ReadME Report

Task 1:

We have been given the financial phrasebook data. This data is a dictionary having two features, namely sentence and label. The 'sentence' part contains 4,846 sentences containing critical information for investors. The 'label' part consists of labels towards the sentence, such as positive or negative sentiment toward a company. Here the labels are 0,1,2, where 0 refers to a negative statement, 1 to a neutral statement and 2 to a positive statement.

1.1

- Chosen sentiment analysis method- its main strengths and limitations:

The method chosen for sentiment analysis is an algorithm called 'Logistic regression. It is a discriminative classifier that directly learns to predict the class labels from the features without a need for conditional independence assumption. The logistic regression model directly estimates the posterior probability of 'y' given 'x'. The equation for a binary logistic regressor is given by: $P(y|\boldsymbol{x}) = \frac{1}{1 + e^{-\sum_{i=1}^{N} \theta_i \cdot x_i}}$

Here, the 'xi' corresponds to a particular feature in a fixed-size vector, where 'i' corresponds to a word in a vocabulary. Logistic regression assumes a weight 'Qi' for each 'xi' and multiplies these two values to weight a feature. Finally, a sigmoid function is applied to squash the value between 0 to 1 to get a probability. The main advantage of logistic regression is that it can work with both continuous and discrete values to predict our target label. The disadvantage of logistic regression is that we can't directly compute the value of 'Q'.

- features choose and how will they affect your results:

The first thing we do is tokenise our sentences. That means it splits the string into a sequence of tokens where tokens might include words, multi-word phrases, parts of words, special characters, punctuations and special tokens. Then to extract a bag of words, we call the CountVectoriser() function. This class outputs the bag of words as a feature vector, where the length of the vector is equal to the vocabulary size, and the values are the counts of each word in a document. The vectoriser function stores our words in the form of:

(x1, x2) y where x1= sentence number x2= Index of the word y= Number of occurrences of the word in the sentence.

- Pre-processing steps your method requires:

The following are the pre-processing steps that I have decided to take:

1) **Lemmatisation:** Lemmatization is all about mapping the words to their root form. This is useful to reduce the size of the vocabulary. E.g., the word changing, change, changed can be labelled as "change".
2) **N-grams:** Another way to improve it could be to use bigrams instead of single words as our features. Bigrams are pairs of words that occur one after another in the text. Bigrams are a kind of 'n-gram', where 'n=2'. To extract bigrams, we again modify our CountVectorizer. This class has a parameter `ngram_range`, which determines the range of sizes of n-grams the vectoriser will include. If we set `ngram_range= (1,1) `, we have our standard bag of words. If we set it to `ngram_range= (2,2) `, we use bigrams instead. Choosing If we set `ngram_range= (1,2) ` will use both single tokens (unigrams) and bigrams. It is mostly used in language identification.
3) **Lexicon Features:** A lexicon is a hand-crafted list of words associated with an effective state with a specific sentiment or connotation. We use prior knowledge to supplement a lack of training data by using lexicon features.

1.2. Implement, train, and test your method:

At first, split the sentences and labels into train and test sets using a train test split from the sklearn library. We will be hiding our test set to check how our model performs with unseen data. We further do another train test split on our train set. This data is divided into four variables: train_sentences, val_sentences, train_labels, and val_labels. The train sentences and labels consist of 75% of our data from our previous 'train sentences, and Val sentences and labels contain the remaining 25% of our data. We will then do all the necessary feature extraction steps, fit our model into a logistic regression model, and predict the results compared to our 'Val' set. In addition to this, we will also use all the pre-processing steps mentioned above and see if these steps impact our model. We will then compute the results in a classification report. After all the feature extraction and pre-processing steps, we will apply our best model to the unseen data and see how our model performs with unseen data.

