Independent Project 1: Text Classification

CS 6501-005 Natural Language Processing
Instructor: Yangfeng Ji

Deadline: September 23, 2018

In this project, you will build systems for classifying Amazon reviews. You will:

- Do some basic text processing, tokenizing your input and converting it into a bag-of-words representation
- Build a simple classifier using Perceptron
- Build more stable classifier with the averaged Perceptron algorithm
- Build the logistic regression classifiers with a rich feature set and more regularization options
- Explore the different combinations of hyperparameters in order to find the best model

What you need for this project

- `Python` 2 or 3
- `NLTK`: for tokenization
- `Numpy and Scipy`: for matrix/vector operations
- `Sklearn`: for implementing bag-of-words representations and logistic regression models in section 3

A convenient way to install Python packages is to use `pip`. To learn more detail, please check out this tutorial `https://packaging.python.org/installing/`

What you are going to submit

- a **report** that describes what you have done in this project
- all your **code**, and
- the **prediction results** on the test data.

Please follow the specified format to name your files.

1 Data Description

As you may find from the texts, all data have been partially pre-processed. For example, punctuation has been separated from the texts. Depending what you need, you may still need `NLTK` for further preprocessing.

	# Documents	# Tokens
Training	30K	3.7M
Development	10K	1.2M
Test	10K	1.2M

2 Perceptron Algorithm

In this section, you need to implement your own Perceptron and averaged Perceptron algorithms. You may need `numpy` or `scipy` for matrix/vector operations, but are not allowed to use an existing implementation of these two algorithms in `sklearn` or any other packages.

1. (2 points) Implement the feature function $\boldsymbol{f}(\boldsymbol{x}, y)$ with the bag-of-words representations discussed in lecture 2. You can use some preprocessing tricks to reduce the vocabulary size, such as (1) convert all characters into lowercase, and (2) discard low-frequency words or map them into a special token UNK.
   - Submit your code with name `[computingID]-bag-of-words.py`
   - In your report, describe what you do reduce the feature size and also report the size of the feature set.
2. (2 points) Implement the Perceptron algorithm described in JE section 2.2.1 Algorithm 3.
   - Submit your code with name `[computingID]-perceptron.py`
   - Plot the accuracy curves on both training and development sets per training epochs, for at least *five* training epochs. One training epoch is when the algorithm sees the entire training set.

   Hint: shuffle the training set after each epoch, see whether it gives any difference.
3. (2 points) Implement the averaged Perceptron algorithm described in JE section 2.2.2.
   - Submit your code with name `[computingID]-averaged-perceptron.py`
   - Plot the accuracy curves on both training and development sets per training epochs, for at least *five* training epochs.
4. (2 points) Run the test data with your averaged Perceptron model and submit the predicted results with file name `[computingID]-averaged-perceptron-test.pred`.

3 Logistic Regression

In this section, you can use the `LogisticRegression` function and the `CountVectorizer` function in `sklearn` for the following questions.

1. (1 point) Use both the `LogisticRegression` function and the `CountVectorizer` function with their *default* settings to train a classifier and report
   - the size of your feature set
   - the classification accuracy on both training and development sets.
2. (1 point) Change the argument `ngram_range` in function `CountVectorizer` from $(1,1)$ to $(1,2)$, then re-train your classifier with this large feature set. Report

- the size of your feature set
- the classification accuracy on both training and development sets.

3. (2 points) The regularization parameter $\lambda = \frac{1}{C}$ in the `LogisticRegression` function is 1.0. In practice, we need to tune this parameter in order to find the best model. Try different λ's with the rich feature set built in step 2. For all the λ's, report

   - the corresponding classification accuracy on both training and development sets.

   For your first try, *recommended* λ values are in exponential scale, e.g., $\lambda \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 100\}$, which will help you to narrow down the range of λ for fine-tuning.

4. Similar to L_2 regularization, L_1 regularization adds the parameter constraint with its L_1 norm as

$$\ell_{L_1}(\boldsymbol{\theta}) = -\sum_{i=1}^{N} \log P(y^{(i)}|\boldsymbol{x}^{(i)}; \boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1 \quad (1)$$

   Theoretically, L_1 regularization tends to give sparse solutions, which means most of components in $\boldsymbol{\theta}$ will be 0 or close to 0.

   - (2 points) In lecture 3, we used contour plots to explain how L_2 works. Please use a similar way to explain why L_1 regularization prefers sparse solutions.
   - (1 point) Try different λ's and report the corresponding classification accuracy on both training and development sets.

5. In lecture 3, we talked different ways to refine the feature set in order to obtain a better classification performance on the development set. Together with the options provided in the `LogisticRegression`, please try different combinations of these tricks/arguments and find the best model as you can do. *Classification accuracy* will be an important criterion for evaluating your answers here.

   - (3 points) Report the accuracy on both training and development sets with your *best* model, and explain how you obtain this model.
   - (2 points) Submit the predicted results on the test set with file name `[computingID]-lr-test.pred`.