CS 272: Statistical NLP: Winter 2020

Homework 1: Semi-supervised Text Classification

Sameer Singh
https://canvas.eee.uci.edu/courses/22668

Many real-world applications contain a small number of labeled instances but a large number of unlabeled instances. Machine learning algorithms that are able to utilize the information from unlabeled instances are known as *semi-supervised* approaches. The first programming assignment will require you to implement such an algorithm that benefits from large amounts of unlabeled text.

The submissions are due by midnight on **January 21, 2020**.

1 Task: Presidential Candidate Speech Classification

The primary objective for the classifier is to identify the presidential candidate (one of 19) from their speeches. For this, we have assembled a collection of speeches from the 2008 and 2012 presidential races, and to makes the task more interesting (i.e. difficult), we have split the speeches into segments of 140 characters or less.

1.1 Data

The data for this task is available on the Kaggle website[1]. The primary data file is named `speech.tar.gz`, which contains the following contents:

- `train.tsv`: List of files and associated labels, to be used for training.
- `dev.tsv`: List of files and associated labels, to be used for development (do not use for training).
- `labeled/`: Folder of text files containing the speeches that are part of training or development sets.
- `unlabeled/`: Folder of text files containing many speeches that are not labeled.

1.2 Kaggle

Kaggle is a website that hosts machine learning competitions, and we will be using it to evaluate and compare the accuracy of your classifiers. The hidden set of gold labels used to evaluate the classifiers is a subset of the unlabeled instances, and thus your submission file to Kaggle should contain a predicted label for all the $43,342$ unlabeled instances. In particular, the submission file should have the following format:

- Start with a single line header: `Fileindex,Category`
- For each of the unlabeled speech (sorted by name) there is a line containing an increasing integer index (i.e. line number-1), then a comma, and then the string label prediction of that speech.
- See `speech-pred.csv` and `speech-basic.csv` on the Kaggle site for examples.

You can make at most *five* submissions everyday, so I 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 that you want to be judged as your final submissions (one for each of the requirements below).

1.3 Source Code

We have made some initial code available for you to develop your submission, available at https://github.com/sameersingh/uci-statnlp/tree/master/hw1. The file `classify.py` contains a basic logistic regression classifier from `sklearn` and a simple example of using the accuracy metrics (this is the metric used on Kaggle to evaluate your results). The file `speech.py` contains methods for loading the dataset (without untarring the data file), creating basic features, and calling the methods to train and evaluate the classifier. It also contains the code to output the submission file for Kaggle from a classifier (`speech-pred.csv`) and the code that has used to generate the solution file and a basic benchmark (`speech-basic.csv`).

[1] https://www.kaggle.com/c/presidential-candidate-classification-w20/

2 What to submit?

Prepare and submit a single write-up (**PDF, maximum 5 pages**) and relevant source code (compressed in a single `zip` or `tar.gz` file; we will not be building or compiling it, nor will we be evaluating its quality) to Canvas. Do not submit iPython or Jupyter notebooks as your writeup, and do not include any code in the writeup. **Do not include your student ID number**, since we might share it with the class if it's worth highlighting. Further, you have to make **two** submissions to Kaggle, one for § 2.1 and one for § 2.2 (you can make multiple submissions to Kaggle and get your scores, however you have to select one submission in each category by the deadline). The write-up and code should contain the following.

2.1 *Supervised:* Improve the Basic Classifier (25 points)

As described above, we have included a basic logistic regression classifier and included its predictions as a benchmark on Kaggle. This classifier uses the default hyper-parameters for regularization and optimization from `sklearn` and uses the basic `CountVectorizer` to create the features. As you can see from the development set accuracy, this is not a trivial task!

Your task is to familiarize yourself with the data and the classification task by improving this supervised text classifier. There are a number of different things you can try, such as hyper-parameter search (using accuracy on the development data), different tokenization, adding/removing features, TF-IDF weighting (using only the training data), and so on[2]. The primary guideline here is to utilize only your intuitions and the training data, and the performance on the development data, in order to make these modifications. Use of unlabeled data or any other external resource is not allowed.

