CS 189: Introduction to Machine Learning

Spring 2018

•

Homework 8

Due on Friday, March 16th, 2018 at 10pm

•

Solutions by

Firstname Lastname

StudentID

In collaboration with

None

Problem 1: Getting Started

Read through this page carefully. You may typeset your homework in latex or submit neatly handwritten/scanned solutions. Please start each question on a new page. Deliverables:

1. Submit a PDF of your writeup, **with an appendix for your code**, to assignment on Gradescope, "HW8 Write-Up". If there are graphs, include those graphs in the correct sections. Do not simply reference your appendix.
2. If there is code, submit all code needed to reproduce your results, "HW8 Code".
3. If there is a test set, submit your test set evaluation results, "HW8 Test Set".

After you've submitted your homework, watch out for the self-grade form.

a. Who else did you you work with on this homework? In case of course events, just describe the group. How did you work on this homework? Any comments about the homework?

Solution

b. Please copy the following statement and sign next to it. We just want to make it *extra* clear so that no one inadvertently cheats.

I certify that all solutions are entirely in my words and that I have not looked at another student's solutions. I have credited all external sources in this write up.

Solution

Problem 2: SGD on OLS

In this problem, we carefully walk through the key ingredients of a proof for SGD convergence for the specific example of a loss function which we are very familiar with: Ordinary Least Squares with positive definite matrix $\mathbf{X}^\top\mathbf{X}$. In particular we show that in this case, even though gradient descent converges to the optimal solution of OLS, with small enough constant step-size, SGD is not guaranteed to do so! We then show that in contrast, when applying SGD with decaying step, as outlined in lecture, it does converge to the same optimum as gradient descent.

This phenomenon that constant step-size gets stuck and decaying step-sizes are the way out is analogous to common observations in practical machine learning problems, where decreasing the learning rate using heuristic learning rate schedules helps to decrease the training error.

Recall that the ordinary least squares problem can be written as:

$$\min_{\mathbf{w}} f(\mathbf{w}) = \min_{\mathbf{w}} \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 = \min_{\mathbf{w}} = \frac{1}{2}\sum_{i=1}^{n}\left(\mathbf{x}_i^\top\mathbf{w} - \mathbf{y}_i\right)^2 = \min_{\mathbf{w}} \frac{1}{2}\sum_{i=1}^{n} f_i(\mathbf{w})$$

where $f-(\mathbf{w}) := \left(\mathbf{x}_i^\top\mathbf{w} - \mathbf{y}_i\right)^2$. Here $\mathbf{x}_i^\top$ is the ith row of matrix $\mathbf{X}$ and f_i is the loss of the training example i. We implement SGD as follows:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \alpha_t \nabla f_{i_t}\left(\mathbf{w}^t\right)$$

where i_t is uniformly sampled from all samples $\{1, 2, \ldots, n\}$ (and is independently drawn for each iteration t). Let?s define the short hand $G_t = \nabla f_{i_t}(\mathbf{w}^t)$ to represent the random gradient at step t. We are interested in how $\mathbf{w}^t$ approaches the optimal solution $\mathbf{w}^* := \arg\min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w})$. One way to characterize this is to monitor the squared distance of the iterate to the optimum, i.e. $\|\mathbf{w}^t - \mathbf{w}^*\|_2^2$. Throughout, we will assume that the following bound holds for the squared norm of the stochastic gradient:

$$\mathbb{E}\|G(\mathbf{w})\|_2^2 \leqslant M_g^2\|\mathbf{w} - \mathbf{w}^*\|_2^2 + B^2 \tag{1}$$

where M_g and B are are constants dependent on the model and loss function f. We will find concrete values of them later in this problem for specific examples. (Notice that in lecture we assumed M_g to be zero, which is too restrictive even for the most basic least squares loss.)

