IN4080 – 2023
Mandatory assignment 2

Thursday 28 September 2023

Submission deadline: Monday 16 October 2023 23:59 CEST

General requirements

We assume that you have read and are familiar with IFI's requirements and guidelines for mandatory assignments:
https://www.uio.no/english/studies/examinations/compulsory-activities/mn-ifi-mandatory.html
https://www.uio.no/english/studies/examinations/compulsory-activities/mn-ifi-guidelines.html

Note that any use of pretrained language models (such as ChatGPT) is forbidden for this assignment.

This is an individual assignment. You should not deliver joint submissions.

You may upload several submissions to Devilry, but only the last submission will be evaluated. Therefore, make sure to include all files in the last submission. If you submit more than one file, put them into a zip-archive and name your submission as follows: **<username>_in4080_mandatory_2**

Your submission should contain:

- One or several Jupyter notebooks containing your code, and where you answer the text questions in markup.
- A PDF version of the notebook(s) where all the results of the runs are included.

The assignment consists of three parts with a total of 100 points. You are required to get at least 60 points to pass. It is more important that you try to answer all questions rather than that you get everything correct.

Goals

In this assignment, we will experiment with sequence classification models for the part-of-speech tagging task. In the first part, we will evaluate some pre-implemented sequence classification models from NLTK. In the second part, we will experiment in detail with a greedy logistic regression tagger and investigate the impact of different feature types. Finally, in the third part, we will evaluate the best taggers on held-out data and perform more fine-grained evaluation.

Part 1 – Exploring the NLTK tagger landscape [30 pts]

The NLTK toolkit provides various methods to train and test part-of-speech taggers. Most relevant content is described in chapters 5 and 6. For our experiments, we will use the Brown corpus, which is annotated with part-of-speech tags and is also made available through NLTK.

Exercise 1a: Data Split [5 pts]

For our first experiments, we limit ourselves to the *news* section of the Brown corpus and split it into a training (90%) and a validation (10%) set. (We don't need a test set for the moment, but will build one in part 3.) Moreover, we use the universal tagset instead of the default one.

Sections 5.5.1 and 5.5.2 of the NLTK book should give you a starting point for this. (You may shuffle the data if you wish, but don't have to.) You can switch to the universal tagset with the following command:

```
sents = brown.tagged_sents(categories='news', tagset='universal')
```

You should store your data split in the variables `news_train` and `news_val`. Note that you may need to download the corpus first.

Exercise 1b: Most Frequent Class Baseline [5 pts]

The distribution of part-of-speech tags is typically quite skewed, with the most frequent class in general being common nouns. As a simple baseline, we should thus know how a model that always predicts the same (most frequent) class performs. This can be done with `nltk.DefaultTagger`. The NLTK book, section 5.4.1, shows how to "train" such a tagger. Note that we are using the universal tagset, so the most frequent tag is *not* named NN. Evaluate it on the validation set and report the accuracy.

Exercise 1c: Naïve Bayes Unigram Tagger [5 pts]

One of the first models discussed in course is a Naïve Bayes classifier that relies only on the current word and does not take any context into account. This model is available as `nltk.UnigramTagger`. Section 5.5.1 of the NLTK book shows how to train and evaluate such a tagger. Report the accuracy on the validation set. How does the accuracy on the universal tagset differ from the one reported on the default tagset in the NLTK book?

Exercise 1d: Bigram HMM Tagger [5 pts]

In the lectures, we spent quite some time on the HMM tagger. NLTK comes with a bigram HMM tagger which can be trained with the following command:

```
hmm = nltk.HiddenMarkovModelTagger.train(news_train)
```

Evaluate it on the validation set and report the result.

Exercise 1e: Perceptron with greedy decoding [10 pts]

In the lectures, we have shortly discussed Matthew Honnibal's proposal of a structured perceptron tagger with greedy decoding. He argued that an extended set of features is more helpful for tagging than exact (Viterbi) decoding. NLTK provides a re-implementation of this tagger that you can train in the following way:

```
perc = nltk.PerceptronTagger(load=False)
perc.train(news_train)
```

Evaluate it on the validation set and report the result.

Summarize the results of the previous exercises and discuss them in a few sentences. Do the accuracies correspond to your expectations?

Part 2 – Greedy LR taggers and feature engineering [35 pts]

In this part, we will dig a bit deeper in the implementation part and try to partially replicate the perceptron tagger, using logistic regression as its base classifier.

Exercise 2a: Getting started with a greedy logistic regression tagger [5 pts]

NLTK contains a `ConsecutivePosTagger` class, which actually corresponds to a greedy bigram HMM with additional features. It is described in detail in the NLTK book, chapter 6.1.6. The following code snippet reimplements this class to use the logistic regression classifier from Scikit-Learn as a basis:

```
import nltk
import numpy as np
import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction import DictVectorizer

class ScikitGreedyTagger(nltk.TaggerI):

    def __init__(self, features, clf=LogisticRegression()):
        self.features = features
        self.classifier = clf
        self.vectorizer = DictVectorizer()

    def train(self, train_sents):
        train_feature_sets = []
        train_labels = []

        for tagged_sent in train_sents:
            history = []
            untagged_sent = nltk.tag.untag(tagged_sent)

            for i, (word, tag) in enumerate(tagged_sent):
                feature_set = features(untagged_sent, i, history)
```

```
                    train_feature_sets.append(feature_set)
                    train_labels.append(tag)
                    history.append(tag)
            x_train = self.vectorizer.fit_transform(train_feature_sets)
            y_train = np.array(train_labels)
            self.classifier.fit(x_train, y_train)

        def tag(self, sentence):
            test_features = []
            history = []
            for i, word in enumerate(sentence):
                featureset = self.features(sentence, i, history)
                test_features.append(featureset)
            X_test = self.vectorizer.transform(test_features)
            tags = self.classifier.predict(X_test)
            return zip(sentence, tags)
```

This class requires an additional function that defines the features to be used. This function returns a dictionary. This dictionary will then be converted into a set of one-hot vectors (one vector per dictionary key), and concatenated into a single vector.

