Exercise Sheet 3

Due Date: December 19, 8 pm

Note on Submission

All solutions have to be uploaded together as a single zip file to LernraumPlus. Solve the exercises by implementing the functions in the file `exercise_sheet3.py`.

In this exercise sheet, you will create your own maximum entropy model by implementing the methods of the class `MaxEntModel` in the file `exercise_sheet3.py`. We use features of the following form (we already used them in the example discussed in the lecture):

$$f_{w/t}(x_i, y_i) = \begin{cases} 1 & \text{iff } x_i = w \quad \wedge \quad y_i = t \\ 0 & \text{otherwise} \end{cases}$$

$$f_{t_1/t_2}(x_i, y_i) = \begin{cases} 1 & \text{iff } y_{i-1} = t_1 \quad \wedge \quad y_i = t_2 \\ 0 & \text{otherwise}, \end{cases}$$

where w is a word, t, t_1, t_2 are labels, x_i is the word of a sentence at position i, y_i is a label we assigned to this word and y_{i-1} is the label of the word at position $i-1$. That is, $f_{w/t}(x_i, y_i)$ is 1 if and only if the i-th word of a given sentence is w and if we assign the label t to this word. Similarly, $f_{t_1/t_2}(x_i, y_i)$ is 1 if and only if the label of the $(i-1)$-th word of a given sentence is t_1 and if we assign the label t_2 to the i-th word.
Use the provided corpus `corpus_pos.txt` for training your model. You can use the Python function `import_corpus` for importing this corpus.

Exercise 1 – Building the Feature Set [3+2 points]

First of all, you need to build the set $\mathcal{F}$ of all features. Therefore, do the following things:

a) Build the set $\mathcal{X}$ of all words and the set $\mathcal{Y}$ of all labels occurring in the corpus. Use the variable `labels` to save the set of all labels. Then build the set of all features

$$\mathcal{F} = \{f_{w/t} | w \in \mathcal{X}, t \in \mathcal{Y}\} \cup \{f_{t_1/t_2} | t_1, t_2 \in \mathcal{Y}\}\,.$$

You can use pairs of strings to represent the features $f_{w/t}$ and f_{t_1/t_2}. Note that there is no previous word for the first word of a sentence. In this case, the label of the previous word is `start`.

For example, assume we are given the ``sentence'' ((the, DT), (dog, NN)) and we decided to represent features as pairs of strings, then we would build the following sets:

$$\begin{aligned}
\mathcal{X} &= \{\texttt{the}, \texttt{dog}\} \\
\mathcal{Y} &= \{\texttt{start}, \texttt{DT}, \texttt{NN}\} \\
\mathcal{F} &= \left\{ \begin{array}{l}
(\texttt{the}, \texttt{start}), (\texttt{the}, \texttt{DT})(\texttt{the}, \texttt{NN}), \\
(\texttt{dog}, \texttt{start}), (\texttt{dog}, \texttt{DT}), (\texttt{dog}, \texttt{NN}), \\
(\texttt{start}, \texttt{start}), (\texttt{start}, \texttt{DT}), (\texttt{start}, \texttt{NN}), \\
(\texttt{DT}, \texttt{start}), (\texttt{DT}, \texttt{DT}), (\texttt{DT}, \texttt{NN}), \\
(\texttt{NN}, \texttt{start}), (\texttt{NN}, \texttt{DT}), (\texttt{NN}, \texttt{NN})
\end{array} \right\}
\end{aligned}$$

Then, assign each feature to a unique index $i \in \{1, \ldots, |\mathcal{F}|\}$. Then the feature assigned to index i can be referenced by f_i. Finally, initialize the vector θ of parameters. Do all this inside the method `initialize`.

b) Implement the method `get_active_features` which returns a vector $\vec{f} \in \{0,1\}^{|\mathcal{F}|}$ indicating which features are active for a given word x_j. I.e., for a given word x_j and a label y_j for this word, the i-th element of this vector is set to 1 if and only if $f_i(x_j, y_j) = 1$ and 0 otherwise.

Hint: Use a Python dictionary to store the features and their indices.

Exercise 2 – Computing Conditional Probabilities [2+2 points]

a) Implement the method `cond_normalization_factor`, which returns the normalization factor $1/Z(x_i)$, where

$$Z(x_i) = \sum_{y' \in \mathcal{Y}} e^{\vec{\theta} \cdot \vec{f}(x_i, y')} .$$

Note that the label y_{i-i} of the previous word is implicitly given by the index i, but you have to provide it explicitly in the code.

b) Implement the method `conditional_probability`, which computes the conditional probability

$$P(y|x_i) = \frac{1}{Z(x_i)} e^{\vec{\theta} \cdot \vec{f}(x_i, y)} .$$

of a label y given the word x_i.

Exercise 3 Empirical and Expected Feature Count [1 + 3 points]

Implement the method `empirical_feature_count`, which returns the vector $E[\vec{f}(x_i, y_i)]$ of the empirical feature count, and the method `expected_feature_count`, which returns the vector $E_{\vec{\theta}}[\vec{f}(x_i)]$ of the expected feature count, given the parameters $\vec{\theta}$ of the current model.

Exercise 4 Training the Model [2+3+2 points]

a) Implement the method `parameter_update`, which performs one learning step according to

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha(E[\vec{f}(x_i, y_i)] - E_{\vec{\theta}}[\vec{f}(x_i)]) ,$$

where α is a learning rate.

b) Implement the method `train` for training your model via gradient descent. Use the method `parameter_update` you implemented above.

c) Implement the method `predict`, which predicts the most probable label of a given word x_i at position i in a sentence.

Exercise 5 Using Full Sentences for Training [2+2+4 points]

So far, we only considered one word and the corresponding label and the label of the previous word in each iteration of the training procedure. Now we are going to use n sentences from the training corpus in each iteration.

a) Implement the methods `empirical_feature_count_batch` and `expected_feature_count_batch` to compute the empirical and the expeceted feature count, respectively, given a set of sentences from the training corpus. You are allowed to use the functions you implemented in exercises above.

b) Implement the method `train_batch` for training your model via gradient descent. In each iteration, randomly select n sentences from the training corpus (n is given by the argument `batch_size`) and use these sentences to compute the gradient. If you choose $n = |\mathcal{D}|$ (where $|\mathcal{D}|$ is the number of sentences in the training corpus), then the training procedure is equivalent to the one presented in the example in the lecture.

c) Compare the training procedures you implemented in `train` and `train_batch` in terms of the convergence rate. To be more precise, do the following things:

1 Create a test set $\mathcal{T}$ by randomly selecting 10% of all sentences from the provided corpus $\mathcal{C}$. Use the set $\mathcal{D} = \mathcal{C} - \mathcal{T}$ for training.

2 Create two instances A and B of the class `MaxEntModel`. A will be trained by `train` and B by `train_batch`. Use the training corpus $\mathcal{D}$ for initialization.

3 Do the following for N iterations (N should be large to guarantee convergence of each training procedure): Train A by `train` and B by `train_batch` for one iteration, use only one sentence in `train_batch` at each iteration. Save the number of words w_A and w_B used so far in each training process. For example, when training A, w_A increases by 1 after each iteration. But when training B, w_B increases by the length of the sentence used in the current iteration. Then compute and save the accuracy (see Wikipedia) of each model on the test set with respect to the number of words used so far.

4 Finally, plot the data you generated during training (accuracy against number of words). Describe and interpret the plot. Don't forget to include this plot in the set of files you finally submit.

Put all your code for evaluation into the function `evaluate`. If the training procedures do not converge, thing about using a smaller learning rate. You may use different learning rates in the two training methods.