15-780: GRADUATE AI (SPRING 2018)

Homework 2: Optimization and Machine Learning

Release: February 19, 2018,
Last Updated: February 22, 2018, 11:15pm,
Due: March 2, 2018, 11:59pm

1 LP Duality [5+5+20 = 30 points]

Suppose that there is a set of n items (indexed as $1, 2, \ldots, i, \ldots, n$) that you would like to buy. However, these items are only sold in bundles. There are m bundles (indexed as $1, 2, \ldots, j, \ldots, m$), and each bundle j corresponds to a particular subset of items $S_j \subseteq \{1, 2, \ldots n\}$. If you buy bundle j, then you get all the items in S_j. Since the salesperson believes that items of higher indices are more valuable, they determine the cost of a bundle w_j as the product of the number of items in the bundle and the largest index of an item in that bundle:

$$w_j = |S_j| \max_{i \in S_j} i. \tag{1}$$

Being a poor grad student, you want to buy at least one copy of each of the n items while spending as little money as possible. You recall from your Grad AI course that your situation can be formulated as a binary integer program over the binary variables x_j (where $x_j = 1$ indicates that you buy bundle S_j and $x_j = 0$ indicates that you do not):

$$\begin{aligned}
&\text{minimize} && \sum_{j=1}^{m} w_j x_j && \\
&\text{subject to} && \sum_{j\,:\,i \in S_j} x_j \;\geq 1, \;\; i = 1, \ldots, n && \text{(i.e., at least one copy of every item must be bought)} \\
& && x_j \in \{0, 1\}, \;\; j = 1, \ldots, m && \text{(i.e., all variables must be binary).}
\end{aligned}$$

Since you recall from class that linear programs are easier to solve than integer programs, you then relax this

into a linear program:

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{m} w_j x_j \\ \text{subject to} \quad & \sum_{j\,:\,i \in S_j} x_j \ \geq 1, \ \ i = 1, \ldots, n \\ & x_j \geq 0 \ \ j = 1, \ldots, m. \end{aligned}$$

For all the following questions, we consider this linear program relaxation to be our primal problem.

1) Consider the following n bundles (for this example, $m = n$): $S_1 = \{1, 2, \ldots, n\}, S_2 = \{2, 3, \ldots, n\}, \ldots, S_n = \{n\}$. What is the optimal value of the primal LP objective in terms of n? Where is it attained? (You must argue why it is optimal.)

2) Consider the following n bundles (for this example, $m = n$): $S_j = \{j\}$ for all j. What is the optimal value of the primal LP objective in terms of n? Where is it attained? (You must argue why it is optimal.)

3) By making use of the dual formulation, show that regardless of m, the primal LP objective can never be lower than $n(n+1)/2$.[1]

2 Least Squares Estimation [5 points]

In class, we saw that the closed form solution of the least squares regression problem is given as

$$\theta^\star = \left(X^T X\right)^{-1} X^T y.$$

Unfortunately, this solution does not work if $X^T X$ is not invertible. However, we can always define a solution in terms of the pseudoinverse X^+ of the matrix X. Show that

$$\theta^\star = X^+ y$$

is always the minimizer of the least squares regression problem.

Note: Recall that the pseudoinverse of X (denoted by X^+) is a matrix satisfying the following properties:

1. $XX^+X = X$
2. $X^+XX^+ = X^+$
3. $\left(XX^+\right)^T = XX^+$
4. $\left(X^+X\right)^T = X^+X$.

[1] The lower bound was incorrectly stated as $n(n-1)/2$ and this has been corrected.

3 VC Dimension [45 points]

3.1 Bounding Boxes in $\mathbb{R}^d$ [5+5=10 points]

In this question, we will determine the VC-dimension of the class of axis-aligned boxes $\mathcal{F}$ in d dimensions. Specifically, each $f_{a,b} \in \mathcal{F}$ can be defined by parameters $a = (a_1, a_2 \ldots, a_d) \in \mathbb{R}^d$ and $b = (b_1, b_2 \ldots, b_d) \in \mathbb{R}^d$ such that $f_{a,b}$ labels a point $x = (x_1, x_2 \ldots, x_d) \in \mathbb{R}^d$ via:

$$f_{a,b}(x) = \begin{cases} + & \text{if } a_i \leq x_i \leq b_i \text{ for all } i = 1, 2, \ldots d \\ - & \text{otherwise.} \end{cases} \tag{2}$$

1) Prove that the VC-dimension of $\mathcal{F}$ is *at least* $2d$.

2) Prove that the VC-dimension of $\mathcal{F}$ is *at most* $2d$. Conclude that the VC-dimension is thus exactly $2d$.

