TTIC 31190: Natural Language Processing
Assignment 1: Text Classification

Kevin Gimpel

Assigned: Jan. 7, 2016

~~Due: 11:59 pm, Jan. 20, 2016~~

Due: 11:59 pm, Jan. 21, 2016

Submission: email to `kgimpel@ttic.edu`

Submission Instructions

Please include your write-up (in any file format) containing answers to the questions in Sections 1 and 2 below, as well as your report for Section 3. Package your write-up file and code in a single zip file or tarball, name the file with your last name followed by "_hw1", and email the file to `kgimpel@ttic.edu` by 11:59 pm on Jan. 21, 2016.

Also, in your write-up, please include an estimate of approximately how many hours you spent working on this assignment. This will help us better calibrate future assignments. It will not affect your grade.

1 Data Collection

Brainstorm ways to collect **naturally-occurring annotated data** from the web for each of the following text classification tasks:

1. classify a news article as genuine or satirical
2. classify text as friendly, hostile, or neutral
3. classify text as written by a male or a female
4. classify whether a piece of text was written by a native English speaker or a non-native English speaker

For each of the above tasks, briefly describe a procedure you could use to collect naturally-occurring annotated data. You can assume the capability to crawl/scrape publicly-accessible websites and also assume you have access to billions of tweets. The only thing you can't do is pay people to annotate the data for you.

2 Classifiers, Datasets, and Features

For each of the following, create a text classification training dataset (i.e., a set containing textual inputs and their labels) that fits the given property. Hint: think small.

1. Create a nonempty training dataset that no classifier (as defined by Eq. 1) could ever classify completely correctly. That is, all such classifiers would achieve a training accuracy that is strictly less than 100%.
2. Suppose classifier A uses unigram binary features and classifier B uses bigram binary features. Classifiers A and B are both defined by Eq. 1. Create a training dataset for which classifier A could never reach 100% training accuracy, but for which classifier B would be able to achieve 100% training accuracy.

3 Implementation and Experimentation with Linear Text Classifiers

You will implement and experiment with simple ways of building text classifiers.
Consider a classifier defined by the function classify:

$$\text{classify}(\boldsymbol{x}, \boldsymbol{\theta}) = \underset{y \in \mathcal{L}}{\text{argmax}} \ \text{score}(\boldsymbol{x}, y, \boldsymbol{\theta}) \tag{1}$$

where $\mathcal{L}$ is the space of classification labels (classes) and where the score function is defined:

$$\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta}) = \sum_i \theta_i f_i(\boldsymbol{x}, y) \tag{2}$$

We will denote our dataset of training data by $\mathcal{T} = \{\langle \boldsymbol{x}^{(i)}, y^{(i)} \rangle\}_{i=1}^{|\mathcal{T}|}$, where $y^{(i)} \in \mathcal{L}$ is the label of instance $\boldsymbol{x}^{(i)}$. I have prepared several text classification datasets that you can choose from. Each is divided into training (TRAIN, also denoted $\mathcal{T}$), development (DEV), and development test (DEVTEST) portions. You should do optimization on TRAIN, tune hyperparameters and do preliminary development on DEV, and report your final test results on DEVTEST. I didn't give you any of the official test sets for these tasks.

Features:

Implement **unigram binary** features for your text classifier. Use a feature count cutoff of 1. That is, create a binary feature for each word in the vocabulary of the training set $\mathcal{T}$, paired with its observed label.

Learning:

Consider the **perceptron loss function**:

$$\text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} \ \text{score}(\boldsymbol{x}, y', \boldsymbol{\theta}) \tag{3}$$

with subgradient entry j as follows:

$$\frac{\partial \text{loss}_{\text{perc}}(\boldsymbol{x}, y, \boldsymbol{\theta})}{\partial \theta_j} = -f_j(\boldsymbol{x}, y) + f_j(\boldsymbol{x}, \hat{y}) \tag{4}$$

where $\hat{y} = \text{classify}(\boldsymbol{x}, \boldsymbol{\theta})$.
We want to minimize the perceptron loss function on training set $\mathcal{T}$, i.e.:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{argmin}} \sum_{i=1}^{|\mathcal{T}|} \text{loss}_{\text{perc}}(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \tag{5}$$

Once you have an estimate $\hat{\boldsymbol{\theta}}$, use it to classify the held-out data (DEV or DEVTEST) by calling $\text{classify}(\boldsymbol{x}, \hat{\boldsymbol{\theta}})$ for each $\boldsymbol{x}$ in the held-out set.

