Natural Language Processing Final: Day 1

Date: 22nd Dec 2014

Time: 8:30 pm to 5:45 pm IST

Problem # 1: Develop a Named Entity Recognizer for the Mobile Corpus supporting the specified tags

In this project you will build 3 building blocks:

1. A MaxEnt Classifier: A reference implementation was already provided to you that you may customize for the requirements of this problem
2. Feature functions that support the tags:

   tag_set = {

   "Company": "Org",

   "Product Family": "Family",

   "Model": "Model", # map it to Version

   "OS": "OS",

   "OS Version": "Version",

   "Date": "Date", # map it to other

   "Price": "Price",

   "Smartphone": "Phone",

   "Location": "Place", # map it to other

   "Size": "Size", # map it to feature

   "App": "App", # map it to other

   “Feature": “Feature“,

   "Other": "Other“,

   }
3. A Maximum Entropy Markov Model (MEMM) that implements the algorithms discussed in the class.

Specifically, you are required to perform the following:

1. Use the tool http://jnresearch.com/edit_corpus to:
   a. Perform any corrections to the data that you entered already

   b. Tag any references to a mobile device feature using the feature tag. For example: tag the words like “front camera” as front/Feature camera/Feature. Please refer the document from Shruti for examples.
2. Once the step 1 as above is completed, go to url: http://jnresearch.com/show You will see a json formatted list of records. You are required to do the following:
   a. Copy this data in to a text file using a text editor
   b. Embed this data (list) as: {“root”: data_received_from_url}
   c. Save this as a JSON file: all_data.json
3. The file, all_data.json contains the Mobile Corpus in the format below (example):


```
{"root":[
        {"data": [
            {"updates": [
                    {"tag": "Other", "word": "Which"},
                    {"tag": "Phone", "word": "phones"},
                    {"tag": "Other", "word": "were"},
                    {"tag": "Other", "word": "released"},
                    {"tag": "Other", "word": "this"},
                    {"tag": "Date", "word": "year"},
                    {"tag": "Other", "word": "?"}
              ],
              "sentence": "Which phones were released this year?"
            },
            {"updates": […], "sentence": "… "}
            …
            ]
            "short_name": “Meghana”,
            "user_id": "1PI11CS096"
            },
            …
            ]}
```

The number of elements under root key is the number of people who contributed to the corpus. For 75 students and 4 faculty this should be 79 (assuming everyone contributed). Each of these elements is a dictionary having 3 keys: data, user_id and short_name. The element data is a list of dictionaries, each dictionary element having keys: updates, sentence. The key updates is mapped to a list of elements of the form: {“tag”:…, “word”: …}.

The first step in this assignment is to build a list of “history tuples” where each tuple is of the form: (tag_t_minus_2, tag_t_minus_1, word_tokens, current_index_i). We will support 3 tags: (“Org”, “Family”, “Other”). When you build the history tuples please do the following transformations to the JSON input:

- Replace occurrence of tags as per the supported tags listed in step 2 above. For example: replace any tag “Model” with “Family”

The space of all these tuples constitute the input space X. An input x_i is a point in the input space.

The space of all tags is the output space Y. In our case Y is a set of 3 elements: Y = {“Org”, “Family”, “Other”}

NOTE: You can use the build_history function that we will provide.

You can use this function as below:

```
data = json.loads(open(json_file).read())['root']

(history_list, sents, expected, )  = build_history(data, supported_tags_list)
```

history_list is the list of tuples needed as inputs for the feature functions.

sents is the list of sentence tokens, for example:

```
[
    ["I", "need", "a", "Microsoft", "Galaxy", "2", "Smartphone"],
    ["Samsung", "released", "a", "Android", "2", "Smartphone"],
    ["I", "have", "a", "Blackberry", "OS", "Smartphone"],
    ["Microsoft", "announced", "the", "quarterly", "results", "today"],
]
```

expected is the list of expected values taken from the training data. This is required when you compute metrics like precision, recall etc.

