

# Homework #2

CSE 546: Machine Learning

Prof. Kevin Jamieson

Due: 11/2 11:59 PM

## 1 A Taste of Learning Theory

1. [5 points] For  $i = 1, \dots, n$  fix  $x_i \in \mathbb{R}^d$  and let  $y_i = x_i^T w_* + \epsilon_i$  where  $\epsilon_i \stackrel{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2)$ . All answers should be given in matrix notation:  $X = [x_1, \dots, x_n]^T$ ,  $y = [y_1, \dots, y_n]^T$ ,  $\epsilon = [\epsilon_1, \dots, \epsilon_n]^T$ . Let  $\mathbb{E}_\epsilon[\cdot]$  denote the expectation with respect to the randomness in  $\epsilon$ .

- Let  $\hat{w} = \arg \min_{w \in \mathbb{R}^d} \sum_{i=1}^n (y_i - x_i^T w)^2$ . Give a simple expression for  $\hat{w}$  in terms of  $X, y, \epsilon, w_*$
  - Some times we are interested in *identifying* the underlying coefficients of  $w_*$ . What are  $\mathbb{E}_\epsilon[\hat{w}]$  and  $\mathbb{E}_\epsilon[\|\hat{w} - w_*\|_2^2]$ ? (Hint: recall the properties of the trace operator; for  $u \in \mathbb{R}^n$  and  $A, B, C \in \mathbb{R}^{n \times n}$  we have  $u^T ABCu = \text{Trace}(u^T ABCu) = \text{Trace}(uu^T ABC) = \text{Trace}(Cuu^T AB)$ .)
  - Other times we just want to be able to *predict* future data. Suppose we observe an additional set of observations: for  $i = 1, \dots, n$  let  $\tilde{y}_i = x_i^T w_* + \tilde{\epsilon}_i$  where  $\tilde{\epsilon}_i \stackrel{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2)$ . Using the  $\hat{w}$  learned from  $\{(x_i, y_i)\}_{i=1}^n$  in part a., what are  $\mathbb{E}_{\tilde{\epsilon}}[\tilde{y}_i]$  and  $\mathbb{E}_{\tilde{\epsilon}}[\mathbb{E}_\epsilon[\frac{1}{n} \sum_{i=1}^n (\tilde{y}_i - x_i^T \hat{w})^2]]$ ? (Hint: add and subtract  $\mathbb{E}_{\tilde{\epsilon}}[\tilde{y}_i]$  inside the square.)
  - Compare your answers to part b. and c. and provide an explanation for why it is generally accepted that prediction is easier than identification for real-world data?
2. [5 points] For  $i = 1, \dots, n$  let  $(x_i, y_i) \stackrel{i.i.d.}{\sim} P_{XY}$  where  $y_i \in \{-1, 1\}$  and  $x_i$  lives in some set  $\mathcal{X}$  ( $x_i$  is not necessarily a vector). The 0/1 loss, or *risk*, for a classifier  $f : \mathcal{X} \rightarrow \{-1, 1\}$  is defined as:

$$L(f) = \mathbb{E}_{XY}[\mathbf{1}(f(X) \neq Y)]$$

where  $\mathbf{1}(\mathcal{E})$  is the indicator function for the event  $\mathcal{E}$  (the function takes the value 1 if  $\mathcal{E}$  occurs and 0 otherwise). The expectation is with respect to the underlying distribution  $P_{XY}$  on  $(X, Y)$ . Unfortunately, we don't know  $P_{XY}$  exactly, but we do have our i.i.d. samples  $\{(x_i, y_i)\}_{i=1}^n$  drawn from it. Define the *empirical risk* as

$$\hat{L}(f) = \frac{1}{N} \sum_{i=1}^N \mathbf{1}(f(x_i) \neq y_i)$$

which is just an empirical estimate of our loss. Recall Hoeffding's inequality: if  $Z_1, \dots, Z_m$  are i.i.d. random variables where  $\mathbb{P}(Z_i = 1) = 1 - \mathbb{P}(Z_i = 0) = p$  and  $\hat{p} = \frac{1}{m} \sum_{i=1}^m Z_i$ , then

$$\mathbb{P}(|p - \hat{p}| \geq \epsilon) \leq 2 \exp(-2m\epsilon^2).$$

- Suppose a domain expert who has not seen your freshly drawn data  $\{(x_i, y_i)\}_{i=1}^n$  gives you a classifying rule  $\tilde{f} : \mathcal{X} \rightarrow \{-1, 1\}$ . We wish to estimate the true risk  $L(\tilde{f})$  of this classifying rule using the empirical risk  $\hat{L}(\tilde{f})$ . Using Hoeffding's inequality, find a value of  $A$  such that

$$\mathbb{P}(|\hat{L}(\tilde{f}) - L(\tilde{f})| \leq A) \geq 1 - \delta.$$

- After seeing the confidence interval, the domain expert is not satisfied with her classifying rule  $\tilde{f}$ . Because she understands the underlying process that generated the data, she proposes a finite set of alternative classifying rules  $\mathcal{F} = \{f_1, \dots, f_k\}$  that may fit the data better. The "best in class" function  $f^*$  is:

$$f^* = \arg \min_{f \in \mathcal{F}} L(f).$$

Note that  $L(f^*)$  is the best true loss we could hope to achieve using functions in  $\mathcal{F}$ . If we replace  $\tilde{f}$  with  $f^*$  in part a., does the same confidence interval hold? Why or why not?