Problem outline: Parts (a) - (c) help to derive a recursive formula of the form

$$\mathbb{E}\|\mathbf{w}^{t+1} - \mathbf{w}^*\|_2^2 \leqslant \gamma\mathbb{E}\|\mathbf{w}^t - \mathbf{w}^*\|_2^2 + \tilde{\gamma}$$

In part (d) we show that we achieve linear (also called geometric) convergence of $\mathbf{w}^t$ to the optimum $\mathbf{w}^*$ for SGD with constant step-size when $f(\mathbf{w}^*) = 0$. That means we can write

$$\mathbb{E}\|\mathbf{w}^t - \mathbf{w}^*\|_2^2 \leqslant \gamma^t\|\mathbf{w}^0 - \mathbf{w}^*\|_2^2$$

for some $\gamma < 1$. In part (e) we explore what happens with the iterates of SGD with constant step-size if instead $f(\mathbf{w}^*) > 0$. In part (f) we visualize how the convergence bounds translate to actual training error decrease for OLS using SGD and GD.

The bonus parts (g) - (h) are for those students who "love math" according to the midterm preliminary questions, where you are asked to prove that SGD with decaying step-size converges at $\frac{1}{t}$ for "nice" functions using induction.

a. **Show that following relation between the $t+1$-step error and the t-step error:**

$$\|\mathbf{w}^{t+1} - \mathbf{w}^*\|_2^2 = \|\mathbf{w}^t - \mathbf{w}^*\|_2^2 - 2\alpha_t\langle G_t, \mathbf{w}^t - \mathbf{w}^*\rangle + \alpha_t^2\|G_t\|_2^2$$

Solution

b. *Here we prove the key relation which underlies the success of stochastic gradient methods. This is where we use that stochastic gradients are unbiased!* Since we have stochastic gradients, we want to make guarantees in terms of expectations. For notational convenience let us define the short hand $\Delta_t := \mathbb{E}_t\|\mathbf{w}^t - \mathbf{w}^*\|_2^2$, the

average squared error, where the expectation is taken over all the random indices drawn by the stochastic gradient method up to time t. We want to show a clean relation between Δ_t and Δ_{t+1} when the stochastic gradient satisfies (1).

Remind yourself which random indices $\mathbf{w}^t$ depends on and use the unbiasedness of the stochastic gradient to **show that**

$$\mathbb{E}\left[\langle G(\mathbf{w}^t), \mathbf{w}^t - \mathbf{w}^*\rangle\right] = \mathbb{E}\left[\langle \nabla f(\mathbf{w}^t), \mathbf{w}^t - \mathbf{w}^*\rangle\right] \tag{2}$$

Now use equality (2) and (a) to prove that

$$\Delta_{t+1} \leqslant (1 + \alpha_t^2 M_g^2)\Delta_t - 2\alpha_t \cdot \mathbb{E}\left[\langle \nabla f(\mathbf{w}^t), \mathbf{w}^t - \mathbf{w}^*\rangle\right] + \alpha_t^2 B^2 \tag{3}$$

Hint 1: You may take the law of iterated expectation (also called tower property) as given, i.e.

$$\mathbb{E}\left[\langle G(\mathbf{w}^t), \mathbf{w}^t - \mathbf{w}^*\rangle\right] = \mathbb{E}_{i_1,\ldots,i_{t-1}}\left[\mathbb{E}\left[\langle G(\mathbf{w}^t), \mathbf{w}^t - \mathbf{w}^*\rangle \mid i_1, \ldots, i_{t-1}\right]\right]$$

See Discussion for the derivation.

Hint 2: Unbiased stochastic gradient means that $\mathbb{E}_{i_t}\left[\nabla f_{i_t}(\mathbf{w})\right] = \nabla f(\mathbf{w})$ for any $\mathbf{w}$ independent of the random index, where the expectation is with respect to the random index i_t at time t. Refer to the Discussion for brief intro on conditional expectations.

