Train

Model 1

In the assignment, we were required to implement the (Ratnaparkhi, 96) features as seen in the lectures. After completing the task, we implemented features of our own, as follows:

1. Character features [f_characters]:

   - A set of features, one for each tag seen in the train set, such that its corresponding word contains only lowercase letters [all_lower].
   - A set of features, one for each tag seen in the train set, such that its corresponding word contains only uppercase letters [all_upper].
   - A set of features, one for each tag seen in the train set, such that its corresponding word contains both lowercase and uppercase letters [lower_and_upper].
   - A set of features, one for each tag seen in the train set, such that its corresponding word contains both a letter (lowercase/uppercase) and a punctuation mark (!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~) [letters_and_punctuation].
   - A set of features, one for each tag seen in the train set, such that its corresponding word starts with an uppercase letter and it's the only uppercase letter [starts_upper].
   - A set of features, one for each tag seen in the train set, such that its corresponding word ends in a period [ends_period].

   We wanted to implement features that capture character-based differences between different words, and that are also broad (represent a wide variety of words).

2. Numeric features [f_num]:

   - A set of features, one for each tag seen in the train set, such that its corresponding word contains only numbers (and maybe some punctuation marks, but with no letters at all) [all_num].
   - A set of features, one for each tag seen in the train set, such that its corresponding word contains both numbers and letters (lowercase/uppercase) [num_and_letter].

   It is reasonable to assume that digits within words could have a large impact on the tag of said word. In particular, in case the word contains only numbers or both numbers and letters.

3. Word length features [f_word_len]:

   - A set of features, one for each tag seen in the train set, and the length of its corresponding word.

   It appears that a word's length has an impact on its probable tags. Thus, we wanted to implement features that capture this information.

4. Position features [f_position]:

- A set of features, one for each tag seen in the train set, and its position in the sentence (e.g. third word of the sentence) [absolue_position].
- A set of features, one for each tag seen in the train set, and its relative position in the sentence [relative_position]. We divided each sentence into three parts:
    - Start – first third of the sentence.
    - Middle – second third of the sentence.
    - End – last third of the sentence.

From our knowledge, the position of the word in the sentence has a great impact on its tag. Therefore, we implemented features that capture both the tag's absolute position and its relative position in the sentence.

5. Frequency features:

- A set of features, one for each tag seen in the train set, and the frequency of its corresponding word in the sentence.

It appears that if a word appears in a sentence several times, it might hint towards a specific subset of tags.

6. Question features:

- A set of features, one for each tag seen in the train set, such that the sentence is a question (ends with a question mark).

As known, the grammatical structure of a question is quite different from that of a non-question sentence. Therefore, there might be an effect on the tags.

7. Next word features:

- A set of features, one for each tag seen in the train set, and its corresponding next word.

We wanted to generalize the idea of features set f107.

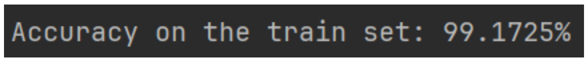
Moreover, we chose to keep all the features which appeared in the train set at least once (threshold=1) after testing several different thresholds. In addition, we set the regularization factor to be 0.3 (lambda=0.3). For more information on the hyper-parameter tuning process, please refer to the "Hyper-Parameter Tuning Appendix" at the end of the report.

For the feature count of this model, see "Feature Count Appendix" at the end of the report.

The training process took **around 20 minutes**, while being run on hardware with the following properties:

- **Operating System:** Microsoft Windows 10 Pro
- **Processor:** Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz, 3192 Mhz, 6 Core(s), 12 Logical Processor(s)
- **Installed Physical Memory (RAM):** 16.0 GB

After testing this model on the train set "train1.wtag", we received the following train accuracy:



```
Accuracy on the train set: 99.1725%
```

Note that we ran the inference procedure with B=1 when performing Beam Search, since the train set is very large (contains 5000 sentences).

