score': 0.20743443071842194},
{'label': 'javascript', 'score': 0.319701611995697},
{'label': 'php', 'score': 0.696204662322998},
{'label': 'php', 'score': 0.6857956647872925},
{'label': 'iphone', 'score': 0.3356183171272278},
{'label': 'php', 'score': 0.6794134974479675},
{'label': 'php', 'score': 0.7141058444976807},
{'label': 'php', 'score': 0.4912610650062561},
{'label': 'iphone', 'score': 0.23382185399532318},
{'label': 'php', 'score': 0.43430033326148987},
{'label': 'php', 'score': 0.634160041809082},
{'label': 'php', 'score': 0.7016432285308838},
{'label': 'php', 'score': 0.47805169224739075},
{'label': 'php', 'score': 0.6772809028625488},
{'label': 'php', 'score': 0.7031415104866028},
{'label': 'php', 'score': 0.4289731979370117},
{'label': 'php', 'score': 0.6903135776519775},
{'label': 'php', 'score': 0.7052801847457886},
{'label': 'c#', 'score': 0.24384552240371704},
{'label': 'php', 'score': 0.6359375715255737},
{'label': 'php', 'score': 0.7072029709815979},
{'label': 'php', 'score': 0.6972955465316772},
{'label': 'php', 'score': 0.7167026400566101},
{'label': 'php', 'score': 0.6970607042312622},
{'label': 'php', 'score': 0.7030326724052429},
{'label': 'php', 'score': 0.7077276110649109},
{'label': 'php', 'score': 0.6675412654876709},
{'label': 'php', 'score': 0.6879643797874451},
{'label': 'asp.net', 'score': 0.19536103308200836},
{'label': 'php', 'score': 0.7069859504699707},
{'label': 'php', 'score': 0.7089122533798218},
{'label': 'php', 'score': 0.6764530539512634},
{'label': 'php', 'score': 0.7061066627502441},

{'label': 'php', 'score': 0.6933153867721558},
{'label': 'php', 'score': 0.7082496881484985},
{'label': 'php', 'score': 0.678881049156189},
{'label': 'php', 'score': 0.6979261636734009},
{'label': 'php', 'score': 0.5091409683227539},
{'label': 'php', 'score': 0.6893354654312134},
{'label': 'asp.net', 'score': 0.39814993739128113},
{'label': 'php', 'score': 0.6442790031433105},
{'label': 'php', 'score': 0.710310161113739},
{'label': 'php', 'score': 0.6740219593048096},
{'label': 'php', 'score': 0.5152489542961121},
{'label': 'php', 'score': 0.6546308398246765},
{'label': 'php', 'score': 0.6870229840278625},
{'label': 'php', 'score': 0.6828672885894775},
{'label': 'php', 'score': 0.7173230648040771},
{'label': 'php', 'score': 0.6782715916633606},
{'label': 'php', 'score': 0.7100273370742798},
{'label': 'php', 'score': 0.6295971274375916},
{'label': 'php', 'score': 0.6751106977462769},
{'label': 'php', 'score': 0.6759113073348999},
{'label': 'php', 'score': 0.6937704682350159},
{'label': 'php', 'score': 0.6932650804519653},
{'label': 'php', 'score': 0.7053393125534058},
{'label': 'php', 'score': 0.7011265754699707},
{'label': 'php', 'score': 0.7031576037406921},
{'label': 'php', 'score': 0.6991036534309387},
{'label': 'php', 'score': 0.6940980553627014},
{'label': 'php', 'score': 0.7071652412414551},
{'label': 'php', 'score': 0.7091761827468872},
{'label': 'php', 'score': 0.6905404329299927},
{'label': 'php', 'score': 0.7018958926200867},
{'label': 'php', 'score': 0.5146055817604065},
{'label': 'php', 'score': 0.6836174130439758},
{'label': 'php', 'score': 0.7092174887657166},
{'label': 'php', 'score': 0.26801997423171997},
{'label': 'php', 'score': 0.6742852926254272},
{'label': 'php', 'score': 0.6921175718307495},
{'label': 'php', 'score': 0.4964448809623718},
{'label': 'php', 'score': 0.6672448515892029},
{'label': 'php', 'score': 0.6848413944244385},
{'label': 'jquery', 'score': 0.5797058939933777},
