# Exercise 1

## Part a

*How is the result compared to using the full brown tagset in the introduction?  
Why do you think one of the tagsets yields higher scores than the other one?*

Result from the run, using the Universal Tagset instead of the Brown Tagset: **0.8689**

Result from the previous run with the given Brown Tagset: 0.7915

The large difference in the scores achieved by using two different tagsets might be a result of the different tags applied and their number, too. The Brown tagset uses 87 different tags to classify the parts of speech it encounters and therefore it is a very elaborate tagset. On the other hand, the Universal Tagset only applies 12 tags, which makes it a more reduced tool to use. However, this reduction to 12 “core” tags for classifying parts of speech might lead to higher accuracies in classification tasks, since the tagger is then less prone to wrong tagging of words, or – to put it differently – the probability that the tagger tags a word correctly is higher when there are only 12 tags then when there are 87 tags to choose from.

## Part b

Result of the baseline tagger: 0.7582

As we can see from the results of Part a and Part b, the tagger used in Part a outperforms the Baseline tagger already by more than 10% difference in accuracy. Even though the feature extractor used in Part a only looks at the most basic indicators for POS tags, mostly leaving out the context of the word or the word itself, the tagger in Part a works more precisely than the baseline tagger. The baseline tagger only assigns words a POS-tag by using the most frequently seen POS-tag for the respective word in the training data. Therefore, the baseline tagger does not take the context of the word into account at all and following that, it might be considered very rudimental.

# Exercise 2

## Part a

*Train the ScikitConsecutivePosTagger on the \*news\_train\* set and test on the \*news\_dev\_test\* set with the \*pos\_features\*. Do you get the same result as with the same data and features and the NLTK code in exercise 1a?*

Results from running the code on the ScikitConsecutivePosTagger: 0.857

The result when running the same data

features with the ScikitConsecutivePosTagger differ slightly from the ones achieved with the NLTK code. To be more precise, the tagger implemented in exercise 2a achieves a slightly lower accuracy than the one implemented in exercise 1a.

## Part b

With the best choice of alpha, do you get the same results as with the NLTK code in exercise 1a, worse results or better results?

|  |  |
| --- | --- |
| **Value of alpha-parameter with BernoulliNB** | **Result** |
| 1 | 0.857 |
| 0.5 | 0.8749 |
| 0.1 | 0.8695 |
| 0.01 | 0.8683 |
| 0.001 | 0.8651 |
| 0.0001 | 0.8631 |

The best choice of the alpha-parameter is alpha=0.5, which yields an accuracy of 0.8749. This is slightly better than what the NLTK-code gives as a result. Moreover, the parameter alpha=0.1 yields a better result as well, while all the other parameters fall behind the NLTK-result from exercise 1a.

## Part c

Did the extended feature selector beat the baseline? – Yes, the feature selector which was extended in this Part c of Exercise 2 beat the baseline trained on *news\_train* and tested on *news\_dev\_test.* The baseline accuracy (0.7582) was outperformed by at least 12% by every of the taggers using various parameters.

The extended feature selector should intuitively beat the baseline because for the baseline calculation a very rudimental method was used, by assigning the words’ most common POS-tag to the respective word. Therefore, the feature selector doesn’t take a word’s context into account. However, the context (or other features, for that matter) can be a very helpful indicator when it comes to POS-tagging as it might help to decide which POS is the correct one.

|  |  |  |  |
| --- | --- | --- | --- |
| **Value of alpha-parameter with BernoulliNB** | **Feature extractor from Part b: Results** | **Results from extended feature extractor** | **Improvement of accuracy from Part b to Part c?** |
| 1 | 0.857 | 0.8874 |  |
| 0.5 | 0.8749 | 0.9166 |  |
| 0.1 | 0.8695 | 0.9244 |  |
| 0.01 | 0.8683 | 0.9303 |  |
| 0.001 | 0.8651 | 0.933 |  |
| 0.0001 | 0.8631 | 0.934 |  |

Running this experiment with the extended feature extractor and various alpha-values, the best results are achieved when using alpha=0.0001. In general, the extended feature extractor performs better than the not-extended one which only looks at the previous word instead of the previous and current word.

