ShritejShrikant_Chavan_HW_5_PartB

October 12, 2023

1 HW5 PartB - MultiClass Classification with different checkpoints- 6 Points

Homework Instructions: Model Performance Comparisons

Objective:
Compare the performance of different model checkpoints on a given task using the provided functions and notebook setup.

Instructions:

1. **Setup & Initialization:**
   - Start by setting up your environment. Ensure you have all necessary dependencies installed.
   - All functions and required code blocks are pre-written for you.
2. **Experiment 1 - Using DistilBERT:**
   - In the specified code block, set the model checkpoint name to `"distilbert-base-uncased"`.
3. **Experiment 2 - Choosing a New Model:**
   - Search for a model similar in size to DistilBERT that have a similar or better performance.
   - Get the model check point name from the Hugging Face Model Hub.
   - In this experiment, replace the model checkpoint name with the one you've chosen from the Model Hub.
4. **Experiment 3 - Another Model Choice:**
   - Go back to the Hugging Face Model Hub.
   - Select another model (different from your choice in Experiment 2) with a similar size to DistilBERT.
   - Update the model checkpoint name in the cell for experiment 3.
5. **Conclusion:**
   - Analyze the results of all three experiments.
   - Discuss any differences in performance between the models. What might be causing these differences?
   - Conclude by summarizing your findings and providing insights on which model checkpoint performed best and why.

Note: It's essential to only change the model checkpoint name in each experiment. All other parameters and functions should remain unchanged to ensure a fair comparison between the models.

- **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_PartB**

This notebook has been modified to include functions that allow switching between different pre-trained models for training and evaluation.

1.1 Set Up Environment

[1]:
```
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") # CHANGE BASED ON YOUR␣
↪SETUP
else:
    base_folder = Path("/home/harpreet/Insync/google_drive_shaannoor/data") #␣
↪CHANGE BASED ON YOUR SETUP


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
import gc

from datasets.utils.logging import disable_progress_bar
disable_progress_bar()
```

```
data_folder = base_folder/'datasets/Classification_HW/csv_files' # CHANGE BASED␣
↪ON YOUR SETUP
model_folder = base_folder/'models/nlp_spring_2023/HW5' # CHANGE BASED ON YOUR␣
↪SETUP
model_folder.mkdir(exist_ok=True)
```

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

kB 7.9 MB/s eta 0:00:00
                                          7.7/7.7 MB
37.7 MB/s eta 0:00:00
                                          81.4/81.4 kB
10.9 MB/s eta 0:00:00
                                          2.1/2.1 MB
67.2 MB/s eta 0:00:00
                                          258.1/258.1 kB
28.3 MB/s eta 0:00:00
                                          115.3/115.3 kB
14.2 MB/s eta 0:00:00
                                          194.1/194.1 kB
23.9 MB/s eta 0:00:00
                                          134.8/134.8 kB
18.0 MB/s eta 0:00:00
                                          302.0/302.0 kB
35.5 MB/s eta 0:00:00
                                          3.8/3.8 MB
86.8 MB/s eta 0:00:00
                                          1.3/1.3 MB
76.0 MB/s eta 0:00:00
                                          190.0/190.0 kB
24.6 MB/s eta 0:00:00
                                          241.0/241.0 kB
30.8 MB/s eta 0:00:00
  Preparing metadata (setup.py) … done
                                          62.7/62.7 kB
8.3 MB/s eta 0:00:00
                                          295.0/295.0 kB
34.5 MB/s eta 0:00:00
  Building wheel for pathtools (setup.py) … done
```

———DO NOT CHANGE ANY OF THE FUNCTIONS————

1.2 Function to Load Dataset

[2]:
```
def load_custom_dataset(data_path, label_columns_name, text_column_name,␣
↪class_names=None):
    from datasets import load_dataset
    dataset = load_dataset('csv', data_files=str(data_path))
    selected_columns = {
    'text': dataset['train'][text_column_name],
    'label': dataset['train'][label_columns_name]
    }