{'label': 'php', 'score': 0.7036151885986328},
{'label': 'php', 'score': 0.6926997900009155},
{'label': 'php', 'score': 0.7035225629806519},
{'label': 'php', 'score': 0.6835882067680359},
{'label': 'php', 'score': 0.3648107647895813},
{'label': 'php', 'score': 0.7029073238372803},

{'label': 'jquery', 'score': 0.6830176115036011},
{'label': 'javascript', 'score': 0.6079242825508118},
{'label': 'javascript', 'score': 0.616793692111969},
{'label': 'javascript', 'score': 0.6211775541305542},
{'label': 'javascript', 'score': 0.580828070640564},
{'label': 'javascript', 'score': 0.611118495464325},
{'label': 'javascript', 'score': 0.5990285277366638},
{'label': 'javascript', 'score': 0.6207003593444824},
{'label': 'javascript', 'score': 0.6147968173027039},
{'label': 'javascript', 'score': 0.6226347088813782},
{'label': 'c++', 'score': 0.30520200729370117},
{'label': 'javascript', 'score': 0.6152518391609192},
{'label': 'javascript', 'score': 0.6159002780914307},
{'label': 'java', 'score': 0.5359857678413391},
{'label': 'javascript', 'score': 0.6287047863006592},
{'label': 'javascript', 'score': 0.6223008632659912},
{'label': 'c#', 'score': 0.19625280797481537},
{'label': 'c++', 'score': 0.269807904958725},
{'label': 'jquery', 'score': 0.6889773607254028},
{'label': 'javascript', 'score': 0.5995703339576721},
{'label': 'jquery', 'score': 0.6775515079498291},
{'label': 'javascript', 'score': 0.5848907828330994},
{'label': 'javascript', 'score': 0.6200507879257202},
{'label': 'javascript', 'score': 0.6170188188552856},
{'label': 'iphone', 'score': 0.22872337698936462},
{'label': 'javascript', 'score': 0.6164118647575378},
{'label': 'javascript', 'score': 0.6020052433013916},
{'label': 'jquery', 'score': 0.7032718658447266},
{'label': 'c#', 'score': 0.1926666796207428},
{'label': 'java', 'score': 0.5566807985305786},
{'label': 'javascript', 'score': 0.6179170608520508},
{'label': 'jquery', 'score': 0.6014814972877502},
{'label': 'javascript', 'score': 0.6170163750648499},
{'label': 'javascript', 'score': 0.6099667549133301},
{'label': 'javascript', 'score': 0.6129487156867981},
{'label': 'javascript', 'score': 0.6132144331932068},
{'label': 'javascript', 'score': 0.6050900816917419},
{'label': 'javascript', 'score': 0.5032682418823242},
{'label': 'c#', 'score': 0.19679217040538788},
{'label': 'javascript', 'score': 0.6021708846092224},
{'label': 'javascript', 'score': 0.6271852850914001},
{'label': 'javascript', 'score': 0.588515043258667},
{'label': 'iphone', 'score': 0.15354543924331665},
{'label': 'javascript', 'score': 0.604826033115387},
{'label': 'javascript', 'score': 0.596237301826477},
{'label': 'javascript', 'score': 0.6180965304374695},
{'label': 'javascript', 'score': 0.6165375709533691},

```
{'label': 'javascript', 'score': 0.5921661853790283},
{'label': 'javascript', 'score': 0.6257862448692322},
{'label': 'javascript', 'score': 0.6195117831230164},
{'label': 'jquery', 'score': 0.5081155300140381},
{'label': 'javascript', 'score': 0.6197485327720642},
{'label': 'c#', 'score': 0.2227008193731308},
{'label': 'jquery', 'score': 0.6769303679466248},
{'label': 'javascript', 'score': 0.5731130242347717},
{'label': 'javascript', 'score': 0.6152353286743164},
{'label': 'asp.net', 'score': 0.229902982711792},
{'label': 'javascript', 'score': 0.2904086709022522},
{'label': 'c++', 'score': 0.26300516724586487},
{'label': 'javascript', 'score': 0.6110489964485168},
{'label': 'javascript', 'score': 0.6257302165031433},
