TP3 - IBM1

Martino Ferrari

Project Structure

```
.
+-- core/                < folder containing all the functional code of the project
|   +-- em_ibm.py        < EM algorithm
|   +-- ibm1.py          < IBM Translation Model 1
|   +-- train.py         < training pipeline
+-- resources/           < folder containing the optional static translation tables
+-- test/                < folder containing the optional static translation tables
|   +-- myalignments     < output of main.py
|   +-- test.align.es    < gold file for translation es -> en (inversed index)
+-- training/            < folder containing the training data set
+-- conf.json            < example of configuration (optional)
+-- main.py              < main script with all the required functinality
+-- generate_tables.py   < script to generate static translation table (optional)
```

Usage

To run my code please execute:

```
python3 main.py
```

There are two mode to use this code: one with parameters and one using a persistent json config file (e.g.: *conf.json* in the main folder). The arguments are the following:

```
usage: main.py [-h] [--conf CONF] [-i EM_MAXITERS] [-n NSENTENCES]
               [-t TRAINDATA] [-s TESTDATA]

IBM1 Em & translation model

optional arguments:
  -h, --help      show this help message and exit
  --conf CONF     use a json config file (default: None)
  -i EM_MAXITERS  em max iterations (default: 10)
  -n NSENTENCES   em max iterations (default: -1 [all])
  -t TRAINDATA    training file (default: t../europarl_50k_es_en)
  -s TESTDATA     test file (default: t../test)
```

if a configuration file is used **all the other arguments will not be taken in account!**

Soft EM

The first part of the TP consisted in implementing a soft version of the EM algorithm to be able to create the translation table for the IBM1.

The first step is to reading the parallel corpus (`core.train.read_training`) and initialize uniformly the translation table (`core.train.initialize_translation_table`).

Once the table is initialized the EM algorithm will optimize it and find the translation table. In our case the EM stop condition is a maximum number of iterations (and not a convergence value) and at every iteration the EM will do the follow (`core.em_ibm.__step`):

```
    # counter
    count = defaultdict(lambda: defaultdict(float))
    # Normalizer factor for class
    total = defaultdict(float)
    # for every couple of sentences
    for c in data:
        # extract the sentences
        s = c['source'].split()
        t = c['target'].split()
        # temp normalizing variable
        s_total = defaultdict(float)
        # for each word w of the source sentence
        for w in s:
            # for each word k of the target sentence
            for k in t:
                # accumulate the probability of the word w
                s_total[w] += self.__t__[w][k]
        # for each word w of the source sentence
        for w in s:
            # for each word k of the target sentence
            for k in t:
                # compute the current probability of the word w knowing k
                p = self.__t__[w][k]
                # accumulating the value in the counter and normalizing variable
                count[w][k] += p/s_total[w]
                total[k] += p/s_total[w]
        del s_total

    # normilze probability for class
    for e in count:
        for f in count[e]:
            count[e][f] /= total[f]
    del total
    # return the new normalized translation table
    return count
```

At every new iteration the EM will find a better translation table and it will asymptotically converge to a local optimum of the problem.

IBM1

The IBM Model 1 implementation for this TP is used to find the best possible alignment between two sentences such:

EN : *those guidelines are presented below .* ES : *más abajo figuran dichas directrices .*

To do so using the computed translation table is relatively easy as this model is a world by world translation model and where as well all the alignments of two sentences have the same probability.

The code to do so is the following (`core.ibm1.best_alignment`):

```
    # computed alignment
    selected = []
    # computed probability
    probability = 1
    # for each word of the source sentence
    for i in range(0, len(s)):
        # best translation (temp var)
        sel = -1
        # probability of best translation (temp var)
        prob = -1
```

```
        # for each word of the target sentence
        for j in range(0, len(t)):
            # if probability of translation of the couple of word is higher of the
selected
            if self.t(f[i], e[j]) > prob:
                # select new alignment
                sel = j
                prob = self.t(f[i], e[j])
        # accumulate probability
        probability *= prob
        # append alignment
        selected.append(sel)
    # return alignment and its probability
    return selected, probability
```

Results

Performances with different sizes of training data:

Language	# Training	# EM Iterations	Precision	Recall	F1 Score	Time
English → Spanish	50000	5	0.596	0.487	0.536	292s
English → Spanish	20000	5	0.575	0.470	0.517	137s
English → Spanish	10000	5	0.540	0.441	0.485	68s
English → Spanish	5000	5	0.508	0.415	0.457	34s
English → Spanish	1000	5	0.398	0.325	0.358	7s