Solution

c. *In this step we see that we need for our analysis that our loss function is "nice" (strongly convex to be precise), which holds for OLS when* $\lambda_{min}\left(\mathbf{X}^\top\mathbf{X}\right) > 0$. *That is, all the guarantees for SGD we discuss in this problem only hold for such "nice" functions. In general, at least convexity is needed for any convergence proof.* **Using inequality (3), now show that**

$$\Delta_{t+1} \leqslant \left(1 + \alpha t^2 M_g^2 - 2\alpha_t m\right)\Delta_t + \alpha_t^2 B^2$$

where we assume that the minimum eigenvalue of $\mathbf{X}^\top\mathbf{X}$ denoted by m is positive, i.e.

$$m := \lambda_{\mathbf{min}}\left(\mathbf{X}^\top\mathbf{X}\right) > 0$$

Hint: Use the fact that $\nabla f(\mathbf{w}^*) = \mathbf{0}$, and hence

$$\langle \nabla f(\mathbf{w}^t), \mathbf{w} - \mathbf{w}^*\rangle = \langle \nabla f(\mathbf{w}^t) - \nabla f(\mathbf{w}^*), \mathbf{w} - \mathbf{w}^*\rangle$$

Solution

d. We now examine how close we can get to the optimal solution $\mathbf{w}^*$ using SGD with **constant step-size**, i.e. $\alpha_t = \alpha$ for all t in two different scenarios. First, for this question we assume that $\mathbf{X}\mathbf{w}^* = \mathbf{y}$, or in other words, that the minimum loss is 0. Show that inequality (1) holds with $B = 0$ and $M_g = \max_i \|\mathbf{x}_i\|_2^4$ in this case. Find the optimum learning rate and show that

$$\Delta_t \leqslant \left(1 - \frac{m^2}{M_g^2}\right)^t \Delta_0$$

This means that with $B = 0$ we have linear (geometric) convergence!

Hint: You may want to use that for any matrix $\mathbf{A}$ and vector $\mathbf{w}$ it holds that

$$\|\mathbf{A}\mathbf{w}\|_2^2 \leqslant \lambda_{\max}(\mathbf{A}^\top\mathbf{A})\|\mathbf{w}\|_2^2$$

and the same bound applies when taking the expectation on both sides.

Solution

e. Instead of assuming that there exists a $\mathbf{w}^*$ that satisfies $\mathbf{X}\mathbf{w}^* = \mathbf{y}$ exactly, we now consider the case where $f(\mathbf{w}^*) = \frac{1}{2}\|\mathbf{X}\mathbf{w}^* - \mathbf{y}\|^2 > 0$. One can show that in this case

$$\mathbb{E}\left[\|G(\mathbf{w})\|_2^2\right] \leqslant M_g^2\|\mathbf{w} - \mathbf{w}^*\|_2^2 + B^2$$

holds for some M_g and B, where $M_g \neq 0$ and $B \neq 0$. Suppose that you are given the constants M_g and B and the step-size is still constant, i.e. $\alpha_t = \alpha$ for all t.

We define the short hand notation

$$\gamma := 1 + \alpha^2 M_g^2 - 2\alpha m$$

and assume α is large enough such that $\gamma < 1$. **Using part (c), prove that**

$$\Delta_t \leqslant \gamma^t \Delta_0 + \frac{\alpha B^2}{\alpha M_g^2 - 2m}$$

We now want to interpret this bound. **Does it guarantee convergence $\mathbf{w}^t \to \mathbf{w}^*$ when t goes to infinity? Explain.**

Hint: You may find the following inequality helpful:

$$\sum_{t=1}^{n} \gamma^t \leqslant \frac{1}{1-\gamma} \quad \text{for } \gamma < 1$$

Solution

f. We now want use simulations to compare the first order methods for solving OLS for the cases in (d) and (e). In particular, we want to plot the estimation error $\|\mathbf{w}^t - \mathbf{w}^*\|_2$ against the gradient descent step in the following 3 scenarios:

- When there exists an exact solution $\mathbf{X}\mathbf{w}^* = \mathbf{y}$, plot the errors when you are using SGD and a constant learning rate.
- When there does not exist an exact solution $\mathbf{X}\mathbf{w}^* = \mathbf{y}$, plot the errors when you are using SGD and a constant learning rate. Also plot the errors for GD and the same constant learning rate.
- When there does not exist an exact solution $\mathbf{X}\mathbf{w}^* = \mathbf{y}$, plot the errors when you are using SGD and a decaying learning rate.

