2. Task 2.3.2 Using the task 2.2 as an pytorch example, please build your own Multiclass classifier on this new dataset with the following main changes

- Multiclass and New Dataset
- Use Word Embedding and Deep Average Network.

Your code should report the Accuracy, F1 score for each label, and macro F1 for a combined score. (You don't need to reimplement all your metrics in Task 2.2. Please directly use classification_report to report the performance on devset and test set. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html)

Training this may take around 20 minutes according your implementation. Hence we will not train your model from scratch to obtain a results. Please write a report to demonstrate your training and results.

Hence, your submission for Task 2.3:

1. A single pdf with both your exploration for the dataset, and your experiments for the semsentiment classification
2. The whole folder (remove the data folder, and the cache folders, such as **pycache**, .env .conda)

Hints:

1. To start, simply copy the code snippets from Task 2.1 and 2.2 into your own code hotel_sentiment_classifier.py. Make it runnable, and replicate the results you have done in the above tutorial.
2. PAY ATTENTION!!! You have to adapt those code to support your "multiclass classifer", including dataset reading, using softmax function, cross-entropy loss, and the model, and many details not listed here. Please explore that by yourself.

Task 2.3.1 Exploratory Data Analysis on SST-5

```python
# Load the dataset
from datasets import load_dataset

ds = load_dataset("SetFit/sst5")
```

```
Repo card metadata block was not found. Setting CardData to empty.
```

```python
train_dataset = ds['train']
dev_dataset = ds['validation']
test_dataset = ds['test']
train_dataset[0]
```

```
{'text': 'a stirring , funny and finally transporting re-imagining of
beauty and the beast and 1930s horror films',
 'label': 4,
 'label_text': 'very positive'}
```

```python
# The dataset is already split into train, dev, and test. So you don't
need to split it again.
len(train_dataset), len(dev_dataset), len(test_dataset)
```

```
(8544, 1101, 2210)
```

```python
import pandas as pd
df_train = ds["train"].to_pandas()
print(df_train.head(5))
```

```
                                                text  label
label_text
0  a stirring , funny and finally transporting re...      4  very
positive
1  apparently reassembled from the cutting-room f...      1
negative
2  they presume their audience wo n't sit still f...      1
negative
3  the entire movie is filled with deja vu moments .      2
neutral
4  this is a visually stunning rumination on love...      3
positive
```

```python
import pandas as pd
df_train = ds["train"].to_pandas()
print(df_train.tail(5))
```

