ML Project

Lin Huiqing, Tey Shi Ying, Tjoa Jing Sen

December 2020

1 Part 2

	E Precision	E Recall	E F	S Precision	S Recall	S F
EN	0.5116	0.724	0.5996	0.4535	0.6416	0.5313
CN	0.0812	0.4929	0.1395	0.0393	0.2386	0.0675
SG	0.195	0.5548	0.2885	0.1251	0.356	0.1851

Table 1: Part 2 - Precision, Recall, F Score for Entity and Sentiment

2 Part 3

	E Precision	E Recall	E F	S Precision	S Recall	S F
EN	0.8032	0.7693	0.7859	0.7649	0.7327	0.7484
CN	0.2503	0.2914	0.2693	0.1485	0.1729	0.1597
SG	0.5195	0.5099	0.5147	0.4276	0.4197	0.4236

Table 2: Part 3 - Precision, Recall, F Score for Entity and Sentiment

3 Part 4

We modify the viterbi algorithm for the purpose of calculating part 4.

1. Initialisation

$$\pi(0,u) = \begin{cases} 0 & \text{if } u = START \\ -\infty & \text{otherwise} \end{cases}$$

2. For $j = 0...n-1$, for each $u \in T$ where route r_u is the total path up to and including u, calculate top 3 probabilities and keep track of their paths.

$$\pi(j+1, r_u) = topthree_v\{\pi(j,v) + log(b_u(x_{j+1})) + log(a_{v,u})\}$$

3. Retrieve the third best path.

$$\pi(n+1, r_{STOP}) = topthree_v\{\pi(n,v) + log(a_{v,STOP})\}$$

The results which we obtained for this path is shown in Table 3.

E precision	E recall	E F	S precision	S recall	S F
0.7693	0.7379	0.7533	0.7273	0.6976	0.7122

Table 3: Scores for third best path

4 Part 5

To improve on the sentiment analysis system, a few modifications to part 3's code were tried:

1. Varying k values
2. Smoothing Techniques
   (a) Laplace Smoothing
   (b) Good Turing Estimate Smoothing
3. Alternative Models
   (a) Structured Perceptron

4.1 Varying k values

To find the optimal k value, k value was experimented in the form of $k = 10^x$ where $-1 < x < 9$ is a random value. Scores are calculated as seen in Table 4. "E" stands for "Entity" while "S" stands for "Sentiment". Scores are then plotted on a scatter graph as seen in Figure 1 for better visualisation.

Based on the scores calculated, the optimal k value, $k' = 10^{x'}$ is likely occur when $3.16 < x < 4.90$. However, while optimal k can be pinpointed, this is likely to over-fit on existing development set.

log10(k)	E precision	E recall	E F	S precision	S recall	S F
-0.7533047598	0.8031	0.7693	0.7858	0.7648	0.7326	0.7483
-0.7482957644	0.8031	0.7693	0.7858	0.7648	0.7326	0.7483
0.3239369788	0.8065	0.7708	0.7883	0.7682	0.7342	0.7508
0.6333971634	0.8068	0.771	0.7885	0.7685	0.7344	0.751
1.132140394	0.8089	0.7721	0.7901	0.7707	0.7356	0.7527
1.835393503	0.8117	0.7746	0.7927	0.7748	0.7394	0.7567
2.593819808	0.8138	0.7765	0.7947	0.7794	0.7436	0.7611
2.713162059	0.8145	0.7776	0.7956	0.7798	0.7444	0.7617
3.163601528	0.8181	0.7792	0.7982	0.7842	0.7469	0.7651
3.202743162	0.8182	0.7788	0.798	0.7841	0.7464	0.7648
4.336691749	0.8279	0.7771	0.8017	0.7949	0.7461	0.7697
4.892136649	0.8249	0.7688	0.7959	0.791	0.7372	0.7632
5.023381063	0.8234	0.7668	0.7941	0.7894	0.735	0.7612
5.495538	0.8217	0.7645	0.7921	0.7866	0.7318	0.7582
6.374596786	0.8227	0.7632	0.7918	0.7868	0.7299	0.7573
6.92982277	0.8224	0.7627	0.7914	0.7866	0.7296	0.757
7.157155681	0.8224	0.7627	0.7914	0.7866	0.7296	0.757
7.201623881	0.8224	0.7627	0.7914	0.7866	0.7296	0.757
8.729569481	0.822	0.7623	0.791	0.7864	0.7292	0.7567
8.762235074	0.822	0.7623	0.791	0.7864	0.7292	0.7567

