CSCI544, Spring 2020: Assignment 2

Due Date: April 13th, before midnight

Introduction

The goal of this assignment is to get some experience with sequence labeling. Specifically, you will be assigning dialogue acts to sequences of utterances in conversations from a corpus. In sequence labeling it is often beneficial to optimize the tags assigned to the sequence as a whole rather than treating each tag decision separately. With this in mind, you will be using a machine learning technique, conditional random fields, which is designed for sequence labeling. You will be using the toolkit, CRFsuite.

The raw data for each utterance in the conversation consists of the speaker name, the tokens and their part of speech tags. Given a labeled set of data, you will first create a baseline set of features as specified below, and measure the accuracy of the CRFsuite model created using those features. You will then experiment with additional features in an attempt to improve performance. The best set of features you develop will be called the advanced feature set. You will measure the accuracy of the CRFsuite model created using those features. Your programs should also be able to assign dialogue act tags to unlabeled data. We will use your code to evaluate your baseline and advanced features on unseen test data.

Data

The Switchboard (SWBD) corpus was collected from volunteers and consists of two person telephone conversations about predetermined topics such as child care. SWBD DAMSL refers to a set of dialogue act annotations made to this data. This (lengthy) annotation manual defines what these dialogue acts mean. In particular, see section 1c (The 42 Clustered SWBD-DAMSL Labels). Note, the statistics in this manual use a different training set than our experiments here but give you a rough idea of how frequently each dialogue act appears. We recommend that you skim the annotation manual to get an understanding of what the tags mean and help you think of good features.

If you have a question about the corpus and can't find an answer in the annotation manual, please post on Piazza or ask a TA or instructor. **DO NOT use resources from the web other than the annotation manual.**

Download a zip file from Blackboard of SWBD dialogues labeled with SWBD-DAMSL dialogue acts. Individual conversations are stored as individual CSV files. These CSV files have four columns and each row represents a single utterance in the conversation. The order of the utterances is the same order in which they were spoken. The columns are:

- act_tag - the dialogue act associated with this utterance. Note, this will be blank for the unlabeled test data we use to test your code.

- speaker - the speaker of the utterance (A or B).
- pos - a whitespace-separated list where each item is a token, "/", and a part of speech tag (e.g., "What/WP are/VBP your/PRP$ favorite/JJ programs/NNS ?/."). When the utterance has no words (e.g., the transcriber is describing some kind of noise), the pos column may be blank, consist solely of "./.", have a pos but no token, or have an invented token such as MUMBLEx. You can view the text column to see the original transcription.
- text - The transcript of the utterance with some cleanup but mostly unprocessed and untokenized. This column may or may not be a useful source of features when the utterance solely consists of some kind of noise.

You will also notice a Python file **hw2_corpus_tool.py** included with the data. We have written this code to help you read the data. You should use this code to ensure that you will be reading the test files in the proper order. You should include the code with your submission to ensure that it imports smoothly when we grade your programs. Read the documentation in the file for more information or use Python's help() function.

CRFsuite

You will need to install pycrfsuite (https://pypi.python.org/pypi/python-crfsuite), a Python interface to CRFsuite (http://www.chokkan.org/software/crfsuite/). As discussed in the CRFsuite tutorial, and the pycrfsuite tutorial, you add training data to the **Trainer** object using the **append** method which takes two arguments (feature_vector_list,label_list) and loads the training data for a single sequence. In our case, each sequence corresponds to a dialogue, and the feature_vector_list is a list of feature vectors (one for each utterance in the dialogue). The label_list corresponds to the dialogue acts for those utterances. Each feature vector is a list of individual features which are binary. The presence of a feature indicates that it is true for this item. Absence indicates that the feature would be false. Here are the features for a training example using features for whether a particular token is present or not in an utterance.

```
['TOKEN_i', 'TOKEN_certainly', 'TOKEN_do', 'TOKEN_.']
```

After loading the training data, you need to set the CRFsuite training parameters. The following parameters are taken from the pycrfsuite tutorial. You are not expected to optimize these hyper parameters. The official solution will use these hyper parameters so it is safer to use them given the unseen test data. The max_iterations parameter is particularly important to keep training times reasonable. We will talk more about conditional random fields and these parameters in class.

```
trainer.set_params({
    'c1': 1.0,   # coefficient for L1 penalty
    'c2': 1e-3,  # coefficient for L2 penalty
    'max_iterations': 50,  # stop earlier

    # include transitions that are possible, but not observed
    'feature.possible_transitions': True
})
```

The last step is to train the model using the **train** method which takes a single argument, the name of the file in which to save the model. As discussed below, the two programs you will submit will both train models and use them to tag data. Thus, you can give the model whatever name you like because you will be creating a **Tagger** object and using its **open** method to read the model. The **tag** method of the Tagger object processes a single sequence at a time (i.e., one dialogue) represented as a list of feature value vectors (i.e., one per utterance) in the same format used by the Trainer object. The tagger will output a list of labels (i.e., dialogue acts): one per utterance. **You will need to print them one per line with a blank line separating sequences/dialogues.** It is okay to print a blank line after the last dialogue.

For labeled data, you will also be able to generate a true label list for the utterances of each dialogue, compare to the tagger output and at the end print an accuracy score which you will need to complete your report. However, your programs should be robust and handle unlabeled data (i.e., skip calculating and printing accuracy but continue to output the label list generated by the tagger).