4. Features are at the heart of the MaxEnt classification process. For each of the tags to be supported you are required to identify one or more features. A feature in MaxEnt model is a function $f_k(x, y)$, where x is an element in X and y is an element in Y. In our case, x is a history tuple and y is a tag. You are

required to implement this function which should return 0 or 1 as the value of the feature. The total number of features is the dimensionality of the input to MaxEnt model. NOTE: For a dimensionality of 10, you will need to write 10 such functions. These functions are called indicator functions when they return binary value. Typically you may end up writing around 50 functions to support about 10 tags (assuming there are 5 features per tag).

5. For this exercise we will select 7500 history tuples for training and another 50 sentences for testing (from sents as above). Choose a random number r in the range of 0 to (number_of_history_tuples – number_of_trg_examples). Use this to select the training examples. Similarly select test examples using a random number. The test sentences should not be from the training set. Please note that we need to experiment with various datasets to measure accuracy.
6. Please refer the reference implementation for MaxEnt that we provided during Unit 4 evaluation. If you are building this on your own, design and implement the class: MyMaxEnt() with the following methods:
    a. __init__ should accept history list and the list of feature functions as input. You can use a method create_dataset() that takes the history and produces the training examples for the MaxEnt. The training examples are feature vectors where each element of this vector is a binary.
    b. Initialize the model (v in our slides) to a vector of zeros. Note that model will have same dimensionality as feature vector. For each feature there is a model parameter. ($f_k(x, y)$, v_k)
    c. cost(model): Given the model, compute the cost as per the equations given in the slides.
    d. train() should train the MaxEnt model using the 50 tuples selected as above and transformed to a dataset using the feature functions. Once you have written the cost function you can invoke for example:

```
from scipy.optimize import minimize as mymin
def train(self):
        params = mymin(self.cost, self.model, method = 'L-BFGS-B')
        self.model = ….
```

    e. The method p_y_given_x(h, tag) should take the history tuple and the required tag as the input and return the probability.
    f. You may also (optionally) write the function gradient() that maximizes the log-likelihood (or minimizes the negative of this) and set jac = gradient when you invoke the minimization function of scipy
    g. The method classify(h) performs the classification by determining the tag that maximizes the probability. That is call p_y_given_x(h, t) for various values of t and select the t that returned the maximum value.
7. Run the classifier classify() method with the test samples and record the result
8. Build the MEMM: Implement the Viterbi algorithm to determine the correct tag sequence for the sentences in the training set.

9. Evaluate the accuracy of the implementation. Benchmark with the reference implementation by looking at: http://jnresearch.com/show_perf. You can also try using http://jnresearch.com/ner_tool and check the outputs generated by the tool in order to benchmark.
10. For your test samples compute at each tag level Precision, Recall and F1 score. You can use the class NerMetrics that will be provided to you.
11. You are required to achieve an F1 Score of minimum 60% for Org tag, 85% for Other tag, 60% for Feature tag, 80% for OS tag, 70% for Price tag. For other tags in the supported tags you need to achieve a minimum 40% F1 score.
12. Once you are done with the development upload your files using the upload web service provided in the ner_client.py module.

NOTE

You should implement all these using the files as below and upload them for evaluation. Please make the file names all lowercase letters as in Linux file names are case sensitive.

1. mymaxent.py : Implements MyMaxEnt class
2. memm.py : Implements Memm class
3. feature_functions.py : Implements all feature functions as FeatureFunctions class
4. ner_main.py : the main program that:
    a. Prepares the input data for MaxEnt
    b. Trains the MaxEnt using train() method
    c. Performs tagging using the MEMM
    d. Evaluates the performance using NerMetrics()
5. After training the classifier create a pickle file with the name: all_data.p

Upload all the above 5 files using ner_client.upload()

Deliverables:

1. Source code of all your py modules (use ner_client.upload())
2. Pickle file of your classifier (use ner_client.upload())

Best wishes from your faculty ☺