3.2 Decision Tree Classifiers [10 points]

In this problem, we will consider the VC dimension of decision tree classifiers on the real line. Specifically, each hypothesis $h : \mathbb{R} \to \{+, -\}$ is a full binary tree (i.e., a tree in which every non-leaf node has exactly two children), in which each node n is assigned a scalar-valued parameter $\text{value}(n)$. (That is, the parameters of each hypothesis are the scalar values assigned to each node in the tree.) For each node n_t with left and right children n_ℓ and n_r, respectively, the nodes' scalar parameters satisfy the relationship that $\text{value}(n_\ell) \leq \text{value}(n_t) \leq \text{value}(n_r)$.

Given some input $v \in \mathbb{R}$, the classification $h(v)$ assigned to v by the tree h can be computed as follows: Start at the root n of the tree. If $v < \text{value}(n)$, then we proceed to the left child of n; otherwise, we go right. We repeat the same process for internal nodes until reaching a leaf n_{leaf} of the tree. If $v < \text{value}(n_{\text{leaf}})$, then we classify it as $-$; otherwise, we classify it as $+$.

Let H_d be the class containing all decision trees of depth d (where a tree's root node is considered to be at a depth of 1). For instance, a generic decision tree of depth $d = 2$ is shown in Figure 1, and the corresponding hypothesis class would be $H_2 = \{h_{t,\ell,r} | t, \ell, r \in \mathbb{R}, \ell \leq t \leq r\}$.

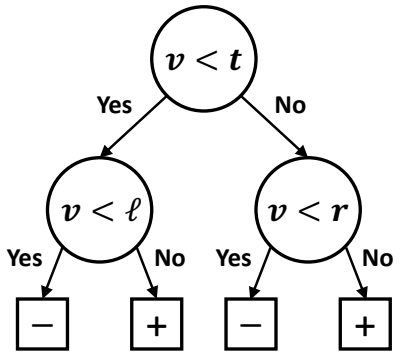

Figure 1: Decision tree of depth 2, with parameters t, ℓ, and r such that $\ell \leq t \leq r$.

Prove that the VC dimension of H_d is $2^d - 1$ for $d \in \mathbb{N}$.

Hint: One way to prove the lower bound is via induction.

3.3 Union of Function Classes [25 points]

In this section, we will analyze the VC dimension of the union of two function classes.

3.3.1 Upper Bound [8 points]

For any $d \in \mathbb{N}$, show that if two function classes $\mathcal{F}_1$ and $\mathcal{F}_2$ each have a VC dimension of d, their union $\mathcal{F}_1 \cup \mathcal{F}_2$ has a VC dimension of at most $2d + 1$.

3.3.2 Tightness of Upper Bound [17 points]

For each $d \in \mathbb{N}$, show two function classes $\mathcal{F}_1$ and $\mathcal{F}_2$ each with a VC dimension of d, whose union has a VC dimension of $2d + 1$.

Hints:

- Consider the input space X to have just $2d+1$ points. And, let $\mathcal{F}$ be the class of all possible functions on this space. Can $\mathcal{F}$ be partitioned into two classes such that both have a VC dimension of at most d?
- For a space of n points $\{x_1, x_2, \ldots, x_n\}$, suppose a function class $\mathcal{F}$ is such that every $f \in \mathcal{F}$ satisfies

$$\sum_{i=1}^{n} \mathbb{1}\{f(x_i) = +\} \leq m,$$

  where $\mathbb{1}\{\cdot\}$ denotes the indicator function and m is some non-negative integer. What can be said about the VC dimension of $\mathcal{F}$?

4 Classification Programming [20 points]

In this section, you will develop a few different multi-class classifiers to classify digits from the MNIST data set. We will extend the notation used in class a bit, and use a loss function that *directly* captures a k-class classification tasks, called the softmax or cross-entropy loss. In our new setting, we have a training set of the form $(x^{(i)}, y^{(i)})$, $i = 1, \ldots, m$, with $y^{(i)} \in \{0, 1\}^k$ (remember that k is the number of classes we're trying to predict), where $y_j^{(i)} = 1$ when j is the target class, and 0 otherwise. That is, if output values can take on one of 10 classes (as will be the case in the digit classification task), and the target class for this example is

class 4, then corresponding $y^{(i)}$ is simply

$$y^{(i)} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{3}$$

This is sometimes called a "one-hot" encoding of the output class.

Under the model, our hypothesis function $\hat{y} = h_\theta(x)$ will now output *vectors* in $\mathbb{R}^k$, where the relative size of the $\hat{y}_j$ corresponds roughly to how likely we believe that the output is really class j (this will become more concrete when we formally define the class function). For instance, the (hypothetical) output

$$\hat{y} = h_\theta(x^{(i)}) = \begin{bmatrix} 0.1 \\ -0.2 \\ 2.0 \\ 5.0 \\ 0.1 \\ -1.0 \\ -5.0 \\ 1.0 \\ 0.4 \\ 0.2 \end{bmatrix} \tag{4}$$

would correspond to a predict that the example $x^{(i)}$ is probably from class 4 (the element with the largest entry). Analogous to binary classification (where, if we wanted binary prediction we would simply take the sign of the hypothesis function), if we want the to predict a single class label for the output, we simply predict class class j for which $\hat{y}_j$ takes on the largest value.