1.3

- Performance metrics and their limitations:

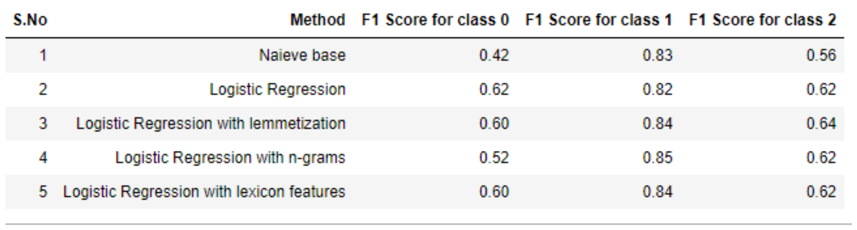
I have decided to go with the performance metric for all my models is f1-score. It is the harmonic mean between precision and recall. Here we check the f1- score for each label.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

The only disadvantage the F1 score has is it requires both the metrics to be reasonably high to result in a high score. In other words, if one of the metrics is close to 0, then the F1 score is also 0.

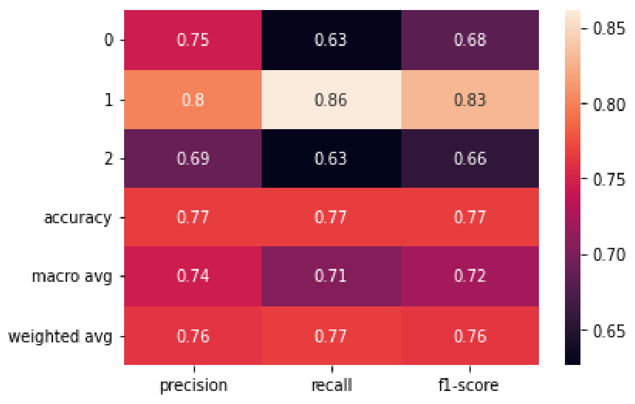
- Results:

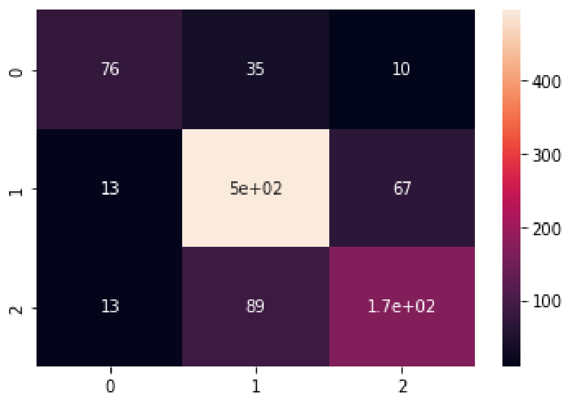
The following were the results of f1 scores of different classes after applying other models with feature extraction, pre-processing etc.:



S.No	Method	F1 Score for class 0	F1 Score for class 1	F1 Score for class 2
1	Naieve base	0.42	0.83	0.56
2	Logistic Regression	0.62	0.82	0.62
3	Logistic Regression with lemmetization	0.60	0.84	0.64
4	Logistic Regression with n-grams	0.52	0.85	0.62
5	Logistic Regression with lexicon features	0.60	0.84	0.62

As we can see, we got the bust results with Logistic Regression with lemmatisation. We will now apply this model to our X_test(validation set) to see how our model perfoms on unseen data. The results were as follows:





As seen from the similar scores, our model performed the same in our testing dataset (Val data). This means our model is good and not overfitting.

- How could you improve the method or experimental process? Errors that your method makes:

We have more neutral sentences in our dataset, and only 12% of the sentences are negative sentences. Our method's error is that it performs well on identifying positive or neutral sentences but performs poorly comparatively on identifying negative sentences. We can improve our experiment process by increasing the number of negative sentences and having our data distributed equally for future work.

Task 2:

2.1:

- Chosen named entity recognition method- its main strengths and limitations:

The name entity method that I have chosen to go with was Conditional Random Fields (CRF). CRF is a discriminative model which is used for the sequence labelling model. Given the sequence labels, the CRF learns to optimise a predictive distribution, i.e., the probability of a sequence of labels given the tokens. The main advantage of CRF is that this model takes each sequence as modelled to depend on the whole sequence of tokens. When predicting a sequence label, the model can consider all the tokens. The disadvantage of CRF is that it is very accurate. It is slower to learn and is less suitable for online learning scenarios where you must update the model regularly.

