Homework 7
CS 221 (Autumn 2012–2013)

Submission instructions: Write your answers in one PDF file named `hw7.pdf`. Remember to include your name and SUNet ID. Copy the PDF file onto `corn.stanford.edu`, ssh in to the machine, type `/usr/class/cs221/WWW/submit`, and follow the instructions.

1. Loss minimization (*7 points*)

a. (*2 points*)
Suppose we're doing least squares regression and our training data is:

$$\text{Train} = \{([0,1],5),([0,1],9),([1,0],8),([1,0],2)\}.$$

Suppose our features are just the raw vector x: $\phi(x) = x \in \mathbb{R}^2$. What is the weight vector $\mathbf{w}$ that minimizes the sum of the squared losses over the training examples?

b. (*1 point*)
Suppose we're training a spam classifier, where $+1$ is spam and -1 is not spam. If we classify a message x as spam when it's not spam, we suffer a loss which is the length of the message $|x|$. However, a message is actually spam but we classify it as non-spam, we suffer a loss of 1. Write the expression for the corresponding loss function $\text{Loss}(x, y, \mathbf{w})$.

c. (*2 points*)
Suppose we have a loss function $\text{Loss}(x, y)$ (either hinge, logistic, or squared). Let

$$L_{\text{opt}}(\phi) = \min_{\mathbf{w}} \sum_{(x,y)\in \text{Train}} \text{Loss}(x, y, \mathbf{w})$$

be the minimum training error obtained by minimizing over all weight vectors $\mathbf{w}$.

Show that adding new features can only decrease the training error—that is, if the components of $\phi'(x)$ are a superset of $\phi(x)$, that $L_{\text{opt}}(\phi') \leq L_{\text{opt}}(\phi)$. Your justification should be short and precise. Will test error also be guaranteed to not go up?

d. (*1 point*)
If we obtain a weight vector $\mathbf{w}$ that achieves zero hinge loss on all training examples, what can we say about the number of mistakes (zero-one loss) that $\mathbf{w}$ makes on the training examples?

e. (*1 point*)
Suppose we have the following loss function for regression:

$$\text{Loss}(x, y, \mathbf{w}) = \begin{cases} r^2 & \text{if } |r| \leq 1 \\ |r| & \text{otherwise,} \end{cases}$$

where $r = (\mathbf{w} \cdot \phi(x)) - y$ is the residual. Compute the gradient $\nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$.

2. Website classification (*12 points*)

In this problem, you will implement a binary classifier to detect whether a website is someone's homepage (output label +1) or not (output label −1). The raw input x will consist of two strings, the URL of the website and the title of the website. For example, the URL `http://cs221.stanford.edu` with title *CS221: Artificial Intelligence* would have label −1, whereas the URL `http://www-cs-faculty.stanford.edu/~uno` with title *Donald E. Knuth* would have label +1.

You will only need to edit `learning.py` (although looking at `util.py` might be helpful), and that's the only code file you will submit. Note that all feature vectors and weight vectors are represented as `util.Counter` objects.

a. (*6 points*)

You will start out by implementing the loss and the gradient functions for three functions: the hinge loss, logistic loss, and the squared loss. The different implementations should look quite similar.

You should implement the functions:

```
logisticLoss
logisticLossGradient
hingeLoss
hingeLossGradient
squaredLoss
squaredLossGradient
```

Note that multiplying two counters will take the dot product of the counters; multiplying a counter by a constant will multiply every element of the counter by a constant. When multiplying a counter by a constant, remember to place the constant second, or else you will try to multiply Python's native `Float` type by a `Counter`!

b. (*3 points*)

You will now implement a general stochastic gradient descent algorithm that will work for any of the above loss functions. Start by implementing the `predict` function, which uses the current weights to predict the output of an input x. Then fill in the code of `learn`, which consists of two blocks of code, one for updating the weights using the training examples, and another for evaluating the objective function. Further instructions are given in the comments of the code. You should be able to run your learning algorithm by just typing `python learning.py`. This will run your code on the toy dataset (default) to help you debug your code (you should get 0 error on both train and validation). Tip: if the regularization is 0, don't bother to update the weights with 0 times the gradient of the regularizer.

