notebook

April 23, 2019

0.1 Introduction

- Tutorial on basics of natural language processing.
- Presenter: **Milad Alshomary**
- **Task:** Predicting the importance score of a premise in relation to the conclusion.

0.2 Importance Estimation:

- Inspired by Wang and Ling 2016 .
  - Paper: Neural Network-Based Abstract Generation for Opinions and Arguments.
  - Input: Set of premises (each premise is a sentence)
  - Output: Conclusion statment.
- **Importance estimation**: is a component of this approach in which each premise gets a score that reflects its importance in relation to the conclusion.

Example from the dataset:

- Premises:
  - Why should governments allow an activity that helps their citizens lose the money they have worked so hard to earn ?
  - Gamblers may win money from time to time , but in the long run , the House always wins .
  - The internet has made gambling so much easier to do and encouraged lots of new people to place bets so dramatically multiplying the harm .
- Conclusion:
  - Gambling is bad for you.

0.2.1 The Task:

Given a set of premises, generate for each an importance score that reflects its importance in relation to the conclusion.

Steps:

```
1. Data preparation
2. Feature Representation
3. Model Training
4. Evaluation
```

0.2.2 Prepare your environment:

Installing Python:

- On Linux, most likely you have it. Otherwise:

```
xx install python3 # xx is (apt-get, dnf, ...) based on you linux distribution
```

- On Windows/MacOS, download the installer
- Test if python is working:

```
python3 --version
```

- Doesn't work? Google it or simply ask us for help!

Installing Jupyter:

- For writing code and presenting it along with text in an easy and interactive way!
- Install Jupyter via pip command:

```
pip install jupyter
```

- Run Jupyter:

```
jupyter notebook
```

- Access the notebook on under localhost:8888

0.2.3 Important Libraries:

- Any library can be downloaded by using the pip tool:

```
pip install library-name    # library-name = matplotlib, numpy, nltk ....
```

- In this lecture we will be using the following libraries:
  - numpy http://www.numpy.org/
  - scikit-learn https://scikit-learn.org
  - nltk https://www.nltk.org/
  - matplotlib https://matplotlib.org/

0.2.4 The code framework

- The code framework contains:
  - A jupyter notebook `notebook.ipynb` to present the results.
  - A Python file `importance_estimation.py` where you will add your code.
  - A json file called `idebate.json` contains the dataset.

Execute the following cell (`shift+enter`) to initialize an instance of the ImportanceEstimation-Model class (the class resides in `importance_estimation.py` where you will add our code).

```
In [1]: from importance_estimation import ImportanceEstimationModel
```

0.2.5 Data Preparation:

Task: Load data from the json file `idebate.json` and split it based on the `debate_id` field into 80% training and 20% testing. - Implement the `load_data` function (in the `importance_estimation.py` file): - Input: `path` to the data file - Output: `train_dataset` and `test_dataset` where each item is a tuple of a list of premises and a conclusion. - Steps: 1. Using `json` library load the data into a json object. 2. Create a set of all debate ids and use scikit-learn to split this set into train and test. 3. Create the train and test datasets by filtering the data based on the train/test debate ids and only selecting `_argument_sentences` and `_claim` (conclusion) fields.

Initializing the `ImportanceEstimation` model and loading the data:

```
In [2]: model = ImportanceEstimationModel()
        train_data, test_data = model.load_data('./idebate.json')

        print('Number of train arguments:', len(train_data))
        print('Number of test arguments:', len(test_data))
```

```
Number of train arguments: 1805
Number of test arguments: 454
```

Print a random sample of the `train_data`:

```
In [3]: #we only print the first 5 premises from the instance
        model.print_train_sample(train_data)
```

```
Conclusion: The Japanese people do not want the bases on their soil.
Premises:
1 . Without reason to be there , and unwanted by the people , the United States should remove
2 . This is demonstrated in every opinion poll and is reflected in the fact that current rulin
3 . For all of these reasons , the Japanese people have resoundingly stated their desire for t
4 . Most of the soldiers who commit these crimes never see justice since American soldiers sta
5 . The presence of American military personnel is particularly onerous in light of the multit
```

0.2.6 Ground truth scores:

Since we don't have ground truth scores for the premises to reflect their relevancy to the conclusion, we take the token overlap between an argument's premise and its conclusion as a measure to reflect the relevancy.

Task: For each premise compute the token overlap with the conclusion (after excluding stopwords).

