Natural Language Processing [CS4120/CS6120] Assignment 2

Instructor: Professor Lu Wang

Deadline: March 20th at 11:59pm on Blackboard

For the programming questions, you can use Python (preferred), Java, or C/C++. You also need to include a README file with detailed instructions on how to run the code and commands. Failure to provide a README will result in a deduction of points. **Violation of the academic integrity policy is strictly prohibited, and any plausible case will be reported once found**. This assignment has to be done individually. If you discuss the solution with others, you should indicate their names in your submission.

1 Parsing [14 points]

Use Version 3.8.0 or higher of the Stanford parser: `https://nlp.stanford.edu/software/lex-parser.shtml#History`, to parse the following corpus:
`https://www.dropbox.com/s/sqak9125n3uw044/corpus.zip?dl=0`
For sentences containing 50 words or less (punctuation does not count), obtain the part-of-speech tags, the context-free phrase structure grammar representation, and the typed dependency representation, as shown in the following sample output: `https://nlp.stanford.edu/software/lex-parser.shtml#Sample`

Submit your code and the output.

1.1 [2 points]

Using the part-of-speech tags that the parser has given you, report the average number of verbs per sentence in the overall corpus (i.e. total number of verbs divided by total number of sentences in the corpus). Also report the part-of-speech tags that you are using to identify the verbs.

1.2 [2 points]

Report the number of sentences parsed; do so by searching for "ROOT" in either the dependency representation or in the context-free phrase structure grammar representation.

1.3 [4 points]

Using the dependency representation (or the context-free phrase structure grammar representation) that the parser has given you, report the total number of prepositions found in each file. In addition, report the most common three preposition overall.

1.4 [6 points]

Take a look at the constituent parsing and dependency parsing results. List out two common errors made in each type of parsing results, and briefly discuss potential methods to reduce each type of error.

2 CKY parser (10 points)

Using the rules and probabilities below, draw a probabilistic CKY chart for the following sentence: **"A school boy bought the ancient coin"**:

1. Det → a 0.3
2. Det → the 0.7
3. NP → ancient 0.1
4. NP → school 0.2
5. NP → boy 0.2
6. NP → coin 0.1
7. VP → bought 0.3
8. VP → coin 0.1
9. Adj → ancient 0.7
10. Adj → school 0.3
11. N → ancient 0.2
12. N → school 0.2
13. N → boy 0.4
14. N → coin 0.2
15. V → coin 0.2
16. V → bought 0.8
17. S → DP VP 1.0

18. DP → Det NP 0.5
19. DP → Adj NP 0.3
20. DP→ N N 0.2
21. NP → N N 0.1
22. NP → Adj NP 0.3
23. VP → V DP 0.6

3 Sentiment Analysis [40 points]

The following training data contains 1133 IMDb movie reviews, where each review is rated either -1 or +1 (-1 - negative, +1 - positive). Reviews for training dataset has already been stored in the corresponding folders (i.e. '/neg' for negative reviews and '/pos' for positive reviews).

The training data is available at: `https://www.dropbox.com/s/o8u0jgmkvyp4s5x/train.zip?dl=0`

You can use the following tools for implementation:

1. TensorFlow `https://www.tensorflow.org/`
2. Scikit-learn `http://scikit-learn.org/stable/index.html`
3. PyTorch `http://pytorch.org/`

In addition to output and results, please submit your code along with a README file explaining how to run your code.

3.1 [10 points]

Train a Multilayer Perceptron to predict sentiment score . Using unigram features as input, call the training and testing functions for the Multilayer Perceptron from the tool. You do not need to implement the learning (i.e., back-propagation) algorithm. You should have an input layer, two hidden layers, and an output layer; the second hidden layer should have 10 nodes. Use 10-fold cross-validation to optimize any parameters (e.g. activation function or number of nodes in the first hidden layer). Use accuracy as the metric for parameter selection. Describe your parameter optimization process, and report the parameters for your best model.

3.2 [5 points]

Using the parameters for the best performing model from 3.1, re-train it on the whole training set, and report the accuracy on the **training** set.

3.3 [10 points]

Use pre-trained word embeddings GoogleNews-vectors-negative300.bin.gz from Word2vec `https://code.google.com/archive/p/word2vec/`, and compute the review feature vector by using the average word embeddings. Do the same tasks as described in 3.1: Use 10-fold cross-validation to optimize any parameters (e.g. activation function or number of nodes in the first hidden layer). Use accuracy as the metric for parameter selection. Report the parameters for your best model. Then re-train the best performing model on the whole training set, and report the accuracy on the **training** set.

3.4 [5 points]

In addition to the word embeddings, add one type of features by your own design (e.g. POS tags distribution) to the model in 3.3. Then re-train this model on the whole training set, and report the accuracy on the **training** set.

3.5 [10 points]

Using the best model from above (based on results from 3.2, 3.3., and 3.4), predict the sentiment scores for all 267 reviews in this test set: `https://www.dropbox.com/s/qbahy541nmriv4r/test.zip?dl=0`
Classifying the reviews into two categories based on your predicted sentiment score, as shown in the training data. Submitting two text files, "pos.txt" with a list of file names where their reviews are predicted as positive, similarly for "neg.txt" with file names for reviews predicted as negative. Put each file name in a separate line. Submit the zip file of these two text files and name it as labels.zip. (For grading purpose, please DO NOT change the file names of the reviews.)