{'label': 'javascript', 'score': 0.6109887361526489},
{'label': 'jquery', 'score': 0.6900594830513},
{'label': 'javascript', 'score': 0.48764944076538086},
{'label': 'javascript', 'score': 0.6098031997680664},
{'label': 'javascript', 'score': 0.6193440556526184},
{'label': 'javascript', 'score': 0.20894181728363037},
{'label': 'javascript', 'score': 0.6190096735954285},
{'label': 'javascript', 'score': 0.6255688071250916},
{'label': 'jquery', 'score': 0.32474485039711},
{'label': 'javascript', 'score': 0.6112467646598816},
{'label': 'javascript', 'score': 0.6165375709533691},
{'label': 'javascript', 'score': 0.6078915596008301},
{'label': 'javascript', 'score': 0.6201697587966919},
{'label': 'javascript', 'score': 0.5836775302886963},
{'label': 'jquery', 'score': 0.66949462890625},
{'label': 'javascript', 'score': 0.598888635635376},
{'label': 'jquery', 'score': 0.5536185503005981},
{'label': 'javascript', 'score': 0.6016958951950073},
{'label': 'c#', 'score': 0.19796480238437653},
{'label': 'javascript', 'score': 0.6211577653884888},
{'label': 'javascript', 'score': 0.6201188564300537},
{'label': 'jquery', 'score': 0.4118015170097351},
{'label': 'javascript', 'score': 0.6039242148399353},
{'label': 'javascript', 'score': 0.5825278162956238},
{'label': 'javascript', 'score': 0.6089004278182983},
{'label': 'c++', 'score': 0.28347522020339966},
{'label': 'javascript', 'score': 0.6011570692062378},
{'label': 'javascript', 'score': 0.5471031665802002},
{'label': 'jquery', 'score': 0.3378503918647766},
{'label': 'javascript', 'score': 0.615505576133728},
{'label': 'javascript', 'score': 0.6200714707374573},
{'label': 'javascript', 'score': 0.6275813579559326},
{'label': 'javascript', 'score': 0.5903646349906921},
```

{'label': 'javascript', 'score': 0.6092986464500427},
{'label': 'asp.net', 'score': 0.23050053417682648},
{'label': 'javascript', 'score': 0.6108306646347046},
{'label': 'iphone', 'score': 0.3927921652793884},
{'label': 'javascript', 'score': 0.604378342628479},
{'label': 'javascript', 'score': 0.6155899167060852},
{'label': 'php', 'score': 0.16397684812545776},
{'label': 'javascript', 'score': 0.5936707258224487},
{'label': 'javascript', 'score': 0.21129417419433594},
{'label': 'javascript', 'score': 0.6079893708229065},
{'label': 'javascript', 'score': 0.6193433403968811},
{'label': 'javascript', 'score': 0.6159656643867493},
{'label': 'c#', 'score': 0.23192265629768372},
{'label': 'php', 'score': 0.20566970109939575},
{'label': 'javascript', 'score': 0.6166580319404602},
{'label': 'javascript', 'score': 0.6062904000282288},
{'label': 'javascript', 'score': 0.5494560599327087},
{'label': 'javascript', 'score': 0.6202976107597351},
{'label': 'javascript', 'score': 0.5927433967590332},
{'label': 'jquery', 'score': 0.7055507898330688},
{'label': 'javascript', 'score': 0.615017831325531},
{'label': 'javascript', 'score': 0.5990434288978577},
{'label': 'javascript', 'score': 0.5689058303833008},
{'label': 'javascript', 'score': 0.6041862964630127},
{'label': 'c++', 'score': 0.30459076166152954},
{'label': 'javascript', 'score': 0.5714124441146851},
{'label': 'javascript', 'score': 0.6136137247085571},
{'label': 'c#', 'score': 0.24179412424564362},
{'label': 'javascript', 'score': 0.595827579498291},
{'label': 'javascript', 'score': 0.5635451078414917},
{'label': 'javascript', 'score': 0.605962872505188},
{'label': 'javascript', 'score': 0.2216506004333496},
{'label': 'javascript', 'score': 0.6019830107688904},
{'label': 'c++', 'score': 0.25740155577659607},