What to do

You will be writing two dialogue act taggers for SWBD DAMSL. You will use your labeled data to debug them and pick the best features for the “advanced” tagger. You could simply split the labeled data by randomly putting roughly 25% of the data in the development set and using the rest to train your classifier. In this case, you would include entire conversations in either the training or development sets. In this assignment, it is up to you how you use your labeled data to evaluate different features. You could use a certain percentage of conversations for development, or you could use k-fold cross-validation.

You should try a set of features that we'll call baseline. In the baseline feature set, for each utterance you include:

- a feature for whether or not the speaker has changed in comparison with the previous utterance.
- a feature marking the first utterance of the dialogue.
- a feature for every token in the utterance (see the description of CRFsuite for an example).
- a feature for every part of speech tag in the utterance (e.g., POS_PRP POS_RB POS_VBP POS_.).

You'll need to create a Python program (**baseline_tagger.py**) that reads in a directory of CSV files (INPUTDIR), trains a CRFsuite model, tags the CSV files in (TESTDIR), and prints the output labels to OUTPUTFILE

```
>python3 baseline_tagger.py INPUTDIR TESTDIR OUTPUTFILE
```

You should try at least one other set of features that we'll call advanced. The advanced feature set should include more information than the baseline feature set. The idea is that you want to improve performance. As discussed in the grading section, part of your grade depends on developing a feature set better than the baseline. You'll need to create a Python program (**advanced_tagger.py**) that reads in a directory of CSV files (INPUTDIR), trains a CRFsuite model using the advanced features, tags the CSV files in (TESTDIR), and prints the output labels to OUTPUTFILE

```
>python3 advanced_tagger.py INPUTDIR TESTDIR OUTPUTFILE
```

Your programs, baseline_tagger.py and advanced_tagger.py, will need to be able to tag CSV files with blank labels as well as labeled development data.

What to turn in and Grading

You need to submit the following on Vocareum:

- **SWBD-DAMSL-Report.txt**.
    - Download and fill in the template from the class website. Include the results from your experiments. Keep the format text or you can use PDF.
        1. Describe how you used the labeled data to evaluate the features (e.g., moved 25% of labeled data into development directory).
        2. Accuracy of baseline features during your evaluation
        3. Describe your advanced features
        4. Describe any advanced features you tried and rejected
        5. Accuracy of advanced features during your evaluation
    - Make sure that none of your code writes to a file called SWBD-DAMSL-Report.txt. We don't want to accidently damage the file when we test your code.
    - You can round off the values to 2 decimal points, but rounding is not mandatory.
- **All your code**: baseline_tagger.py, advanced_tagger.py, and any necessary support code including hw2_corpus_tool.py (if not directly added to your code). This is required and the academic honesty policy (see rules below) applies to all your code. Do not submit data.

10% of your grade (10 points) is based on the report. Make sure to answer every question. 80% of the grade is based on performance on tagging unseen data using your baseline features. We

will develop a specific grading rubric for assigning partial credit to cases where performance does not match the TAs' baseline model. 10% of the grade is based on performance on tagging unseen data using your advanced features. This 10% will be assigned by comparing your performance to that of the model developed by the TAs. We will develop a specific grading rubric for assigning partial credit to cases where performance does not match the TA's model.

Each time you submit your code, Vocareum will run your baseline_tagger.py and advanced_tagger.py programs on a small set of data to ensure that they work. So submit early to check your code.

Late Policy

- On time (no penalty): before April 13th, before midnight.
- One day late (10% penalty): before April 14th, before midnight.
- Two days late (20% penalty): before April 15th, before midnight.
- Three days late (30% penalty): before April 16th, before midnight.
- Zero credit after April 16th.

Other Rules

- DO NOT look for any kind of help on the web outside of the SWBD-DAMSL annotation manual, and documentation for Python, pycrfsuite and CRFsuite.
- DO NOT use any external libraries other than the default Python libraries, pycrfsuite and hw2_corpus_tool.py.
- Questions should go to Piazza first not email. This lets any of the TAs or instructors answer, and lets all students see the answer.
- When posting to Piazza, please check first that your question has not been answered in another thread.
- DO NOT wait until the last minute to work on the assignment. You may get stuck and last minute Piazza posts may not be answered in time.
- This is an individual assignment. DO NOT work in teams or collaborate with others. You must be the sole author of 100% of the code and writing that you turn in.
- DO NOT use code you find online or anywhere else.
- DO NOT turn in material you did not create other than hw2_corpus_tool.py
- All cases of cheating or academic dishonesty will be dealt with according to University policy.

FAQ

1. How will the TAs/Professors grade my HW?

We will be writing scripts which will run on Vocareum. Since we automate the process of grading, make sure that you write your code to match the output format "exactly". If you do not do this there will be a penalty.

2. I found a piece of code on the web. Is it ok to use it?

No! We run plagiarism checks on the code you write. The plagiarism detector is a smart algorithm which can tell if you've copied the code from elsewhere/others. Instead please contact TAs/Professor during the office hours with your questions.

3. Vocareum terminates my code before it finishes, but it finishes executing on my computer.

This usually means that your implementation is inefficient. Keep an eye out for places where you iterate and write to files. Run time issues can be solved by using efficient data structures (Hashmap, Hashset etc.).

4. My code works on my computer but not on Vocareum. Do I get an extension?

The late policy above still applies, and is automatically enforced by the system. Submit your code incrementally to make sure it functions on Vocareum.

5. When are the office hours ?

Please refer to the course website. http://nld.ict.usc.edu/cs544-spring2020/