Natural Language Processing [CS4120/CS6120] Assignment 2

Instructor: Professor Lu Wang

Deadline: November 12th, 2019 at 11:59pm on Blackboard

For the programming questions, you can use Python (preferred), Java, or C/C++. Please include a README file with detailed instructions on how to run your code. Failure to provide a README file will result in **deduction of points** (5 to 10 points per problem). Your final deliverable for this homework should consist of one zipped folder with i) one text file for each of the four problems with your answers for questions requiring textual answers (name each file as instructed in each question) ii) zipped folders with code (one folder per problem), and iii) README describing how to run your code for each of the programming problem.

This assignment has to be done individually. If you discuss the solution with others, you should indicate their names in your submission. If you use ideas/code from any online forums, you should cite them in your solutions. **Violation of the academic integrity policy is strictly prohibited, and any plausible case will be reported once found. No exceptions will be made.**

1 Parsing [14 points]

In class, we explored several parsing algorithms for natural language. Now, you will apply some of those to parse a given corpus. For this problem, you will use Spacy: https://spacy.io/, to parse the following corpus:
http://bit.ly/2lWhdzl
Combine the text from all the given files into one raw text variable. Next, process this raw text using spacy to obtain the part-of-speech tags, dependency tree representation and named entities.

Report your answers in file named Q1.txt. Please submit your code along with a README file explaining how to run your code.

1.1 [2 points]

Report the number of sentences parsed; do so by searching for "ROOT" in the dependency representation or use the sentence segmentation method of spacy parser.

1.2 [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). Use the total number of sentences from 1.1. Also report the part-of-speech tags that you are using to identify the verbs.

1.3 [3 points]

Using the dependency tree representation that the parser has given you, report the total number of prepositions found in the overall corpus. In addition, report top three most commonly used preposition.

1.4 [3 points]

Using the named entity recognition method of the parser, report the total number of entities found in the overall corpus. Also, list out all the unique entity labels found in the corpus (e.g. PERSON, ORG etc.).

1.5 [4 points]

Take a look at the 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]

As shown in class, CKY is a parsing algorithm for context-free grammars which employs bottom-up parsing and dynamic programming. Probabilistic CKY allows to recover the most probable parse tree given the probabilities of all productions. Using the rules and probabilities given below, draw a probabilistic CKY chart for the sentence : "beautiful girl dances in a park". Please submit your answer in the format as shown in lecture slides in file named Q2.txt .

- $Det \rightarrow a\ 1.0$
- $NP \rightarrow beautiful\ 0.1$
- $NP \rightarrow girl\ 0.2$
- $NP \rightarrow park\ 0.1$
- $NP \rightarrow dances\ 0.2$
- $NP \rightarrow NP\ VP\ 0.2$
- $NP \rightarrow NP\ P\ 0.1$
- $NP \rightarrow NP\ PP\ 0.1$

- $Adj \rightarrow beautiful$ 1.0
- $V \rightarrow park$ 0.2
- $V \rightarrow dances$ 0.8
- $VP \rightarrow park$ 0.2
- $VP \rightarrow dances$ 0.2
- $VP \rightarrow V\ DP$ 0.4
- $VP \rightarrow VP\ P$ 0.2
- $P \rightarrow in$ 1.0
- $PP \rightarrow P\ Det$ 1.0
- $S \rightarrow DP\ NP$ 1.0
- $DP \rightarrow Det\ NP$ 0.3
- $DP \rightarrow Adj\ NP$ 0.3
- $DP \rightarrow N\ N$ 0.1
- $DP \rightarrow PP\ DP$ 0.2
- $DP \rightarrow P\ DP$ 0.1

3 Sentiment Analysis [40 points]

Sentiment analysis refers to the use of natural language processing to systematically identify, extract, quantify, and study affective states and subjective information. As shown in class it is widely applied to reviews and survey responses, and social media content for applications that range from marketing to customer service. In this problem, we will explore a basic task in sentiment analysis, which is classifying the polarity of a given text to be positive or negative.
For this problem, we will use the training data containing 10,000 Yelp restaurant reviews, where each review is labelled based on their ratings. Reviews for training dataset are in the corresponding folders ('/neg' for negative reviews and '/pos' for positive reviews). Label the negative sentiment reviews from the /neg folder as 1 and the positive sentiment reviews from the /pos folder as 0, these 1 and 0 labels will be your target labels for sentiment analysis.

