Student: Maor Shliefer
ID: 305206898

NLP – Assignment 1

General info:
* Because of required directory structure, files were duplicated between folders.
  Each folder has mandatory files+duplicated from previous folders (only what's needed).
* Project structure:

```
zip
  writeup.pdf
  -run
    .sh files – can be used to run all files at once – not required. Use if you want
  -src
   -hmm1,hmm2,hmm3,memm1,memm2,memm3,ner
  -data-test – input for the assignment
  -data-ner – input for the assignment
  -output-test
    sklearn_model trained on test data
```

* Notes on how to use the run folder appear at the end of this file. It is not required to run the project.
* data folders (input) are included to make running with run scripts easier (if you choose to use it)

Question 1: Describe how you handled unknown words in hmm1
Answer:

* Define rare words. A rare word is a word that appeared less then 5 times during training (as described in the memm paper).
* For a rare word w, convert it to either *UNK*_suffix if w ends with a common suffix. *UNK* otherwise.
* common suffixes were picked to be {ed, ing}
* During loading of text, all number appearances (numeric or textual) were replaced with *Number*.
  e.g three-year-old → *Number*-year-old

Question 2: Describe your pruning strategy in the viterbi hmm.
Answer:

* I implemented beam search, as seen in pdf from a lecture from Rochester university:
  http://www.cs.rochester.edu/u/james/CSC248/Lec9.pdf (Beam search: page 2)
* The idea is pretty simple and beautiful.
  If a Viterbi algorithm looks like this:

Algorithm:

- For $i = 1, \ldots, n$
  - For $t \in T_1, r \in T_2$
    $V(i,t,r) = \max_{t'} V(i-1,t',t)q(r|t',t)e(w_i|r)$

return: $\max_{t \in T, r \in T} V(n,t,r)$

Insert prunning here.
Modify T_1, T_2 to contain only likely tags.

* Picking likely tags for T_1, T_2 is done using beam search.
* Look at the highest score for step i-1, call it Mt, multiply that by percentage constant k. That is our threshold.
* When calculating V(i), only look at V(i-1) where V(i-1) > threshold.
* Formally (where the line is) add:

  Mt = max(V(i-1,:,:))
  k = 0.5
  T_1, T_2 = {t_1,t_2 where V(i-1,t_1,t_2) > Mt*k}

* Note: this code might seem like it implies T_1=T_2, but it's not. T_1 is for one dimension, T_2 for the other.
* This method is useful, because the threshold **is not constant**. It always uses the max probability at the last step as reference.
* Testing showed that even with aggressive prunning (k=0.5). Performance was not damaged at all (hmm+memm),
  and MEMM+Viterbi for test data took less then a minute to run.
* This method was used in both hmm and memm, same code too, and located at viterby.py file.
* Calculations in viterbi+prunning was done using logs, to avoid computational errors.

Question 3: Report your test scores when running the each tagger (hmm-greedy, hmm-viterbi, maxent-greedy, memm-viterbi) on each dataset. For the NER dataset, report token accuracy accuracy, as well as span precision, recall and F1.
Answer:

Algorithm	Test dataset	NER dataset
Greedy HMM	Per token acc: 0.891	Per token acc: 0.91 All-types Prec: 0.729 Rec: 0.515

		LOC Prec: 0.876 Rec: 0.638 F1: 0.36 MISC Prec: 0.71 Rec: 0.624 F1: 0.33 PER Prec: 0.784 Rec: 0.397 F1: 0.263 ORG Prec: 0.519 Rec: 0.433 F1: 0.236
Viterbi HMM	Per token acc: 0.895	Per token acc: 0.916 All-types Prec:0.762 Rec:0.529 LOC Prec: 0.890 Rec: 0.660 F1: 0.379 MISC Prec: 0.775 Rec: 0.653 F1: 0.354 PER Prec: 0.781 Rec: 0.400 F1: 0.264 ORG Prec: 0.570 Rec: 0.444 F1: 0.249
Greedy MEMM	Per token acc: 0.927	Per token acc: 0.934 All-types Prec: 0.710 Rec: 0.617 LOC Prec: 0.833 Rec: 0.714 F1: 0.384 MISC Prec: 0.792 Rec: 0.654 F1: 0.358 PER Prec: 0.583 Rec: 0.539 F1: 0.280 ORG Prec: 0.673 Rec: 0.567 F1: 0.308
Viterbi MEMM	Per token acc: 0.955	Accuracy: 0.953 All-types Prec: 0.923 Rec: 0.720 LOC Prec: 0.940 Rec: 0.775 F1: 0.425 MISC Prec: 0.930 Rec: 0.705 F1: 0.401 PER Prec: 0.899 Rec: 0.750 F1: 0.409 ORG Prec: 0.928 Rec: 0.614 F1: 0.369