Using the attached starter code, **implement the gradient updates** in the function `sgd()`, while all the plotting functions are already there. **Show the 3 plots you obtain using the starter code. Report the average squared error computed in the starter code. What?s your conclusion?**

Solution

g. (Bonus) **Prove that there is some constant S and k_0 such that if the learning rate is decaying, the error Δ_k satisfies**

$$\Delta_k \leqslant \frac{S}{k + k_0}$$

In this part, prove for the case of $M_g = 0$ and $B \neq 0$.

Hint: Induction is your friend.

Solution

h. (Bonus) **Prove that the same conclusion holds for the case of $M_g \neq 0$ and $B \neq 0$, for some different constants M_g and B.**

Hint: This general case can be reduced to the case above.

Solution

Problem 3: Gradient Descent Framework

In HW1, you modeled the classification of digit numbers of the MNIST dataset as a linear regression problem and solved it using its closed-form solution. In this homework, you will model it better by using classification models such as logistic regression and neural networks, and solve it using stochastic gradient descent. The goal of this problem is to show the power of modern machine learning frameworks such as TensorFlow and PyTorch, and how they can solve a wide range of problems. TensorFlow is the recommended framework in this homework and we also provide the starter kit in PyTorch.

a. The starter code contains an implementation of linear regression for the MNIST dataset to classify 10 digits. Let $\mathbf{x}_i$ be the feature vector and $\mathbf{y}_i$ be a one-hot vector encoding of its class, i.e. $y_{i,j} = 1$ if and only if ith images is in class j and $y_{i,j} = 0$ if not. In order to use linear regression to solve a multi-class classification, we will use the *one-vs-all* strategy here. In particular, for each j we will fit a linear regression model $(\mathbf{w}_j, b_j)$ to minimize $\sum_i \left(\mathbf{w}_j^\top \mathbf{x}_i + b_j - y_{i,j}\right)_2^2$. Then for any image $\mathbf{x}$, the prediction of its class will be $\arg\max_j \left(\mathbf{w}_j^\top \mathbf{x} + b_j\right)$.

Read the implementation and run it with batch size equal to 50, 100, and 200. **Attach the "epoch vs validation accuracy" plot. Report the running time for all the batch sizes. Explain the potential benefits and drawbacks of using small batch size, i.e., SGD vs GD.**

Solution

b. **Implement the `train_logistic_function` to do the multi-class logistic regression using softmax function. Attach the plot of the "epoch vs validation accuracy" curve.** The loss function of the multi-class logistic regression is

$$\ell = -\sum_{i=1}^{n} \log \left[\mathbf{softmax}\left(\mathbf{W}\mathbf{x}_i + b\right)^\top \mathbf{y}_i\right] \tag{4}$$

where the softmax function $\mathbf{softmax} : \mathbb{R}^d \to \mathbb{R}^d$ is defined as

$$\mathbf{softmax}(\mathbf{z})_j = \frac{\exp(z_j)}{\sum_k \exp(z_k)} = \frac{\exp(z_j - z')}{\sum_k \exp(z_k - z')} \tag{5}$$

Here $z' = \max_j z_j$. The expression on the right is a numerical stable formula to compute the softma. You may NOT use any functions in `tf.nn.*`, `tf.losses.*`, and `torch.nn.*` for all parts of this problem.

Solution

c. Copy your code from `train_logistic` to `train_nn` and add an additional `tanh` nonlinear layer to it. Your loss function should be something like

$$\ell = -\sum_{i=1}^{n} \log \left[\mathbf{softmax}\left(\mathbf{W}^{(2)}\mathbf{tanh}\left(\mathbf{W}^{(1)}\mathbf{x}_i\right) + \mathbf{b}\right)^\top \mathbf{y}_i\right] \tag{6}$$

Attach the plot of of the "epoch vs validation accuracy" curve. You have the freedom but **are NOT required to** choose the hyper-parameters to get the best performance.

