CS 475 Machine Learning: Project 2
Analytical Part

Yichuan Tang

9th Mar 2018

1) Decision Tree and Logistic Regression (10 points) Consider a binary classification task (label y) with four features (x):

x_1	x_2	x_3	x_4	y
0	1	1	-1	1
0	1	1	1	0
0	-1	1	1	1
0	-1	1	-1	0

(a) Can this function be learned using a decision tree? If so, provide such a tree (describe each node in the tree). If not, prove it.

Answer: This function can be learned using decision tree. The first node on first level is x_2, the first and second nodes on second level are x_4 and x_4.

(b) Can this function be learned using a logistic regression classifier? If yes, give some example parameter weights. If not, why not.

Answer: This function cannot be learned by using logistic regression classifier. The hypothesis used in logistic regression is that the probability that label is 1 equals:

$$p(y = 1) = \frac{1}{1 + e^{-w^T x}}$$

If we want to use logistic regression to learn with given data and label, the $w^T x_1$ and $w^T x_3$ should be larger than zero and at the same time $w^T x_2$ and $w^T x_4$ should be smaller than zero, assume $w = (w_1, w_2, w_3, w_4)^T$. However, $w^T x_1 > 0$ and $w^T x_2 < 0$ suggests that $w_4 < 0$, but $w^T x_3 > 0$ and $w^T x_4 < 0$ suggests $w_4 > 0$, which is a contradiction. Therefore we cannot find a suitable w to do logistic regression.

(c) For the models above where you can learn this function, the learned model may over-fit the data. Propose a solution for each model on how to avoid over-fitting.

Answer: Here only decision tree model is discussed as logistic regression cannot be used. We can randomly flip Y with a certain probability (like adding noisy data to make the predictor more robust) and duplicate training data into test. The tree we have in this question is too small, for larger trees pruning can also help to avoid overfitting.

2) Stochastic Gradient Descent (10 points) In the programming part of this assignment you implemented Gradient Descent. A stochastic variation of that method (Stochastic Gradient Descent) takes an estimate of the gradient based on a single sampled example, and takes a step based on that gradient. This process is repeated many times until convergence. To summarize:

1. Gradient descent: compute the gradient over all the training examples, take a gradient step, repeat until convergence.
2. Stochastic gradient descent: sample a single training example, compute the gradient over that training example, take a gradient step, repeat until convergence.

In the limit, will both of these algorithms converge to the same optimum or different optimum? Answer this question for both convex and non-convex functions. Prove your answers.

Answer:
For convex functions, both GD and SDG converges to same optimum. For a convex function, there is only one optimum, thus using GD and SGD are actually equivalent, they will finally find the optimum, only the path to optimum using two algorithms would be different.

For non-convex functions, two algorithm may end up finding same optimum, or finding difference optimum, both are likely. as SGD has a more aggressive and unstable update scheme than GD, therefore it has the ability to go across local optimum, and find better optimum than using GD (given same initialization condition). If we are lucky and both algorithms are initialized near the global optimum, they may end up with finding same global optimum.

3) Kernel Trick (10 points) The kernel trick extends SVMs to learn nonlinear functions. However, an improper use of a kernel function can cause serious over-fitting. Consider the following kernels.

(a) Inverse Polynomial kernel: given $\|x\|_2 \leq 1$ and $\|x'\|_2 \leq 1$, we define $K(x, x') = 1/(d - x^\top x')$, where $d \geq 2$. Does increasing d make over-fitting more or less likely?

Answer: For this inverse polynomial kernel, the dot product between training and testing data indicated the similarity between these two entries. If d in increased, then this similarity will not dominate, thus has less influence on the value of kernel. The over-fitting is less likely in this case.

(b) Chi squared kernel: Let x_j denote the j-th entry of x. Given $x_j > 0$ and $x'_j > 0$ for all j, we define $K(x, x') = \exp\left(-\sigma \sum_j \frac{(x_j - x'_j)^2}{x_j + x'_j}\right)$, where $\sigma > 0$. Does increasing σ make over-fitting more or less likely?

Answer: For this Chi squared kernel, its formulation could be changed into:

$$K(x, x') = \exp \prod_j \left(-\frac{(x_j - x'_j)^2}{\frac{x_j + x'_j}{\sigma}} \right)$$

Which has a similar form as Gaussian kernel, and if σ is increased, this kernel makes over-fitting more likely.

We say K is a kernel function, if there exists some transformation $\phi : \mathbb{R}^m \to \mathbb{R}^{m'}$ such that $K(x_i, x_{i'}) = \langle \phi(x_i), \phi(x_{i'}) \rangle$.

(c) Let K_1 and K_2 be two kernel functions. Prove that $K(x_i, x_{i'}) = K_1(x_i, x_{i'}) + K_2(x_i, x_{i'})$ is also a kernel function.

