Homework 3

Hamzeh Alzweri

Dr Kristen Johnson

1 Q1

For this question, I trained in native Pytorch, I used the RoBERTa variation to obtain an accuracy of 0.6488. This whole framework is new to me to be honest, so I learned alot from it and I also used it to solve the bonus part from Q2. The summary of the model was too lengthy to include here, therefore I included it in the summary.txt file in the zipped folder. Code for this question is also in the zipped folder as HW3_Q1.ipynb

2 Q2

I will be building a Named Entity Recognition model, I choose this task because its very commonly used for data extractions in general so I thought its a good chance to learn it. BERT-CRF, to the best of my knowledge, is the current best BERT model for NER task, the paper can be found here : https://paperswithcode.com/paper/focusing-on-possible-named-entities-in-active. I didn't find a table with recall, precision and accuracy but they report values were mostly F1 scores so I will be comparing to that but I will report all metrics for my models, all required packages area in the venv within the project files so as long as you are running using that environment it should be fine. I am mentioning this because I had edited a code for a library function which was throwing errors, namely the flat_classification_report in the CRF.py was throwing a positional argument error so I just replaced the "labels" argument with "labels = labels" for the classification_report function inside it and it worked. Just in case this error pops up Professor, please just replace that argument and it should work.

I will try 3 different models; SVM, NaiveBayes, and CRF, I will explain my implementation and results in the below:

-Dataset: I used an annotated GMB(Groningen Meaning Bank) dataset for NER from Kaggle: https://www.kaggle.com/datasets/abhinavwalia95/entity-annotated-corpus which includes 17 IOB (inside chunk, outside chunk, beginning of a chunk) unique tags. I chose this dataset because it already has the POS tag for each word which is a very useful feature to do NER and I will demonstrate that by only using the POS and the word to predict the tag

in SVM, and NaiveBayes but I will add extra features for the CRF model to increase performance.

1. SVM: The features I used for this model are the POS tag and the word, I converted them to vectors using the DictVectorizer which essentially one-hot encodes them such that we get 17 new binary (0,1) columns for the unique POS for each row (word). Same method applies to the words, this gives us the new feature vector for each word which is 18923 dimensional (includes NER tag label and sentence).

   I initially extracted 200 thousand rows from the entire dataset as I had memory limitation issues as it kept throwing OOM error, I noticed that the samples are heavily imbalanced, 169381 words have the tag "O" which means out of chunk or simply not a named entity. This will cause overfitting and overly optimistic evaluations, so I removed 120000 of those words and kept 49381 only which is still the dominant class. I printed out these numbers as tables when you run the SVM.py file. I used cross validation with 5 folds to obtain the following results:
   average test precision weighted 0.8449471289396644
   average test accuracy 0.8418875
   average test recall weighted 0.8418875
   average test f1 weighted 0.8399226298858684

   The metrics above are the "weighted" metrics provided by sklearn, if calculates metrics for each label, and find their average weighted by the number of true instances for each label. Which reduces the class imbalance issue for our evaluations, for this part of the question all you need to do is run the SVM.py file and it should print out the results mentioned above as well as data distributions for each class.

2. Naive Bayes: For this model, I used the same features and pre-processing explained above for the SVM model and I obtained the following results:
   average test precision weighted 0.8306722220157248
   average test accuracy 0.8199125
   average test recall weighted 0.8199125
   average test f1 weighted 0.8137217871251531

   The results slightly degraded from the SVM model as the context is not present within the features, I will try the CRF model and and I will also add context by adding features of previous and next word to each row with the help of the package. Code for the Naive Bayes model can be found in NaiveBayes.py, again you only need to run the file.

3. CRF: For the CRF model, I am using the sklearn-crfsuite https://sklearn-crfsuite.readthedocs.io/en/latest/api.html#module-sklearn_crfsuite which requires a list of lists of dicts which where the dictionaries contain the features, i.e "feature named": "feature value", of each word. Here I decided

to add more useful features using the crfsuite which capture context across consecutive words, here is a sample of the features for each word :
{'bias': 1.0,
'word.lower()': 'arabic',
'word[-3:]': 'bic',
'word[-2:]': 'ic',
'word.isupper()': False,
'word.istitle()': True,
'word.isdigit()': False,
'postag': 'NNP',
'postag[:2]': 'NN',
'-1:word.lower()': 'friday',
'-1:word.istitle()': True,
'-1:word.isupper()': False,
'-1:postag': 'NNP',
'-1:postag[:2]': 'NN',
'BOS': False,
'+1:word.lower()': 'al-jazeera',
'+1:word.istitle()': False,
'+1:word.isupper()': False,
'+1:postag': 'NNP',
'+1:postag[:2]': 'NN',
'EOS': False}

+1 and -1 mean the next and previous words, respectively, which means we are incorporating features of the surrounding words into our model. BOS means whether its a begging of a sentence while EOS represents whether the word is an end of sentence, other features have their full names.

Here I report the metrics for each class separately, because I suspect that "O" class is still giving us an unrealistic estimate. I also ran the metrics function again and reported the average without the "O" class. (Note that the code will sometimes show warning about a zero division which is caused by the fact that the model didn't guess any sample correctly for a particular class, these classes had very low number of samples compared to the other classes).

Results are shown in Figures 1 and 2.