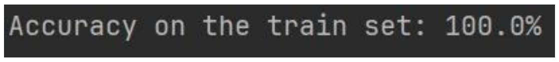
Model 2

In this model, after several experiments, we chose to use the same features that we used in Model 1, with threshold=1 as well. Furthermore, we set the regularization factor to be 0.13 (lambda=0.13). For more information on the hyper-parameter tuning process, please refer to the "Hyper-Parameter Tuning Appendix" at the end of the report.

For the feature count of this model, see "Feature Count Appendix" at the end of the report.

The training process took **a few seconds**, while being run on the same hardware described previously for Model 1.

After testing this model on the train set "train2.wtag", we received the following train accuracy:



```
Accuracy on the train set: 100.0%
```

Note that we ran the inference procedure with B=10 when performing Beam Search.

In addition, we noticed that certain words in the train set have the same POS tag each time they appear. In particular, a significant part of those words were punctuation symbols (period, comma, etc.). Thus, when training **both models**, we defined words that appeared in the train set with a **single** POS tag as "deterministic words" if they satisfied at least one of the following conditions:

- The word's corresponding tag is a punctuation symbol
- The word appeared in the train set over 500 times (we chose 500 as the threshold after testing different ones).

We will use these words when performing inference to speed up the process.

Inference

We were asked to implement the Viterbi algorithm as seen in the lecture. In order to control the algorithm's running time, we incorporated Beam Search as part of it.

Moreover, when predicting "deterministic words", we predict the tag they were assigned in the train set instead of running the usual inference procedure. In addition, when predicting "non-deterministic words", we only iterate the "possible tags"* in the Viterbi algorithm.

These changes saved unnecessary computations during inference, and therefore reduced the runtime of the procedure.

*"possible tags" - all the tags except symbol tags that their corresponding words appeared in the train set with a **single POS tag and it's a symbol tag**.

Test

Model 1

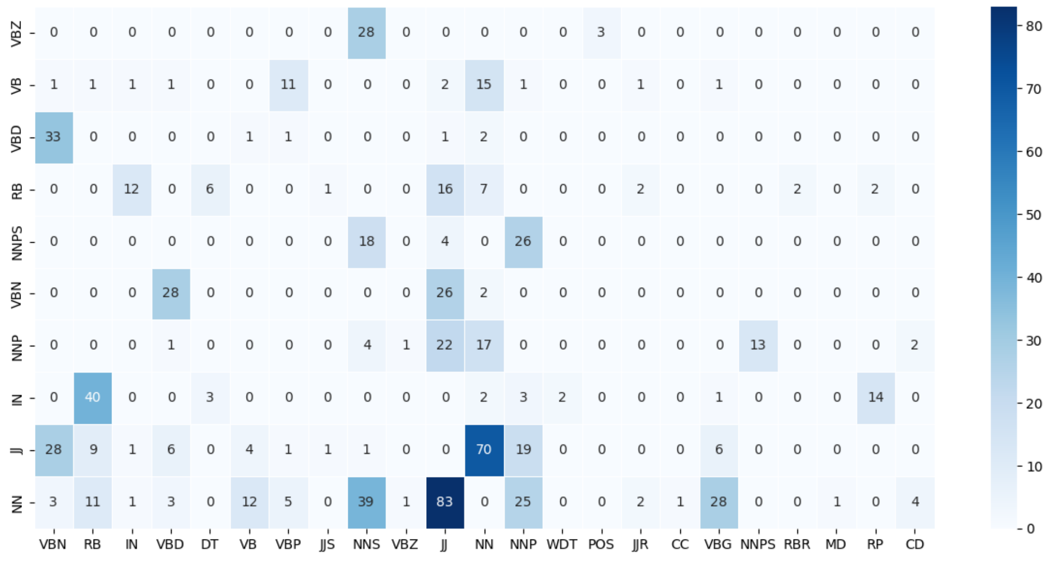
After training Model 1 on the train file (train1.wtag) we received the following (word-based) accuracy on the test file (test1.wtag):

```
Accuracy on the test set: 96.1139%
```

Note that we ran the inference procedure with B=10 when performing Beam Search.