- c. Of course, determining  $f^*$  would require exact knowledge of  $L(f)$ , which requires exact knowledge of  $P_{XY}$ , which we don't have. As an alternative, you propose to use the *empirical risk minimizer* (ERM):

$$\hat{f} = \arg \min_{f \in \mathcal{F}} \hat{L}(f).$$

If we replace  $\tilde{f}$  with  $\hat{f}$  in part a., does the same confidence interval hold? Why or why not?

- d. Come up with a **simple** example (i.e., give a  $\mathcal{X}, P_{XY}, \mathcal{F}$ ) where the empirical risk is not indicative of the true risk of the empirical risk minimizer  $\hat{f}$ . (Hint: consider a case where  $\mathbb{P}(Y = 1|X = x) = \mathbb{P}(Y = -1|X = x) = 1/2$  for all  $x \in \mathcal{X}$ , and show that  $L(f) = 1/2$  for all  $f \in \mathcal{F}$ , but  $\hat{L}(\hat{f}) = 0$ ).
- e. Provide a confidence interval that simultaneously holds for the losses of all  $f \in \mathcal{F}$ , with probability of error  $\delta$ . In other words, provide a value  $B$  so that:

$$\mathbb{P}(\text{for all } f \in \mathcal{F}, |\hat{L}(f) - L(f)| \leq B) \geq 1 - \delta$$

Show your steps. (Hint: for events  $\mathcal{E}_1, \mathcal{E}_2, \dots, \mathcal{E}_k$ , the “union bound” states that  $\mathbb{P}(\mathcal{E}_1 \text{ or } \mathcal{E}_2 \text{ or } \dots \text{ or } \mathcal{E}_k) = \mathbb{P}(\bigcup_{i=1}^k \mathcal{E}_i) \leq \sum_{i=1}^k \mathbb{P}(\mathcal{E}_i)$ .)

- f. Noting that  $\hat{f} \in \mathcal{F}$ , provide a confidence interval for the loss of  $\hat{f}$  that holds with probability greater than  $1 - \delta$ . In other words, we are seeking a bound on  $|L(\hat{f}) - \hat{L}(\hat{f})|$  that holds with probability greater than  $1 - \delta$ .
- g. Provide a bound on how close your loss, using the ERM, is to the best possible loss (in the hypothesis space). Specifically, provide a value  $R$  so that the following holds with probability greater than  $1 - \delta$ ,

$$L(\hat{f}) - L(f^*) \leq R$$

The quantity  $L(\hat{f}) - L(f^*)$  is often referred to as the *regret*, as it is how close you are to the best possible. Here,  $R$  is our regret bound.

- h. Let us understand how large a hypothesis class we can utilize, i.e. how large  $|\mathcal{F}|$  can be? Suppose we know we will be provided with a training set of size  $n$ , and, before we look at our training data, we choose the hypothesis class  $\mathcal{F}$  as a function of the sample size  $n$ . As we get more data, we would expect that we can utilize a larger hypothesis class. Let us examine this more quantitatively. We can think of learning being possible if our regret tends to 0 as  $n$  becomes large (with a probability of error less than  $1 - \delta$ ). Let us determine if learning is possible in each of the following cases. For the following cases, does the above suggest that we are able to learn, and if so, what is our regret as a function of  $n$  in big-O notation?

- (a)  $|\mathcal{F}|$  is a constant.
- (b)  $|\mathcal{F}| = n^p$  for some constant  $p$ .
- (c)  $|\mathcal{F}| = \exp(\sqrt{n})$ .
- (d)  $|\mathcal{F}| = \exp(10n)$ .

Your answers to the last problem are one interpretation as to why learning complex models is possible (from a statistical perspective). Of particular interest is when  $\mathcal{F}$  is allowed to be infinite in which more advanced tools are necessary. To provide intuition we note that  $n$  points in  $d$ -dimensions can be labeled by a hyperplane  $w$  in at most  $O(n^d)$  ways (i.e.,  $|\{\text{sign}(Xw) : w \in \mathbb{R}^d\}| = O(n^d)$  for fixed  $X \in \mathbb{R}^{n \times d}$ ). While more advanced tools (e.g., VC dimension, covering numbers, etc.) are necessary to formally prove the learning rates for infinite classes, thinking about this number of sign patterns and how this relates to  $|\mathcal{F}|$  is instructive.

