Machine Learning and Computational Statistics, Spring 2015
Homework 3: SVM and Sentiment Analysis

Due: Monday, February 23, 2015, at 4pm (Submit via NYU Classes)

Instructions: Your answers to the questions below, including plots and mathematical work, should be submitted as a single PDF file. You may include your code inline or submit it as a separate file. You may either scan hand-written work or, preferably, write your answers using software that typesets mathematics (e.g. LaTeX, LyX, or MathJax via iPython).

1 Introduction

In this assignment we'll be working with natural language data. In particular, we'll be doing sentiment analysis on movie reviews. This problem will give you the opportunity to try your hand at feature engineering, which is one of the most important parts of many data science problems. From a technical standpoint, this homework has two new pieces. First, you'll be implementing Pegasos. Pegasos is essentially SGD for the SVM, and it even comes with a schedule for the step-size, so nothing to tweak there. Second, because in natural langauge domains we typically have huge feature spaces, we work with sparse representations of feature vectors, where only the non-zero entries are explicitly recorded. This will require coding your gradient and SGD code using hash tables (dictionaries in Python), rather than nympy arrays.

2 The Data

We will be using "polarity dataset v2.0", constructed by Pang and Lee (`http://www.cs.cornell.edu/People/pabo/movie%2Dreview%2Ddata/`). It has the full text from 2000 movies reivews: 1000 reviews are classified as "positive" and 1000 as "negative." Our goal is to predict whether a review has positive or negative sentiment from the text of the review. Each review is stored in a separate file: the positive reviews are in a folder called "pos", and the negative reviews are in "neg". We have provided some code in `load.py` to assist with reading these files. You can use the code, or write your own version. The code removes some special symbols from the reviews. Later you can check if this helps or hurts your results.

1. Load all the data and randomly split it into 1500 training examples and 500 test examples.

3 Sparse Representations

The most basic way to represent text documents for machine learning is with a "bag-of-words" representation. Here every possible word is a feature, and the value of a word feature is the number of times that word appears in the document. Of course, most words will not appear in any particular document, and those counts will be zero. Rather than store a huge number of zeros, we use a sparse representation, in which we only store the counts that are nonzero. The counts are stored in a key/value store (such as a dictionary in Python). For example, "Harry Potter and Harry Potter II" would be represented as the following Python dict: x={'Harry':2, 'Potter':2, 'and':1, 'II':1}. We will be using linear classifiers of the form $f(x) = w^T x$, anad we can store the w vector in a sparse format as well, such as w={'minimal':1.3,'Harry':-1.1,'viable':-4.2,'and':2.2,'product':9.1}. The inner product between w and x would only involve the features that appear in both x and w, since whatever doesn't appear is assumed to be zero. For this example, the inner product would be x[Harry] * w[Harry] + x[and] * w[and] = 2*(-1.1) + 1*(2.2). Although we hate to spoil the fun, to help you along, we've included code to take the dot product between two vectors represented in this way, and to increment one sparse vector by a scaled multiple of another vector, which is a very common operation. These functions are located in `util.py`.

1. Write a version of `generic_gradient_checker` from Homework 1 (see Homework 1 solutions if you didn't do that part) that works with sparse vectors represented as dict types. Since we'll be using it for stochastic methods, it should take a single (x, y) pair, rather than the entire dataset. Be sure to use the dotProduct and increment primitives we provide, or make your own. [You'll eventually be using with for the SVM loss and gradient.]

4 Support Vector Machine via Pegasos

In this question you will build an SVM using the Pegasos algorithm. To align with the notation used in the Pegasos paper[1], we're considering the following formulation of the SVM objective function:

$$\min_{w \in \mathbf{R}^n} \frac{\lambda}{2}\|w\|^2 + \frac{1}{m}\sum_{i=1}^{m} \max\left\{0, 1 - y_i w^T x_i\right\}.$$

Note that for simplicity we are leaving off the unregularized bias term b, and the expression with "max" is just another way to write $\left(1 - y_i w^T x\right)_+$. Pegasos is stochastic subgradient descent using a step size rule $\eta_t = 1/(\lambda t)$. The pseudocode is given below:

```
Input: $\lambda > 0$. Choose $w_1 = 0, t = 0$
While epoch $\leq$ max_epochs
  For $j = 1, \ldots, m$ (assumes data is randomly permuted)
    $t = t + 1$
    $\eta_t = 1/(t\lambda)$;
    If $y_j w_t^T x_j < 1$
      $w_{t+1} = (1 - \eta_t \lambda) w_t + \eta_t y_j x_j$
    Else
      $w_{t+1} = (1 - \eta_t \lambda) w_t$
```

[1]Shalev-Shwartz et al.'s "Pegasos: Primal Estiamted sub-GrAdient SOlver for SVM" `http://ttic.uchicago.edu/~nati/Publications/PegasosMPB.pdf`

1. [Written] Compute a subgradient for the "stochastic" SVM objective, which assumes a single training point. Show that if your step size rule is $\eta_t = 1/(\lambda t)$, then the the corresponding SGD update is the same as given in the pseudocode.

2. Implement the Pegasos algorithm to run on a sparse data representation. The output should be a sparse weight vector w. [As should be your habit, please check your gradient using `generic_gradient_checker` while you are in the testing phase. That will be our first question if you ask for help debugging. Once you're convinced it works, take it out so it doesn't slow down your code.]

3. Write a function that takes the sparse weight vector w and a collection of (x, y) pairs, and returns the percent error when predicting y using $\text{sign}(w^T x)$ (that is, report the 0-1 loss).

4. Using the bag-of-words feature representation described above, search for the regularization parameter that gives the minimal percent error on your test set. A good search strategy is to start with a set of lambdas spanning a broad range of orders of magnitude. Then, continue to zoom in until you're convinced that additional search will not significantly improve your test performance. Once you have a sense of the general range of regularization parameters that give good results, you do not have to search over orders of magnitude everytime you change something.

5 Error Analysis

The natural language processing domain is particularly nice in that one can often interpret why a model has performed well or poorly on a specific example, and it is often not very difficult to come up with new features that would help fix the problem. The first step in this process is to look closely at the errors that our model makes.

6 Features

https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews

7 Feedback (not graded

1. Approximately how long did it take to complete this assignment?

2. Did you find the Python programming challenging (in particular, converting your code to use sparse representations)?

3. How did you find this assignment, compared to the first two?

4. Any other feedback?