{'label': 'c++', 'score': 0.36238420009613037},
{'label': 'c#', 'score': 0.1537732183933258},
{'label': 'javascript', 'score': 0.6049597859382629},
{'label': 'javascript', 'score': 0.6269087791442871},
{'label': 'javascript', 'score': 0.6086750626564026},
{'label': 'c++', 'score': 0.26656726002693176},
{'label': 'c++', 'score': 0.33755621314048767},
{'label': 'javascript', 'score': 0.30149200558662415},
{'label': 'c++', 'score': 0.31283193826675415},
{'label': 'javascript', 'score': 0.6163541078567505},
{'label': 'javascript', 'score': 0.5963265895843506},
{'label': 'javascript', 'score': 0.5853301286697388},
{'label': 'javascript', 'score': 0.539530336856842},

{'label': 'jquery', 'score': 0.4737502634525299},
{'label': 'javascript', 'score': 0.575910210609436},
{'label': 'jquery', 'score': 0.2801506221294403},
{'label': 'c#', 'score': 0.24043866991996765},
{'label': 'javascript', 'score': 0.5505529046058655},
{'label': 'javascript', 'score': 0.6198399662971497},
{'label': 'asp.net', 'score': 0.21587806940078735},
{'label': 'javascript', 'score': 0.6105730533599854},
{'label': 'javascript', 'score': 0.6001565456390381},
{'label': 'javascript', 'score': 0.6226856708526611},
{'label': 'javascript', 'score': 0.6164095401763916},
{'label': 'javascript', 'score': 0.585765540599823},
{'label': 'javascript', 'score': 0.5921544432640076},
{'label': 'javascript', 'score': 0.6073955297470093},
{'label': 'php', 'score': 0.69828861951828},
{'label': 'javascript', 'score': 0.6309557557106018},
{'label': 'javascript', 'score': 0.504911482334137},
{'label': 'javascript', 'score': 0.6218494772911072},
{'label': 'javascript', 'score': 0.6111397743225098},
{'label': 'javascript', 'score': 0.6175492405891418},
{'label': 'python', 'score': 0.3171026110649109},
{'label': 'php', 'score': 0.4400644600391388},
{'label': 'javascript', 'score': 0.5923292636871338},
{'label': 'c#', 'score': 0.1785585880279541},
{'label': 'javascript', 'score': 0.6067209839820862},
{'label': 'javascript', 'score': 0.6215793490409851},
{'label': 'javascript', 'score': 0.6202128529548645},
{'label': 'javascript', 'score': 0.6134735345840454},
{'label': 'javascript', 'score': 0.6165505647659302},
{'label': 'javascript', 'score': 0.6128037571907043},
{'label': 'javascript', 'score': 0.6212300658226013},
{'label': 'javascript', 'score': 0.6198769807815552},
{'label': 'javascript', 'score': 0.5794166922569275},
{'label': 'c++', 'score': 0.29195132851600647},
{'label': 'c++', 'score': 0.27992966771125793},
{'label': 'javascript', 'score': 0.6179161071777344},
{'label': 'asp.net', 'score': 0.2674875855445862},
{'label': 'jquery', 'score': 0.6437039971351624},
{'label': 'javascript', 'score': 0.5982268452644348},
{'label': 'javascript', 'score': 0.6278131604194641},
{'label': 'javascript', 'score': 0.6237597465515137},
{'label': 'javascript', 'score': 0.6084674000740051},
{'label': 'javascript', 'score': 0.5925342440605164},
{'label': 'javascript', 'score': 0.6127628087997437},
{'label': 'asp.net', 'score': 0.30108097195625305},
{'label': 'c++', 'score': 0.33513185381889343},
{'label': 'javascript', 'score': 0.6116468906402588},

```
{'label': 'javascript', 'score': 0.6167373061180115},
{'label': 'javascript', 'score': 0.585217297077179},
{'label': 'javascript', 'score': 0.6016601324081421},
{'label': 'javascript', 'score': 0.6083412766456604},
{'label': 'javascript', 'score': 0.6004777550697327},
{'label': 'asp.net', 'score': 0.3121016025543213},