Solution

d. In our previous lecture, we learned how to solve the total least squares by doing the SVD decomposition. Now we will try to model the problem from the probabilistic perspective and solve it using stochastic gradient descent. Let the probabilistic model

$$y_i - z_y = \mathbf{w}^\top (\mathbf{x}_i - \mathbf{z}_x) \tag{7}$$

where $\mathbf{x}_i$ and y_i is the observed data, $y_i - z_y$ is y_{true}, $\mathbf{x}_i - z_{\mathbf{z}}$ is x_{true}, $z_y \sim \mathcal{N}(0,1)$, and $z_{\mathbf{x},j} \sim \mathcal{N}(0,1)$ for all j. **Prove that the log-likelihood for the model in Equation (7) is**

$$\sum_{i=1}^{n} \log P_{\mathbf{w}}(y_i \mid \mathbf{x}_i) = C - \frac{n}{2} \log\left(\|\mathbf{w}\|_2^2 + 1\right) - \frac{1}{2\left(\|\mathbf{w}\|_2^2 + 1\right)} \sum_{i=1}^{n} \left(y_i - \mathbf{w}^\top \mathbf{x}_i\right)^2 \tag{8}$$

where n is the number of samples, and C is a constant that is not related to $\mathbf{w}$, y, and x. *Note that the maximum likelihood estimation of this probabilistic model is not the same as the SVD solution of TLS as we did in lecture.*

Solution

e. **Implement the stochastic gradient descent to find the MLE for the model above.** In the starter code, we generate some data for you and you will need to recover $\mathbf{w}$ using the observed data. **Report the error** $\|w^* - w_{\text{true}}\|_2^2$ where w^* is the one that you recover. Try to play with hyper-parameters such as the batch size and the learning rate. **Is the solution of SGD sensitive to its hyper-parameters in this problem?**

Solution

[BONUS] Problem 4: Genome-Wide Association Study

All the following text is present in the accompanying Jupyter notebook, but the notebook has additional explanations and accompanying figures and code. We recommend you do not read this pdf, but go directly to the Jupyter notebook. This pdf was included for quick reference.

Overall goal: This real world problem is one in computational biology which uses many of the techniques and concepts you have been introduced to, all together, in particular, linear regression, PCA, non-iid noise, diagonalizing multivariate Gaussian covariance matrices, and bias-variance trade-off. We will also tangentially introduce you to concepts of statistical testing. **This homework problem is effectively a demo in that we will ask you to execute code and answer questions about what you observe. You are not required to code anything at all.**

Setup and problem statement: Given a set of people for whom genetics (DNA) has been measured, and also a corresponding trait for each person, such as blood pressure, or "is-a-smoker", one can use data-driven methods to deduce which genetic effects are likely responsible for the trait. We have collected blood from n individuals who either smoke ($y_i = 1$) or do not smoke ($y_i = 0$). Their blood samples have been sequenced at m positions along the genome, yielding the feature matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$, composed of the genetic variants (which take on values 0, 1, or 2). Specifically, $\mathbf{X}_{i,j}$ is a numeric encoding of the DNA for the i^{th} person at genetic feature j.[1] We want to deduce which of the m genetic features are associated with smoking, by considering one at a time. *In the data we give you, it will turn out that there is no true signal in the data; however, we will see that without careful modeling, we would erroneously conclude that the data were full of signal.*

Overall modeling approach: The basic idea will be to "test" one genetic feature at a time and assign a score (a p-value) indicating our level of belief that it is associated with the trait. We will start with a simple linear regression model, and then build increasingly more correct linear predictive models. The three models we will use are (1) linear regression, (2) linear regression with PCA features, (3) linear regression with all genetic variants as features. The first model is naive, because it assumes the individuals are iid. The fundamental modeling challenges with these kinds of analyses is that the individuals in the study are not typically not iid, owing to differences in race and family-relatedness (e.g., sisters, brothers, patients, grandparents in the data set), which violate the iid assumption. Left un-modeled, such structure creates misleading results, yielding signal where none exists. Luckily, as we shall see, one can visualize these modeling issues via quantile-quantile plots, which will soon be briefly introduced.

