1 Analytical (50 points)

1) Decision Tree and Logistic Regression (8 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.

Yes this function can be learned using a decision tree. For example, using prefix notation:

x_1
—L: x_3 with path $x_1 = 0$
——LL: x_2 with path $x_3 = 1$
———LLL: x_4 with path x_2 = -1
————LLLL: output 0 with path x_4 = -1
————LLLR: output 1 with path $x_4 = 1$
———LLR: x_4 with path $x_2 = 1$
————LLRL: output 1 with path x_4 = -1
————LLRL: output 0 with path $x_4 = 1$

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

No this function cannot be learned using a logistic regression classifer because when we use the link function $g(z) = \frac{1}{(1+e^{-wx})}$ we cannot assign weights that will be able to take in the input combination of $x_2 = 1$, $x_4 = -1$, then $y = 1$ with the combination $x_2 = -1$, $x_4 = 1$, then $y = 1$ since the input values are essentially negative copies of each other, and yet the output y must still be the same. Furthermore, the inputs for x_1 and x_3 are always 0 and always 1 respectively, so they do not resolve this problem. That is to say, they cannot be assigned weights to correctly describe this function even in combination with x_2 and x_4. Therefore, the function here cannot be learned by a logistic regression classifier.

(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.

For the decision tree model, you can split the data into training and testing data to avoid over-fitting. This will reduce the depth and complexity of the decision tree and will thus decrease the training accuracy, but it will make the decision tree more generalizable. Furthermore, you could also set a manual limit for the maximum depth of the tree so that it does not over-fit itself on the training data.

2) Stochastic Gradient Descent (8 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.

For convex functions, in the limit, both of these algorithms will converge to the same optimum because convex functions are guaranteed to have a minimum. Even if the points for gradient descent are chosen stochastically (randomly) they will all point to the minimum. So the convex function will still reach the same optimum.

For non-convex functions, in the limit, the algorithms may converge to different optimum. This is because non-convex functions may have multiple minima. Thus, while the gradient descent algorithm will always end up at one optimum, the stochastic gradient descent algorithm may randomly choose different points and could end up at a different minimum.

3) Regularizer of Regression (10 points) In linear regression we want to minimize the sum of square loss

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} ||y - X\beta||_2^2 \tag{1}$$

To address overfitting, we might plug in a regularizer $||\beta||_q$ as:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} ||y - X\beta||_2^2 + \lambda||\beta||_q \tag{2}$$

where $||\cdot||_q$ is the q-norm and λ is the regularization parameter

(a) What is the effect on β when $q = 0$, $q = 1$ and $q = 2$? Explain why this is the case.

When $q = 0$, the regularizer becomes the intersection of the axes at 0. In order to find the intersection of the unregularized error function with this regularization function, each term of β will be forced to 0. This leads to a trivial solution which is not useful.

When $q = 1$, the regularizer takes the form of a lasso where the edge points are along an axis. Thus, since we are trying to find the intersection of the unregularized error function with the regularization function, this means that the intersection will be along an axis where some of the coefficients of β will be driven to 0. This leads to a more sparse model for β.

When $q = 2$, we get a quadratic regularizer which has a more circular form. This allows us to minimize the square error by setting weights in β without changing many of them to zero. Thus, we can regularize the function without changing it significantly as with the lasso regularizer $q = 1$ where coefficients in β go to zero.

(b) What is the effect of λ in terms of variance/bias trade-off? How do we usually select a proper λ?

As λ increases, this is less variance and more bias. As λ decreases, the variance increases and there is less bias.

Thus, we choose λ to have a balance of variance and bias so that we have high accuracy (using bias) while still maintaining generalizability to new data (using variance). By separating data into train, dev, and test data and using cross-validation, we can adjust and choose λ to find a good balance of variance and bias.

(c) Suppose each example has three features and the corresponding parameters are β_0, β_1 and β_2. If we formulate the regularizer as $||\beta_{\{0,1\}}||_2 + |\beta_2|$, where $\beta_{\{0,1\}}$ is a 2 dimensional vector containing the first two elements of β. Describe the effect of this regularizer.

This regularizer seeks to keep the values of the features β_0, β_1 close together while also minimizing the value of β_2. So it essentially correlates β_0 and β_1 and drives the value of β_2 to 0 which means the model will be biased towards β_0 and β_1.

4) Constructing Generalized linear models. (12 points) Generalized linear models (GLMs), especially logistic regression are heavily used by banks, credit card companies and insurance companies. Actually, when you apply for a credit card, banks may put your information into a logistic regression model to decide whether you are eligible.

(a) The GLMs are closely related to the exponential distribution family, which has the probability density/mass function $f(y;\theta)$ in the form

$$f(y;\theta) = H(y)e^{\eta(\theta)\cdot T(y)-A(\theta)}, \tag{3}$$

where H, η, T, A are some known functions.