    # Create a new dataset with the selected columns
    dataset_selected_columns = Dataset.from_dict(selected_columns)
    dataset_selected_columns.set_format(type='pandas')
    df = dataset_selected_columns[:]
    df['text'] = df['text'].fillna('')
    dataset_selected_columns.reset_format()

    if class_names:
        dataset_selected_columns = dataset_selected_columns.
↪cast_column('label', ClassLabel(names = class_names))

    return dataset_selected_columns
```

1.3 Function to Split Dataset

[3]:
```
def split_dataset(dataset, train_size, val_size, test_size):
    test_val_splits = dataset.train_test_split(train_size=train_size, seed=42,␣
↪stratify_by_column='label')
    train_split= test_val_splits['train']
    test_size_new =test_size/(test_size + val_size)
    test_val_splits = test_val_splits['test'].
↪train_test_split(test_size=test_size_new, seed=42,␣
↪stratify_by_column='label')
    val_split = test_val_splits['train']
    test_split = test_val_splits['test']

    train_val_dataset = DatasetDict({'train': train_split, 'val': val_split})
    test_dataset = DatasetDict({'test': test_split})

    return train_val_dataset, test_dataset
```

1.4 Function to Create smaller subset

[4]:
```
def get_small_balanced_subset(dataset, num_samples_per_class):
    subset = DatasetDict()

    for split in list(dataset.keys()):
        texts = []
        labels = []
        for label in range(10):
            label_texts = dataset[split].filter(lambda x: x['label'] ==␣
↪label)['text']
            label_subset = random.sample(list(label_texts),␣
↪num_samples_per_class)
            texts.extend(label_subset)
            labels.extend([label]*len(label_subset))

        subset[split] = Dataset.from_dict({'text': texts, 'label': labels})

    return subset
```

1.5 Function for Tokenization

[5]:
```
def get_tokenized_dataset(checkpoint, dataset):
    tokenizer = AutoTokenizer.from_pretrained(checkpoint)

    def tokenize_fn(batch):
        # Debug print statements
        # print(f"Type of batch['text']: {type(batch['text'])}")
        # print(f"First item in batch['text']: {batch['text'][0]}")
        return tokenizer(batch["text"], truncation=True)

    tokenized_dataset = dataset.map(tokenize_fn, batched=True)
    return tokenized_dataset
```

1.6 Function to Create Datasets

[6]:
```
def setup_dataset(data_folder, class_names, num_samples_per_class):

    # Constants for loading and splitting
    data_path = data_folder / 'multiclass_hw_basic_clean.csv'
    label_columns_name = 'Tag_Number_final'
    text_column_name = 'basic_cleaned_text'

    # 1. Load Dataset
    dataset = load_custom_dataset(data_path, label_columns_name,␣
↪text_column_name, class_names=class_names)
```

```
    # 2. Split Dataset
    train_val_dataset, test_dataset = split_dataset(dataset, train_size=0.6,␣
↪val_size=0.2, test_size=0.2)

    # 3. Get Small Balanced Subset
    train_val_subset = get_small_balanced_subset(train_val_dataset,␣
↪num_samples_per_class=num_samples_per_class)

    return train_val_subset, train_val_dataset, test_dataset
```

1.7 Function to Initialize Model

[7]:
```
def initialize_model(checkpoint, class_names):
    config = AutoConfig.from_pretrained(checkpoint)
    id2label = {}
    for id_, label_ in enumerate(class_names):
        id2label[str(id_)] = label_

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

    config.id2label = id2label
    config.label2id = label2id

    model = AutoModelForSequenceClassification.from_pretrained(checkpoint,␣
↪config=config)
    return model
```

1.8 Function to Compute Metrics

[8]:
```
def compute_metrics(eval_pred):

    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)

    f1_metric = evaluate.load("f1", average="macro")
    accuracy = evaluate.load("accuracy")

    evaluations = {}
    evaluations.update(f1_metric.compute(predictions=predictions, references =␣
↪labels, average="macro"))
    evaluations.update(accuracy.compute(predictions=predictions, references =␣
↪labels))

    return evaluations
```

1.9 Function to set Trainer

[9]:
```python
def get_trainer(model, training_args, train_dataset, eval_dataset,␣
↪compute_metrics, tokenizer, data_collator):
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=eval_dataset,
        compute_metrics=compute_metrics,
        tokenizer=tokenizer,
        data_collator= data_collator
    )
    return trainer
```

1.10 Function to plot confusion matrix

[10]:
```python
def log_and_plot_confusion_matrix(trainer, tokenized_val_dataset, class_names):
    # Perform prediction using the trainer
    valid_output = trainer.predict(tokenized_val_dataset)