- Features are chosen and how they will affect your results:

For this task, two features have been selected along with the standard CRF, they are:

1) Custom CRF tagger: in this tagging scheme, we will be extracting basic features about a word, including factors like the Current word. Is it capitalised? Does it have punctuation? Does it have a number? Suffixes up to length 3.
   This sounds very similar to stemming, and by doing this, we are reducing our vocabulary by removing unwanted features and training our data in a better way to predict a better sequence of labels.
2) CRF with POS tagging: For languages where the same word can have different parts of speech or meaning, POS tagging helps us identify those parts of speech and help us tag the words accordingly. Adding POS tagging in our CRF categorises words in a text (corpus) in correspondence with a particular part of speech, depending on the definition of the word and its context. This helps our tagger to identify labels more quickly.

- Tagging scheme for labelling entities in this dataset:

Our dataset had five tags, and the tagging schemes were as follows:

1) I-LOC: If our tokens contained locations of places, those tokens were tagged as “I-LOC”.
2) I-MISC: If our tokens contained any Miscellaneous entities, e.g., events, nationalities, products, or works of art. Then those tokens were tagged as “I-MISC”.
3) I-ORG: If our tokens contained names of organisations, those tokens were tagged as “I-ORG.”
4) I-PER: If our tokens contained names of people, those tokens were tagged as “I-PER”.
5) O: If our token contained characters other than the abovementioned, these tokens were tagged as “O”. These tokens include characters like any parts of speech etc.

2.2 Implement, train, and test your method:

We will first be splitting our sentences into a train and test set. This can be done using a train test split. We further split our train set into train and Val set. Our model will learn and predict based on this data. The test set will be used as our validation set to see how our model performs on unseen data. We will then train our data based on the training set, predict the tags, and compare them with our validation set. We first need to generate our training data so that each token already has a tagging scheme. The same thing should be done to our validation set as well. Once the desired outputs are achieved, we will begin our model training process on our training set. For implementing CRF, we first need to define a function in which a tagger is limited, and this tagger then trains our training set using the CRF model. Once the model is implemented, we will then predict the values using the tagger function on our validation set. The predicted tags are then scored based on the evaluation metric. The exact process is repeated for our two other CRFs based on features, i.e., Custom CRF tagger and CRF with POS tagging. The scores are evaluated for these models, and then we compare all the scores in the end and decide which model fits the best.

2.3 Evaluating and interpreting results

- Choice of performance metrics and their limitations:

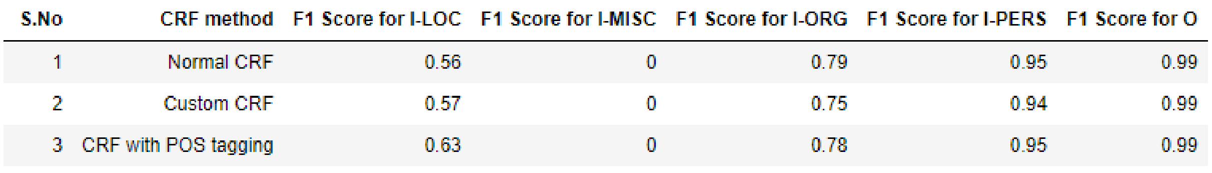
I first took out a classification report to compare our predicted tags' performance and our validation set. I have decided to go with the F1 scores as our performance metrics. We could have gone for other metrics such as accuracy or precision. Still, these metrics consider the false positives and false negatives to a certain extent, whereas F1 scores try to minimise them as much as possible. The F1 score combines the precision and recalls into a single metric by taking their harmonic mean. A disadvantage of the F-score is that **it does not reveal mutual information among features**. It gives equal importance to precision and recall, and if one of those values is equal to zero, our total f1 score is equal to 0.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

For our models, an F1 score is evaluated for all the five tags present in our dataset, and it gives us insight into how good our model performs as a tagger while detecting an entity. The same is done for our other 2 CRFs as well.