As we can see, the accuracy received is indeed above 90% as requested in the assignment.

In addition, we received the following confusion matrix for the top 10 "worst tags"* (shown in rows of the confusion matrix):



	VBN	RB	IN	VBD	DT	VB	VBP	JJS	NNS	VBZ	JJ	NN	NNP	WDT	POS	JJR	CC	VBG	NNPS	RBR	MD	RP	CD
VBZ	0	0	0	0	0	0	0	0	28	0	0	0	0	0	3	0	0	0	0	0	0	0	0
VB	1	1	1	1	0	0	11	0	0	0	2	15	1	0	0	1	0	1	0	0	0	0	0
VBD	33	0	0	0	0	1	1	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0
RB	0	0	12	0	6	0	0	1	0	0	16	7	0	0	0	2	0	0	0	2	0	2	0
NNPS	0	0	0	0	0	0	0	0	18	0	4	0	26	0	0	0	0	0	0	0	0	0	0
VBN	0	0	0	28	0	0	0	0	0	0	26	2	0	0	0	0	0	0	0	0	0	0	0
NNP	0	0	0	1	0	0	0	0	4	1	22	17	0	0	0	0	0	0	13	0	0	0	2
IN	0	40	0	0	3	0	0	0	0	0	0	2	3	2	0	0	0	1	0	0	0	14	0
JJ	28	9	1	6	0	4	1	1	1	0	0	70	19	0	0	0	0	6	0	0	0	0	0
NN	3	11	1	3	0	12	5	0	39	1	83	0	25	0	0	2	1	28	0	0	1	0	4

*"Worst tags" – after discussion on the course forums, we chose to define that a tag is "worse" than another tag **if its absolute number of wrong predictions is higher**.

It appears that the model commonly confuses NN with JJ and vice versa. Upon researching this issue, we found that in many cases in English language, it's hard to distinguish between the two. Therefore, our model's confusion is a known issue.

Model 2

In Model 2, no designated test set was provided. In order to deal with that problem, we chose to test the performances of Model 2 by using Cross Validation. In that process, we divided our data into 5 folds (80%/20% as our train/test split). The average accuracy across all folds is **94.55%** (with B=10 during Beam Search).

Competition

The creation and hyper-parameter choosing of the models that were used to perform inference on the competition files were described in the previous sections. Let us summarize the important points for each model:

Model 1

- Incorporates all discussed features.
- Threshold = 1
- Regularization factor (Lambda) = 0.3
- We chose to train our model on the train set (train1.wtag) **as well as on the test set (test1.wtag)** *

* Note: Before doing so, we wanted to make sure that the training on the test set as well, did not harm our model's accuracy. In order to achieve that, we performed Cross Validation on the test set **while also** training on the train set as follows:

1. We split the test set into two halves (2 folds).
2. We trained our model on one of the halves (train fold) **and** on the train set.
3. We tested the trained model performance on the second half (test fold)

The average CV accuracy received is 96.48% which is higher than the previous accuracy.

Model 2

- Incorporates all discussed features.
- Threshold = 1
- Regularization factor (Lambda) = 0.13
- We chose to train our model only on train set (train2.wtag).

In addition, **for both models**, the parameter **B=100** was used when performing the Beam Search procedure. We purposefully chose the B value to be as large as the computational limitations of our hardware allowed us, for the best results possible.

Moreover, we did not make any changes in the optimization file provided in the assignment.

Predicting Our Competition Accuracy

For **Model 1**, we predict the accuracy on the competition set to be around **95%**.

The performance of Model 1 (competition) was evaluated using 2-fold cross validation on a test set that contains 1,000 sentences (each test fold contains 500 sentences). Since the size of the test set is quite large, it's reasonable to assume that it represents a board range of examples from the domain. Therefore, we predict our model's accuracy on the competition set to be similar to the received CV average accuracy (96.48%).

