CSC 411: Introduction to Machine Learning
ASSIGNMENT # 1

Out: Jan 31, 2016

Due: Feb 12 (Friday), 2016 [noon Toronto time]

Submission

Package your code using *zip* or *tar.gz* in a file called `CSC411-A1-*your-student-id*.[zip|tar.gz]`. Only include functions and scripts that you modified.

Submit the code on MarkUs by February 12, 2016, noon. Hand in the written part of your assignment in class (tutorial) on February 12.

1 Logistic Regression (30 points)

1.1 (10 points) Bayes' Rule

Suppose you have a D-dimensional data vector $\mathbf{x} = (x_1, \ldots, x_D)^T$ and an associated class variable $y \in \{0, 1\}$ which is Bernoulli with parameter α (i.e. $p(y = 1) = \alpha$ and $p(y = 0) = 1 - \alpha$). Assume that the dimensions of $\mathbf{x}$ are conditionally independent given y, and that the conditional likelihood of each x_i is Gaussian with μ_{i0} and μ_{i1} as the means of the two classes and σ_i as their shared standard deviation.

Use Bayes' rule to show that $p(y = 1|\mathbf{x})$ takes the form of a logistic function:

$$p(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{D} w_i x_i - b\right)}$$

Derive expressions for the weights $\mathbf{w} = (w_1, \ldots, w_D)^T$ and the bias b in terms of the parameters of the class likelihoods and priors (i.e., μ_{i0}, μ_{i1}, σ_i and α).

1.2 (10 points) Maximum Likelihood Estimation

Now suppose you are given a training set $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$. Consider a binary logistic regression classifier of the same form as before:

$$p(y = 1|\mathbf{x}^{(n)}, \mathbf{w}, b) = \sigma(\mathbf{w}^T\mathbf{x}^{(n)} + b) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{D} w_i x_i^{(n)} - b\right)}$$

Derive an expression for $E(\mathbf{w}, b)$, the negative log-likelihood of $y^{(1)}, \ldots, y^{(n)}$ given $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$ and the model parameters, under the i.i.d. assumption. Then derive expressions for the derivatives of E with respect to each of the model parameters.

1.3 (10 points) **L2 Regularization**

Now assume that a Gaussian prior is placed on each element of $\mathbf{w}$, and b such that $p(w_i) = \mathcal{N}(w_i|0, 1/\lambda)$, and $p(b) = \mathcal{N}(b|0, 1/\lambda)$. Derive an expression that is proportional to $p(\mathbf{w}, b|\mathcal{D})$, the posterior distribution of $\mathbf{w}$ and b, based on this prior and the likelihood defined above. The expression you derive must contain all terms that depend on $\mathbf{w}$ and b.

Define $L(\mathbf{w}, b)$ to be the negative logarithm of this posterior. Show that $L(\mathbf{w}, b)$ takes the following form:

$$L(\mathbf{w}, b) = E(\mathbf{w}, b) + \frac{\lambda}{2}\sum_{i=1}^{D} w_i^2 + \frac{\lambda}{2}b^2 + C(\lambda)$$

where $C(\lambda)$ is a term that depends on λ but not on either $\mathbf{w}$ or b. What are the derivatives of L with respect to each of the model parameters?

2 Logistic Regression vs. KNN (25 points)

In this section you will compare the performance and characteristics of different classifiers, namely Logistic Regression and k-Nearest Neighbors. You will extend the provided code and experiment with these extensions. Note that you should understand the code first instead of using it as a black box.

Both Matlab and Python[1] versions of the code have been provided. You are free to work with whichever you wish.

The data you will be working with are hand-written digits, 4s and 9s, represented as 28x28 pixel arrays. There are two training sets: `mnist_train`, which contains 80 examples of each class, and `mnist_train_small`, which contains 5 examples of each class. There is also a validation set `mnist_valid` that you should use for model selection, and a test set `mnist_test`.

Code for visualizing the datasets has been included in `plot_digits`.

2.1 (5 points) ***k*-Nearest Neighbors**

Use the supplied kNN implementation to predict labels for `mnist_valid`, using `mnist_train` as the training set.

Write a script that runs kNN for different values of $k \in \{1, 3, 5, 7, 9\}$ and plots the classification rate on the validation set (number of correctly predicted cases, divided by total number of data points) as a function of k.

Comment on the performance of the classifier and argue which value of k you would choose. What is the classification rate for k^*, your chosen value of k? Also compute the rate for $k^* + 2$ and $k^* - 2$.

[1] If you choose to work with Python, you should use Python 2.7 with both the Numpy and Matplotlib packages installed.

Does the test performance for these values of k correspond to the validation performance?[2] Why or why not?

2.2 (10 points) **Logistic regression**

Look through the code in `logistic_regression_template` and `logistic`. Complete the implementation of logistic regression by providing the missing part of `logistic`. Use `checkgrad` to make sure that your gradients are correct.

Run the code on both `mnist_train` and `mnist_train_small`. You will need to experiment with the hyperparameters for the learning rate, the number of iterations (if you have a smaller learning rate, your model will take longer to converge), and the way in which you initialize the weights. If you get Nan/Inf errors, you may try to reduce your learning rate or initialize with smaller weights.

Report which hyperparameter settings you found worked the best and the final cross entropy (also called log loss) and classification error on the training, validation and test sets. Note that you should only compute the test error once you have selected your best hyperparameter settings using the validation set.

Next look at how the cross entropy changes as training progresses. Submit 2 plots, one for each of `mnist_train` and `mnist_train_small`. In each plot show two curves one for the training set and one for the validation set. Run your code several times and observe if the results change. If they do, how would you choose the best parameter settings?

2.3 (10 points) **Penalized logistic regression**

Implement the penalized logistic regression model you derived in **1.3** by modifying `logistic` to include a regularizer. Call the new function `logistic_pen`. You should only penalize the weights and not the bias term, as it only controls the height of the function but not its complexity. Note that you can omit the $C(\lambda)$ term in your error computation, since its derivative is 0 w.r.t. the weights and bias. Use `checkgrad` to verify the gradients of your new `logistic_pen` function.

Repeat part 2.2, but now with different values of the penalty parameter λ. Try $\lambda \in \{0.001, 0.01, 0.1, 1.0\}$. At this stage you should not be evaluating on the test set as you will do so once you have chosen your best λ.

To do the comparison systematically, you should write a script that includes a loop to evaluate different values of λ automatically. You should also re-run logistic regression at least 10 times for each value of λ.

So you will need two nested loops: The outer loop is over values of λ. The inner loop is over multiple re-runs. Average the evaluation metrics (cross entropy and clasification error) over the different re-runs. In the end, plot the average cross entropy and classification error against λ. So for each of `mnist_train` and `mnist_train_small` you will have 2 plots. One plot for cross entropy and another plot for classification error. Each plot will have two curves - one for training and one for validation.

How do the cross entropy and classification error change when you increase λ? Do they go up, down,

[2] In general you shouldn't peek at the test set multiple times, but for the purposes of this question it can be an illustrative exercise.

first up and then down, or down and then up? Explain why you think they behave this way. Which is the best value of λ, based on your experiments? Report the test error for the best value of λ.

Compare the results with and without penalty. Which one performed better for which data set? Why do you think this is the case?