In the writeup, describe the changes, your reasoning behind them, and the results. You should use figures, examples, tables, and graphs to illustrate your ideas and results, for example plotting accuracy as the regularization strength is varied. Submit the relevant code (of your changes, not the analysis) as `sup.tar.gz` or `sup.zip`. The Kaggle submission description should start with "*Supervised:*" (you are free to add whatever else afterwards).

2.2 *Semi-supervised:* Exploiting the Unlabeled Data (70 points)

In the second part of this homework, you will focus on improving your supervised classifier by using the information from the large collection of unlabeled speeches that are available to you. There are many different approaches to do this, I will describe two of the types below. You are free to choose any approach you are interested in. Note that semi-supervised learning is tricky, and it's not always easy to get a substantially higher accuracy than the supervised classifier. Do what you can.

As with the previous submission, you have to describe the approach that you are using, and provide the plots, tables, and graphs to analyze and discuss the results in the write-up. For example, a useful plot for you to include might be the performance of your classifier (on development data) as the amount of unlabeled data is varied, i.e. 0% (same as supervised), 10%,20%,..,100%. Part of the analysis should also include some error analysis and examples of the highest and lowest weighted features for one or more labels. You should also refer to the relevant readings and published papers, and cite them as appropriate, when describing your approach. Include the relevant portions of your code that implements your approach as `semisup.tar.gz` or `semisup.zip`. The description for the submission on Kaggle should start with "*Semi-supervised:*" (you are free to add whatever else afterwards).

2.2.1 Expanding the Labeled Data

One popular approach to semi-supervised learning is to iteratively expand the labeled data (by using the classifier to predict on the unlabeled data) and retrain the classifier. The following pseudocode illustrates this structure.

[2] You might find this useful: http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

```
Require $D_u$ (unlabeled data), $D_l$ (labeled data)
  $\hat{D}_l \leftarrow D_l$
  loop
    $\theta \leftarrow$ TRAIN($\hat{D}_l$)                ▶ train supervised classifier
    $\hat{D}_u \leftarrow$ PREDICT($D_u$,$\theta$)         ▶ predict on unlabeled data
    $\hat{D}_l \leftarrow$ EXPAND($\hat{D}_u$)             ▶ expand the labeled data
    if STOP(_) then return $\theta$                ▶ stopping criterion
    end if
  end loop
```

There are many variations of this algorithm, known as *self-training*, that you can use. The first choice is to decide which labels to include in $\hat{D}_l$ in every iteration: every prediction? most "confident" predictions? predictions with a "soft" label (kind of like EM)? only points that an ensemble of simple classifiers agree on? nearest neighbors of previously labeled points? or something else? The second choice is to decide the stopping criterion, should it be a fixed number of iterations (determined using development data)? should it be when the set of labels stop changing (or only a small proportion changes)? or something else?

When analyzing this approach, I would expect you to include the accuracy and the size of the labeled set (if appropriate) as they vary across iterations. It may also be useful to identify a few features/words whose weight changed significantly, and hypothesize why that might have happened.

2.2.2 Designing Better Features

The primary concern when using a small set of labeled data for training text classifiers is the sparsity of the vocabulary in the training data; irrelevant words might look incredibly discriminative for a label, while relevant words may not even appear in the training data, because of small sample and high dimensional statistics. A way to counteract this is to utilize the corpus of unlabeled text to learn something about the word semantics, and use it to identify the words that are likely to be uttered by the same person. In order words, knowledge from unlabeled documents can allow us to *spread* the labels to words that do not even appear in the training data.

For example, suppose a word like "healthcare" does not appear in the training data (and is quite indicative of a particular candidate, say *Obama 2008*), but "health", "insurance", and "coverage" do (and are as indicative). From co-occurrence statistics on the unlabeled set of speeches, we can determine that "healthcare" seems to co-occur with "health", "insurance", and "coverage" with a much higher probability than with other words. Thus, we may hypothesize that "healthcare" should be indicative of *Obama 2008* as well, even if it never appears during training.