    # Extract the predicted labels and true labels
    valid_preds = np.argmax(valid_output.predictions, axis=1)
    valid_labels = np.array(valid_output.label_ids)

    # Log the confusion matrix to wandb
    wandb.log({
        "conf_mat": wandb.plot.confusion_matrix(
            preds=valid_preds,
            y_true=valid_labels,
            class_names=class_names
        )
    })

    # Plot the confusion matrix using Matplotlib
    fig, ax = plt.subplots(figsize=(8, 6))
    ConfusionMatrixDisplay.from_predictions(
        y_true=valid_labels,
        y_pred=valid_preds,
        ax=ax,
        normalize="true",
        display_labels=class_names,
        xticks_rotation=90
    )
    plt.show()
```

1.11 Function to free memory

[11]:
```python
def free_memory():
    """
    Attempts to free up memory by deleting variables and running Python's␣
↪garbage collector.
    """
    gc.collect()
    for device_id in range(torch.cuda.device_count()):
        torch.cuda.set_device(device_id)
        torch.cuda.empty_cache()
    gc.collect()
```

1.12 Function to tokenize dataset and, train and eval models

ALLOWED TO CHANGE THE BLOCK IN THE FUNCTION BELOW

[12]:
```python
def tokenize_train_evaluate_log(training_args, checkpoint, base_folder,
                                class_names, train_val_subset, compute_metrics):
    # 1. Free memory
    free_memory()

    # 2. Setup wandb
    wandb.login()
    %env WANDB_PROJECT = nlp_course_fall_2023-HW5-Part-B-Colab

    ######################## ALLOWED TO CHANGE THIS BLOCK␣
↪################################################

    # MAKE SURE THE BASE FOLDER IS SETUP CORRECTLY
    #  YOU CAN CHANGE THIS LINE IF YOU WANT TO SAVE IN A DIFFERENT FOLDER

    model_folder = base_folder / "models" / "nlp_spring_2023/stack"/checkpoint
    model_folder.mkdir(exist_ok=True, parents=True)

    ######################## ALLOWED TO CHANGE THIS BLOCK␣
↪################################################


    # 3. Get Tokenized Dataset and Data Collator
    train_val_tokenized_dataset = get_tokenized_dataset(checkpoint,␣
↪train_val_subset)

    # 4. Initialize Model and Tokenizer
    model = initialize_model(checkpoint, class_names)
    tokenizer = AutoTokenizer.from_pretrained(checkpoint)
```

```
    # 5. Initialize Trainer
    data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
    trainer = get_trainer(model, training_args,␣
↪train_val_tokenized_dataset['train'],
                          train_val_tokenized_dataset['val'], compute_metrics,␣
↪tokenizer, data_collator)

    # 6. Train and Evaluate
    trainer.train()
    trainer.evaluate(train_val_tokenized_dataset['val'])

    # 7. Log Metrics and Plot
    log_and_plot_confusion_matrix(trainer, train_val_tokenized_dataset['val'],␣
↪class_names)

    best_model_checkpoint_step = trainer.state.best_model_checkpoint.
↪split('-')[-1]
    wandb.log({"best_model_checkpoint_step": best_model_checkpoint_step})
    print(f"The best model was saved at step {best_model_checkpoint_step}.")