The training data is available at: http://bit.ly/2nuzyUM

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, results and your answers (in a file named Q3.txt), please submit your code along with a README file explaining how to run your code.

3.1 [10 points]

Train a Multilayer Perceptron (MLP) to predict sentiment score using unigram features (e.g. word counts or one hot encoding of words) as input. You can use the training and testing functions for the MLP from the tool of your choice. You are not required to implement the learning algorithm (i.e., back-propagation) for this problem. Your MLP 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 of your best model.

Note: Implement the 10-fold cross validation and unigram features without using any library.

3.2 [2 points]

Using the parameters for the best performing model from 3.1, re-train the model on the whole training set, and report the accuracy on the same **training** set. (This tells you how the selected model performs on the full training data.)

3.3 [8 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 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 of your best model. Then re-train the best performing model on the whole training set, and report the accuracy on the same **training** set.

3.4 [10 points]

Compute training set's review feature vector: i) first generate TF-IDF, and ii) then apply Singular value decomposition (SVD) on it. Do the same as described in 3.1: Use 10-fold cross-validation to optimize the SVD's **number of components** and model's 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 of your best model. Then re-train the best performing model on the whole training set, and report the accuracy on the **training** set.

Hint: You can use the SVD implementation of Sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html. The *n_components* parameter is the **number of components**.

3.5 [5 points]

Using the optimal number of components you found in 3.4, apply TF-IDF and SVD on the whole training set. Report the top 5 topics and the 20 most important words in each topic.

Hint: Each topic is the component you get from SVD. To get top 5 components: i) first sort the singular values in decreasing order (use the singular value function from sklearn's SVD), ii) then, pick the first five components corresponding to those sorted singular values.

3.6 [5 points]

Using the best model from above (based on results from 3.2, 3.3., and 3.4), predict the sentiment scores for all 495 reviews in this test set: http://bit.ly/2m0Hxsg
Classify the reviews into two categories based on your predicted sentiment score, as shown in the training data. Submit two text files: i) "pos.txt", with a list of file names for reviews predicted as positive, ii) "neg.txt", with a list of file names for reviews predicted as negative. Put each file name in a separate line. Submit a folder of these two text files and name the folder as **labels**. (For grading purpose, please DO NOT change the file names of the reviews.)

4 Summary Evaluation [36 points]

A summary is usually a brief overview of a longer document. A good summary is supposed to be grammatically correct, non-redundant and coherent. We define these three properties below.

Grammaticality A grammatically correct summary should have no datelines, system-internal formatting, capitalization errors or obviously ungrammatical sentences (e.g., fragments, missing components) that make the text difficult to read. For this problem, grammaticality score can range from -1 [grammatically poor] to 1 [grammatically correct].

Non-redundancy A non-redundant summary should have no unnecessary repetition in the summary, which might take the form of whole sentences that are repeated, or repeated facts, or the repeated use of a noun or noun phrase (e.g., Bill Clinton) when a pronoun (he) would suffice. For this problem, non-redundancy scores can range from -1 [highly redundant] and 1 [no redundancy].

Coherence A coherent summary should be well-structured and well-organized. The summary should not just be a heap of related information, but should build from sentence to sentence to a coherent body of information about a topic or entity. For this problem, coherence score can range from -1 [not coherent] to 1 [highly coherent].

In this question, you will design classifiers to evaluate the summary quality based on aforementioned qualities. You will be given a training set and a test set, both in json files. Here's the link to the data: https://www.dropbox.com/s/c9kyap6xlqn86v2/summary_quality.zip?dl=0. The zipped folder contains the following:

- *summaries*: This folder contains all (training and test) the sentence-segmented summaries (each line in a file is one sentence).

- *train_data.json*: This json file contains 1737 training instances in form of dictionary, where key is the file name of the summary and value is a dictionary of all three scores. [You may want to further divide the training set into training and validation set for your classifiers].
- *test_data.json*: This json file contains 193 test instances in form of dictionary, where key is the file name of the summary and value is a dictionary of all three scores.
- *readData.py* : This is sample code to read the training or test data.