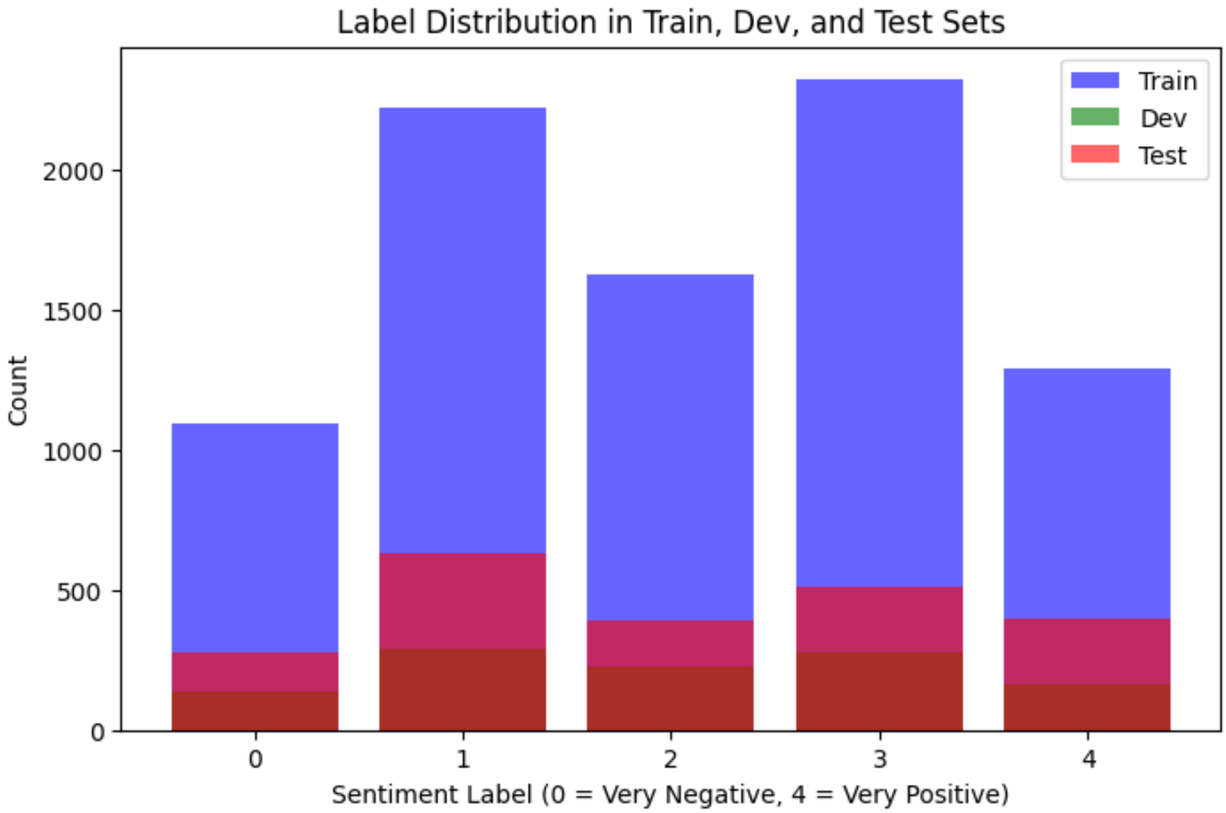
```
                                                   text  label
label_text
8539  take care is nicely performed by a quintet of ...      1
negative
8540  the script covers huge , heavy topics in a bla...      1
negative
8541  a seriously bad film with seriously warped log...      1
negative
8542   it 's not too racy and it 's not too offensive .      2
neutral
8543  a deliciously nonsensical comedy about a city ...      4  very
positive
```

```python
### ENTER CODE FOR EXPLORATORY HERE ###
# Check the structure of one example in the dataset
print(" Sample Data from Training Set:")
print(train_dataset[0])

# Check dataset sizes
print("\n Dataset Sizes:")
print(f"Train: {len(train_dataset)}, Dev: {len(dev_dataset)}, Test:
{len(test_dataset)}")

# Check keys in dataset
print("\n Available Keys in Dataset:")
print(train_dataset.features)
```

```
 Sample Data from Training Set:
{'text': 'a stirring , funny and finally transporting re-imagining of
beauty and the beast and 1930s horror films', 'label': 4,
'label_text': 'very positive'}

 Dataset Sizes:
Train: 8544, Dev: 1101, Test: 2210

 Available Keys in Dataset:
{'text': Value(dtype='string', id=None), 'label': Value(dtype='int64',
id=None), 'label_text': Value(dtype='string', id=None)}
```

```python
import matplotlib.pyplot as plt
from collections import Counter

# Count labels in each dataset
train_labels = [example['label'] for example in train_dataset]
dev_labels = [example['label'] for example in dev_dataset]
test_labels = [example['label'] for example in test_dataset]

train_label_counts = Counter(train_labels)
dev_label_counts = Counter(dev_labels)
test_label_counts = Counter(test_labels)

# Print label distributions
print("\n Label Counts:")
print("Train:", train_label_counts)
print("Dev:", dev_label_counts)
print("Test:", test_label_counts)

# Plot label distribution
plt.figure(figsize=(8, 5))
plt.bar(train_label_counts.keys(), train_label_counts.values(),
color='blue', alpha=0.6, label="Train")
plt.bar(dev_label_counts.keys(), dev_label_counts.values(),
color='green', alpha=0.6, label="Dev")
plt.bar(test_label_counts.keys(), test_label_counts.values(),
color='red', alpha=0.6, label="Test")
plt.xlabel("Sentiment Label (0 = Very Negative, 4 = Very Positive)")
plt.ylabel("Count")
plt.xticks([0, 1, 2, 3, 4])
plt.legend()
plt.title("Label Distribution in Train, Dev, and Test Sets")
plt.show()
```

```
 Label Counts:
Train: Counter({3: 2322, 1: 2218, 2: 1624, 4: 1288, 0: 1092})
Dev: Counter({1: 289, 3: 279, 2: 229, 4: 165, 0: 139})
Test: Counter({1: 633, 3: 510, 4: 399, 2: 389, 0: 279})
```



```
# Function to print example sentences from each sentiment class
def print_examples(dataset, num_examples=2):
    label_groups = {i: [] for i in range(5)}

    for example in dataset:
        label_groups[example['label']].append(example['text'])

    print("\n Example Sentences from Each Sentiment Class:\n")
    for label, texts in label_groups.items():
        print(f" Sentiment {label} ({len(texts)} samples):")
        for text in texts[:num_examples]:
            print(f"  ➡ {text}")
        print("-" * 50)

# Print examples from training dataset
print_examples(train_dataset)


 Example Sentences from Each Sentiment Class:

 Sentiment 0 (1092 samples):
  ➡ final verdict : you 've seen it all before .
  ➡ lacks the inspiration of the original and has a bloated plot that
stretches the running time about 10 minutes past a child 's interest
and an adult 's patience .
```

```
--------------------------------------------------
 Sentiment 1 (2218 samples):
  ➡ apparently reassembled from the cutting-room floor of any given
daytime soap .
  ➡ they presume their audience wo n't sit still for a sociology
lesson , however entertainingly presented , so they trot out the
conventional science-fiction elements of bug-eyed monsters and
futuristic women in skimpy clothes .
--------------------------------------------------
 Sentiment 2 (1624 samples):