For **Model 2**, we predict the accuracy on the competition set to be around **70%**.

The performance of Model 2 was evaluated using 5-fold cross validation on a dataset that contains 250 sentences (each test fold contains 50 sentences). In contrary to the previous prediction, the size of this dataset is relatively small. Thus, we cannot assume anymore that it reliably represents the domain. Therefore, we predict a much lower accuracy than the received CV average accuracy (94.55%)

Coworking

In order to work together on the assignment, we held numerous zoom sessions in which we brainstormed and wrote the code together.

Feature Count Appendix

Below are the features count of all the features incorporated in each of our models.

Model 1 (non-competition)

f100 – 15,415

f101 – 13,265

f102 – 22,393

f103 – 8,150

f104 – 1,060

f105 – 44

f106 – 17

f107 – 31

f_characters – 156

- all_lower – 34
- all_upper – 22
- lower_and_upper – 39
- letters_and_punctuation – 21
- starts_upper – 32
- ends_period – 8

f_num – 9

- all_num – 4
- num_and_letter – 5

f_word_len – 305

f_position – 160

- absolute_position – 76
- relative_position - 84

f_frequency – 164

f_question – 33

f_ctag_nword – 30,793

Model 1 (competition)

f100 – 17,148

f101 – 14,175

f102 – 24,135

f103 – 8,772

f104 – 1,097

f105 – 44

f106 – 18

f107 – 31

f_characters – 156

- all_lower – 34
- all_upper – 22
- lower_and_upper – 39
- letters_and_punctuation – 21
- starts_upper – 32
- ends_period – 8

f_num – 9

- all_num – 4
- num_and_letter – 5

f_word_len – 310

f_position – 161

- absolute_position – 76
- relative_position - 85

f_frequency – 172

f_question – 33

f_ctag_nword – 35,054

Model 2 (non-competition + competition)

f100 – 1,805

f101 – 2,556

f102 – 3,978

f103 – 1,168

f104 – 316

f105 – 32

f106 – 11

f107 – 14

f_characters – 80

- all_lower – 28
- all_upper – 6
- lower_and_upper – 19
- letters_and_punctuation – 9
- starts_upper – 16
- ends_period – 2

