Assignment 3: Text Classification

Due: 11:59 pm Apr 20th, 2018

Naive Bayes, Logistic Regression, Support Vector Machines, and Random Forests

This is an analysis assignment, where you will investigate the utility of different learning algorithms for a text classification task. You will be using the implementations available in the scikit python library here:
http://scikit-learn.org/stable/supervised_learning.html#supervised-learning

The task is to learn classifiers that can classify input texts into one of the K-classes. You will create classifiers with three configurations:

1. Unigram Baseline (UB) -- Basic sentence segmentation and tokenization. Use all words.
2. Bigram Baseline (BB) -- Use all bigrams. (e.g. I ran a race => {I ran, ran a, a race}.)
3. My best configuration (MBC) -- You can create your own configuration based on the design choices below.

Dataset

For this assignment we're using the "20 newsgroup dataset" which contains 20,000 newsgroup documents partitioned into 20 news categories. We treat each of the categories as classes. We will use only 4 classes for this assignment.

1. rec.sport.hockey
2. sci.med
3. soc.religion.christian
4. talk.religion.misc

In each class, there two sets of documents, one for ***training*** and one for ***test***. The format of each document is as follows:

Header	Consists of fields such as <From>, <Subject>, <Organization> and <Lines> fields. The <lines> field includes the number of lines in the document body.
Body	The main body of the document. This is where you should extract features from.

Note that you can ignore the header when you're extracting features.

Implementation Requirements

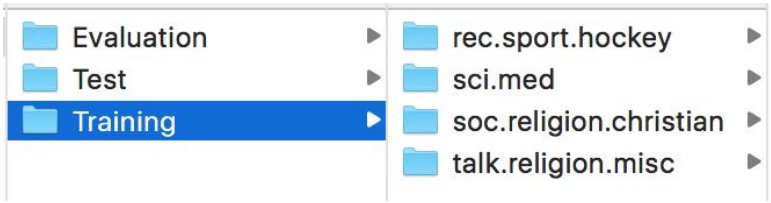
- Your implementation must be in Python 3x, up to 3.5
- All code will be tested by running through plagiarism detection software.
- You should include cite any web resources or friends you used/discussed with for your implementation. **Failure to do so will result in an F.**

What to turn in?

1. **Report:** Include all results/observations below into your report.
2) ***Basic Comparison with Baselines*** (50 points) -- For all four methods (NB, LR, SVM, and RF) you should run both unigram and bigram baselines. You should turn in two results:
    a. **UB_BB.py** (20 points) should run all 4 methods, each with 2 representations. The output of each run is macro-average of precision/recall and F1 value.

python UB_BB.py trainset testset output display_LC

Where 'trainset' and 'testset' are the parent folders to your training data and test data respectively. For example, if your training data is in the 'Training' folder as in the picture below, the parent folder in the argument should be 'Training'.



'display_LC' is an option to display the learning curve (part b). '1' to show the plot, '0' to NOT show the plot.

'output' is a **comma-separated values** format ("YourAlgorithmName,Config(UB/BB),Precision,Recall,F1"), containing 8 rows corresponding to 8 runs. Sample output file as below. Note that all elements are UPPER-CASE letters. The numeric values are precision, recall, and F1 respectively.

```
NB,UB,0.67,0.6,0.63
NB,BB,0.67,0.6,0.63
LR,UB,0.67,0.6,0.63
NB,BB,0.67,0.6,0.63
SVM,UB,0.67,0.6,0.63
SVM,BB,0.67,0.6,0.63
RF,UB,0.67,0.6,0.63
RF,BB,0.67,0.6,0.63
```

Include a table of result as above into the report.

b. (20 points) A learning curve (LC) result, where you show the performance of each classifier with just the unigram representation. The learning curve is a plot of the performance of the classifier (F1 on the y-axis) on the dev fold, when trained on different amounts of training data (size of training data on the x-axis). Include the plot of LC into the report.

c. (10 points) Describe your findings and make arguments to explain why this is the case. You can use any online source to understand the relative performance of algorithms for varying training data sizes. Make arguments in your own words. Provide a citation.

3) ***My best configuration*** (50 points) -- Pick the best performing classification algorithm from the above experiments and then use the design choices below to find the best possible result on the dev set. You will create a model that we can run on a hidden test data.
    a) **MBC_exploration.py** (20 points) use the best classification algorithm from section 2, each with one of 8 representations picking two choices from each of the options below. Output the result to a file as command below.

python MBC_exploration.py trainset testset output

Where 'trainset' and 'testset' are the parent folders to your training data and test data respectively.
'output' is a **comma-separated values** format**,** containing **8 rows** corresponding to 8 runs. Sample of each row is "YourAlgorithmName,YourFeature,Precision,Recall,F1". For example "SVM,CountVectorizer,0.67,0.6,0.63". You can name "YourFeature" to be anything you want but "YourAlgorithmName" must be one of "NB, LR, SVM, or RF".

b **MBC_final.py** (20 points) -- Export the best configuration training model (you can use any combinations) and give us code that we can run to get performance of your model on a test set. You can call helper functions from MBC_exploration.py if you want. The organization of the test data we use is the same as the test data you are given.

python MBC_final.py trainset testset output

Where 'trainset' is the parent folder to your train data; 'testset' is the parent folder to your test data.
'output' contains a single number for your result. Include the result in the report.
The grade for this part is decided by competing all results among students. It means that the threshold for grading this question is based on results of the whole class.

c Explanation (10 points) -- Explain your result based on your best understanding of the configuration options. Should be no longer than 2 paragraphs. You can use any online source to understand the options and what they mean. Provide a citation of the sources.

4. Submission format

- Report must be in pdf format, named "Firstname_lastname_SBUID.pdf"
- Only submit 4 files: report, UB_BB.py, MBC_exploration.py, and MBC_final.py
- Put all files in the same folder named Firstname_Lastname_SBUID, then zip it (please name the folder correctly before zipping, otherwise, the unzip will generate a different folder name). Your submission is one 1 zip file **Firstname_Lastname_SBUID.zip**
- Do NOT submit any data.

Design Choices for your best configuration:

Feature representations

Scikit provides few implementations of feature extractors. The extractors first segment the text into sentences, and tokenize them into words. Each document is then represented as a vector based on the words that occur in it.

1. CountVectorizer -- Uses the number of times each word was observed.
2. TFIDFVectorizer -- Uses relative frequencies normalized by the inverse of the number of documents in which the word was observed.

See Section 4.2.3 in http://scikit-learn.org/stable/modules/feature_extraction.html for details. You can also explore many different choices for getting a good representation. Preprocessing often has a big impact in text tasks. Some choices include:

1. Lower case and filter out stopwords (e.g., I ran a race => I, ran, race).
2. Apply stemming (e.g, {running, runs, ran} ⇒ run). You can find a stemmer in nltk. Use porter stemmer shown here: http://www.nltk.org/howto/stem.html

Feature selection

This is often key to the performance of any ML based application. Depending on the type of algorithm used, you will have different choices for doing feature selection. For instance, you can do an external feature selection where you find the most informative features or you can try L1, L2 or Lasso regularizers. You should read this blog and pick any two regularizations.

http://scikit-learn.org/stable/modules/feature_selection.html

Hyperparameters

For most learning algorithms there are many different options that can be tuned. Finding the best set of options can be tricky and require many rounds of exploration. Try the following:

1. Naive Bayes -- No hyperparameters.
2. Logistic Regression -- Regularization constant, num iterations
3. SVM -- Regularization constant, Linear, polynomial or RBF kernels.
4. RandomForest -- Number of trees and number of features to consider.