* HMM results are pretty low (<94), I tried playing with it **alot**. I tried picking most common X words, instead of a minimum occurrence filter. I tried adding more common suffixes other then {ed, ing}. Removing prunning in Viterbi. Nothing helped. The algorithm does not miss "unknown" words, but words that have several tags, and are common. I think it is a bug, as I tried all the options proven by other students and it didn't help. I tried really hard to find it, but to no avail.
* F1 equation in classroom was different from wiki, I used one from the classroom (F1=prec*rec/(prec+rec))

Question 4: Is there a difference in behavior between the hmm and maxent taggers? discuss.
Answer:

* Big difference, hmm treat each word as a black box, and ignores it's structure ("capital letters", "contain numbers", "suffix", "prefix", "hyphen") - all have to be engineered by hand into the representation of w, and no matter how we do it, we loose a lot of information.
For instance: Let's take the word "Influential", and let's assume it is a rare word. Appearing in the training with capital letter only once.
Assuming we keep each word as is, we treat it is as *UNK*, even though it is pretty common as lower case word.
Assuming we lower the cases of all words, we loose information of how 'Influencial' might act differently then 'influential'.
Assuming we do case sensitive search for a word, and only if it is UNK, then try search case insensitively. Then it does not help. I tried it.
* Another disadvantage of hmm, is that it is impossible to combine features and retain connection. (UNK+capital+ing, UNK+ing. Are two different words)
* The big disadvantage of hmm, is that it is too inflexible. MaxEnt taggers however, can mix and combine the features as they see fit, giving more weight to the relevant ones.

Question 5: Is there a difference in behavior between the datasets? discuss.
Answer: I found several differences:

* For NER there are only 8 tags. So in terms of per token accuracy, it is no surprise NER performed better then POS.
* The POS input was given with the assumption: line=sentence. The NER file have some sentences spaning more then a line.
* NER file had a DOCSTART word per paragraph, and was much smaller.
* Big possible disadvantage of NER, is that even though the algorithms might guess if a new word is a location, person or organization from the context around it. It still needs to find the beginning and end of the span. Making this more difficult then per token tagging.
* For per token : NER performed slightly better (less tags).
* For span : precision, recall and F1 values (for spans) are consistently lower then per token accuracy. Which supports my idea that span tagging is harder then single tag tagging. (Even if we take F1 formula from wikipedia, which is 2*F1 here).

Question 6: What will you change in the hmm tagger to improve accuracy on the named entities data?
Answer:

Since not every line is a sentence, I will have either implemented an algorithm for sentence segmentation, and apply it for each paragraph. Or used the tagger with every paragraph (DOCSTART), as a giant sentence. I suspect the second option is not very good, because sentence boundaries are very important for it.

Question 7: What will you change in the memm tagger to improve accuracy on the named entities data, on top of what you already did?
Answer:

* The MEMM paper mentioned dropping features that appear less then 10 times.
* Add more data (always good!)
* Analyze missed words, and engineer features to help the algorithm.
  Example of features: length of word, count of capital letters in word, contain dot, is the word all capital letters, or a mix or lower and upper case?
* Add a dictionary of known names, and classification (person, location etc..), add classification from dictionary as a feature for input.
* Add more caching (more then I already did), and pruning strategies to help MEMM run fast with all those extra features.

Question 8: Why are span scores lower than accuracy scores?
Answer: see question 5.

How to use .sh files to run all the project files at once (**not required**):

* Give execution rights to scripts (chmod a+x for .sh files + virtualenv\bin\activate file)
* In general.config, modify the settings at the start (set paths for virtual env and project folder).
* run ./train.sh ./predict.sh ./eval.sh (one after the other)

* Note: The script implements profiling. It has "ner.config" for NER run, "test.config" for test data run.
* They are used to tell the script where to find the data and where to output the results.
  "ner.config" relies on having "data-ner" folder inside project folder.
  "test.config" relies on having "data-test" folder.
* You set the active profile in general.config