f_num – 5

- all_num – 1
- num_and_letter – 4

f_word_len – 199

f_position – 80

- absolute_position – 47
- relative_position - 33

f_frequency – 68

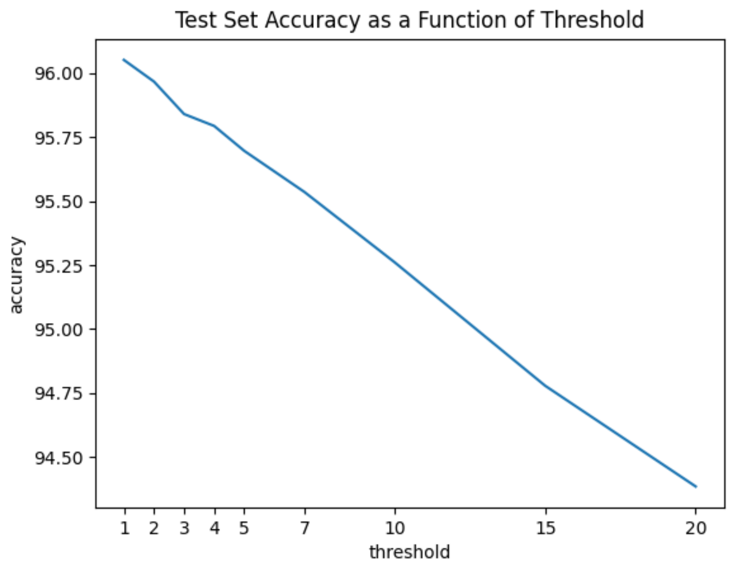
f_question – 0

f_ctag_nword – 2,694

Hyper-Parameter Tuning Appendix

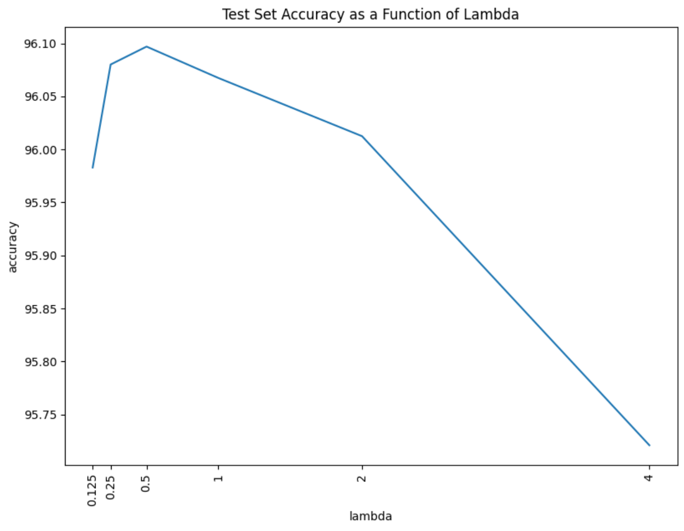
Model 1

For this model, we performed hyper-parameter tuning by choosing the threshold and lambda values that yield the best accuracy on the test set (test1.wtag). First, we set lambda to be 1 and tested different threshold values. We received the following accuracy plot:

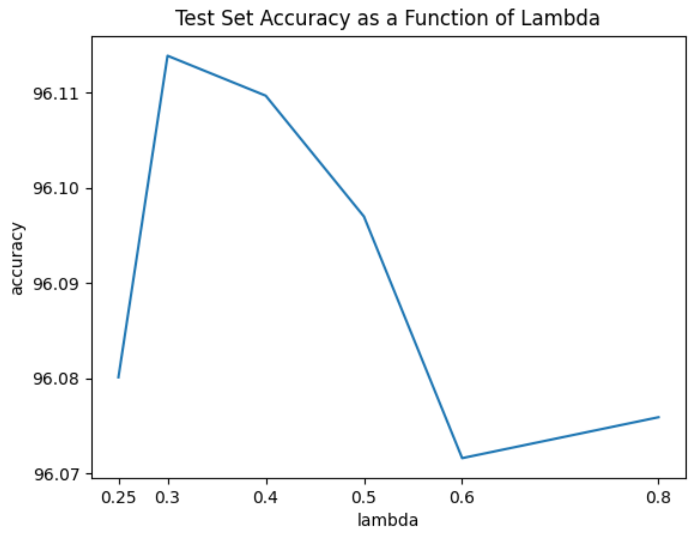


The highest accuracy was received for **threshold=1**, so we will choose that value as our threshold.

Then, we proceeded with this threshold value and tested different lambda values. We received the following accuracy plot:



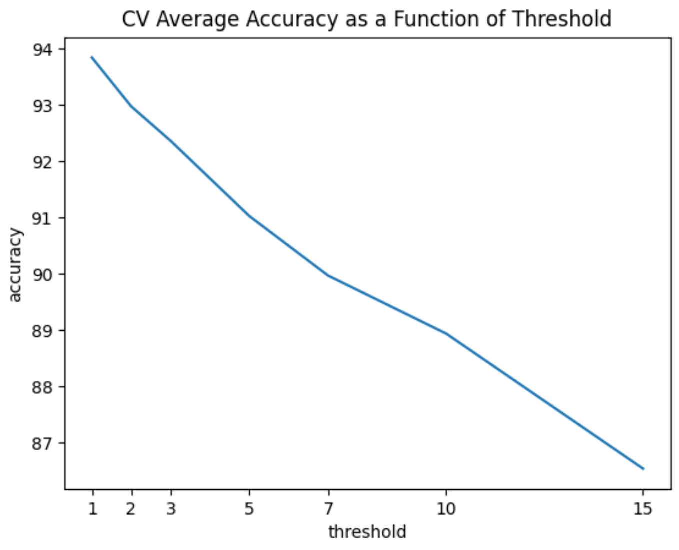
It appears that the lambda value which yields the best accuracy is between 0.25 and 1. We tested different lambda values within that range and received the following plot:



Finally, it appears that the lambda value which yields the best accuracy is **0.3**, so we will choose that value as our regularization factor, lambda, for this model.