4 Summary Evaluation [36 points]

In this question, you will design classifiers to evaluate summary quality based on its *non-redundancy* and *fluency*. You will be given a training data and a test data, both in the following format:

1. Column 1: summary for a news article
2. Column 2: non-redundancy score [a score that indicates the conciseness of the summary]
   Non-redundancy scores range from -1 [highly redundant] to 1 [no redundancy].
3. Column 3: fluency [a score that indicates whether a sentence is grammatically correct in the summary]
   Fluency can range from -1 [grammatically poor] to 1 [fluent and grammatically correct].

For further discussion on the two aspects, please refer to https://duc.nist.gov/pubs/2006papers/duc2006.pdf, see Section 3.1 on "Grammaticality" and "Non-Redundancy".

Your task is to build classifiers of your own choice (e.g. Support vector regression, linear regression, neural networks, or other classifiers) on the training dataset to predict the non-redundancy and fluency of the summaries in the test dataset.

Here's the datasets:
Training set: https://drive.google.com/file/d/1c5n6hc8prmiY9F7dlrPmSx6OpmIiSRg2/view?usp=sharing
Test set https://drive.google.com/file/d/1ljBSAoP7tmsFLg3RsqtcWzHvSN1aWE2k/view?usp=sharing

In addition to output and results, please submit your code along with a README file explaining how to run your code.

4.1 Building Non-Redundancy Scorers

4.1.1 [12 points]

Train your classifier with the following three features on training data, with summary as input and "non-redundancy" as gold-standard scores, and report evaluation performance on test data. For evaluation, please use metrics of Mean Squared Error (MSE) and Pearson correlation, both calculated between your classifier's predictions and gold-standard scores of samples in the test data.
Each feature implementation is 3 points, and building classifiers is 3 points.

1. Maximum repetition of unigrams: calculate the frequencies of all unigrams (remove stop words), and use the maximum value as the feature.

2. Maximum repetition of bigrams: calculate the frequencies of all bigrams, and use the maximum value as the feature.

3. Maximum sentence similarity: each sentence is represented as average of word embeddings, then compute cosine similarity between pairwise sentences, use the maximum similarity as the features. Use word embeddings GoogleNews-vectors-negative300.bin.gz from Word2vechttps://code.google.com/archive/p/word2vec/ as input for each word. Words in a summary that are not covered by Word2vec should be discarded.

4.1.2 [6 points]

Design two new features for this task. Add each feature to the classifier built in 4.1.1, and report MSE and Pearson correlation. At least one of your proposed features should get better MSE AND Pearson. Take a look at the training samples and explain why your features can improve your classifier's performance.

4.2 Building Fluency Scorers

4.2.1 [12 points]

Train your classifier with the following three features on training data, with summary as input and "fluency" as gold-standard scores, and report evaluation performance on test data. For evaluation, please use metrics of Mean Squared Error (MSE) and Pearson correlation, both calculated between your classifier's predictions and gold-standard scores of samples in the test data.
Each feature implementation is 3 points, and building classifiers is 3 points.

1. Total number of repetitive unigrams: count how many unigrams are the same as the previous unigrams. For example, for a summary "**The the** article **talks talks** about language understanding", the value should be 2.

2. Total number of repetitive bigrams: count how many bigrams are the same as the previous bigrams. For example, for a summary "**The article the article** talks about about language understanding", the value should be 1.

3. Minimum Flesch reading-ease score: use tool from `https://pypi.org/project/readability/` to get readability score for each sentence, and use the minimum value as the feature.

4.2.2 [6 points]

Design two new features for this task. Add each feature to the classifier built in 4.2.1, and report MSE and Pearson correlation. At least one of your proposed features should get better MSE AND Pearson. Take a look at the training samples and explain why your features can improve your classifier's performance.

Here are the instructions for preprocessing

1. Replace tab or new line characters with space

2. Lowercase words

3. Remove extra spaces

4. If you want to use a stop word list, it can be found in NLTK.

5. Mean squared error is the error of the results obtained from your classifier compared to the true values of the labels.
   The output ranges from 0 to 1 where

   (a) 0 indicates that your predicted values match the gold-standards perfectly.

   (b) 1 indicates that your prediction are not very accurate.

   To know more, visit: `https://en.wikipedia.org/wiki/Mean_squared_error#Interpretation`
   You can use sklearn package to compute the mean squared error: `http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html`

6. Pearson Correlation Coefficient calculates the correlation between the predicted values and the gold-standards.
   The result ranges from -1 to 1 where

   (a) 1 indicates that your predicted values positively correlate with the gold-standards perfectly.

   (b) 0 indicates that there is no correlation between predictions and the gold-standards.

   (c) -1 indicates that your predicted values negatively correlate with the gold-standards perfectly.

   To learn more, visit: `https://en.wikipedia.org/wiki/Pearson_correlation_coefficient#Interpretation`
   You can use sklearn package to compute the pearson coefficient: `https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.stats.pearsonr.html`