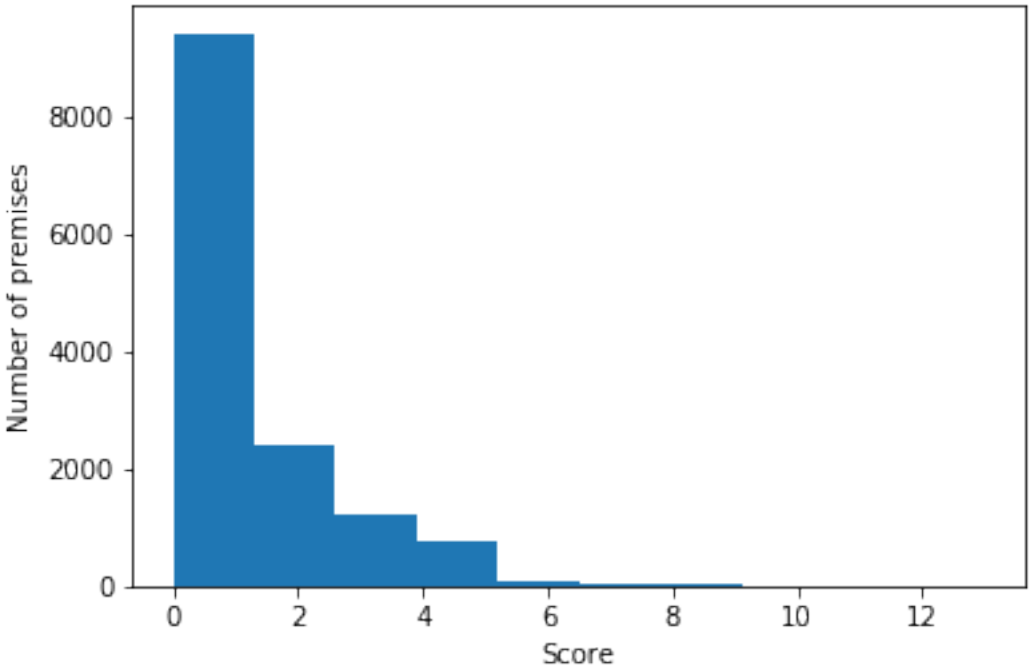
- Implement the `instance_scores` function:
  - Input: A list of premises and the corresponding conclusion.
  - Output: A list of scores (score for each premise).
  - Steps:
    1. Using NLTK library, tokenize each premise as well as the conclusion and remove stopwords (use NLTK stopwords).

2. For each premise compute the token overlap with the conclusion.

Ground truth scores (number of tokens shared between a premise and a conclusion) distribution:

```
In [4]: from   matplotlib import pyplot as plt

        instances_scores = [model.instance_scores(instance[0], instance[1]) for instance in tra
        all_scores = [score for x in instances_scores for score in x]
        plt.hist(all_scores)
        plt.xlabel('Score')
        plt.ylabel('Number of premises')
        plt.show()
```



0.2.7 Feature Representation:

Task: For each premise construct a features vector containing the following features: - Number of words: Implement `_num_of_words_feature` function to return number of words in the claim. You may use the `nltk.word_tokenize` for this. - Avg./Max. tf-idf scores: For this 1. First implement the function `_build_tfidf_model` that builds a tfidf model (use scikit-learn for this) over a corpus of texts. Consider each set of claims (you might concatenate the claims as one string) as one document. 2. Implement the function `_tfidf_features` that uses the tfidf_model to compute for each claim the average tf-idf value of its tokens as well as the maximum tf-idf.

- Number of positive/negative/neutral words: Implement the function `_sentiment_features` that uses `sentiwordnet` lexicon (you might use the implementation by NLTK).

Encoding the `train_data` into features vectors as well as computing the corresponding ground truth scores:

```
In [5]: train_X, train_Y = model.feature_representation(train_data)

        print('train_X shape :', train_X.shape)
        print('train_Y shape :', train_Y.shape)
```

```
train_X shape : (13901, 6)
train_Y shape : (13901,)
```

0.2.8 Model Training:

Task: Train a support vector regression (SVR) model using a grid search (over the cost parameter C) and cross validation of 5. - Implement the function `train_grid_search_svr`: - Input: `train_X` and `train_Y` - Output: `best_svr` the best `SVR` model and `best_score` the mean absolute error of the best `SVR` model. - Steps: 1. Initialize a support vector regression model using Scikit Learn. 2. Initialize a grid search model using Scikit Learn library over the SVR model with a cv (cross validation) equal 5 and `mean absolute error` for scoring. 3. Fit the model on the `train_data` and save the `best_svr` as a property in the `ImportanceEstimation` model.

```
In [7]: best_svr, mean_absolute_error = model.train_grid_search_svr(train_X, train_Y)
        print('Mean absolute error:', mean_absolute_error)
```

```
Mean absolute error: -0.9507702650510839
```

0.2.9 Evaluation:

Task: Evaluate the `best_svr` model by computing the **mean reciprocal rank (MRR)** on the `test_data`. - Implement `mrr_evaluation` function: - Input: `test_data` - Output: `mrr_value` - Steps: 1. For each sample in the `test_data`, predict the scores for its premises using the `best_svr` model and compute the overlap with the conclusion (you may call `instance_score`). 2. Sort the premises according to the predicted scores. 3. Compute the rank considering a relevant premise as the premise that overlap at least with one token with the conclusion.

Computing the `MRR` value on the test dataset:

```
In [8]: model.mrr_evaluation(test_data)
```

```
Out[8]: 0.8242828543214002
```

0.2.10 Extra tasks (optional):

- Think of a new feature, that can be useful in predicting the score of a premise and implement it.
- Perform an ablation study by removing some of the features and check how that affects the model performance in terms of `MRR` score.