Natural Language Processing – Assignment #2

Task #1: POS-tagging and Viterbi decoding

Given the sentence "fish swim" and the following probabilities:

tags: N (noun) and V (verb) with initial probabilities: `P(N)=0.6, P(V)=0.4`

<u>`transition:`</u>

- `P(N → N) = 0.2   P(N → V) = 0.8`
- `P(V → N) = 0.5   P(V → V) = 0.5`

<u>`emission:`</u>

- `P(fish|N) = 0.7        P(fish|V) = 0.1`
- `P(swim|N) = 0.1        P(swim|V) = 0.4`
- `P(another word | N)    P(another word | V)`
- …

Run one full iteration of the Viterbi algorithm and produce:

- the Viterbi decoding matrix C (values for each word-tag pair)
- the final most-likely tag sequence

Task #2: predicting words valence (sentiment) with distributional semantics

This task involves a study on one of the psycholinguistic word properties: valence (sentiment). A common way to study emotions in the psycholinguistic literature groups affective states into a few major dimensions. The Valence-Arousal-Dominance (VAD) representation has been widely used to conceptualize an individual's emotional spectrum, where <u>valence</u> refers to the degree of positiveness of the affect (from negative to positive), <u>arousal</u> to the degree of its intensity, and <u>dominance</u> represents the degree of feeling in control.

In this assignment we focus on the dimension of <u>valence</u>, which spans the continuous range of [-1, 1]. Specifically, we make use of the dataset by Saif Mohammad et al., where over 50K words are manually ranked by multiple annotators for various dimensions – valence among others. For example, the word "fabulous" is ranked high on the valence dimension, while "deceptive" is rated low.

Word representations (embeddings) carry over various aspects of word meaning (including sentiment); there are reasons to believe that a model trained on representations of a set of words (and their valence scores as labels), will generalize to unseen words, being able to rank them with valence scores based on their distributional representations. In this assignment you will explore two types of representations, as discussed during the lecture: (1) long and sparse, and (2) short and dense.

The goal is to train a linear regression model using the train set, infer valence scores to words in the test set, and evaluate the model using standard evaluation metrics (see below).

Detailed description:

Two datasets are attached to this assignment: (1) nrc-valence-scores.trn.csv and (2) nrc-valence-scores.tst.csv. You are also given two files: config.json and main.py. You can change values in config.json, but not the code in the main() of this assignment.

Your task is to implement the function predict_words_valence():

```
def predict_words_valence(train_file, test_file, data_path, is_dense_embedding):
      …

      return mse, corr
```

where `is_dense_embedding` can have one of the two values: true or false.

(1) Generating long and sparse word representations (is_dense_embededing = false):

We will learn these representations from the English Wikipedia dump from assignment #1, using co-occurrence word matrix, as described in the distributional semantics lecture.

Please only consider the 10K most frequent words in the corpus for your co-occurrence matrix (columns), meaning that 10K will also be the word representation vector length. For the matrix rows – consider only the words in the train + test set, you won't need other words, and your solution will be faster. Also, using only the first 7-8M lines from the Wikipedia dump should be good enough and that will help you meeting the runtime requirements.

Finally, try using various window sizes around the word and see what works best.

Comment: word frequency affects counts in co-occurrence vectors -- this may interfere with your model training. Think how can you overcome this issue, for instance by some type of vector normalization.

With reasonably good implementation, you may expect Pearsons corr of around 0.4 for this part.

(2) Generating short and dense word representations (is_dense_embededing = true):

We will experiment with pre-trained word embeddings. Note that the gensim package allows you to easily load and experiment with multiple embedding types. Select the one that yields the highest performance on the test set; no need to experiment with them all, a couple will do.

With reasonably good implementation, you may expect Pearsons corr of over 0.7 for this part.

Regression model:

We fit a linear regression model of the form Y = a + bX, where the feature vector (X) is a word representation (sparse or dense), and the label (Y) is the word's valence score in the training set. The trained model is further used to infer valence scores on the test set.

Evaluation of the results on the test set:

We use two commonly used metrics for evaluation of the predicted scores (y_pred) against the ground truth -- the actual scores (y_true). The first one measures the mean-square-error (MSE) between y_pred and y_true, and the second one measures the correlation between these two.

MSE: the average of the squares of the errors -- the average squared difference between the estimated (y_pred) and the actual (y_true) value, commonly used as a regression loss function.

Pearsons correlation: measures the the strength of the linear relationship between two variables. The value varies between -1 and 1, where correlation higher than 0.4 is commonly considered moderate, and higher than 0.7 – strong. We compute the correlation between y_pred and y_true.

Implementation details and comments:

- Use python's linear regression implementation. Read through the documentation, make sure you can relate the API to the concepts learned at class, and look for usage examples if necessary.
- Evaluation metrics: use python's MSE and Pearson's correlation. You may want to learn more about Pearson's correlation metric in this documentation.
- For faster development runtime cycles, keep your intermediate results in pickle files where possible.
- Make sure your code runtime doesn't exceed 15 minutes (invocation → results are printed).

Submission

Submit a single zip file – assignment_ xxxxxxxxx_xxxxxxxxx.zip , where "xxxxxxxxx" stands for a student id. Please specify two student ids (your and your partner's). It should include two files:

(1) a pdf (convert word to pdf) document with your solution for task #1

(2) your implementation for task #2: main.py

Grading criteria include: correctness (the major part), code design, readability and documentation.

Good Luck!