Provided Data:

Four text classification datasets are provided to you. Choose **two (2)** of these four for your experiments. Each contains files corresponding to TRAIN, DEV, and DEVTEST. Each line in each file contains a textual input followed by a tab followed by an integer containing the gold standard label. Below are the four datasets:

- `senti.tar.gz`: contains `senti.{train,dev,devtest}`; fine-grained sentiment analysis of movie reviews (Socher et al., 2013); $\mathcal{L}$ ={strongly negative, negative, neutral, positive, strongly positive}. Note that the `train` file contains labels for all constituents of the training sentences, while `dev` and `devtest` contain only full sentences. There is also a file with suffix `train.onlyfull` containing only the full training sentences. (Training on `train` usually works better than `train.onlyfull`.)
- `trec.tar.gz`: contains `trec.{train,dev,devtest}`; TREC question classification (Li and Roth, 2002); $\mathcal{L}$ ={abbreviation, description, entity, human, location, numeric value}.
- `subj.tar.gz`: contains `subj.{train,dev,devtest}`; sentence subjectivity classification (Pang and Lee, 2004); $\mathcal{L}$ ={objective, subjective}.
- `CR.tar.gz`: contains `CR.{train,dev,devtest}`; binary sentiment classification of customer reviews of several products (Hu and Liu, 2004); $\mathcal{L}$ ={negative, positive}.

3.1 Required:

Perform the following sequence of steps. For steps 2 (experiments) and 3 (analysis), use two of the provided text classification datasets.

1. **Implement** the classifier described above, including the unigram binary features, the classify function used to make predictions, and the training procedure described above, using stochastic subgradient descent (SSD) to solve the optimization problem in Eq. 5. Turn in your code as part of your submission. Your code should be documented enough that we can look at it briefly and figure out what's going on.
2. **Experiment** with your implementation. Run SSD for N epochs over TRAIN ($N = 50$ is likely enough), using a mini-batch size of 1 (i.e., online), and a fixed stepsize of 0.005. An epoch is defined as processing $|\mathcal{T}|$ examples; on each epoch, you can either sample training examples with replacement $|\mathcal{T}|$ times or simply loop through TRAIN in order. After each epoch, calculate the value of the loss function on TRAIN and compute the classification accuracy on TRAIN and DEV. Each time you see a new highest DEV accuracy, compute the classification accuracy on DEVTEST.
3. **Analyze** the results:
   (a) Plot the loss function values on TRAIN as a function of training epochs.
   (b) Plot TRAIN accuracy and DEV accuracy as a function of training epochs.
   (c) Report the accuracy on DEVTEST corresponding to the single epoch that achieved the best accuracy on DEV. This is often called **early stopping**. If multiple epochs had the same DEV accuracy, choose one arbitrarily.

(d) Inspect the learned feature weights. For each label, print the 100 features with the largest weights. Do the highest-weighted features make sense for the dataset? Is anything surprising? You can also look at the features with the lowest (i.e., most negative) weights, but they are often harder to interpret.

3.2 Your Choice

After doing the above, choose any **one (1)** of the three choices below. For whichever one you choose, use both of your text classification datasets.

1. **Modeling**: perform the following feature exploration steps.
   - Implement unigram count features. Compare them empirically with unigram binary features. Is there much difference in performance?
   - Implement **bigram** and **trigram** binary features. Experiment with each feature template by itself and also combined with the unigram binary features. Do higher-order features improve performance compared with unigram binary features alone? What is the effect on runtime?
   - Implement the ability to use different feature count cutoffs. Try combining different count cutoffs with different feature templates. Do you see any improvement in performance by using a cutoff other than 1?
   - Brainstorm a new kind of feature template, implement it, and report the results.

2. **Learning**: compare perceptron loss and hinge loss.
   - Let's define **hinge loss** as follows:

     $$\text{loss}_{\text{hinge}}(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\text{score}(\boldsymbol{x}, y, \boldsymbol{\theta}) + \max_{y' \in \mathcal{L}} \left(\text{score}(\boldsymbol{x}, y', \boldsymbol{\theta}) + \delta \mathbb{I}[y \neq y']\right) \tag{6}$$

     where δ is a hyperparameter of the learning procedure.
   - Derive/write out the subgradient of the hinge loss (as we did for the perceptron loss in Eq. (4) above).
   - Implement and experiment with minimizing hinge loss instead of perceptron loss. Compare different values of δ, including values like 0, 0.5, 1, 5, and 10.
   - What is the effect of increasing or decreasing δ on the training accuracy? What is the effect on held-out accuracy?
   - How does changing δ affect what you see when printing the highest-weighted features?

3. **Analysis**: error analysis and feature design.
   - Print the errors made by your classifier on DEV.
   - Look through them manually. After looking at a few, you will likely start to observe certain patterns in the errors; define your own error classification scheme based on what you observe and then manually categorize each error into one of the error categories.
   - If there are too many errors to go through manually, go through a sample of at least 50.
   - Based on your analysis, brainstorm a new feature template to address one of the error categories you observed.

- Implement your new feature template and check its effect on the DEV instances in that error category.
- Describe your error analysis, your new feature template, and report the results of your experiments in your write-up.
- Repeat for the other dataset.

3.3 Extra Credit: Your Second Choice

If you're interested in extra credit, do a second one of the three choices from Section 3.2.

References

Hu, M. and Liu, B. (2004). Mining and summarizing customer reviews. In *Proceedings of KDD*. [3]

Li, X. and Roth, D. (2002). Learning question classifiers. In *Proceedings of COLING*. [3]

Pang, B. and Lee, L. (2004). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of ACL*. [3]

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP*. [3]