You task is to build classifiers of your own choice (e.g. Support vector regression, logistic regression, neural network, or other classifiers) on the training dataset to predict the grammaticality, non-redundancy and coherence of the summaries in the test dataset.

In addition to the output and results for each question (named Q4.txt), please submit you code along with a README file explaining how to run your code.

4.1 Building Grammaticality Scorer

4.1.1 [9 points]

Train your classifier with the following three features on the training data, with summary as input and "grammaticality score" as the gold label, and report the performance of your classifier on the test data. For evaluation, please report Mean Squared Error (MSE) and Pearson correlation, both calculated between your predicted and gold labels (scores) for each sample in the test data.

1. *Total number of repetitive unigrams*: count how many unigrams were repeated in a given summary. For instance, for a summary "**The the** article **talks talks** about language understanding", the feature value should be 2.
2. *Total number of repetitive bigrams*: count how many bigrams were repeated in a given summary. For instance, for a summary "**The article the article** talks about language understanding", the feature 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.

4.1.2 [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 the classifier's performance.

4.2 Building Non-Redundancy Scorer

4.2.1 [9 points]

Train your classifier with the following three features on the training data, with summary as input and "non-redundancy score" as the gold label, and report the performance of your classifier on the test data. For evaluation, please report Mean Squared Error (MSE) and Pearson correlation, both calculated between your predicted and gold labels(scores) for each sample in the test data.

1. *Maximum repetition of unigrams*: calculate the frequencies of all the unigrams (remove stop words), and use the maximum value as the feature value.
2. *Maximum repetition of bigrams*: calculate the frequencies of all the bigrams, and use the maximum value as the feature value.
3. *Maximum sentence similarity*: represent each sentence as average of its word embedding, then compute cosine similarity between pairwise sentences, use the maximum similarity as the features. Use word embeddings GoogleNews-vectors-negative300.bin.gz from Word2Vec : https://code.google.com/archive/p/word2vec/ as input for each word. Words in a summary that are not covered by Word2Vec should be discarded.

4.2.2 [6 points]

Again, 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 the classifier's performance. Please do not repeat the features from 4.1.2.

4.3 Building Coherence Scorer

4.3.1 [6 points]

Train your classifier with the following two features on the training data, with summary as input and "coherence score" as the gold label, and report the performance of your classifier on the test data. For evaluation, please report Mean Squared Error (MSE) and Pearson correlation, both calculated between your predicted and gold labels (scores) for each sample in the test data.

1. *Total number of repetitive noun phrases*: count how many noun phrases were repeated in a given summary. You can consider phrases that have a noun as their head in the dependency parse tree representation of the given text. In the sentence "Autonomous cars shift insurance liability toward manufacturers", "Autonomous cars", "insurance liability" and "manufacturers" are the noun phrases. You can use spacy to extract the noun phrases (https://spacy.io/usage/linguistic-features#noun-chunks).
2. *Total number of coreferred entities*: count how many entities or textual units are coreferred in a given summary.

Coreference occurs when two or more expressions in a text refer to the same person or thing; they have the same referent, e.g. "Bill said he would come"; the proper noun "Bill" and the pronoun "he" refer to the same person, namely to "Bill". When two expressions are coreferential, one is usually a full form (the antecedent) and the other is an abbreviated form (a proform or anaphor). Coreference resolution is the task of correctly matching the antecedent with its referent. For sentence, "My sister has a dog. She loves him.", here is the result of spacy's coreference resolution: https://bit.ly/2nm8RBG [Try it here https://bit.ly/2meo8UV if the link doesn't work.] You can use NeuralCoref (https://github.com/huggingface/neuralcoref) to do coreference resolution and get the total number of coreferred entities for constructing your feature.

Hints

1. Tokenize using spacy since you will be using spacy for 4.3. https://spacy.io/usage/linguistic-features

2. Lowercase words (Exception: For Coherence scorer since you require to run a parser on your summaries, you will need to use non-lowercased version of your summaries.)

3. Remove extra spaces

4. If you want to use a stop word list, spacy has one.

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