How to test each variant: Herein we provide a minimal exposition to allow you to do the homework To estimate how implicated each genetic feature is we will use a score, in the form of a p-value. One can think of the p-value as a proxy for how informative the genetic feature is for prediction of the trait (e.g. "is smoker"). More precisely, to get the p-value for the j^{th} genetic feature, we first include it in the model (for a given model class, such as linear regression) and compute the maximum likelihood, LL_j (this is our alternative hypothesis). Then we repeat this process, but having removed the genetic feature of interest from the model, yielding LL_{-j} (this is our null hypothesis). To be clear, the null hypothesis will have none of the m genetic variants that are being tested. You can refer to the Jupyter notebook for a brief explanation of hypothesis testing. The p-value is then a simple monotonic decreasing function of the difference in these two likelihoods, `diff_ll` $= LLj - LL_{-j}$-one that we will give you. p-values lie in $[0, 1]$ and the smaller the p-value, the larger the difference in the likelihoods, and the more evidence that the genetic marker is associated with the trait (assuming the model is correct).

a. To diagnose if something is amiss in our genetic analyses, we will make use of the following: (1) we assume that if any genetic signal is present, it is restricted to a small fraction of the genetic features, (2) p-values corresponding to no signal in the data are distributed as $p \sim \text{Unif}[0, 1]$. Combining these two assumptions, we will assume that p-values arising from a valid model should largely be drawn from Unif$[0, 1]$, and also that large deviations suggest that our model is incorrect. Quantile-quantile plots let us visualize if we have deviations or not. Quantile-quantile plots are a way to compare two probability distributions by comparing their quantile values (e.g. how does the smallest p-value in each distribution compare, and then the second smallest, etc.). In the quantile-quantile plot, you will see m points, one for each genetic marker. The x-coordinate of each point corresponds to the theoretical quantile we would expect to see if the distribution was in fact a Unif$[0, 1]$ and the y-coordinate corresponds to the observed value of the quantile. An example is shown in Figure 1 1, where the line on the diagonal results from an analysis where the model is correct, and hence the theoretical and empirical p-value quantiles match, while the other line, which deviates from the diagonal, indicates that

[1] Technically, the entries of the matrix correspond to having zero, one or two mutant versions of the DNA, but we will treat them as real-valued in this problem.

we have likely made a modeling error. If there are genetic signals in the data, these would simply emerge as a handful of outlier points from the diagonal (not shown).

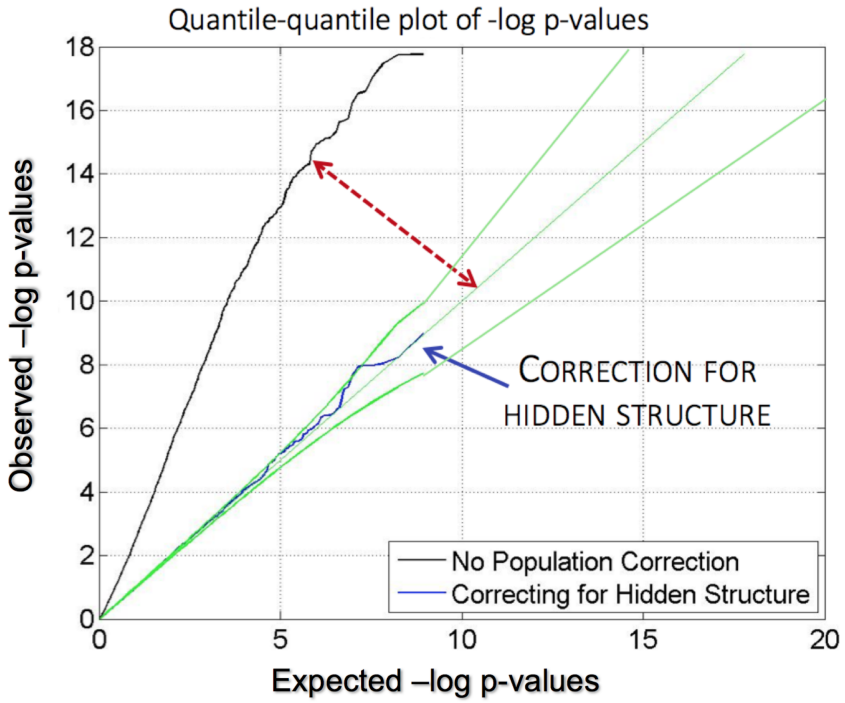

Figure 1: Example of quantile-quantile plot that shows large deviation from expectation.