# Exercise 3

## Part a

Result of running with the extended feature extractor from the previous Exercise 2 in combination with Logistic Regression as classification algorithm: 0.9514

This result is even better than the best result with the Naïve Bayes classifier (best one here: 0.934).

## Part b

|  |  |
| --- | --- |
| **Value of C-parameter for smoothing in Logistic Regression (regularization)** | **Result** |
| 0.01 | 0.8499 |
| 0.1 | 0.9265 |
| 1.0 | 0.9514 |
| 10.0 | 0.9545 |
| 100.0 | 0.9531 |
| 1000.0 | 0.954 |

The classification with the logistic regression and various parameters for regularization yields the best results when using C=10.0. As smaller C values stand for stronger regularization, the best results are achieved with medium regularization in this case here. With the smallest C-value, the worst result is obtained with only 85% of accuracy in the POS-tags.

# Exercise 4

## Part a

Result with added feature in feature extractor – the next word in the sentence – and the best C from Exercise 3: 0.9651, which is a further improvement in accuracy

## Part b

|  |  |
| --- | --- |
| **Features observed by feature extractor** | **Accuracy** |
| Including all previous features plus checking whether word is a number | 0.9658 |
| Including all previous features plus checking whether word is capitalized | 0.9646 |
| Combining the previous two settings | 0.9646 |
| Checking whether all letters in word are capitalized, or only the first or none | 0.9644 |
| Previous setting + checking if string is a number | 0.9644 |

The best results can be obtained by checking whether the string is a number and also considering the following, the previous and the word itself.

Speed of running the experiment: 25sek

# Exercise 5

## Part b

Baseline Tagger Accuracy: 0.8446

## Part c

Accuracy of the tagger with the whole data set and the best settings from the previous exercises: 0.969  
Speed of running the experiment: 3:16 min

# Exercise 6

## Part a

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **.** | **ADJ** | **ADP** | **ADV** | **CONJ** | **DET** | **NOUN** | **NUM** | **PRON** | **PRT** | **VERB** | **X** |
| **.** | 12724 | . | . | . | . | . | . | . | . | . | . | . |
| **ADJ** | . | 6539 | 2 | 161 | . | . | 374 | 2 | . | 4 | 162 | . |
| **ADP** | 1 | 11 | 12547 | 70 | 5 | 16 | 8 | . | 15 | 189 | 17 | . |
| **ADV** | . | 211 | 113 | 4406 | 9 | 8 | 47 | . | . | 20 | 17 | . |
| **CONJ** | . | . | . | 7 | 3225 | 1 | . | . | . | . | 1 | . |
| **DET** | . | 1 | 34 | 14 | 3 | 12083 | 13 | . | 26 | . | . | . |
| **NOUN** | . | 294 | 5 | 39 | . | 3 | 23724 | 41 | 2 | 7 | 244 | 7 |
| **NUM** | . | 5 | 1 | . | . | . | 36 | 1242 | . | . | 2 | . |
| **PRON** | . | . | 49 | 3 | . | 33 | 9 | . | 3983 | . | 1 | 1 |
| **PRT** | . | 3 | 102 | 10 | . | 2 | 35 | . | 2 | 2387 | 3 | . |
| **VERB** | . | 83 | 11 | 16 | . | 2 | 426 | . | . | 4 | 15317 | . |
| **X** | 5 | 5 | 4 | 1 | . | 3 | 74 | . | 3 | . | 4 | 28 |

(row = reference; col = test)