{'label': 'javascript', 'score': 0.5964862704277039},
{'label': 'javascript', 'score': 0.612449049949646},
{'label': 'javascript', 'score': 0.604738712310791},
{'label': 'javascript', 'score': 0.624538242816925},
{'label': 'asp.net', 'score': 0.23794153332710266},
{'label': 'javascript', 'score': 0.5903770327568054},
{'label': 'android', 'score': 0.6716528534889221},
{'label': 'android', 'score': 0.7424478530883789},
{'label': 'android', 'score': 0.7412161827087402},
{'label': 'android', 'score': 0.7482861876487732},
{'label': 'android', 'score': 0.7201215624809265},
{'label': 'android', 'score': 0.7504886388778687},
{'label': 'android', 'score': 0.7290259003639221},
{'label': 'android', 'score': 0.7450205683708191},
{'label': 'android', 'score': 0.7560275197029114},
{'label': 'android', 'score': 0.7576512098312378},
{'label': 'android', 'score': 0.7562099099159241},
{'label': 'iphone', 'score': 0.24817471206188202},
{'label': 'android', 'score': 0.7305458784103394},
{'label': 'android', 'score': 0.7275570034980774},
{'label': 'iphone', 'score': 0.3354015350341797},
{'label': 'java', 'score': 0.6307052969932556},
{'label': 'android', 'score': 0.747502863407135},
{'label': 'android', 'score': 0.7562099099159241},
{'label': 'iphone', 'score': 0.3670744299888611},
{'label': 'android', 'score': 0.7497662305831909},
{'label': 'android', 'score': 0.7494368553161621},
{'label': 'iphone', 'score': 0.3911207914352417},
{'label': 'iphone', 'score': 0.34888213872909546},
{'label': 'android', 'score': 0.74811190366745},
{'label': 'android', 'score': 0.7439162731170654},
{'label': 'android', 'score': 0.7223688364028931},
{'label': 'php', 'score': 0.6734864711761475},
{'label': 'android', 'score': 0.6214867830276489},
{'label': 'java', 'score': 0.62017422914505},
{'label': 'android', 'score': 0.7542790174484253},
{'label': 'android', 'score': 0.7404230833053589},
{'label': 'php', 'score': 0.38133636116981506},
{'label': 'android', 'score': 0.7222671508789062},
{'label': 'android', 'score': 0.7353474497795105},
{'label': 'android', 'score': 0.7418168187141418},
```

{'label': 'iphone', 'score': 0.45167458057403564},
{'label': 'android', 'score': 0.7502014636993408},
{'label': 'iphone', 'score': 0.3414381742477417},
{'label': 'android', 'score': 0.7509039044380188},
{'label': 'iphone', 'score': 0.320442795753479},
{'label': 'android', 'score': 0.7439302802085876},
{'label': 'android', 'score': 0.7322848439216614},
{'label': 'android', 'score': 0.7468224167823792},
{'label': 'asp.net', 'score': 0.21375460922718048},
{'label': 'iphone', 'score': 0.267074316740036},
{'label': 'android', 'score': 0.7490736246109009},
{'label': 'android', 'score': 0.736701488494873},
{'label': 'android', 'score': 0.7540411949157715},
{'label': 'android', 'score': 0.7356501221656799},
{'label': 'java', 'score': 0.6193434596061707},
{'label': 'asp.net', 'score': 0.24286438524723053},
{'label': 'asp.net', 'score': 0.27151182293891907},
{'label': 'android', 'score': 0.7492297291755676},
{'label': 'android', 'score': 0.7407581806182861},
{'label': 'android', 'score': 0.741920530796051},
{'label': 'android', 'score': 0.7020562887191772},
{'label': 'android', 'score': 0.7458301186561584},
{'label': 'iphone', 'score': 0.35026514530181885},
{'label': 'android', 'score': 0.7374668121337891},
{'label': 'android', 'score': 0.6759082674980164},
{'label': 'android', 'score': 0.7379442453384399},
{'label': 'iphone', 'score': 0.3833078145980835},
{'label': 'android', 'score': 0.7251616716384888},