    wandb.finish()
```

1.13 Initial Training Arguments

DO NOT CHANGE ANY ARGUMENTS

[13]:
```
training_args = TrainingArguments(
    # Training-specific configurations
    num_train_epochs=2,
    per_device_train_batch_size=16,
    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= 'exp1',  # Experiment name for Weights & Biases
)
```

2 Experiments

2.1 Dataset hyperparameters

[14]:
```
class_names = ['c#', 'java', 'php','javascript', 'android', 'jquery', 'c++', 
 ↪'python', 'iphone', 'asp.net']
num_samples_per_class = 200
train_val_subset, train_val_dataset, test_dataset = setup_dataset(data_folder,
 ↪class_names, num_samples_per_class)
```

2.2 Experiment 1 : distilbert-base-uncased

2.2.1 Trainer hyperparameters

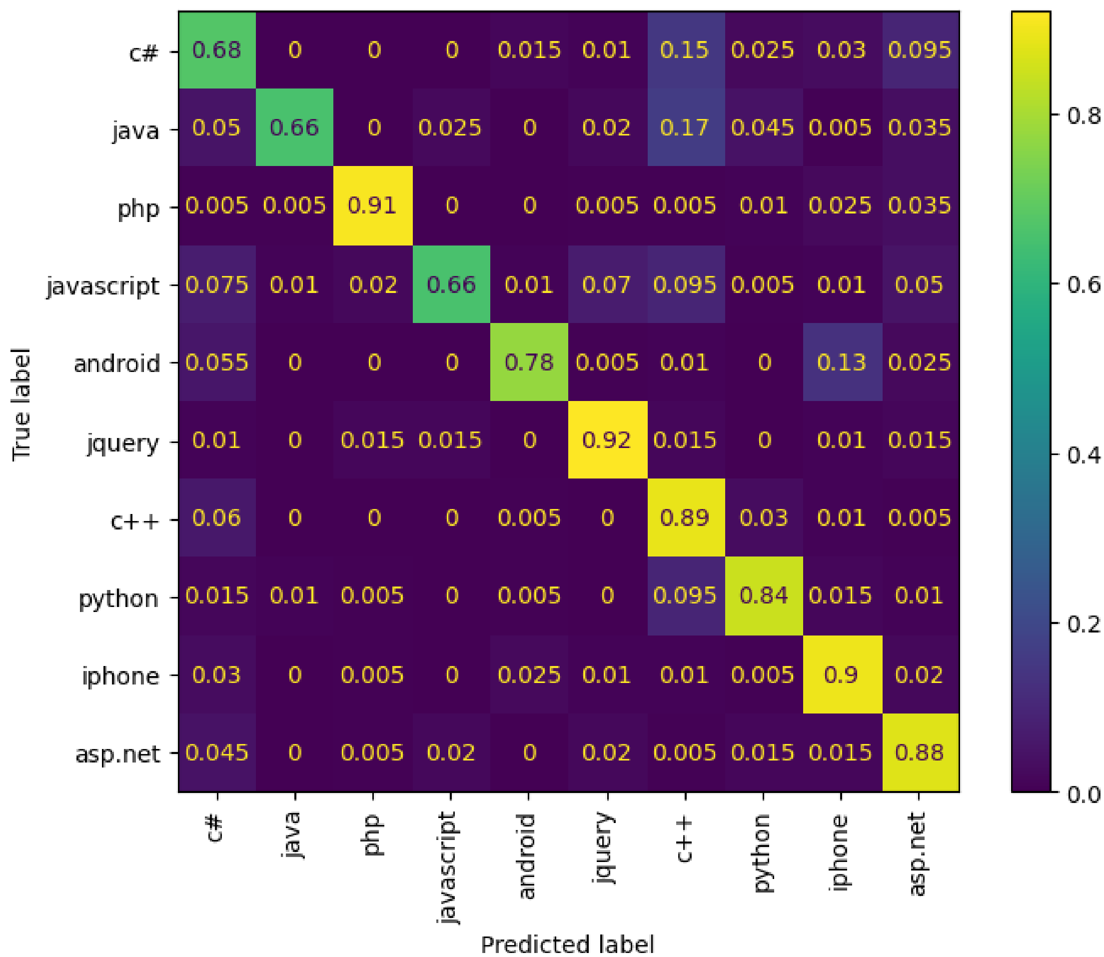
[ ]:
```
checkpoint = 'distilbert-base-uncased' # CODE HERE
training_args_dict = training_args.to_dict() # Convert TrainingArguments to
 ↪dictionary
training_args_dict['run_name'] = f'{checkpoint}-{num_samples_per_class}' #
 ↪Update the run_name
new_training_args = TrainingArguments(**training_args_dict)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1711:
FutureWarning: `--push_to_hub_token` is deprecated and will be removed in
version 5 of  Transformers. Use `--hub_token` instead.
  warnings.warn(
```

[ ]:
```
tokenize_train_evaluate_log(training_args= new_training_args,
 ↪checkpoint=checkpoint, base_folder=base_folder,
                            class_names=class_names,
 ↪train_val_subset=train_val_subset,
                            compute_metrics=compute_metrics)
```

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

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

env: WANDB_PROJECT=nlp_course_fall_2023-HW5-Part-B-Colab

Downloading (…)okenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]

Downloading (…)lve/main/config.json:   0%|          | 0.00/483 [00:00<?, ?B/s]
```

```
Downloading (…)solve/main/vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