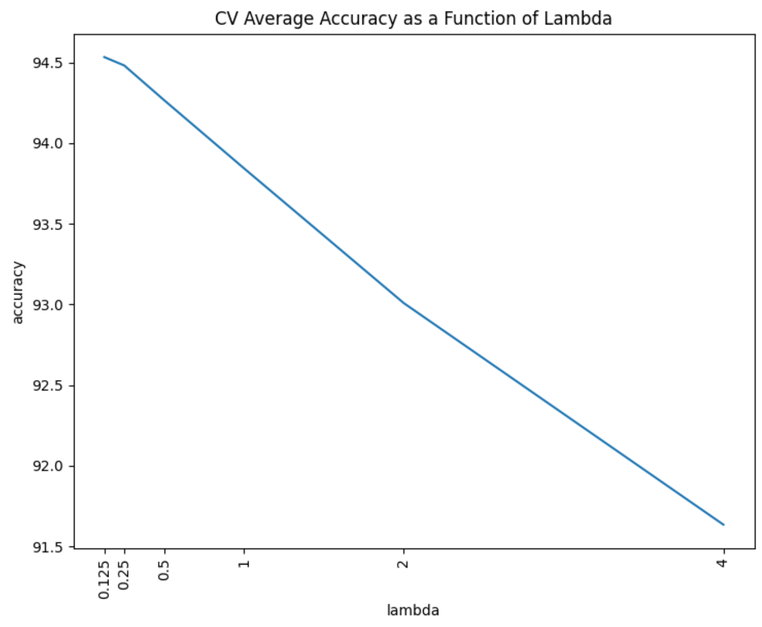
Model 2

For this model, we performed hyper-parameter tuning by choosing the threshold and lambda values that yield the best **average** accuracy, when performing 5-fold CV on the train set (train2.wtag). First, we set lambda to be 1 and tested different threshold values. We received the following accuracy plot:

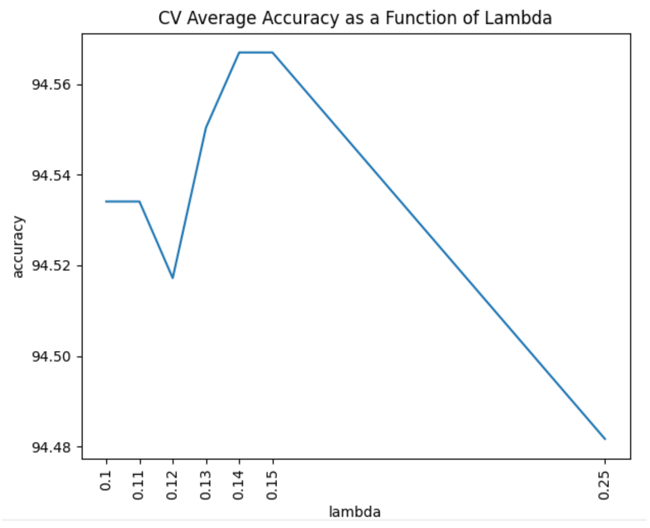


The highest accuracy was received for **threshold=1**, so we will choose that value as our threshold.

Then, we proceeded with this threshold value and tested different lambda values. We received the following accuracy plot:



It appears that the lambda value which yields the best accuracy might be between 0 and 0.25. We tested different lambda values within that range and received the following plot:



Finally, it appears that the lambda value which yields the best accuracy is **0.13**, so we will choose that value as our regularization factor, lambda, for this model.