{'label': 'android', 'score': 0.7519783973693848},
{'label': 'android', 'score': 0.734352707862854},
{'label': 'android', 'score': 0.7383087873458862},
{'label': 'android', 'score': 0.640169620513916},
{'label': 'java', 'score': 0.6029773950576782},
{'label': 'android', 'score': 0.7325422167778015},
{'label': 'iphone', 'score': 0.1745244562625885},
{'label': 'android', 'score': 0.7240275740623474},
{'label': 'android', 'score': 0.737092912197113},
{'label': 'iphone', 'score': 0.331928014755249},
{'label': 'android', 'score': 0.3559122681617737},
{'label': 'android', 'score': 0.7388221025466919},
{'label': 'asp.net', 'score': 0.3713480532169342},
{'label': 'android', 'score': 0.7029090523719788},
{'label': 'iphone', 'score': 0.37668779492378235},
{'label': 'android', 'score': 0.7442251443862915},
{'label': 'android', 'score': 0.7429421544075012},
{'label': 'android', 'score': 0.7318833470344543},
{'label': 'c#', 'score': 0.246387779712677},

```
{'label': 'iphone', 'score': 0.4434128701686859},
{'label': 'asp.net', 'score': 0.2037971019744873},
{'label': 'iphone', 'score': 0.19792386889457703},
{'label': 'iphone', 'score': 0.42470163106918335},
{'label': 'android', 'score': 0.7529975175857544},
{'label': 'android', 'score': 0.7558742165565491},
{'label': 'android', 'score': 0.7265777587890625},
{'label': 'android', 'score': 0.7416625618934631},
{'label': 'android', 'score': 0.7394019961357117},
{'label': 'android', 'score': 0.7555270791053772},
{'label': 'android', 'score': 0.7431713938713074},
{'label': 'android', 'score': 0.7380423545837402},
{'label': 'android', 'score': 0.7444494962692261},
{'label': 'android', 'score': 0.7395526170730591},
{'label': 'android', 'score': 0.724953293800354},
{'label': 'c#', 'score': 0.3132789433002472},
{'label': 'android', 'score': 0.728074848651886},
{'label': 'android', 'score': 0.7471926212310791},
{'label': 'android', 'score': 0.7419967651367188},
{'label': 'iphone', 'score': 0.25487884879112244},
{'label': 'android', 'score': 0.7370752692222595},
{'label': 'iphone', 'score': 0.46234697103500366},
{'label': 'iphone', 'score': 0.29052799940109253},
{'label': 'android', 'score': 0.7509146332740784},
{'label': 'android', 'score': 0.7232123613357544},
{'label': 'android', 'score': 0.7564176321029663},
{'label': 'android', 'score': 0.7236469388008118},
{'label': 'android', 'score': 0.7119142413139343},
{'label': 'android', 'score': 0.7420920133590698},
{'label': 'android', 'score': 0.7362130880355835},
{'label': 'android', 'score': 0.7258647084236145},
{'label': 'android', 'score': 0.7292203307151794},
{'label': 'android', 'score': 0.7226149439811707},
{'label': 'iphone', 'score': 0.2894640564918518},
{'label': 'android', 'score': 0.712628185749054},
{'label': 'iphone', 'score': 0.34523534774780273},
{'label': 'android', 'score': 0.7497782707214355},
{'label': 'android', 'score': 0.7311979532241821},
{'label': 'c#', 'score': 0.21963021159172058},
{'label': 'android', 'score': 0.7407196760177612},
{'label': 'iphone', 'score': 0.291492760181427},
{'label': 'android', 'score': 0.7200924158096313},
{'label': 'android', 'score': 0.7393238544464111},
{'label': 'android', 'score': 0.7552582621574402},
{'label': 'android', 'score': 0.7368161678314209},
{'label': 'android', 'score': 0.7330918312072754},
{'label': 'iphone', 'score': 0.2864637076854706},
```

{'label': 'java', 'score': 0.4927942454814911},
{'label': 'android', 'score': 0.744785726070404},
{'label': 'android', 'score': 0.7311314940452576},