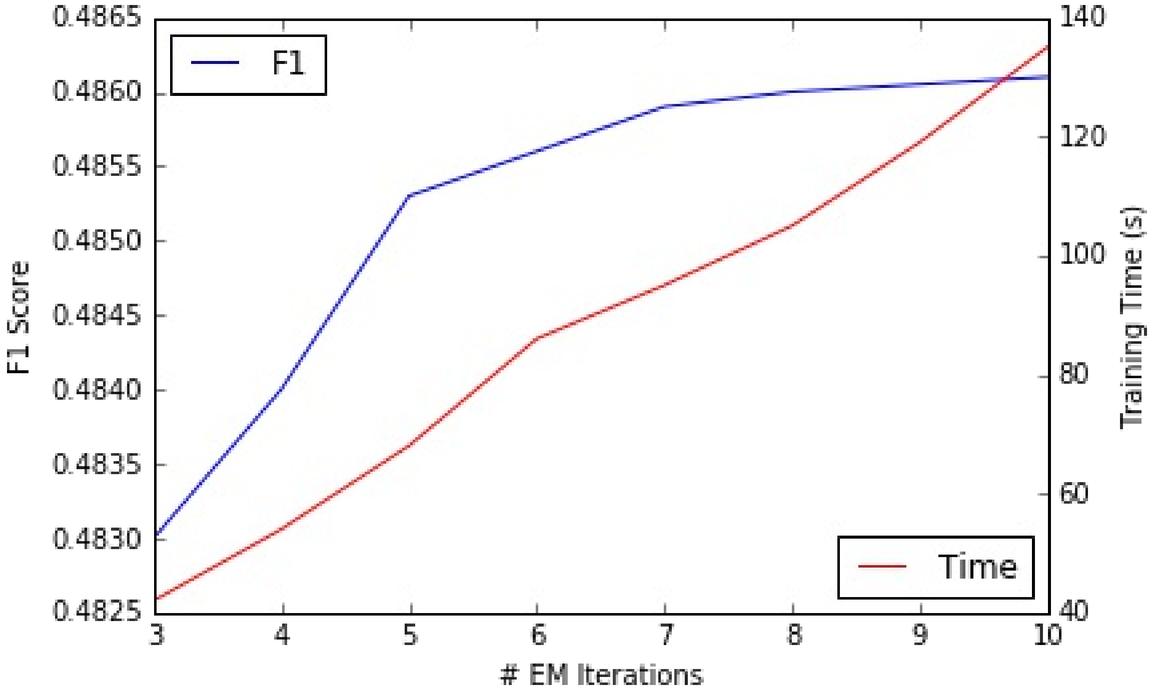
Performances of the inverse translation with different sizes of the training data (note: I created a new gold file swapping the indexes to compute this results):

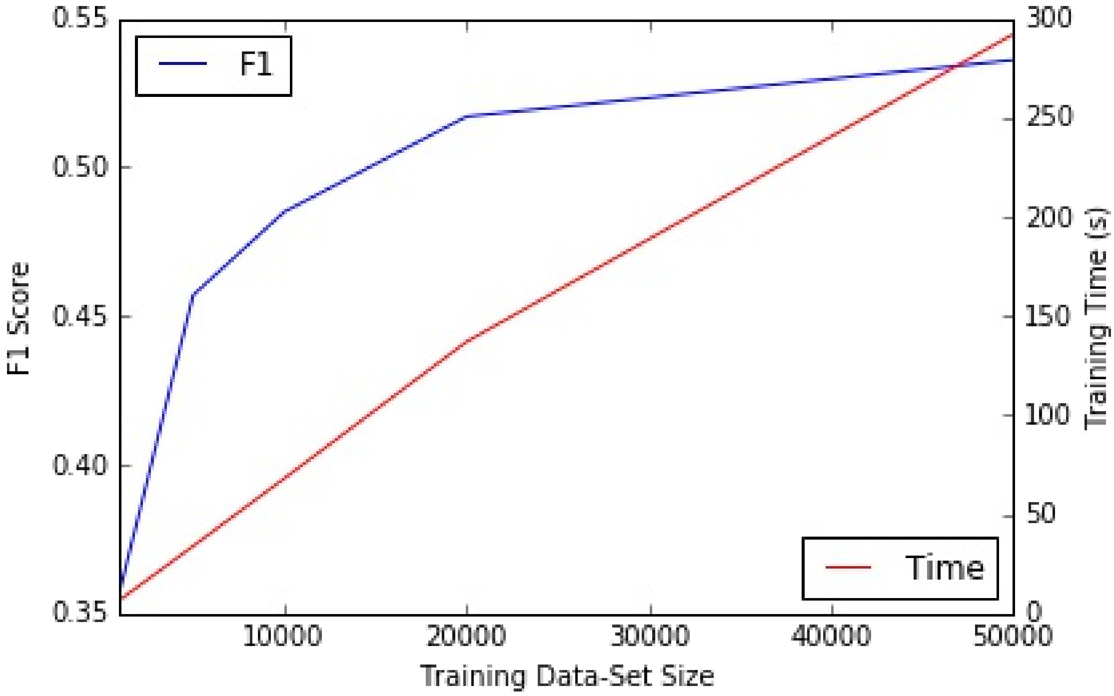
Language	# Training	# EM Iterations	Precision	Recall	F1 Score	Time
Spanish → English	10000	5	0.499	0.446	0.470	67s
Spanish → English	5000	5	0.468	0.418	0.442	33s
Spanish → English	1000	5	0.373	0.334	0.352	8s

Performances with diffrent number of EM Iterations:

Language	# Training	# EM Iterations	Precision	Recall	F1 Score	Time
English → Spanish	10000	10	0.541	0.442	0.486	135s
English → Spanish	10000	9	0.541	0.442	0.486	119s
English → Spanish	10000	8	0.541	0.442	0.486	105s
English → Spanish	10000	7	0.540	0.441	0.485	95s
English → Spanish	10000	6	0.540	0.441	0.485	86s
English → Spanish	10000	5	0.540	0.441	0.485	68s
English → Spanish	10000	4	0.538	0.440	0.484	54s
English → Spanish	10000	3	0.537	0.439	0.483	42s
English → Spanish	10000	2	0.534	0.437	0.481	29s
English → Spanish	10000	1	0.418	0.342	0.376	16s
English → Spanish	10000	0	0.049	0.040	0.044	3s

To understand better the results I choose to plot the dependency of the F1 Score to the number of EM iterations and to the training-set size. Moreover I choose to see the computation time trend.





As expected the computation time increase linearly both depending on the number of EM iterations and both on the size of the training data-set.

Instead the quality of the results (F1 Score) has a logaritmic trend vs the size of the training data-set. However The trend of the F1 score agains the number of EM iterations is more complex and has a phase transition at around 5 iterations. That's mean that after 5 iteration the algorithm converge and there are no big improvments after that.

Finally the translations ordered by its probabilites can be found at `test/mytranslations`.