Before we dive into developing our models, we need to be able to understand whether the p-values we get back from our hypothesis tests look like m random draws from a Unif$[0, 1]$.

Use the qqplot function to make a qq-plot for each of the 3 distributions provided below and explain your findings. What should we observe in our qq-plot if our empirical distribution looks more and more similar to a Unif$[0, 1]$**?** Note that we use two kinds of qq-plots: one in p-value space and one negative log p-value space. The former is for intuition, while the latter is for higher resolution. The green lines in the negative log p-value qq-plots indicate the error bars.

Solution

b. We will use linear models in the genetic marker we are testing. In particular, when testing the j^{th} genetic feature, we have that the trait, y, is a linear function of the genetic variant, i.e. $\mathbf{y} = \mathbf{x}_j w_i + \mathbf{w}_0 + \boldsymbol{\epsilon}$ where ϵ_i is random noise distributed as $\mathcal{N}(0, \sigma^2)$, $w_1 \in \mathbb{R}^1$, $\mathbf{w}_0 \in \mathbb{R}^{n \times 1}$ is a constant vector, and $\mathbf{x}_j$ is the j^{th} column of $\mathbf{X}$, representing data for the j^{th} genetic variant. To simplify matters, we will add a column of ones to the right end of $\mathbf{x}_j$ and rewrite the regression as $\mathbf{y} = [\mathbf{x}_j, \mathbf{1}]\mathbf{w} + \boldsymbol{\epsilon}$ where $[\mathbf{x}_j, \mathbf{1}]$ is the j^{th} column of $\mathbf{X}$ with a vector of ones appended to the end and $\mathbf{w} \in \mathbb{R}^{2 \times 1}$. The model without any genetic information, $\mathbf{y} = \mathbf{1}w + \boldsymbol{\epsilon}$ is referred to as the *null model* in the parlance of statistical testing the *alternative model*, which includes the information we are testing (one genetic marker) is $\mathbf{y} = [\mathbf{x}_j, \mathbf{1}]\mathbf{w} + \boldsymbol{\epsilon}$ where $\mathbf{x}_j$ is the j^{th} column of $\mathbf{X}$, i.e. the data using only the j^{th} genetic variant as a feature. **Plot the quantile quantile plot of p-values using linear regression as just describe, a so-called naive approach, by running the function `naive_model`. From the plot, what do you conclude about the suitability of linear regression for this problem?**

Solution

c. From the quantile-quantile plot in the previous part, it appears that the model is picking up on more association than theoretically expected. The reason for this is owing to the assumption of iid noise being correct. In particular, this data set contains individuals from different racial backgrounds, and also has clusters of individuals from extended families (e.g. grandparents, parents, siblings). This means that their genetics are not iid, and hence linear regression yields spurious results-all the genetic features seem to be implicated in the trait. Thus we need to modify our noise assumptions by somehow accounting for the hidden structure in the data set. The main idea is that when testing one genetic feature, all the other genetic features, jointly, are a proxy to the racial and family background. If we could include them in our model, we could correct the problem. **Ideally we would like to use all the genetic features in the linear regression model, however this is not a good idea. Why not?** Hint: There are roughly 1300 individuals and 7500 genetic variants. A written, English answer is sufficient.

So instead of using all genetic features, we will try using PCA to reduce the number of genetic features. As we saw in class, PCA on a genetic similarity matrix can capture geography, which correlates to race, quite well. So instead of adding all the genetic features, we will instead use only three features[2] $\mathbf{X}_{\text{proj}}$, which are the $\mathbf{X}$ projected onto the top 3 principal components of $\mathbf{X}$. Consequently, the updated null model is $\mathbf{y} = \mathbf{X}_{\text{proj}}\mathbf{w}_{\text{proj}} + \boldsymbol{\epsilon}$ where $\mathbf{w}_{\text{proj}} \in \mathbb{R}^{3\times 1}$, while the alternative model is $\mathbf{y} = [\mathbf{x}_j, \mathbf{X}_{\text{proj}}, \mathbf{1}]\mathbf{w} + \boldsymbol{\epsilon}$ where $\mathbf{w} \in \mathbb{R}^{5\times 1}$ for genetic variant j. **Plot the quantile-quantile plot from obtaining p-values with this PCA linear regression approach by running the function `pca_corrected_model`. How does this plot compare to the first plot? What does this tell you about this model compared to the previous model?**