Downloading (…)/main/tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

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

<IPython.core.display.HTML object>

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

<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>

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.

<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]

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




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

<IPython.core.display.HTML object>

VBox(children=(Label(value='0.007 MB of 0.007 MB uploaded (0.000 MB
 ↪deduped)\r'), FloatProgress(value=1.0, max…

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

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

2.3 Experiment 2 : albert-base-v1

2.3.1 Trainer hyperparameters

[15]:
```
checkpoint = 'albert-base-v1' # CODE HERE
training_args_dict = training_args.to_dict() # Convert TrainingArguments to␣
↪dictionary
training_args_dict['run_name'] = f'{checkpoint}-{num_samples_per_class}' #␣
↪Update the run_name
new_training_args = TrainingArguments(**training_args_dict)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1711:
FutureWarning: `--push_to_hub_token` is deprecated and will be removed in
version 5 of  Transformers. Use `--hub_token` instead.
  warnings.warn(
```

[16]:
```
tokenize_train_evaluate_log(training_args= new_training_args,␣
↪checkpoint=checkpoint, base_folder=base_folder,
                            class_names=class_names,␣
↪train_val_subset=train_val_subset,
                            compute_metrics=compute_metrics)
```

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

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

env: WANDB_PROJECT=nlp_course_fall_2023-HW5-Part-B-Colab

Downloading (…)lve/main/config.json:   0%|          | 0.00/684 [00:00<?, ?B/s]

Downloading (…)ve/main/spiece.model:   0%|          | 0.00/760k [00:00<?, ?B/s]