You can then switch to `python learning.py --dataset websites` and try out different loss functions that you implemented:

```
python learning.py --dataset websites --loss hinge
python learning.py --dataset websites --loss squared
python learning.py --dataset websites --loss logistic
```

You should watch the objective function generally go down—if it doesn't, something's wrong (although least squares regression wobbles around a bit). The training error and the validation error should decrease too, but there will be some randomness. Eventually, your errors should be about 14% and 17%, respectively. You can also try tuning the various hyperparameters (see `util.py` for the options). Describe which loss function works best in a sentence or two. Describe the effect of regularization, step-sizes, etc., in a few sentences.

Our autograder will evaluate your predictor on a different test set which you don't see, to make sure you haven't manually overfit the validation set. This also means that you should debug your algorithm on the training set, rather than the validation set, to avoid unintentionally overfitting.

c. (*3 points*)

The previous algorithm was run on the `basicFeatureExtractor`, which only looks at the URL. You should implement your own `customFeatureExtractor` which can look at both the URL and the title of the website. You can start the program with your feature extractor using the flag `--featureExtractor custom`. For example:

```
python learning.py --dataset websites --loss logistic --featureExtractor custom
```

Experiment around with different features. You can pass the `--verbose 1` flag, which will print out more details about what kind of errors the algorithm is making. Error analysis is an important part of doing machine learning; look through the errors and see if you can add any features to capture phenomena that's not modeled currently. You can also look at the `weights` file to look at the feature weights to see if you are learning something sensible.

When you are done setting your features and fine tuning your algorithm, implement `setTunedOptions` to set the hyperparameters (e.g., stepsizes, loss functions) to what you think are good settings (you shouldn't spend too much time on parameter twiddling—being in the general ballpark will earn you full credit). We will call your `setTunedOptions`, call your learning algorithm, and then evaluate on our held out test set. You can test your options with the command:

```
python learning.py --dataset websites --featureExtractor custom --setTunedOptions True
```

You should be able to get your validation error down to about 8% without much work.

3. Maximum likelihood (*5 points*)

a. (*2 points*)
Suppose we have a simple Bayesian network over variables A, B, C with distribution

$$\mathbb{P}(A = a, B = b, C = c) = p_1(a)p_2(b|a)p_2(c|b).$$

Suppose we get the following samples of (A, B, C):

$$\text{Train} = \{(1, 1, 0), (0, 1, 1), (0, 1, 0), (1, 1, 1), (1, 0, 1)\}.$$

What are the maximum likehood estimates for p_1 and p_2? What are the estimates if we do Laplace smoothing with $\lambda = 2$?

b. (*2 points*)
Consider the Naive Bayes model defined over variables $Y, X_1, \ldots, X_L$:

$$\mathbb{P}(Y = y, X_1 = x_1, \ldots, X_L = x_L) = p_{\text{class}}(y) \prod_{j=1}^{L} p_{\text{word}}(x_j \mid y),$$

where the parameters of the model are $\theta = (p_{\text{class}}, p_{\text{word}})$. Recall that the maximum likelihood objective is:

$$\max_{\theta} \prod_{(x,y) \in \text{Train}} \mathbb{P}(Y = y, X_1 = x_1, \ldots, X_L = x_L).$$

Write the expression for $\text{Loss}(x, y, \theta)$ so that the maximum likelihood objective is equivalent to loss minimization:

$$\min_{\theta} \sum_{(x,y) \in \text{Train}} \text{Loss}(x, y, \theta).$$

c. (*1 point*)
If there are W words and Y can be one of K different classes, what is the total number of parameters as a function of W, K, L (big-Oh notation is sufficient)?