A basic feature function that only uses the current and previous words could look like this:

```
    def pos_features(sentence, i, history):
        features = {"curr_word": sentence[i]}
        if i == 0:
            features["prev_word"] = "<START>"
        else:
            features["prev_word"] = sentence[i-1]
        return features
```

We have all the ingredients in place now to train and evaluate a tagger with this model:

```
    lr_tagger = ScikitGreedyTagger(pos_features)
    lr_tagger.train(news_train)
    lr_tagger.accuracy(news_val)
```

How does the accuracy of this tagger compare to the taggers tested in part 1?

Exercise 2b: Adding word context features [5 pts]

The basic feature function contains the previous and the current word. Also add the next word and the word before the previous one. Describe which combination works best and keep it for the next experiment.

Exercise 2c: Adding transition features [5 pts]

Modify the feature function to include the tag predicted at the previous position. Does this help? What about a trigram model that includes the two previously predicted tags?

Exercise 2d: Even more features [10 pts]

Try to add more features to get an even better tagger. Only the fantasy sets limits to what you may consider. Some ideas: Extract suffixes and prefixes from the current, previous or next word. Is the current word a number? Is it capitalized? Does it contain capitals? Does it contain a hyphen? etc. What is the best feature set you can come up with? Train and test various feature sets and select the best one.

If you use sources for finding tips about good features (like articles, web pages, NLTK code, etc.) make references to the sources and explain what you got from them.

Exercise 2e: Regularization [10 pts]

As in the previous assignment, we will study the effect of different regularization strengths now. In scikit-learn, regularization is expressed by the parameter C. A smaller C means stronger regularization. Try with C in [0.01, 0.1, 1.0, 10.0, 100.0, 1000.0] and see which value which yields the best result. You can also try additional values.

Summarize your experiments to make clear which set of features and parameters provide the best results, and what the corresponding accuracy score is. Did you manage to outperform the perceptron tagger? If not, where do you think the bottleneck of your current tagger lies?

Part 3 – Training and testing on a larger corpus [35 pts]

The Brown corpus covers 15 different genres, but we have only explored the *news* genre so far. In this part, we will retrain the most promising taggers on an extended set of genres and test them on held-out data.

Exercise 3a: Compile the extended training and test data [5 pts]

The NLTK book, chapter 2.1.3, lists the names of the 15 genres available in the Brown corpus. We will set two genres aside for testing: *hobbies* and *adventure*. For training, we will use the *news* training set prepared for the previous exercises, as well as the data from the remaining 12 genres. Prepare the corpus as described and store the datasets in the variables `all_train`, `hobbies_test` and `adventure_test`. We will not use `news_val` in this part. Make sure to use the universal tagset.

Exercise 3b: Evaluate the taggers [5 pts]

Identify the most successful tagger from part 1 and the best setup from part 2. Retrain both of them on `all_train` and evaluate them separately on the two test genres. Report the results and discuss them briefly: Which of the two genres is “easier”? How well do the two taggers generalize to unseen genres?

Exercise 3c: Confusion matrix [5 pts]

The accuracy gives us a high-level overview of the performance of a tagger, but we may be interested in finding out more details about where the tagger makes the mistakes. The universal tagset is reasonably small, so we can produce a confusion matrix. Take a look at https://www.nltk.org/api/nltk.tag.api.html and make a confusion matrix for the results. Pick the results of one test set and one tagger. Make sure you understand what the rows and columns are. Which pairs of tags are most easily confounded?

You can find the documentation of the tagset in the following link, but note that NLTK uses an earlier, slightly different version of the tagset: https://universaldependencies.org/u/pos/index.html

Exercise 3d: Precision, recall and f-measure [10 pts]

Finding hints on the NLTK web page linked above, calculate the precision, recall and f-measure for each tag and display the results in a table.

Also calculate the macro precision, macro recall and macro f-measure across all tags.

Exercise 3e: Error analysis [10 pts]

Sometimes, it makes sense to inspect the output of a machine learning model more thoroughly. Find five sentences in the test set where at least one token is misclassified and display these sentences in the following format, with both the predicted and gold tags.

```
Token                   pred   gold
=================================
The                     DET    DET
panda                   NOUN   NOUN
eats                    VERB   VERB
shoots                  VERB   NOUN
and                     CONJ   CONJ
leaves                  VERB   NOUN
```

Identify the words that are tagged differently. Comment on each of the differences. Would you say that the predicted tag is wrong? Or is there a genuine ambiguity such that both answers are defendable? Or is even the gold tag wrong?