BANA 290: Machine Learning for Text: Spring 2018

Homework 2: Sentiment Classifier

Conal Sathi and Sameer Singh (with help from Yoshitomo Matsubara)
https://canvas.eee.uci.edu/courses/9097

The second programming assignment will familiarize you with the use of machine learning for text classification, and the use of prebuilt lexicons and bag of words to build custom features. The submissions are due by midnight on **May 11, 2018**.

Task: Sentiment Classification

The primary objective for the assignment is same as the first assignment: to predict the sentiment of a movie review. We will be providing you with the dataset containing the text of the movie reviews from IMDB, and for each review, you have to predict whether the review is positive or negative.

Data: The data is available on the Kaggle website[1]. **Please re-download the archive, since it is slightly different**. The file is called `data.zip`, which contains similar structure as the data for the first homework.

Kaggle As with HW1, you can make at most *three* submissions each day, so we encourage you to test your submission files early, and observe the performance of your system. By the end of the submission period, you will have to select the two submissions, one for *default* and one for *custom* (more on this later).

Source Code Code is available for you to develop your submission, at https://canvas.eee.uci.edu/courses/9097/assignments/182967. In order to run the code, unzip `data.zip` to a local folder, say `data/`, and execute:

```
1  python3 ml_sentiment.py data
```

You can also use PyCharm, etc. The file `ml_sentiment.py` loads the data and lexicons, trains the classifiers, and evaluates and saves the results to files (including the CSV file that you need to submit to Kaggle).

Default Features

We are providing code for training a machine learning classifier for sentiment classification using unigrams as features, i.e. `CountVectorizer()`. The first goal is to optimize the hyper-parameters of logistic regression, by modifying `get_tuned_lr`. The regularization weight C is the primary hyper-parameter for logistic regression. Currently, the range for the parameter is `np.arange(0.5, 3.5, 0.5)` but this should be modified. Uncomment:

```
66 tuned_lr = get_tuned_lr(train_data, dev_data, CountVectorizer())
```

to train the classifier. When running this function, you will see both training and dev accuracy printed. There will also be a plot `lr.png` that will be saved that you can view. Based on what these, adjust the range for C.

The next goal is to optimize the parameters for random forest. To do so, you'll need to modify `get_tuned_rf`. `n_estimators` is the parameter of interest used by random forest. Currently, the range is set to `np.arange(5, 35, 5)` but this should also be modified. Uncomment:

```
67 tuned_rf = get_tuned_rf(train_data, dev_data, CountVectorizer())
```

to train the random forest classifier. Like before, when running this function, you will see both training accuracy and dev accuracy, and the plot `rf.png` will be saved. Based on what you see, adjust the parameters accordingly.

Once you are satisfied with the parameters for both the classifiers, run the save function as you see in lines 73-74 of `ml_sentiment.py`. Running `save(tuned_lr, filedir, 'lr_default')` will save the classifier as `lr_default.pkl` which you will need for your error analysis. It will also run the classifier on the test set and save the results as `lr_default_test.csv` which you can upload to Kaggle. Similarly, the next line will output the files `rf_default.pkl` and `rf_default_test.csv`.

[1] https://www.kaggle.com/c/sentiment-analysis-sp18/

Custom Features

The next goal for the assignment is to improved upon these classifiers by introducing your own features. Similar to the in-class activity in week 4, you will design features that utilize lexicons and regular expressions, etc. and experiment with the vectorizer (e.g. unigrams vs. bigrams, counts vs. TF-IDF, etc). To implement these features, you will need to modify the `features` function in `custom_features.py` file, and you can change the vectorizer in `custom_vectorizer`.

As before, tune the parameters for both logistic regression and random forest, but this time with your custom features. To do so, uncomment the following lines in `ml_sentiment.py`:

```
79 tuned_lr = get_tuned_lr(train_data, dev_data, get_custom_features())
80 tuned_rf = get_tuned_rf(train_data, dev_data, get_custom_features())
```

And then run the save methods to save the classifiers (`lr_custom.pkl` and `rf_custom.pkl`) and predictions (`lr_custom_text.csv` and `rf_custom_text.csv`), and upload the latter files to Kaggle.

Analysis

Along with tuning classifiers and designing features to achieve as high of an accuracy as possible, you also need to perform analysis of the classifier in this assignment. For this analysis, open up the `error_analysis.ipynb` python notebook and modify it to get metrics and feature weights for all four of your best classifiers (note, it's currently set up to only compute metrics for two, so you will need to either modify it to get metrics for all four or run it twice). You may need to update `load_classifier('lr_default.pkl')` replacing `lr_default.pkl` with the model you have (the one that was saved when you run the save method earlier).

The rest of the notebook also contains the two primary approaches for analysis that use the `eli5` package: (1) global importance weights of individual classifiers, and (2) weights of individual words for example predictions. The notebook also includes code for easily comparing classifiers to each other.

What to submit?

Prepare and submit a single write-up (**PDF, maximum 3 pages**) and Python source code (`custom_features.py`, `error_analysis.ipynb`, and `ml_sentiment.py`), compressed in a single `zip` file, to Canvas. **Do not include your student ID number**, since we might share it with the class if it's worth highlighting. The write-up and code should address the following.

1. **Preliminaries, 5 points:** Kaggle username and Kaggle accuracy of best default and custom models.

2. **Default Features, 20 points:** Tune classifiers on default features (include the selected range of parameters in the code), and submit your predictions to Kaggle. Include the accuracies of the best classifier for each (LR and RF) and the two plots in the write-up, and a few sentences on how you picked the range.

3. **Custom Features, 35 points:** Implement your custom features in the code, and train and tune the classifiers. Submit the predictions to Kaggle, identify the best classifier for each, and include the accuracy obtained in the report. Include the description of your features and the vectorizer in a few sentences in the write-up.

4. **Analysis, 35 points:** Select the best classifier with default features and the best classifier with custom features. The analysis will focus on comparing these two classifiers. First, use `eli5` to generate the global importance weights for both classifiers in the notebook (`show_weights`), and in a few sentences in the write-up, describe what is different between them. Then, in the notebook, generate 2 examples each where the classifiers disagree: (i) positive reviews, where only the default is correct, (ii) positive reviews, where only the custom is correct, (iii) negative reviews, where only the default is correct, and (iv) negative reviews, where only the custom is correct. Also include 2 examples each of when both classifiers are correct and both are incorrect. In the write-up, in a paragraph, describe the insights these errors provide about the differences between the classifiers, especially advantages/disadvantages of your custom features.

5. **Statement of Collaboration, 5 points**

Statement of Collaboration (5 points)

It is **mandatory** to include a *Statement of Collaboration* in each submission, with respect to the guidelines below. Include the names of everyone involved in the discussions (especially in-person ones), and what was discussed.

All students are required to follow the academic honesty guidelines posted on the course website. For programming assignments, in particular, we encourage the students to organize (perhaps using Piazza) to discuss the task descriptions, requirements, bugs in our code, and the relevant technical content *before* they start working on it. However, you should not discuss the specific solutions, and, as a guiding principle, you are not allowed to take anything written or drawn away from these discussions (i.e. no photographs of the blackboard, written notes, referring to Piazza, etc.). Especially *after* you have started working on the assignment, try to restrict the discussion to Piazza as much as possible, so that there is no doubt as to the extent of your collaboration.