Homework 2

Aleck Zhao

March 9, 2018

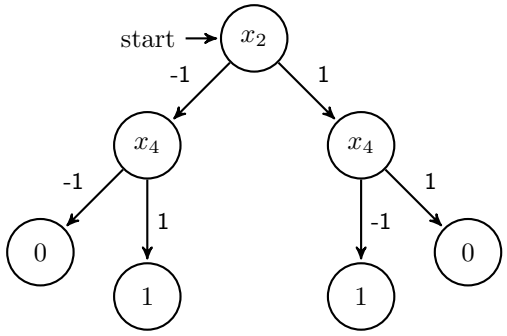
Analytical (50 points)

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.

Solution. Yes, this function can be learned using a decision tree because it is a proper boolean function, that is, $y(x_1) = y(x_2)$ whenever $x_1 = x_2$. The decision tree is given as



□

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

Solution. Suppose this function can be learned using a logistic regression classifier, and let the parameter weight be $\mathbf{w} = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 \end{bmatrix}$. Suppose we label the examples from the table as $x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$. Then if g represents the logistic function, we predict $y^{(i)} = 1$ when $g\left(\mathbf{w} \cdot x^{(i)}\right) > 0.5$ and 0 otherwise.

$$\frac{1}{1+e^{\mathbf{w}\cdot x^{(1)}}} = \frac{1}{1+e^{w_2+w_3-w_4}} > \frac{1}{2} \implies e^{w_2+w_3-w_4} < 1 \implies w_2 + w_3 - w_4 < 0$$

$$\frac{1}{1+e^{\mathbf{w}\cdot x^{(2)}}} = \frac{1}{1+e^{w_2+w_3+w_4}} \leq \frac{1}{2} \implies e^{w_2+w_3+w_4} \geq 1 \implies w_2 + w_3 + w_4 \geq 0$$

$$\frac{1}{1+e^{\mathbf{w}\cdot x^{(3)}}} = \frac{1}{1+e^{-w_2+w_3+w_4}} > \frac{1}{2} \implies e^{-w_2+w_3+w_4} < 1 \implies -w_2 + w_3 + w_4 < 0$$

$$\frac{1}{1+e^{\mathbf{w}\cdot x^{(4)}}} = \frac{1}{1+e^{-w_2+w_3-w_4}} \leq \frac{1}{2} \implies e^{-w_2+w_3-w_4} \geq 1 \implies -w_2 + w_3 - w_4 \geq 0$$

Here, if we add equations 1 and 3, and equations 2 and 4, respectively, then we get $2w_3 < 0$ and $2w_3 \geq 0$, a contradiction. Thus, there do not exist parameter weights that can describe this function. □

(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. For decision tree learning, we can use fewer splits, so for example only split based on x_2.

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.

Solution. In the case of a convex function, there is a single global minimum. Since both gradient descent and stochastic gradient descent converge to a local minimum, they will both converge to the same (and only) global minimum.

In the case of a non-convex function, stochastic gradient can escape the local minimum, whereas gradient descent does not (assuming the learning rate is properly adjusted). Thus, they will not converge to the same local optimum in general. □

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?

Solution. As d increases, K decreases. However, if d is large then $K(x, x')$ is small no matter what x and x' are, so the kernel is very inflexible, and thus will likely lead to under-fitting. □

(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?

Solution. Increasing σ decreases the value of the kernel function, so the support vectors have a smaller influence area, which increases the chance of over-fitting. K is flexible because if x and x' are close, then the value of $K(x, x')$ has a large range, and thus is more prone to over-fitting. □

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.

Proof. Since K_1 and K_2 are kernel functions, there exist transformations $\varphi, \theta : \mathbb{R}^m \to \mathbb{R}^{m'}$ such that $K_1(x_i, x_{i'}) = \langle \varphi(x_i), \varphi(x_{i'}) \rangle$ and $K_2(x_i, x_{i'}) = \langle \theta(x_i), \theta(x_{i'}) \rangle$. Then if $\gamma(x_i) = \begin{bmatrix} \varphi(x_i) & \theta(x_i) \end{bmatrix}$ be φ concatenated with θ, then we have

$$\begin{aligned} K(x_i, x_{i'}) &= K_1(x_i, x_{i'}) + K_2(x_i, x_{i'}) = \varphi(x_i)^T \varphi(x_{i'}) + \theta(x_i)^T \theta(x_{i'}) \\ &= \begin{bmatrix} \varphi(x_i) & \theta(x_i) \end{bmatrix}^T \begin{bmatrix} \varphi(x_{i'}) & \theta(x_{i'}) \end{bmatrix} \\ &= \langle \gamma(x_i), \gamma(x_{i'}) \rangle \end{aligned}$$

so K is also a kernel function. □

4) Dual Perceptron (8 points)

(a) 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?

Solution. If the perceptron had a bounded number of prediction errors, then there exists $N \in \mathbb{N}$ such that the perceptron correctly classifies every example x_n for all $n \geq N$. At this point, $\mathbf{w}$ will not be changing anymore, so it follows that the data is linearly separable, a contradiction. Thus, the number of prediction errors is unbounded. □

(b) 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?

Solution. In the dual formulation, we have $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i x_i$ where $\alpha_i = 1$ if x_i was misclassified, and 0 otherwise. Using the linear kernel, we have

$$\hat{y}_d(x_{T+1}) = \text{sign}\left(\sum_{i=1}^{T} \alpha_i y_i x_i^T x_{T+1}\right)$$

$$\hat{y}(x_{T+1}) = \text{sign}\left(w^T x\right) = \text{sign}\left[\left(\sum_{i=1}^{T} \alpha_i y_i x_i\right)^T x_{T+1}\right] = \text{sign}\left(\sum_{i=1}^{T} \alpha_i y_i x_i^T x_{T+1}\right)$$

so the two functions are the same. □

(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?

Solution. In the dual formulation, we have $\hat{y}(x_{n+1}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i x_i^T x\right)$, where $\alpha_i = 1$ if example i was classified wrong, and 0 otherwise, since that is when w is updated each time. As $T \to \infty$, this becomes a sum with more and more terms since the data is not linearly separable, which will become computationally difficult. The primal perceptron will not have this problem because the number of features of $\mathbf{w}$ is not changing, so at each example only a dot product needs to be computed. □

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.)

Proof. Let $t \in [0, 1]$ and let x, y be in the domain of g. Then since g is convex, we have

$$g\left[tx + (1-t)y\right] \leq tg(x) + (1-t)g(y)$$

and since f is non-decreasing, that means $f(a) \leq f(b)$ whenever $a \leq b$, so it follows that

$$f\big(g\left[tx + (1-t)y\right]\big) \leq f\left[tg(x) + (1-t)g(y)\right]$$

and since f is convex, we finally have

$$f\left[tg(x) + (1-t)g(y)\right] \leq tf(g(x)) + (1-t)f(g(y))$$

so $f(g(x))$ is 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.

Solution. Let $f(x) \equiv 3$ and $g(x) \equiv 2$. Then f and g are both convex, then $f - g \equiv 1$ which is convex. Let $\bar{f}(x) \equiv 0$ and $\bar{g}(x) = x^2$. Then f and g are both convex, but $\bar{f} - \bar{g} = -x^2$, 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. Non-convex functions will not have a single, distinct global minimum. We might get stuck at a local minimum that isn't the global minimum.

(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. If the step size is constant, we may run into the issue where we end up oscillating around the global minimum. If the step size is too small, it may take too long to converge, and if the step size is too large, we may overshoot the minimum