Statistical NLP 2019 - Assignment 2

Due Monday Sept 30 - at 11:59pm. Email code/report to urvish@nyu.edu and aparikh@cs.nyu.edu

Deliverables: Deliverables:

- Result file in github repository.
- Code zip emailed to instructors. You should use Python / numpy to implement the assignment.
- **Submission Format:** Please title your submission email **Assignment 2 submission**. You should submit a zip file (firstname_lastname_hw2.zip) containing the report in pdf format (firstname_lastname_hw2.pdf) and the zipped code directory. Write your netID and your collaborators' netID (if any) in the report.

Git and Leaderboard

On the course webpage there will be a leaderboard for tracking the prediction performance of the models that you will be building during the assignments. It will only show the (anonymized) best submission so far, not your particular one. This is simply to give you a general idea of the performance that others in the class are getting (Note that only the extra credit portion of the assignment requires you to compete with the scores of others in the class. You are eligible for full, normal credit on the assignment as long as you achieve a certain pre-determined score).

Whenever the leaderboard is updated however, it will send you an email with your test set score. Note that this is done only once every 3 hours for this assignment to reduce overfitting.

We will be using public GitHub repositories to manage the submissions. If you are not familiar with *git*, there are plenty of good tutorials online, for example: `http://www.vogella.com/articles/Git/article.html`.

If you don't already have a GitHub account yet, please create one here: `https://github.com/signup/free`. You are welcome to reuse an existing account for this course.

Now fork the master repository `https://github.com/urvishdesai/StatisticalNLP19`. We only be using the repository for submitting system output (**please do not put your code/data there**), so you will find a skeleton structure for all the assignments.

Please edit `hw0/output.txt` so it contains your name, email address, and a screen name for the course leaderboard and submit it via: `git add hw0/output.txt; git commit -m 'my first check-in'; git push;`

Finally, post your repository to Piazza, so that I can link it to your name. Your repository path should be of the form: `https://github.com/username/stat-nlp-fall2019`. There is a manual step that we need to perform after you post your repository to Piazza, so please give me a couple of hours. After that everything will be automatic.

1 Problem Statement

Proper Name Classification: Proper name classification is the task of taking proper names like *Eastwood Park* and deciding whether they are places, people, etc. In the more general task of named entity recognition (NER), which we do not consider here, one also has to detect the boundaries of the phrases. In general, we might use a variety of cues to make these kinds of decisions, including looking at the syntactic environment that the phrases occur in, whether or not

the words inside the phrase occur in lists of known names of various types (gazetteers), and so on. In this assignment, however, you will write classifiers which attempt to classify proper names purely on the basis of their surface strings alone. This approach is more powerful than you might think: for example, there aren't too many people named *Xylex*. Since even the distribution of characters is very distinctive for each of these categories, we will start out trying to make classification decisions on the basis of character n-grams alone (so, for example, the suffix *-x* may indicate drugs while *-wood* may indicate places.

Code Instructions: In this assigment you will implement a logistic regression model to solve this problem in Python. You may also use the numpy library to aid with matrix calculations etc. if you so desire. However, you are required to implement your own version of gradient descent as instructed below and **cannot** use any automatic differentiation libraries such as PyTorch or Tensorflow.

2 Obtaining the data

I will post the link to the data on piazza. There should be 3 files (train, validation (devlopment), test). The train and validation files are two column since both the label (first column) and input (second column) is provided. For the test file, only the input (second column) is given.

3 Implementing the model [30 points]

Representing features: Your first step is to define and represent a set of features that you can easily compute and store from the data. For debugging I would recommend, starting with unigrams as the set of features. Then *Xylexx* would have the following features:

```
["X" : 1.0, "y" : 1.0, "l" : 1.0, "e" : 1.0, "x" : 2.0]
```

You are free to make your own implementation choices. For instance one basic strategy may be to use a dictionary that maps feature strings to integer ids. Then for each example, the counts for each feature can be stored in an array/numpy vector that makes it easy to multiply with the weights.