  ➡ the entire movie is filled with deja vu moments .
  ➡ um , no. .
--------------------------------------------------
 Sentiment 3 (2322 samples):
  ➡ this is a visually stunning rumination on love , memory , history
and the war between art and commerce .
  ➡ jonathan parker 's bartleby should have been the be-all-end-all of
the modern-office anomie films .
--------------------------------------------------
 Sentiment 4 (1288 samples):
  ➡ a stirring , funny and finally transporting re-imagining of beauty
and the beast and 1930s horror films
  ➡ béart and berling are both superb , while huppert ... is
magnificent .
--------------------------------------------------
```

```python
# Compute text lengths
train_lengths = [len(example['text'].split()) for example in
train_dataset]

# Find min, max, average length
print("\n Text Length Statistics:")
print(f"Min Length: {min(train_lengths)} words")
print(f"Max Length: {max(train_lengths)} words")
print(f"Average Length: {sum(train_lengths) / len(train_lengths):.2f}
words")

# Find short and long texts
short_texts = [example['text'] for example in train_dataset if
len(example['text'].split()) < 5]
long_texts = [example['text'] for example in train_dataset if
len(example['text'].split()) > 50]

print("\n Example of Very Short Texts:")
for text in short_texts[:3]: print(f"- {text}")

print("\n Example of Very Long Texts:")
for text in long_texts[:3]: print(f"- {text[:300]}...")  # Show only
first 300 characters
```

```
 Text Length Statistics:
Min Length: 2 words
Max Length: 52 words
Average Length: 19.14 words

 Example of Very Short Texts:
- um , no. .
- too bad .
- a fun ride .

 Example of Very Long Texts:
- if you are curious to see the darker side of what 's going on with
young tv actors -lrb- dawson leery did what ?!? -rrb- , or see some
interesting storytelling devices , you might want to check it out ,
but there 's nothing very attractive about this movie ....
- it may not be as cutting , as witty or as true as back in the glory
days of weekend and two or three things i know about her , but who
else engaged in filmmaking today is so cognizant of the cultural and
moral issues involved in the process ?...
- it 's a bad sign when you 're rooting for the film to hurry up and
get to its subjects ' deaths just so the documentary will be over ,
but it 's indicative of how uncompelling the movie is unless it
happens to cover your particular area of interest ....
```

```python
from collections import Counter
nlp = spacy.load("en_core_web_sm")
# Tokenize all text and count word frequencies
word_counts = Counter()
for example in train_dataset:
    tokens = [token.lemma_.lower() for token in nlp(example['text'])
if token.is_alpha]
    word_counts.update(tokens)

# Most common words
most_common_words = word_counts.most_common(20)

# Least common words (rare words)
least_common_words = word_counts.most_common()[:-20:-1]

print("\n Most Common Words in Training Set:")
for word, count in most_common_words:
    print(f"{word}: {count}")

print("\n Least Common Words in Training Set:")
for word, count in least_common_words:
    print(f"{word}: {count}")
```

```
 Most Common Words in Training Set:
```

```
the: 7353
a: 5305
and: 4517
of: 4456
be: 4340
to: 3052
it: 2428
that: 1955
in: 1917
film: 1306
as: 1299
movie: 1176
but: 1172
with: 1139
have: 1093
for: 1037
this: 998
an: 974
its: 944
you: 860

 Least Common Words in Training Set:
racy: 1
wimmer: 1
warp: 1
surfacey: 1
quintet: 1
mcdormand: 1
intriguingly: 1
indicate: 1
overstuff: 1
analysis: 1
irreparable: 1
enjoys: 1
monument: 1
personable: 1
appealingly: 1
curler: 1
comeback: 1
marcus: 1
embellish: 1
```

Task 2.3.2 Build Your MultiClass Sentiment Classifier With Word Embedding and Deep Neural Networks.

You are given two sources of uncased pretrained embeddings you can use: data/glove.6B.50d-subset.txt and data/glove.6B.300d-subset.txt. These are trained using GloVe (Pennington et al., 2014). We only used a subset of this for a runtime and memory optimization. It also means that