Solution

d. PCA got us part of the way there. However, PCA truncates the eigen-spectrum; if the tail-end of that spectrum is important, as it is for family-relatedness, then it will not fully correct for our problem. So we want a method which (a) is well-behaved in terms of number of parameters that need to be estimated, and (b) includes all of the information we need. So rather than adding the projections as features, we use an modeling approach called linear mixed models which effectively adjust the iid noise in the gaussian by the pairwise genetic similarity of all the individuals. That is, we set Σ in $\mathbf{y} \sim \mathcal{N}\left(y \mid [\mathbf{x}_j, 1]\mathbf{w}, \mathbf{I}\sigma^2 + \mathbf{X}\mathbf{X}^\top\sigma_k^2\right)$.

Specifically, $\mathbf{y} = [\mathbf{x}_j, 1]\mathbf{w} + \mathbf{z} + \boldsymbol{\epsilon}$ where $\mathbf{z} \sim \mathcal{N}(0, \sigma_k^2\mathbf{K})$ where $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$, σ_k, $\mathbf{w} \in \mathbb{R}^{m\times 1}$ and σ are parameters we want to estimate. Notice that $\mathbf{y} \sim \mathcal{N}\left([\mathbf{x}_j, 1]\mathbf{w}, \sigma^2\mathbf{I} + \sigma_k^2\mathbf{K}\right)$. Evaluation of the likelihood is thus on the order of $\mathcal{O}(n^3)$ from the required matrix inverse and determinant of $\sigma^2\mathbf{I} + \sigma_k^2\mathbf{K}$. To test m genetic variants, the time complexity becomes $\mathcal{O}(mn^3)$, which is extremely slow for large datasets. **Given the eigen-decomposition $\mathbf{K} = \mathbf{U}\mathbf{D}\mathbf{U}^\top$, how can we make this faster if we have to test thousands of genetic feature? Would this be computationally more efficient if you only have one genetic feature to test?**

Finally, make the quantile-quantile plot for this last model by running the function `lmm`. What can we conclude about this model relative to the other two models?

HINT: Since the manipulations needed for $\sigma^2\mathbf{I} + \sigma_k^2\mathbf{K}$ is the bottleneck here, we would like a transformation which makes this covariance of the multi-variate gaussian be a diagonal matrix.

Solution

[2] One needs to choose this number, but we have done so for you

Problem 5: Your Own Question

Write your own question, and provide a thorough solution.

Writing your own problems is a very important way to really learn the material. The famous "Bloom's Taxonomy" that lists the levels of learning is: Remember, Understand, Apply, Analyze, Evaluate, and Create. Using what you know to create is the top-level. We rarely ask you any HW questions about the lowest level of straight-up remembering, expecting you to be able to do that yourself. (e.g. make yourself flashcards) But we don't want the same to be true about the highest level.

As a practical matter, having some practice at trying to create problems helps you study for exams much better than simply counting on solving existing practice problems. This is because thinking about how to create an interesting problem forces you to really look at the material from the perspective of those who are going to create the exams.

Besides, this is fun. If you want to make a boring problem, go ahead. That is your prerogative. But it is more fun to really engage with the material, discover something interesting, and then come up with a problem that walks others down a journey that lets them share your discovery. You don't have to achieve this every week. But unless you try every week, it probably won't happen ever.

Solution

Code Appendix

- `starter_P1_SGDtheory.ipynb`

  Solution

- `starter_P3_classification/classification_pytorch.py`

  Solution

- `starter_P3_classification/classification_tensorflow.py`

  Solution

- `starter_P3_classification/tls_pytorch.py`

  Solution

- `starter_P3_classification/tls_tensorflow.py`

  Solution

- `code_P4_genome/gwas problem.ipynb`

  Solution