## Part b

|  |  |  |  |
| --- | --- | --- | --- |
| **Tag** | **Precision** | **Recall** | **F-measure** |
| **.** | 0.9995 | 1.00 | 0.9998 |
| **ADJ** | 0.9143 | 0.9027 | 0.9084 |
| **ADP** | 0.9751 | 0. 9742 | 0. 9746 |
| **ADV** | 0.9321 | 0.9120 | 0.9220 |
| **CONJ** | 0.9948 | 0.9972 | 0.9960 |
| **DET** | 0.9944 | 0.9925 | 0.9935 |
| **NOUN** | 0.9587 | 0.9737 | 0.9661 |
| **NUM** | 0.9665 | 0.9658 | 0.9662 |
| **PRON** | 0.9881 | 0.9765 | 0.9822 |
| **PRT** | 0.9142 | 0.9383 | 0.9261 |
| **VERB** | 0.9714 | 0.9658 | 0.9686 |
| **X** | 0.7778 | 0.2205 | 0.3436 |

## Part c

|  |  |
| --- | --- |
| Macro-Precision | 0.9489083333333331 |
| Macro-Recall | 0.9015999999999998 |
| Macro-F-Measure | 0.9122583333333334 |

# Exercise 7

# Exercise 8

## Part a

Result of running the tagger on the *test* data set: 0.9698

The result of running the tagger on the *dev\_test* data set was 0.969 (see Ex5, Part c), therefore there is hardly any difference between the accuracies achieved on testing on both data sets. Testing on the *test* data set achieves a slightly better result than the other experiment, however, this difference should not play a major role in assessing the quality of the tagger.

## Part b

Result of testing the tagger on the *adventure* set: 0.963

Result of testing the tagger on the *hobbies* set: 0.9546

As we can see, the tagger does not perform as highly on these two data sets as on the *test* data set in Part a before. A reason for that could be that before, we trained the tagger on the same genres of text we would later test it on, therefore the contents of the test set do not hugely differ from the ones on which the tagger was trained in the exercises before. The two genres *adventure* and *hobbies* were left out when creating the training test sets, therefore these two genres are completely new to the tagger and therefore it might not perform as good on them.

It is also perceptible from the results of running the tagger on the *adventure* and on the *hobbies* set, that the tagger performs better on the further than on the latter. A reason for that could be that the genre *adventures* consist of novels and short stories (see <http://icame.uib.no/brown/bcm.html>), which are both included in other genres as well – e.g. *humor, romance, science fiction*. Furthermore, the Brown Corpus includes 126 examples of *imaginative* *prose* to which *adventure* can be counted as well. This leads to a well-trained tagger when it comes to this whole genre. Therefore, the tagger might have been more familiar with this type of writing and therefore was able to better tag the words with a respective POS.

However, the genre *hobbies* consists of material from books and periodicals and belongs to the sub-category of informative prose. This sub-category consists only of 17 examples, which is really few compared to the sub-category of *imaginative prose*. Of course, book texts might be longer, however, the training material for this sub-category is still less than for the other sub-category.

# Exercise 9

## Part a

Result of training the HMM tagger on *news­\_train­* and running it on *news\_test*: 0.8995

Result of training on *train* and running HMM tagger on *test*: 0.9518  
Speed of this experiment: 2:50 mins

Compared to our first tagger, the HMM tagger performs a bit worse with an accuracy of only 95% compared to the 97% achieved before. In terms of speed, the HMM tagger works slightly faster than the tagger which was used for the previous experiments. It runs about half a minute less in time, however, it also performs slightly worse than the previous tagger so this in not really an advantage here.

## Part b

Result of training the perceptron tagger on *news­\_train­* and running it on *news\_test*: 0.9638

Result of training on *train* and running the perceptron tagger on *test*: 0.9793  
Speed of the experiment: 3:54 mins

Running the perceptron tagger achieved higher results than running the best tagger from the previous exercises. It outperforms the previous tagger by 1% with accuracy values of 0.9793 compared to 0.969 (Ex 5, Part C). With an accuracy of almost 98%, this tagger can be evaluated of being really good. As one would expect, the tagger performed better when trained on the larger *train* and *test* sets than when trained and tested on the news sets. This is due to the higher number of examples the tagger can look at in the training sets and therefore it is able to perform better.

When it comes to speed, the perceptron tagger runs about half a minute longer on the whole *train/test data set* than the tagger used in Exercise 5. Since the perceptron tagger achieves better, the 30 seconds are a reasonable “downside”.