{'label': 'java', 'score': 0.5613517165184021},
{'label': 'iphone', 'score': 0.23294208943843842},
{'label': 'android', 'score': 0.7040212750434875},
{'label': 'android', 'score': 0.742196798324585},
{'label': 'android', 'score': 0.738429844379425},
{'label': 'android', 'score': 0.7301720380783081},
{'label': 'android', 'score': 0.7457019686698914},
{'label': 'android', 'score': 0.3440427780151367},
{'label': 'iphone', 'score': 0.22814112901687622},
{'label': 'android', 'score': 0.7553722858428955},
{'label': 'asp.net', 'score': 0.2691638469696045},
{'label': 'android', 'score': 0.7571807503700256},
{'label': 'iphone', 'score': 0.6253128051757812},
{'label': 'android', 'score': 0.7486850023269653},
{'label': 'android', 'score': 0.7370760440826416},
{'label': 'android', 'score': 0.7293469905853271},
{'label': 'android', 'score': 0.7415677309036255},
{'label': 'android', 'score': 0.734533965587616},
{'label': 'android', 'score': 0.7243310213088989},
{'label': 'android', 'score': 0.7241151928901672},
{'label': 'android', 'score': 0.7369401454925537},
{'label': 'android', 'score': 0.7397056818008423},
{'label': 'android', 'score': 0.7403814196586609},
{'label': 'android', 'score': 0.7528306841850281},
{'label': 'android', 'score': 0.7316986322402954},
{'label': 'android', 'score': 0.7328152656555176},
{'label': 'android', 'score': 0.757263720035553},
{'label': 'android', 'score': 0.732794463634491},
{'label': 'android', 'score': 0.7396886944770813},
{'label': 'android', 'score': 0.7297115921974182},
{'label': 'android', 'score': 0.7320613265037537},
{'label': 'android', 'score': 0.7487497925758362},
{'label': 'asp.net', 'score': 0.22340558469295502},
{'label': 'android', 'score': 0.6515841484069824},
{'label': 'android', 'score': 0.7435297966003418},
{'label': 'android', 'score': 0.7372156977653503},
{'label': 'android', 'score': 0.7272995710372925},
{'label': 'android', 'score': 0.7494709491729736},
{'label': 'iphone', 'score': 0.19620312750339508},
{'label': 'android', 'score': 0.7338091731071472},
{'label': 'android', 'score': 0.6415929794311523},
{'label': 'android', 'score': 0.7455134987831116},
{'label': 'android', 'score': 0.7514002323150635},
{'label': 'android', 'score': 0.612357497215271},

```
{'label': 'android', 'score': 0.7506657838821411},
{'label': 'android', 'score': 0.7200497984886169},
{'label': 'android', 'score': 0.7415062785148621},
{'label': 'android', 'score': 0.7397153973579407},
{'label': 'iphone', 'score': 0.2144499272108078},
{'label': 'android', 'score': 0.7182732224464417},
{'label': 'android', 'score': 0.7423161268234253},
{'label': 'android', 'score': 0.750735878944397},
{'label': 'android', 'score': 0.7400001287460327},
{'label': 'android', 'score': 0.7430348992347717},
{'label': 'android', 'score': 0.7427866458892822},
{'label': 'android', 'score': 0.7351692914962769},
{'label': 'asp.net', 'score': 0.3189341127872467},
{'label': 'android', 'score': 0.7362954616546631},
{'label': 'android', 'score': 0.7425981163978577},
{'label': 'android', 'score': 0.7527831196784973},
{'label': 'android', 'score': 0.7482730746269226},
{'label': 'iphone', 'score': 0.3622899651527405},
{'label': 'android', 'score': 0.7400481104850769},
{'label': 'android', 'score': 0.7440089583396912},
{'label': 'android', 'score': 0.7179732918739319},
{'label': 'iphone', 'score': 0.40015360713005066},
{'label': 'android', 'score': 0.21591217815876007},
{'label': 'android', 'score': 0.7427411675453186},
…]
```

[ ]:

```
```