Of course, I am being deliberately vague here in order to not provide a specific solution. If you will explore this direction, you will have to consider how to represent the word contexts (as a word-document matrix? word-word matrix?), how to compute similarity between word representations (cosine distance? pairwise mutual information?), how to represent/encode the notion of similar words (fixed or hierarchical clusters? low-dimensional embeddings? topics?), and finally, how to utilize the labeled data (use as features during training? propagate labels to nearest points using the new distance? directly set weights of new words?), and so on.

If you pick this kind of an approach, the analysis should include why you picked a certain strategy (why you thought it would be a good idea). You should also include examples of words and/or speeches where the propagation of information helped (where it worked) and hurt.

3 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, I encourage the students to organize (perhaps using Campuswire) to discuss the task descriptions, requirements, bugs in my 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 Campuswire, etc.). Especially *after* you have started working on the assignment, try to restrict the discussion to Campuswire as much as possible, so that there is no doubt as to the extent of your collaboration.

4 Potential Concerns / FAQs

Since this kind of an assignment might be unfamiliar to some students (lack of very specific instructions, too many possible solutions, course leaderboard, etc.), I have put together a set of answers that hopefully address some of these concerns. Please post on Campuswire if you have any other questions.

***Question:* Is the grade based on the performance of my submissions relative to the others in the class?**

No, you will primarily be evaluated on the quality and creativity of your write up, but of course, it is expected both your submissions should beat the simple baseline I have provided. If you do get significantly unsatisfactory results, for example your semi-supervised approach performs worse than the supervised classifier, I would expect a discussion and analysis in the writeup (and you can get full credit).

***Question:* Then why have the class leaderboard in the first place?**

The leaderboards are provided for you to be able to evaluate how good your results are relative to others. Some people find this incredibly useful, and a motivation to be more creative about their solutions. To some extent, it also mirrors what happens in the real-world, in academia or industry, when a new task or a dataset is introduced.

***Question:* I do not find this leaderboard motivating, and in fact, am vehemently opposed to reduction of science to a chase after higher accuracy numbers. Will all programming assignments have this competitive structure?**

No, I anticipate maybe only one other programming assignment to have a course leaderboard.

***Question:* This assignment seems too open-ended, and I can see spending many whole days improving my classifier. How will I know I am done?**

The short, but perhaps unsatisfactory, answer is: when you feel you have put an adequate effort. For each part, this would mean you decided on one non-trivial approach you would like to use, implemented it, analyzed the results, and presented the approach in critical light. There is going to be diminishing returns in accuracy with the amount of effort you put in, so I would discourage solely using the accuracy gain in deciding when you are done.

***Question:* I think X (X = SVM, Random forests, neural networks, ...) will do better than logistic regression. Can I use X instead of logistic regression?**

No, you are **not** allowed to use a different classifier than logistic regression (and as much as possible, use the scikit-learn implementation). The aim of this exercise is to train you to get gains using feature engineering and semi-supervised learning algorithms, not from changing the underlying classifier.

***Question:* I am not sure what is a valid external resource that I can use, and what is not?**

In general, no external resource is allowed. There is a lot of speech data online, and in fact you can probably download a similar, or identical, set of speeches. However, using any such resource in any way will constitute as cheating. If you are still in doubt, I suggest posting the question on Campuswire.

***Question:* My code is taking too long to run. What should I do?**

By most standards, this is a modest sized dataset. If you are facing efficiency issues, I suggest you analyze your code for bottlenecks and address them, for example caching feature calculations if runtime feature computations are taking too long. However, I do not want you to waste too much time optimizing your code; if you are struggling, it would be better to *randomly* sub-sample the unlabeled documents or aggressively prune the vocabulary size by frequency, in order to get the results and provide the submissions.

Acknowledgements

This homework is adapted (with minor changes) from one made available by Noah A. Smith from the University of Washington. Thanks, Noah!