Table 4: Effect of varying k value on scores

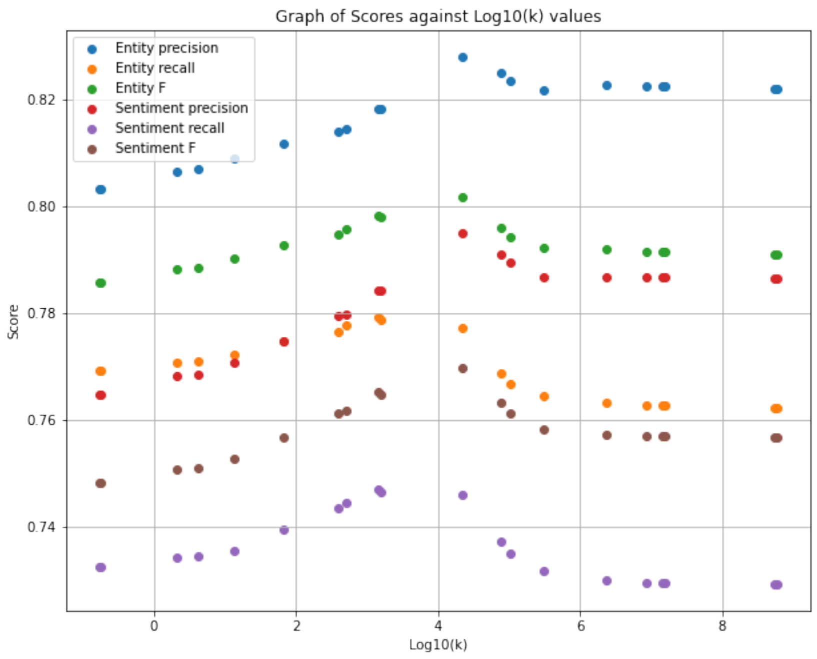

Figure 1: Effect of varying k value on scores

4.2 Smoothing Techniques

Due to the limited dataset used in this exercise, data is sparse. One way to tackle this is to use smoothing techniques to calculate the probabilities. Add-one smoothing was first tried as a simple smoothing method, then Good Turing Estimate Smoothing is tried. As expected based on our research, both smoothing techniques improve the performance of the model, and Good-Turing estimate performed much better than Add-one smoothing.

Model	E precision	E recall	E F	S precision	S recall	S F
HMM	0.8032	0.7693	0.7859	0.7649	0.7327	0.7484
with Add-one	0.8292	0.8098	0.8193	0.7914	0.7729	0.782
with Good-Turing	0.8381	0.8195	0.8287	0.8008	0.783	0.7918

Table 5: Effect of Different Smoothing Techniques on Scores

4.2.1 Add-one Smoothing

Add-one smoothing, also known as Laplace Smoothing, is calculated with the following equation[3]:

$$\begin{aligned} p(x, y) &= \frac{1 + c(y \to x)}{\sum_x [1 + c(y)]} \\ &= \frac{1 + c(y \to x)}{|V| + \sum_x [c(y)]} \end{aligned}$$

where x is the observation, y is the state, and V is the length of vocabulary (including "#UNK#").

This is implemented in calculating the emission probabilities for the HMM version coded in Part 3. Results of the model with Add-one smoothing can be seen in Table 5.

4.2.2 Good Turing Estimate Smoothing

Another smoothing technique we tried was using Good Turing estimation. Good Turing Estimate adjusts actual counts c to expected counts c^* through the following formula[2]:

$$c^* = (c + 1)\frac{N_{c+1}}{N_c}$$

Probabilities are then calculated through expected count c^*, and applied to the HMM in Part 3. Results of the model with Good Turing Estimate smoothing can be seen in Table 5.

4.3 Alternative Models

For alternative models, we chose to try the Structured Perceptron model. Structured Perceptron was chosen as a discriminative model to try out to compare against the performace of our currect HMM model, which is a generative model.

4.3.1 Structured Perceptron

The Structured Perceptron model is a discriminative model which can be used for sequence labelling. It is based on feature vectors representations which provide a conditional probability $p(y|x)$ for decision y given history x. In our case, y refers to the state s while x refers to the observation. Below are the steps to run the structured perceptron algorithm[1].

1. Initialise all weights $w = 0$.
2. For each example number $i = 1...n$,

(a) Use the Viterbi algorithm to calculate as follows:

$$\begin{aligned} s^* &= argmax_{s \in S} w \cdot \phi(x^i, s) \\ &= argmax_{s \in S} \sum_{j=1}^{m} w \cdot \phi(x, j, s_{j-1}, s_j) \end{aligned}$$

(b) Update weights, penalising wrong examples as follows:

$$\begin{aligned} w &= w + \Phi(x^i, s) - \Phi(x^i, s) \\ &= w + \sum_{j=1}^{m} \phi(x, j, s^i_{j-1}, s^i_j) - \sum_{j=1}^{m} \phi(x, j, s^*_{j-1}, s^*_j) \end{aligned}$$

3. Return weights w.

$\Phi(x, s)$ refers to the feature vectors for state s and observation x.

After which, the weights are used to propagate the Viterbi algorithm again for predictions.