Downloading (…)/main/tokenizer.json:   0%|          | 0.00/1.31M [00:00<?, ?B/s]

Downloading model.safetensors:   0%|          | 0.00/47.4M [00:00<?, ?B/s]

Some weights of AlbertForSequenceClassification were not initialized from the
model checkpoint at albert-base-v1 and are newly initialized:
['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

<IPython.core.display.HTML object>

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

<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>

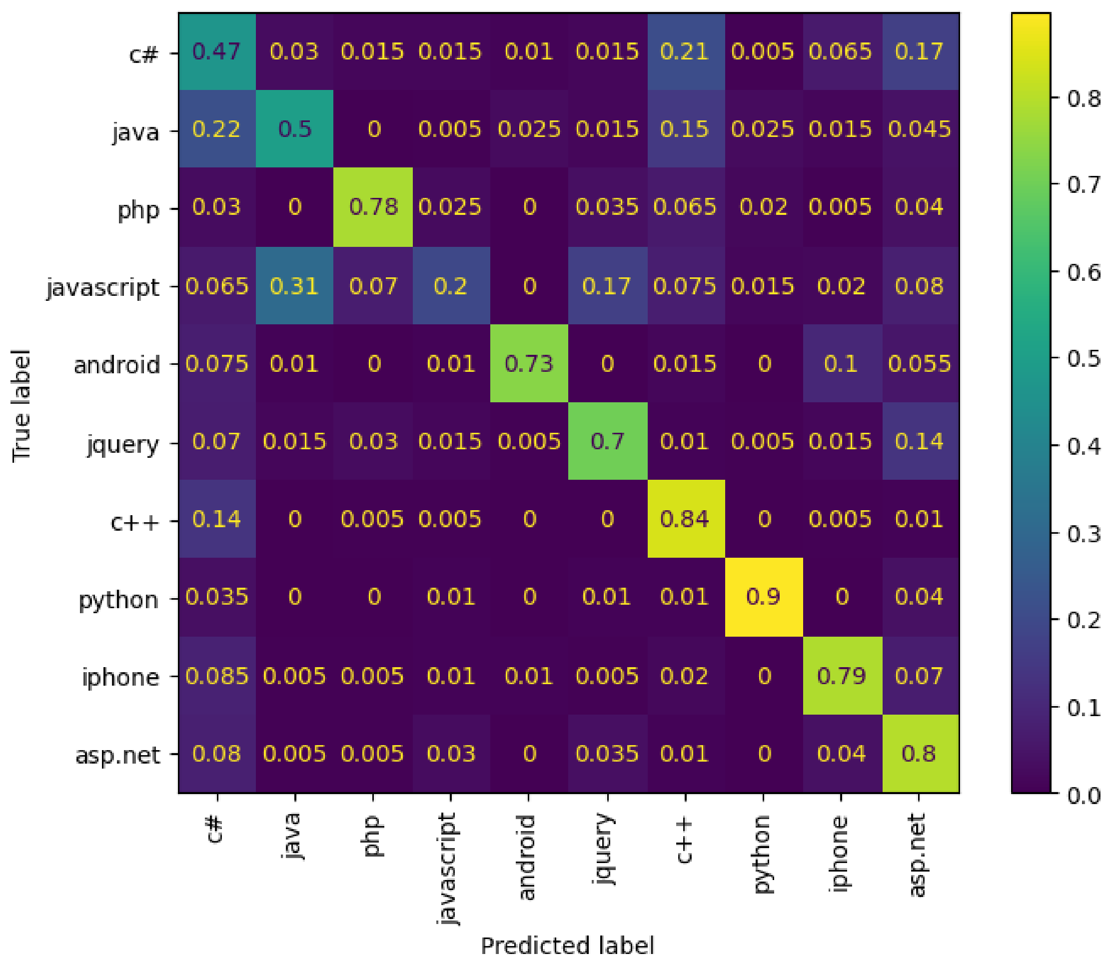
You're using a AlbertTokenizerFast tokenizer. Please note that with a fast tokenizer, using the `__call__` method is faster than using a method to encode the text followed by a call to the `pad` method to get a padded encoding.

<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]

<IPython.core.display.HTML object>



<IPython.core.display.HTML object>

The best model was saved at step 240.

VBox(children=(Label(value='0.007 MB of 0.007 MB uploaded (0.000 MB deduped)\r'), FloatProgress(value=1.0, max…

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

2.4 Experiment 3 - bert-base-uncased

2.4.1 Trainer hyperparameters

[17]:
```
checkpoint = 'bert-base-uncased' # CODE HERE
training_args_dict = training_args.to_dict() # Convert TrainingArguments to␣
↪dictionary
training_args_dict['run_name'] = f'{checkpoint}-{num_samples_per_class}' #␣
↪Update the run_name
new_training_args = TrainingArguments(**training_args_dict)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1711:
FutureWarning: `--push_to_hub_token` is deprecated and will be removed in
version 5 of  Transformers. Use `--hub_token` instead.
  warnings.warn(
```

[18]:
```
tokenize_train_evaluate_log(training_args= new_training_args,␣
↪checkpoint=checkpoint, base_folder=base_folder,
                             class_names=class_names,␣
↪train_val_subset=train_val_subset,
                             compute_metrics=compute_metrics)
```

