CSC 411
Machine Learning & Data Mining
ASSIGNMENT # 1
Out: Sep. 30
Due: Oct. 14 [10am Toronto time]

1 Logistic Regression (30 points)

Suppose you are given a training set with N training examples, $\mathcal{D} = \{(\mathbf{x}^{(1)}, t^{(1)}), \dots, (\mathbf{x}^{(N)}, t^{(N)})\}$, where each input $\mathbf{x}^{(i)}$ is a D-dimensional vector $\mathbf{x}^{(i)} = (x_1^{(i)}, \dots, x_D^{(i)})^T$ and the targets are binary, i.e., $t^{(i)} \in \{0, 1\}$. Assume a model such that for each example $p(t = 1|\mathbf{x}^{(i)}, \mathbf{w}, w_0)$ takes the form of a logistic function:

$$p(t = 1|\mathbf{x}^{(i)}, \mathbf{w}, w_0) = \sigma(\mathbf{w}^T\mathbf{x}^{(i)} + w_0) = \frac{1}{1 + \exp\left(-\sum_{d=1}^{D} w_d \mathbf{x}_d^{(i)} - w_0\right)}$$

The likelihood function is defined as $p(t^{(1)}, t^{(2)}, \dots, t^{(N)}|\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N)}, \mathbf{w}, w_0)$

Further, assume that for regularization purposes, a Gaussian prior is placed on the weights $\mathbf{w} = (w_1, \dots, w_D)^T$ such that $p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|0, \alpha^{-1}\mathbf{I})$.

1. Define the loss function to be the negative loss posterior over the weights. Write the loss function as simplified as possible (show all your derivations). In order to do so, assume that the data is i.i.d (i.e., every example is drawn independently and its identically distributed).
2. Derive what the gradient of the loss is with respect to w_i and w_0, i.e., $\frac{\partial Loss}{\partial w_i}$, $\frac{\partial Loss}{\partial w_0}$.
3. Finally, write the pseudocode for gradient descent using those gradients.

2 Decision Trees (16 points)

Assume you are given data consisting of a training set of 5 examples (Table 1) and a test set of 3 examples (Table 2).

1. Using the training data, construct a decision tree for the binary classification of customers of the restaurant "Mama's Pasta" into 'Satisfied' or 'Unsatisfied'. Use the Information Gain (IG) as the decision criterion to select which attribute to split on. Show your calculations for the IG for all possible attributes for every split.
2. Now use the decision tree you have created to predict whether each of the test users will be satisfied or not after their visit to "Mama's Pasta".

Person ID	Overcooked pasta?	Waiting time	Rude waiter?	Satisfied?
1	Yes	Long	No	Yes
2	No	Short	Yes	Yes
3	Yes	Long	Yes	No
4	No	Long	Yes	Yes
5	Yes	Short	Yes	No

Table 1: Training Data

Person ID	Overcooked pasta?	Waiting time	Rude waiter?
6	No	Short	No
7	Yes	Long	Yes
8	Yes	Short	No

Table 2: Test Data

3 Logistic Regression vs. KNN (54 points)

In this section you will compare the performance and characteristics of different classifiers, namely k-Nearest Neighbors and Logistic Regression. 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. Python[1] version of the code has been provided.

The data you will be working with are hand-written digits, 2s and 8s, represented as 28x28 pixel arrays. There are two training sets: `mnist_train`, which contains 100 examples of each class, and `mnist_train_small`, which contains 5 examples of each class. There is also a validation set `mnist_valid` (with 25 examples each class) that you should use for model selection, and a test set `mnist_test`.

Code for visualizing the datasets has been included in `plot_digits`. `check_grad` can be used for checking gradient computation. You can also find some help functions in `utils` including loading data.

2.1 k-Nearest Neighbors (17 points)

Implement the kNN function inside `run_knn`, you can use the utility function from `l2_distance` to compute distance between different data points.

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

1. Comment on the performance of the classifier and argue which value of k you would choose. What is the classification rate on test set of your chosen value k^*? Also compute the rate for

[1]You should use Python 2.7 with both the Numpy and Matplotlib packages installed.

$k^* + 2$ and $k^* - 2$ on test set.

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

2.2 Logistic regression (17 points)

Look through the code in `logistic_regression_template` and `logistic`. Complete the implementation of logistic regression by providing the missing part of those two files.

Write a separate script or use `logistic_regression_template` so you can experiment with the hyperparameters for the learning rate, the number of iterations, and the way in which you initialize the weights.

As a practice guide, before training, you should use `check_grad` to make sure your implementation of gradient computation is correct; during training, if you get Nan/Inf errors, you may try to reduce your learning rate or initialize with smaller weights. If you have a smaller learning rate, your model will take longer to converge.

1. Report which hyperparameter settings you found worked the best and the final cross entropy and classification error on the training (`mnist_train`), 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.
2. using the your chosen hyperparameter, report how the loss (cross entropy) changes during training. You need to 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. Comment on your results. Run your code several times and observe if the results change, why or why not?

2.3 Regularized logistic regression (20 points)

Implement the penalized logistic regression model you derived in Section **1** by modifying `logistic` to include a regularizer (implement the new function `logistic_pen`). You will need to change the loss function and the corresponding gradient. In your gradient computation, note that you can omit the $C(\alpha)$ term, since its derivative is 0 w.r.t. the weights and bias. Use `check_grad` to verify the gradients of your new `logistic_pen` function.

Write a scripts to do experiments with different hyperparameters including different values of the penalty parameter α. Try $\alpha \in \{0.001, 0.01, 0.1, 1.0\}$. At this stage you should not be evaluating on the test set as you will do so once you have chosen your best α.

1. Similar to Section **2.2**, report the best hyperparameters and corresponding test set error. You will also need to submit two plots for training on `mnist_train` and `mnist_train_small`,

[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.

each should also include two curves of how the loss (negative log posterior) changes, one for training set and the other for validation set.

2. Given other hyperprameters fixed, how does the loss (negative log posterior) and classification error change when you increase α? Do they go up, down, first up and then down, or down and then up? Explain why you think they behave this way. Which is the best value of α, based on your experiments?

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

4. Compare the results (with and without regularizer) against the results with kNNs, what do you find? In general, how would choose one method over the other? What's the pros and cons for logistic regression and kNNs?

Submission

Package your code and your answers to all questions using *zip* or *tar.gz* in a file called `CSC411-A1-*your-stude`
For the code, only include functions and scripts that you modified.

Submit the code on MarkUs by October 14, 2016, 10am.