For feature selection for the feature vectors, we used a collection of features which take into account the prefixes, suffixes, tags, previous tags and other special considerations like dashes. These features are selected based on our knowledge of language. The features which we chose were the following:

```
features = [
    f"PREFIX2+2TAGS_{prefix2}_{previous_tag}_{tag}",
    f"PREFIX3+2TAGS_{prefix3}_{previous_tag}_{tag}",
    f"DASH_{'-' in word}_{tag}",
    f"WORD_LOWER+TAG_{word_lower}_{tag}",
    f"UPPER_{word[0].isupper()}_{tag}",
    f"PREFIX2+TAG_{prefix2}_{tag}",
    f"SUFFIX3+2TAGS_{suffix3}_{previous_tag}_{tag}",
    f"WORD_LOWER+TAG_BIGRAM_{word_lower}_{tag}_{previous_tag}",
    f"SUFFIX3+TAG_{suffix3}_{tag}",
    f"SUFFIX3_{suffix3}",
    f"SUFFIX2+TAG_{suffix2}_{tag}",
    f"SUFFIX2+2TAGS_{suffix2}_{previous_tag}_{tag}",
    f"WORD_LOWER+TAG_{word_lower}_{tag}",
    f"PREFIX3+TAG_{prefix3}_{tag}",
    f"PREFIX2_{prefix2}",
    f"TAG_{tag}",
    f"TAG_BIGRAM_{previous_tag}_{tag}",
    f"SUFFIX2_{suffix2}",
    f"WORD_SHAPE_{self._shape(word)}_TAG_{tag}",
    f"PREFIX3_{prefix3}",
    f"ISPUNC_{word in string.punctuation}"
]
```

We trained the model for 10 epochs, recording the scores for the models at each epoch, as shown in Table 6. Scores are then plotted on a scatter graph as seen in Figure 2 for better visualisation. Based on the results, epoch 8 was the best as it has the best scores.

epoch no.	E precision	E recall	E F	S precision	S recall	S F
1	0.8033	0.8158	0.8095	0.778	0.7901	0.784
2	0.8119	0.8116	0.8117	0.7844	0.7841	0.7843
3	0.8596	0.822	0.8404	0.8367	0.8001	0.818
4	0.8389	0.8605	0.8496	0.815	0.836	0.8254
5	0.848	0.8593	0.8536	0.8235	0.8345	0.829
6	0.8571	0.8655	0.8613	0.8349	0.8432	0.839
7	0.8429	0.8651	0.8538	0.8206	0.8422	0.8312
8	0.8647	0.8605	0.8626	0.8399	0.8358	0.8378
9	0.86	0.8598	0.8599	0.8358	0.8356	0.8357
10	0.854	0.7923	0.822	0.835	0.7746	0.8037

Table 6: Effect of no. of epochs on scores

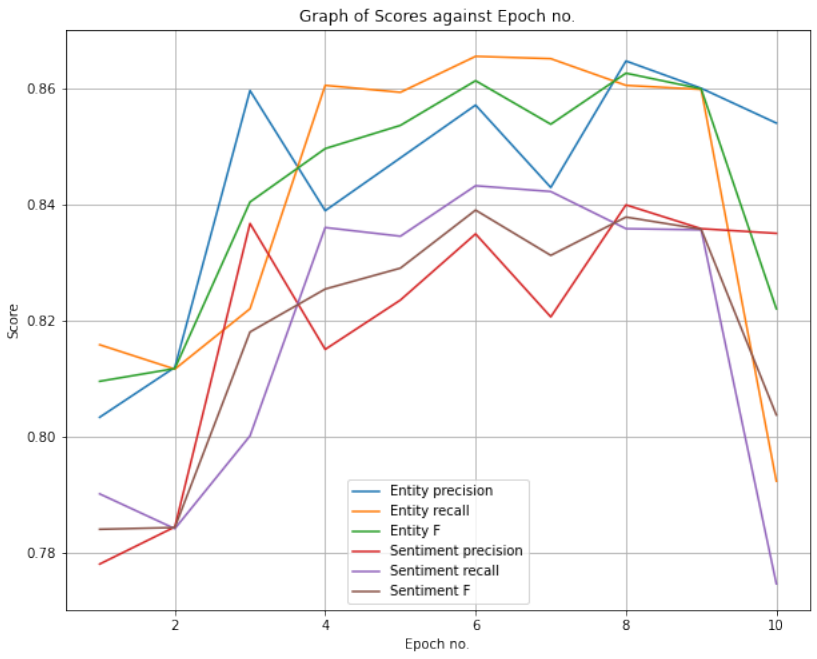

Figure 2: Effect of no. of epochs on scores

4.4 Conclusion

Based on the models which were tried out, the Structured Perceptron was chosen to run the predictions for the test set as it yielded the best scores on the development set. The test predictions were done by first training the Structured Perceptron model for 8 epochs, then parsing in the `test.in` file for prediction.

There are several improvements which can be made to the models but were not implemented.

For instance, the Structured Perceptron can possibly be improved with averaging and a dynamic learning rate based on the performance of the model at each epoch. Further attempts at feature engineering could be made as well, possibly pruning some of the overlapping features, or adding some other features such as the word length.

The smoothing techniques could be combined as well on HMM's calculation of probabilities to possibly yield better results.

References

[1] Michael Collins. The structured perceptron. University Lecture, 2011.

[2] Sharon Goldwater. Introduction to computational linguistics: Smoothing. University Lecture, 2015.

[3] Raymond Mooney. Cs 388: Natural language processing. University Lecture, 2017.