```
env: WANDB_PROJECT=nlp_course_fall_2023-HW5-Part-B-Colab
```

```
Downloading (…)okenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]
```

```
Downloading (…)lve/main/config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]
```

```
Downloading (…)solve/main/vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
```

```
Downloading (…)/main/tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
```

```
Downloading model.safetensors:   0%|          | 0.00/440M [00:00<?, ?B/s]
```

```
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-uncased and are newly initialized:
['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

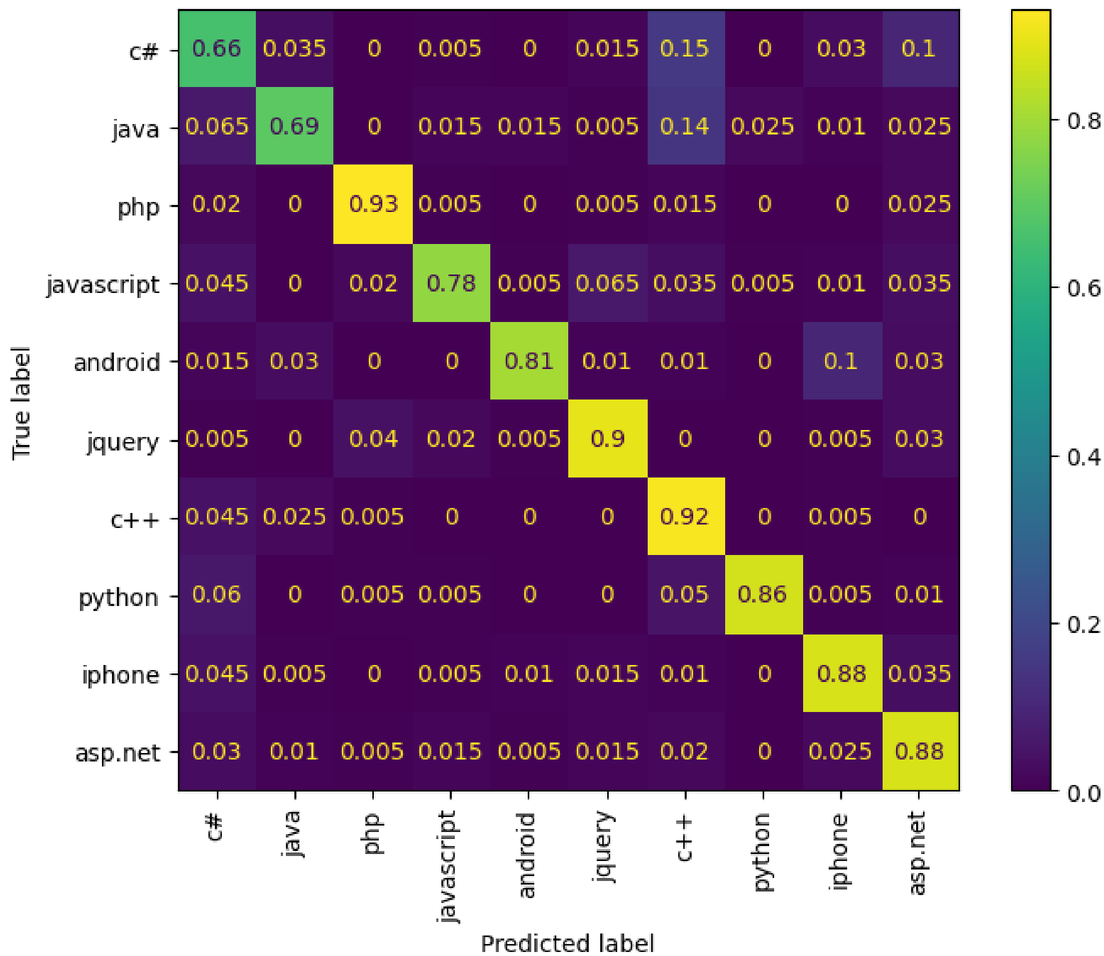
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

You're using a BertTokenizerFast 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.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>



The best model was saved at step 220.

<IPython.core.display.HTML object>

VBox(children=(Label(value='0.005 MB of 0.007 MB uploaded (0.000 MB deduped)\r'), FloatProgress(value=0.651617…

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

```
[ ]:
```

From the above 3 models, we can see according to the F1 score and accuracy on the evaluation dataset, the performance wise:

1. BERT. - 83%
2. DistillBert - 80.9%
3. ALBert. - 66.9%

In almost all cases, due to it's size (number of parameters and sheer complexity) BERT performs better than DistilBert which is derived from BERT using knowledge distillation. The knowledge distillation process can act as a form of regularization, preventing overfitting and promoting better generalization on the given text classification task. The simpler architecture of DistilBERT may encourage better feature reuse and learning of more generalizable representations for the task at hand. So in cases where compuational efficiency is more important than the accuracy, DistilBERT becomes a smart choice.

ALBERT uses a factorized embedding parameterization and cross-layer parameter sharing. The performance of any model, including ALBERT, heavily depends on the specific hyperparameters chosen during training and fine-tuning. Suboptimal hyperparameters or training procedures can affect the model's performance negatively.

The way the model is fine-tuned for a specific task can significantly impact its performance. The fine-tuning process and the task-specific architecture can be better optimized for BERT or DistilBERT than for ALBERT.

```
[22]:
```