The weighted average F1 score increased to %0.92, CRF model was designed to deal with sequences so this is of no surprise. However, we also we note that some tags like "I-art" Which is Inside chunk Artifact are being always misclassified, which is why I decided to show results without the O class in 2. The weighted average F1 score reduced to %0.81 which is most likely the true potential of the model because of the class imbalanced. But its still doing better than the previous models, therefore I will stick with this model.

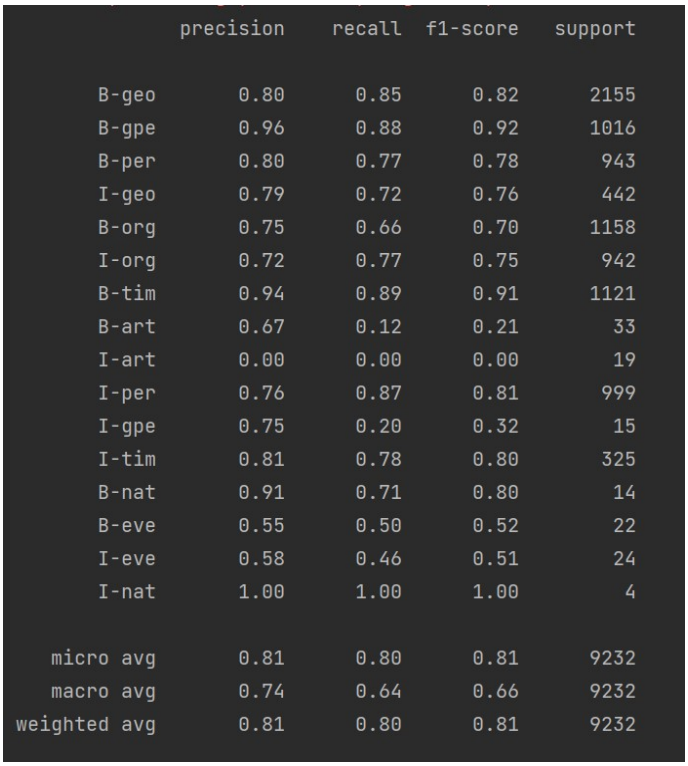
	precision	recall	f1-score	support
B-geo	0.80	0.85	0.82	2155
B-gpe	0.96	0.88	0.92	1016
B-per	0.80	0.77	0.78	943
I-geo	0.79	0.72	0.76	442
B-org	0.75	0.66	0.70	1158
I-org	0.72	0.77	0.75	942
B-tim	0.94	0.89	0.91	1121
B-art	0.67	0.12	0.21	33
I-art	0.00	0.00	0.00	19
I-per	0.76	0.87	0.81	999
I-gpe	0.75	0.20	0.32	15
I-tim	0.81	0.78	0.80	325
B-nat	0.91	0.71	0.80	14
B-eve	0.55	0.50	0.52	22
I-eve	0.58	0.46	0.51	24
I-nat	1.00	1.00	1.00	4
O	0.98	0.99	0.99	15037
accuracy			0.92	24269
macro avg	0.75	0.66	0.68	24269
weighted avg	0.92	0.92	0.92	24269

Figure 1: Results including the O class

2.1 Q2 Branch H

Adding BERT embeddings: For this part, I will add embedding extracted from the last transformer encoder from the BERT-cased model. However, I tried to do it with all my models but it was only compatible with the SVM model, the Naive Bayes model did not accept negative values and the sklearn-crf model didn't work with the list of features I extracted, the SVM model worked with no issues and it showed good results, the features of each word were the BERT embeddings concatenated with the vectorized POS of the word. I removed the words vector because the BERT embedding is already describing those words, results were as follows:
average test precision weighted 0.8518994737219379
average test accuracy 0.857975
average test recall weighted 0.857975
average test f1 weighted 0.8523354009495625

Here I am only using 100k words instead of 200k because of memory limitations, I removed 60k of the "O" tagged words to prevent overfitting, these results are better than the SVM and Naive Bayes mode, but it couldn't outper-



	precision	recall	f1-score	support
B-geo	0.80	0.85	0.82	2155
B-gpe	0.96	0.88	0.92	1016
B-per	0.80	0.77	0.78	943
I-geo	0.79	0.72	0.76	442
B-org	0.75	0.66	0.70	1158
I-org	0.72	0.77	0.75	942
B-tim	0.94	0.89	0.91	1121
B-art	0.67	0.12	0.21	33
I-art	0.00	0.00	0.00	19
I-per	0.76	0.87	0.81	999
I-gpe	0.75	0.20	0.32	15
I-tim	0.81	0.78	0.80	325
B-nat	0.91	0.71	0.80	14
B-eve	0.55	0.50	0.52	22
I-eve	0.58	0.46	0.51	24
I-nat	1.00	1.00	1.00	4
micro avg	0.81	0.80	0.81	9232
macro avg	0.74	0.64	0.66	9232
weighted avg	0.81	0.80	0.81	9232

Figure 2: Results without the O class

form the CRF because I am only using roughly half of the samples from class "O". Table below shows a comparison of all the previous results. Code can be found in bonus_bert.py, you only have to run the file but it will consume some time.

	F1
SVM	0.84
Naive Bayes	0.814
CRF	0.92
BERT-SVM	0.852

Table 1. Overall results