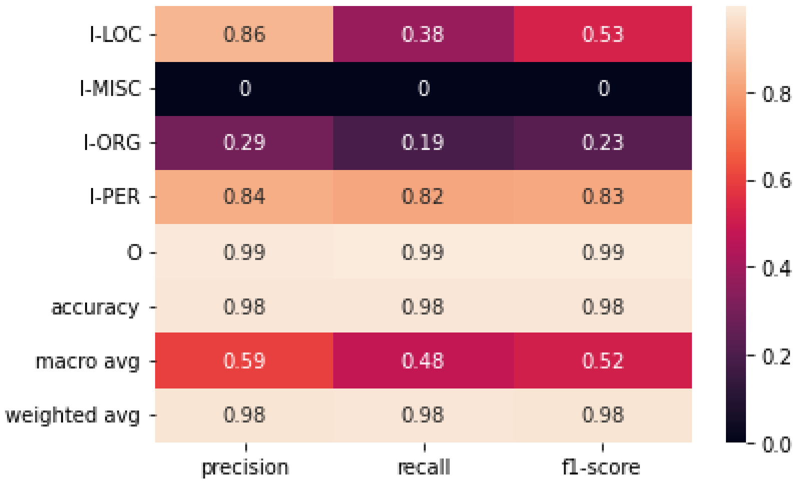
- Results:

The following F1 scores were noticed for our 3 CRF models for all our tags:



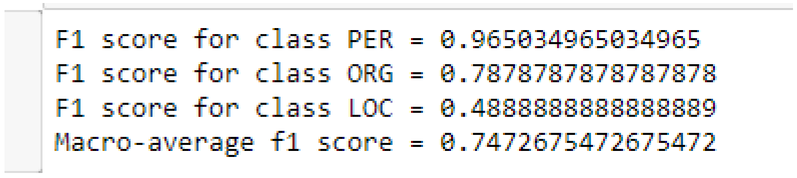
S.No	CRF method	F1 Score for I-LOC	F1 Score for I-MISC	F1 Score for I-ORG	F1 Score for I-PERS	F1 Score for O
1	Normal CRF	0.56	0	0.79	0.95	0.99
2	Custom CRF	0.57	0	0.75	0.94	0.99
3	CRF with POS tagging	0.63	0	0.78	0.95	0.99

<u>Note:</u> We don't get any score for the "I-MISC" labels in the above scores because we only had five tokens with the tags "I-MISC" out of 7899 tags. Since the training data is insufficient, our model couldn't predict the scores for this tag.
As we can see, we got the best scores for CRF with POS tagging. Hence we will be applying this model for our test set, which is an unseen data, to determine how well our model performs with unseen data. The results were as follows:



- How could we improve the method or experimental process? Errors our method makes:

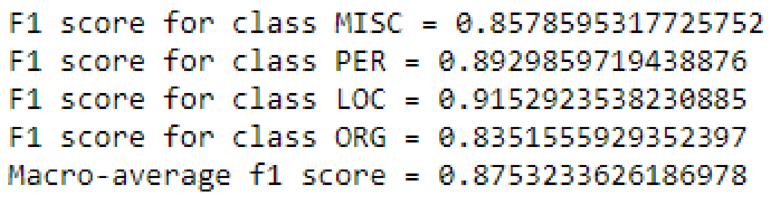
Our limitation in this metric is that each word is recognised as an entity, making it hard to compute a score for organisations and locations. Also, our dataset contains a lot of “O” tags since they’re very generic, and because of this, a bias is created between the “O” tags and the rest of our tags. For this reason, the overall maco avg for F1 was less. This observation is made by observing the F1 scores of individual Tags. To overcome this problem, we can use SPAN functions. A text span is a subsequence within a larger piece of text. We can improve our score by extracting one or more words in succession. E.g., Instead of recognising United Airlines as two entities, we recognise it as one entity using SPAN. After applying the SPAN function for CRF with POS tagging, the scores were found to be:



```
F1 score for class PER = 0.965034965034965
F1 score for class ORG = 0.7878787878787878
F1 score for class LOC = 0.4888888888888889
Macro-average f1 score = 0.7472675472675472
```