Consider a classification or regression problem where we would like to predict the value of some random variable y as a function of x. To derive a GLM for this problem, we will make the following three assumptions about the conditional distribution of y given x and about our model:

1. $y|x;\theta \sim ExponentialFamily(\eta)$. I.e., given x and θ, the distribution of y follows some exponential family distribution, with parameter η.
2. Given x, our goal is to predict the expected value of $T(y)$ given x. In most of our examples, we will have $T(y) = y$, so this means we would like the prediction $h(x)$ output by our learned hypothesis h to satisfy $h(x) = E[y|x]$.
3. The natural parameter η and the inputs x are related linearly: $\eta = \mathbf{w}^T x$

Derive the expression of logistic regression from the Bernoulli distribution:

$$h_{\mathbf{w}}(x) = \frac{1}{1+\exp(-\mathbf{w}^T x)} \tag{4}$$

by following the above three assumptions.

$$\begin{aligned} P(y|h) &= h^y * (1-h)^{1-y} \\ &= exp(\ln(h^y(1-h)^{1-y})) \\ &= exp(y\ln h + (1-y)\ln(1-h)) \\ &= exp(y\ln h + \ln(1-h) - y\ln(1-h)) \\ &= exp(y\ln\frac{h}{1-h} + \ln(1-h)) \end{aligned}$$

to make it in the form of the exponential distribution family

$$f(y;\theta) = H(y)e^{\eta(\theta)\cdot T(y)-A(\theta)},$$

we have

$$H(y) = 1$$

$$A(\theta) = \ln(1-h)$$

$$\eta(\theta) * T(y) = y\ln\frac{h}{1-h}$$

so

$$T(y) = y$$

and

$$\eta = w^T x$$

So solving for h

$$\eta(\theta) = \ln \frac{h}{1-h}$$

$$e^{\eta(\theta)} = \frac{h}{1-h}$$

$$\frac{1-h}{h} = \frac{1}{h} - 1 = e^{-\eta(\theta)}$$

$$\frac{1}{h} = 1 + e^{-\eta(\theta)}$$

$$h = \frac{1}{1+e^{-\eta(\theta)}}$$

$$h = \frac{1}{1+e^{-w^T x}}$$

finally

$$h_{\mathbf{w}}(x) = \frac{1}{1+\exp(-\mathbf{w}^T x)}$$

(b) The GLMs often contain some transformation, which is non-linear such as the log-odds-ratio transformation in the logistic regression. Why do we still call them "linear"?

Even though the transformation may be non-linear, the linear model is still related to the input variable through that link function. Therefore, the predicted output for each input is still a linear function of the given input. So the overall response still varies linearly despite the use of non-linear transformations.

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.)

A function $f(x)$ is convex if for all x and y and for all $0 \leq \lambda \leq 1$, we have

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$$

Thus we need to show f(g(x)) is convex by showing

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

By definition of g convex,

$$g(\lambda x + (1-\lambda)y) \leq \lambda g(x) + (1-\lambda)g(y)$$

and since f is a non-decreasing and convex function,

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

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

and since λ is a constant

$$= \lambda f(g(x)) + (1-\lambda)f(g(y))$$

which is what we needed to show.

(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.

Let $f(x) = 5x^2$ and let $g(x) = 4x^2$ which are both convex. Then $f(x) - g(x) = 5x^2 - 4x^2 = x^2$ which is also convex.

Now suppose $\bar{f}(x) = x^4$ and $\bar{g}(x) = x^2$ which are both convex. Then $\bar{f}(x) - \bar{g}(x) = x^4 - x^2 = x^2(x^2 - 1) = x^2(x + 1)(x - 1)$ where there are two local minima: one at $x = -1$ and one at $x = 1$ and so the difference is non-convex since a graph of the function shows the two local minima.

"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?

Since her function was non-convex, her gradient descent algorithm might not converge. Thus, when solving for the w weight vectors, Jenny's algorithm does not converge to find the maximum likelihood estimate. Instead, it may just pick any estimate which would lead to poor accuracy in prediction and thus poor results.

Also, since her function is non-convex, there may be multiple minima. Thus, her gradient descent algorithm may find a local minima that is not as good as a global minima. Thus, her algorithm finds a more likely estimate instead of finding the maximum likelihood estimate.

(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.

When the stepsize is fixed to a constant regardless of the gradient, it is possible that the gradient descent algorithm will not converge. For example, if the stepsize is too small, the gradient might also be very small and so the function will essentially be stuck and will not be able to move to reach the minima. On the other hand, if the stepsize is too large, the gradient might also be large and thus the gradient descent algorithm may jump back and forth past the minima and would not converge to the optimum.