---

**Algorithm 1:** Coordinate Descent Algorithm for Lasso

---

```
while not converged do
     $w_0 \leftarrow \frac{1}{n} \sum_{i=1}^n \left( y_i - \sum_{j=1}^d w_j x_{i,j} \right)$ 
    for  $k \in \{1, 2, \dots, d\}$  do
         $a_k \leftarrow 2 \sum_{i=1}^n x_{i,k}^2$ 
         $c_k \leftarrow 2 \sum_{i=1}^n x_{i,k} \left( y_i - (w_0 + \sum_{j \neq k} w_j x_{i,j}) \right)$ 
         $w_k \leftarrow \begin{cases} (c_k + \lambda)/a_k & c_k < -\lambda \\ 0 & c_k \in [-\lambda, \lambda] \\ (c_k - \lambda)/a_k & c_k > \lambda \end{cases}$ 
    end
end
```

---

## 2 Programming: Lasso

Given  $\lambda > 0$  and data  $(x_1, y_1), \dots, (x_n, y_n)$ , the Lasso is the problem of solving

$$\arg \min_{w \in \mathbb{R}^d, w_0 \in \mathbb{R}} \sum_{i=1}^n (x_i^T w + w_0 - y_i)^2 + \lambda \sum_{j=1}^d |w_j| \quad (1)$$

$\lambda$  is a regularization tuning parameter. For the programming part of this homework, you are required to implement the coordinate descent method of Algorithm 1 that can solve the Lasso problem.

You may use common computing packages (such as NumPy or SciPy), but do not use an existing Lasso solver (e.g., of Sci-Kit Learn).

Before you get started, here are some hints that you may find helpful:

- For loops can be slow whereas vector/matrix computation in Numpy is very optimized, exploit this as much as possible.
- As a sanity check, ensure the objective value is nonincreasing with each step.
- It is up to you to decide on a suitable stopping condition. A common criteria is to stop when no element of  $w$  changes by more than some small  $\delta$  during an iteration. If you need your algorithm to run faster, an easy place to start is to loosen this condition.
- You will need to solve the Lasso on the same dataset for many values of  $\lambda$ . This is called a regularization path. One way to do this efficiently is to start at a large  $\lambda$ , and then for each consecutive solution, initialize the algorithm with the previous solution, decreasing  $\lambda$  by a constant ratio until finished.
- The smallest value of  $\lambda$  for which the solution  $\hat{w}$  is entirely zero is given by

$$\lambda_{max} = \max_{k=1, \dots, n} 2 \left| \sum_{i=1}^n x_{i,k} \left( y_i - \left( \frac{1}{n} \sum_{j=1}^n y_j \right) \right) \right|$$

This is helpful for choosing the first  $\lambda$  in a regularization path.

3. [5 points] We will first try out your solver with some synthetic data. A benefit of the Lasso is that if we believe many features are irrelevant for predicting  $y$ , the Lasso can be used to enforce a sparse solution, effectively differentiating between the relevant and irrelevant features. Suppose that  $x \in \mathbb{R}^d, y \in \mathbb{R}, k < d$ , and pairs of data  $(x_i, y_i)$  for  $i = 1, \dots, n$  are generated independently according to the model

$$y_i = w_0^* + w_1^* x_{i,1} + w_2^* x_{i,2} + \dots + w_k^* x_{i,k} + \epsilon_i$$

where  $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$  is some Gaussian noise. Note that since  $k < d$ , the features  $k+1$  through  $d$  are unnecessary (and potentially even harmful) for predicting  $y$ .

With this model in mind, let  $n = 50, d = 75, k = 5$ , and  $\sigma = 1$ . Generate some data by drawing each  $x_i \sim \mathcal{N}(0, I)$  so that  $x_i \in \mathbb{R}^d$ . Let  $w_0^* = 0$  and create a  $w^*$  by setting the first  $k$  elements to  $\pm 10$  (choose any sign pattern) and the remaining elements to 0. Finally, generate a Gaussian noise vector  $\epsilon$  with variance  $\sigma^2$  and form  $y = Xw^* + w_0^* + \epsilon$  where  $X = [x_1, \dots, x_n]^T$  and  $y = [y_1, \dots, y_n]^T$ .

With your synthetic data, solve multiple Lasso problems on a regularization path, starting at  $\lambda_{max}$  and decreasing  $\lambda$  by a constant ratio until few features are chosen correctly. Compare the sparsity pattern of your Lasso solution  $\hat{w}$  to that of the true model parameters  $w^*$ . Record values for precision (number of correct nonzeros in  $\hat{w}$ /total number of nonzeros in  $\hat{w}$ ) and recall (number of correct nonzeros in  $\hat{w}/k$ ).