F1 scores on CRF with POS tagging using SPAN

I have also performed NER and SPAN on the conll2003 dataset, a related dataset in a similar format, and got some promising results. The following were the F1 scores on using CRF with POS tagging along with spans.



```
F1 score for class MISC = 0.8578595317725752
F1 score for class PER = 0.8929859719438876
F1 score for class LOC = 0.9152923538230885
F1 score for class ORG = 0.8351555929352397
Macro-average f1 score = 0.8753233626186978
```

F1 scores on CRF with POS tagging using SPAN for the conll2003 dataset

2.4. Applying trained NER tagger to the Financial Phrase bank dataset.

The following is done by first converting the sentences in our datasets into separate tokens. Once we acquire these tokens, we run them through our CRF tagger. By doing this, each token is given a label along with it.

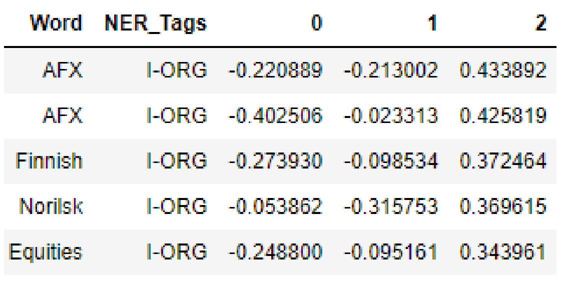
- Compute a sentiment score for each entity that you detect.

The following is done by putting all our words along with their tags in the form of a data frame. Then we get the sentiment score by applying the classifier coefficient function to each word. What the classifier.coef_ method does, is it provides a score for each label present in our data. Where the score is highest, the token will belong to that label. In our case, the score will determine whether the word is positive, negative, or neutral based on the value of coefficients we get with respect to the label.

- For example, show your results by listing the five most positive and five most negative organisations and their scores.

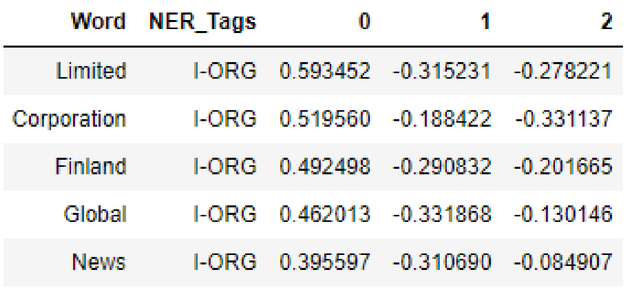
This is done by taking a subset of our data frame, which only consists of the words with the tags “I-ORG” in them. We then arrange our data frame based on descending values of our label “0”. The same should be done with the label “2” as well. What this does is the head data frame will give us the topmost positive and negative organisations.

An important observation is that our model had an F1 score of around 0.6 for “I-ORG”, Which tells us that our model will be accurate about 60% of the time, which explains why our results are not that accurate.



	Word	NER_Tags	0	1	2
	AFX	I-ORG	-0.220889	-0.213002	0.433892
	AFX	I-ORG	-0.402506	-0.023313	0.425819
	Finnish	I-ORG	-0.273930	-0.098534	0.372464
	Norilsk	I-ORG	-0.053862	-0.315753	0.369615
	Equities	I-ORG	-0.248800	-0.095161	0.343961

Top 5 positive organisations



	Word	NER_Tags	0	1	2
	Limited	I-ORG	0.593452	-0.315231	-0.278221
	Corporation	I-ORG	0.519560	-0.188422	-0.331137
	Finland	I-ORG	0.492498	-0.290832	-0.201665
	Global	I-ORG	0.462013	-0.331868	-0.130146
	News	I-ORG	0.395597	-0.310690	-0.084907

Top 5 negative organisations