Implementing objective function Let the dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $y_i \in \mathcal{Y}$ is the class label of input point $\mathbf{x}_i$. Let $f(\mathbf{x}_i)$ denote the feature vector for $\mathbf{x}_i$ (such as the unigrams above). Let $\mathbf{w}_y$ be a set of class specific weights and let $\mathbf{w} = \{\mathbf{w}_y\}_{y\in\mathcal{Y}}$ denote the set of all weights [1] The objective function is then the conditional log likelihood:

$$L(\mathbf{w}) = \sum_{i=1}^{m} \log P(y_i|\mathbf{x}_i, \mathbf{w}) = \sum_{i=1}^{m} \log \left(\frac{\exp\left(\mathbf{w}_{y_i}^\top f(\mathbf{x}_i)\right)}{\sum_{y'\in\mathcal{Y}} \exp\left(\mathbf{w}_{y'}^\top f(\mathbf{x}_i)\right)} \right)$$

You will need to implement this objective function in your code.

Regularization You should augment the above objective function with ℓ_2 regularization.

$$L_\lambda(\mathbf{w}) = \sum_{i=1}^{m} \log P(y_i|\mathbf{x}_i, \mathbf{w}) - \lambda\|\mathbf{w}\|^2$$

[1] This is a slightly different notation from the Lecture slides where y was incorporated into the feature vector. Here the weights are being indexed by y.

4 Learning [30 points]

Computing the gradient. Next, write code that computes the gradient of $L_\lambda(\mathbf{w})$ with respect to

$$\frac{\partial L}{\partial \mathbf{w}_y} = \sum_{i=1}^{m} I(y_i = y) f(\mathbf{x}_i) - \sum_{i=1}^{m} P(y|\mathbf{x}_i, \mathbf{w}) f(\mathbf{x}_i) - 2\lambda \cdot \mathbf{w}$$

Recall that the left sum is the total feature count vector over examples with true class y in the training, while the right sum is the expectation of the same quantity, over all examples, using the label distributions the model predicts.

Implementing gradient ascent

Next you will implement gradient ascent:

$$\begin{aligned}
&t := 0 \\
&\mathbf{w}^{(0)} := \text{ randomly initialized} \\
&\text{while } t == 0 \text{ or } \|\mathbf{w}^{(t)} - \mathbf{w}^{(t-1)}\|_2 > \epsilon : \\
&\quad \mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \alpha \frac{\partial L}{\partial \mathbf{w}} \\
&\quad t \leftarrow t + 1
\end{aligned}$$

where t is the iteration number and generally, $\alpha_t \propto \frac{\alpha_0}{\sqrt{t}}$. α_0 is the initial learning rate and ϵ determines when the algorithm has converged. You may need to play with the learning rate / convergence criterion so that the algorithm converges properly.

A good way to see if your algorithm is working properly is to plot the objective function for each t and observe the trend.

5 Feature Engineering / Evaluation [30 points]

:

Evaluation You will need to implement an evaluation function that computes the accuracy on the validation set (and perhaps also the training set for debugging). Define $\widehat{y}_i$ to be the prediction of your model on example i:

$$\widehat{y}_i = \text{argmax}_{y \in \mathcal{Y}} \mathbf{w}^T f(\mathbf{x}_i)$$

Then, for a dataset of size m:

$$\textbf{accuracy} = \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}[y_i == \widehat{y}_i]$$

i.e. accuracy is the fraction of the time, the model predicts the correct class

Feature engineering / Hyperparameter tuning:

You now should have all you need to be able to train and evaluate your model end-to-end:

- Your model should be training (learning weights) on the training set
- You should be measuring accuracy on the validation set (and perhaps also on the training set to determine overfitting).
- You will want to make design choices (features) and hyperparameter choices to drive up the accuracy on the validation set.

- In the end, you seek a model that performs well on the test set.

Using only unigram features should get you a performance of around 60-65%. Your next task is to boost the performance of your classifier with feature engineering (i.e. defining more complex features). To obtain full credit for this part, you should obtain $>= 80\%$ accuracy on the test set. You will receive partial credit for getting less than this number.

Submitting the test set predictions: Finally, you should output the predictions of your model on the test set and store them in a file, in the same format in the train and validation sets. You should upload this to your github repository in `hw2/output.txt`.

(Small) Writeup (10 points): The recommended write up length is around 1-1.5 pages.

1. Record the performance of your top scoring submission on the validation and test sets.
2. Plot the objective function and accuracy, each as a function of the iteration t as your algorithm trains.
3. Describe the features you added to boost the performance.

Extra Credit [5 points]: Be the highest scoring submission in the class. Submissions using late days do not count for extra credit.