How well are you able to discover the true nonzeros? Comment on how  $\lambda$  affects these results and include plots of precision and recall vs.  $\lambda$ .

4. [5 points] We'll now put the Lasso to work on some real data from Yelp (an old Kaggle competition, see <http://www.kaggle.com/c/yelp-recruiting> for background, but get the data from the class website).

For this competition, the task is to predict the number of useful upvotes a particular review will receive. One of the most important requirements for learning great models is creating great features. We can use our Lasso solver for this as follows. First, generate a large amount of features from the data, even if many of them are likely unnecessary. Afterward, use the Lasso to reduce the number of features to a more reasonable amount.

Yelp provides a variety of data, such as the review's text, date, and restaurant, as well as data pertaining to each business, user, and check-ins. We have preprocessed this data for you into the following files:

<code>upvote_data.csv</code>	Each row is a review, each column is a real-valued feature ( $n \times d$ )
<code>upvote_labels.txt</code>	Each row is the number of useful vote counts for that review ( $n \times 1$ )
<code>upvote_features.txt</code>	Names of each feature for interpreting results ( $d \times 1$ )

To get you started, the Python following code should load the data:

```
import numpy as np
# Load a csv of floats:
X = np.genfromtxt("upvote_data.csv", delimiter=",")
# Load a text file of integers:
y = np.loadtxt("upvote_labels.txt", dtype=np.int)
# Load a text file of strings:
featureNames = open("upvote_features.txt").read().splitlines()
```

Use the first 4000 samples for training, the next 1000 samples for validation, and the remaining samples for testing.

- Solve lasso to predict the number of useful votes a Yelp review will receive. Starting at  $\lambda_{max}$ , run Lasso on the training set, decreasing  $\lambda$  using previous solutions as initial conditions to each problem. Stop when you have considered enough  $\lambda$ 's that, based on validation error, you can choose a good solution with confidence (for instance, when validation error begins increasing or stops decreasing significant). On the same plot, plot the squared error on the training and validation data versus  $\lambda$ . In addition, plot the number of nonzeros in each solution versus  $\lambda$ .
- Find the  $\lambda$  that achieves best validation performance, and test your model on the remaining set of test data. What is the test error?
- Inspect your solution and take a look at the 10 features with weights largest in magnitude. List the names of these features and their weights, and comment on if the weights generally make sense intuitively.

## 2.1 Programming: Binary Logistic Regression

5. [5 points] Let us again consider the MNIST dataset, but now just binary classification, specifically, recognizing if a digit is a 2 or 7. Here, let  $Y = 1$  for all the 7's digits in the dataset, and use  $Y = -1$  for 2. We will use regularized logistic regression. Given a binary classification dataset  $\{(x_i, y_i)\}_{i=1}^n$  for  $x_i \in \mathbb{R}^d$  and  $y_i \in \{-1, 1\}$  we showed in class that the regularized negative log likelihood objective function can be written as

$$J(w, b) = \frac{1}{n} \sum_{i=1}^n \log(1 + \exp(-y_i(b + x_i^T w))) + \lambda \|w\|_2^2$$

Note that the offset term  $b$  is not regularized. For all experiments, use  $\lambda = 10^{-1}$ . Let  $\mu_i(w, b) = \frac{1}{1 + \exp(-y_i(b + x_i^T w))}$ .

- a. Derive the gradients  $\nabla_w J(w, b)$ ,  $\nabla_b J(w, b)$  and Hessians  $\nabla_w^2 J(w, b)$ ,  $\nabla_b^2 J(w, b)$  and give your answers in terms of  $\mu_i(w, b)$  (your answers should not contain exponentials).
- b. Implement gradient descent with an initial iterate of all zeros. Try several values of step sizes to find one that appears to make  $J(w, b)$  on the training set converge the fastest. Run until you feel you are near to convergence.
  - (a) For both the training set and the test, plot  $J(w, b)$  as a function of the iteration number (and show both curves on the same plot).
  - (b) For both the training set and the test, classify the points according to the rule  $\text{sign}(b + x_i^T w)$  and plot the misclassification error as a function of the iteration number (and show both curves on the same plot).

Note that you are only optimizing on the training set. The  $J(w, b)$  and misclassification error plots should be on separate plots.

- c. Repeat (b) using stochastic gradient descent with batch size of 1. Note, the expected gradient with respect to the random selection should be equal to the gradient found in part (a). Take careful note of how to scale the regularizer.
- d. Repeat (b) using stochastic gradient descent with batch size of 100. That is, instead of approximating the gradient with a single example, use 100.
- e. Repeat (b) using Newton's method.