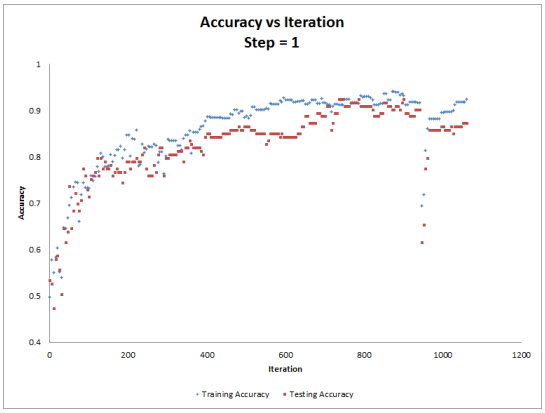
Paul Boschert
09/11/2015
CSCI 5622 - Machine Learning: Homework 2 Analysis

1) The role of the learning rate is to more quickly (or more slowly) get to an optimal solution. It controls the magnitude of the changes to the β_j values. A step size, λ, that is too large can quickly cause the exp() function to overflow because it may create very large β_j values.

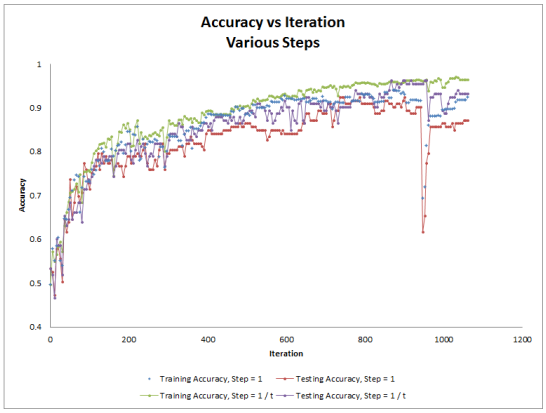
$$\pi_i = \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} \tag{1}$$

Generally speaking, the learning rate is usually chosen by trial and error. If it's not constant, it usually gradually decreases as the iteration number increases.

2) With a constant step size of one, the accuracy of the training data and the test data start to level out around iteration ~ 800. See Figure 1(a) for more details.



(a) The relationship of accuracy and iteration



(b) The relationship of accuracy and iteration different steps

Figure 1: Accuracy and learning rate

3) The words that are the best predictors have the largest $|\beta_j|$ values. I sorted the final β array using numpy.argsort. This gives back indices that would sort the array. Then I used the last five indices and the first five indices to index into the vocab array to get the words. The first five (greatest negative values) are associated with the Hockey classification, and the last five (greatest positive values) are associated with the Baseball classification. An important note here: I used the entire data set, and I also used a $\lambda_t = 1/t$ where t is the iteration. The five best predictors are:

	Hockey		Baseball	
Num	Word	β	Word	β
0	hockey	-1.851942	hit	1.207308
1	playoffs	-1.270941	runs	1.146263
2	pick	-0.995942	bat	0.929948
3	playoff	-0.904541	saves	0.900354
4	points	-0.901229	pitching	0.840741

4) The words that are the worst predictors have the smallest $|\beta_j|$ values. I sorted the final β array using numpy.argsort. This gives back indices that would sort the array. Then I looped through the array until I saw a sign change which indicates a change in classification for either the Hockey classification or the Baseball classification. Once I saw a change in sign, I walked back five values to find the worst predictors for the Hockey classification, and I walked forward five values to find the worst predictors for the Baseball classification. I used these indices to index into the vocab array to get the words. An important note here: I used a $\lambda_t = 1/t$ where t is the iteration. Another potential ambiguity: some might argue that the worst predictors for a particular classification are the best predictors for a different classfication for example a $\beta_j = 2$ is so far away from the Hockey classification that it's the worst predictor. I did not do this, I chose

to interpret the question to mean that we should find the worst predictors for a particular classification that still actually predict that classification. The five worst predictors are:

	Hockey		Baseball	
Num	Word	β	Word	β
0	silence	0.000000	tandem	6.43996e-13
1	everywhere	0.000000	terrific	6.43996e-13
2	riel	0.000000	partly	7.89948e-13
3	intermissions	0.000000	chicago	2.00711e-12
4	tone	0.000000	dive	2.00711e-12

I belive the values for Hockey are 0 instead of slightly negative because of the precision available. Those values do indicate a different sign because that's how I found them, but when printed out they are so small, they show zero.

EC 1) At first I attempted to use a variable learning rate that was simply $1/t$. The effect was generally better. See Figure 1(b) Then, I looked at the scikit-learn python package. When the learning rate is set to 'optimal' it is implemented as $\lambda_t = \frac{1}{\alpha(t_0+t))}$ Where t_0 is chosen heuristically using a method proposed by Léon Bottou. After reading the relevant sections of his paper [1]. I tried using a smaller data set to determine the optimal γ_0 value from the formula given in [1], $\gamma_t = \gamma_0(1+\gamma_0\lambda t)^{-1}$. Unfortunately, this gave me nearly identical results to $\lambda_t = 1/t$.

References

[1] Bottou, Léon *Stochastic Gradient Descent Tricks, http://research.microsoft.com/pubs/192769/tricks-2012.pdf*, 2012. PDF file.