Answer:

$$K_1(x_i, x_{i'}) = \langle \phi_1(x_i), \phi_1(x_{i'}) \rangle$$
$$K_2(x_i, x_{i'}) = \langle \phi_2(x_i), \phi_2(x_{i'}) \rangle$$
$$K_1(x_i, x_{i'}) + K_2(x_i, x_{i'}) = \langle \phi_3(x_i), \phi_3(x_{i'}) \rangle$$
$$\phi_3(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \end{bmatrix}$$

Above formula show the sum of two kernels can be represented by another kernel.

4) Dual Perceptron (8 points)

(c) You train a Perceptron classifier in the primal form on an infinite stream of data. This stream of data is not-linearly separable. Will the Perceptron have a bounded number of prediction errors?

Answer: Perceptron (primal form) will have non-bounded number of prediction errors when dealing with data stream which is not-linearly separable.

(c) Switch the primal Perceptron in the previous step to a dual Perceptron with a linear kernel. After observing T examples in the stream, will the two Perceptrons have learned the same prediction function?

Answer: The two perceptrons will learn the same prediction function, because the dual form is still using linear kernel.

(c) What computational issue will you encounter if you continue to run the dual Perceptron and allow T to approach ∞? Will this problem happen with the primal Percepton? Why or why not?

Answer:
When T approaches infinity, the memory may overflow as all data received are used to make prediction in dual perceptron, also computation will become very slow.

This will not happen in primal form. Primal form uses each piece of incoming data to update model parameters, but this piece of data will not be used in future, thus memory doesn't need to keep them, also computation time in each updating iteration is constant.

5) Convex Optimization (12 points) Jenny at Acme Inc. is working hard on her new machine learning algorithm. She starts by writing an objective function that captures her thoughts about the problem. However, after writing the program that optimizes the objective and getting poor results, she returns to the objective function in frustration. Turning to her colleague Matilda, who took CS 475 at Johns Hopkins, she asks for advice. "Have you checked that your function is convex?" asks Matilda. "How?" asks Jenny.

(a) Jenny's function can be written as $f(g(x))$, where $f(x)$ and $g(x)$ are convex, and $f(x)$ is non-decreasing. Prove that $f(g(x))$ is a convex function. (Hint: You may find it helpful to use the definition of convexity. Do not use gradient or Hessian, since f and g may not have them.)

Answer: For a convex function $f(x)$ we have,

$$f(\theta * x + (1-\theta) * y) \leq \theta * f(x) + (1-\theta) * f(y)$$

$$(0 \leq \theta \leq 1)$$

Then we have for the combined function $h(x) = f(g(x))$,

$$h(\theta * x + (1-\theta) * y) = f(g(\theta * x + (1-\theta) * y))$$

$$g(\theta * x + (1-\theta) * y) \leq \theta * g(x) + (1-\theta) * g(y)$$

We know that $f(x)$ is non-decreasing which means,

$$f(g(\theta * x + (1-\theta) * y)) \leq f(\theta * g(x) + (1-\theta) * g(y))$$

$$f(\theta * g(x) + (1-\theta) * g(y)) \leq \theta * f(g(x)) + (1-\theta) * f(g(y))$$

$$f(g(\theta * x + (1-\theta) * y)) \leq \theta * h(x) + (1-\theta) * f(y)$$

$$h(\theta * x + (1-\theta) * y) \leq \theta * h(x) + (1-\theta) * f(y)$$

Above equation indicates that $h(x) = f(g(x))$ is also convex.

(b) Jenny realizes that she made an error and that her function is instead $f(x) - g(x)$, where $f(x)$ and $g(x)$ are convex functions. Her objective may or may not be convex. Give examples of functions $f(x)$ and $g(x)$ whose difference is convex, and functions $\bar{f}(x)$ and $\bar{g}(x)$ whose difference is non-convex.

Answer:
$f(x) = 2x^2$ and $g(x) = x^2$, their difference is x^2, which is a convex function.
$\bar{f}(x) = x^4$ and $\bar{g}(x) = x^2$, their difference is $x^4 - x^2$, which is not convex.

"I now know that my function is non-convex," Jenny says, "but why does that matter?"

(c) Why was Jenny getting poor results with a non-convex function?

Answer: Jenny's optimization algorithm may get stuck in the local minimum of that non-convex function, and the global minimum (although exist) cannot be reached.

(d) One approach for convex optimization is to iteratively compute a descent direction and take a step along that direction to have a new value of the parameters. The choice of a proper stepsize is not so trivial. In gradient descent algorithm, the stepsize is chosen such that it is proportional to the magnitude of the gradient at the current point. What might be the problem if we fix the stepsize to a constant regardless of the current gradient? Discuss when stepsize is too small or too large.

Answer:
When we have steep gradient, using a small stepsize is okay. In most cases, it will finally find the optimum, but cost more time than using a larger stepsize.
If current position is nearby the optimum point and we are using a large stepsize, it will probably go across the optimum point. It would move back and forth around the optimum point without reaching it.