Our loss function is now defined as a function $\ell : \mathbb{R}^k \times \{0,1\}^k \to \mathbb{R}_+$ that quantifies how good a prediction is. In the "best" case, the predictions $\hat{y}_j$ would be $+\infty$ for the true class (i.e. for the element where $y_j^{(i)} = 1$) and $-\infty$ otherwise (of course, we usually won't make infinite predictions, because we would then suffer very high loss if we ever made a mistake, and we won't get such predictions with most classifiers if we

include a regularization term). The loss function we use is the softmax loss, given by[2]

$$\ell(\hat{y}, y) = \log\left(\sum_{j=1}^{k} e^{\hat{y}_j}\right) - \hat{y}^T y. \tag{7}$$

This loss function has the gradient

$$\nabla_{\hat{y}} \ell(\hat{y}, y) = \frac{e^{\hat{y}}}{\sum_{j=1}^{k} e^{\hat{y}_j}} - y \tag{8}$$

where the exponent $e^{\hat{y}}$ is taken elementwise.

In practice, you would probably want to implemented regularized loss minimization, but for the sake of this problem set, we'll just consider minimizing loss without any regularization (at the expense of overfitting a little bit).

4.1 Linear softmax regression [20 points]

In this section, you'll implement a linear classification model to classify digits. That is, our hypothesis function will be

$$h_\theta(x^{(i)}) = \Theta \begin{bmatrix} x^{(i)} \\ 1 \end{bmatrix} \tag{9}$$

where $\Theta \in \mathbb{R}^{10 \times 785}$ is our vector of parameters. Using a simple application of the chain rule (similar to that applied to derive backpropagation), we can compute the gradient of our loss function for this hypothesis class

$$\nabla_\Theta \ell(h_\theta(x^{(i)}), y) = \nabla_{\hat{y}} \ell(\hat{y}, y) \begin{bmatrix} {x^{(i)}}^T & 1 \end{bmatrix}. \tag{10}$$

where $\hat{y} \equiv h_\theta(x^{(i)})$ and where the gradient $\nabla_{\hat{y}} \ell(\hat{y}, y)$ is given in (8) above (to make this a bit simplier, in practice you can just create a new X matrix with an additional column of all ones added).

Gradient descent Implement the gradient descent algorithm to solve this optimization problem. Recall from the slides that the gradient descent algorithm is given by:

function $\theta = \text{Gradient-Descent}(\{(x^{(i)}, y^{(i)})\}, h_\theta, \ell, \alpha)$
  Initialize: $\theta \leftarrow 0$
  For $t = 1, \ldots, T$:

[2] It won't be relevant to the implementation in this problem, but for those curious where this loss function comes from, softmax regression can be interpreted as a probabilistic model where

$$p(y = j|\hat{y}) = \frac{e^{\hat{y}_j}}{\sum_{l=1}^{k} e^{\hat{y}_l}}. \tag{5}$$

If we look at maximum likelihood estimation under this true model, we get the optimization problem

$$\underset{\theta}{\text{minimize}} - \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)}) \equiv \underset{\theta}{\text{minimize}} \sum_{i=1}^{m} \log\left(\sum_{j=1}^{k} e^{h_\theta(x^{(i)})_j}\right) - h_\theta(x^{(i)})^T y^{(i)}. \tag{6}$$

$g \leftarrow 0$
For $i = 1, \ldots, m$:
$g \leftarrow g + \frac{1}{m} \nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$
$\theta \leftarrow \theta - \alpha g$
return θ

i.e., we compute the gradient with respect to the loss function for all the examples, divide it by the total number of examples, and take a small step in this direction (you can leave all the step sizes α at their default values for the programming part). Each pass over the whole dataset is referred to as an *epoch*.

Specifically, you'll implement the softmax_gd function:

```
def softmax_gd(X, y, Xt, yt, epochs=10, alpha = 0.5):
"""
Run gradient descent to solve linear softmax regression.

Inputs:
X: numpy array of training inputs
y: numpy array of training outputs
Xt: numpy array of testing inputs
yt: numpy array of testing outputs
epochs: number of passes to make over the whole training set
alpha: step size

Outputs:
Theta: 10 x 785 numpy array of trained weights
"""
```

In addition to outputing the trained parameters, your function should print[3] the error (computed by the included `error` function) on the test and training sets, and the softmax loss averaged on the test and training sets (again using the provided function, which will help you in computing the gradient). In your PDF, you should report all these values computed at each iteration (epoch) *before* adjusting the parameters for that iteration[4] . For example, our implementation of gradient descent outputs the following:

```
Test Err  |Train Err |Test Loss |Train Loss
0.9020|    0.9013|    2.3026|    2.3026
0.3192|    0.3276|    1.8234|    1.8318
0.2101|    0.2228|    1.4983|    1.5141
0.2142|    0.2246|    1.2830|    1.3018
0.1844|    0.1935|    1.1341|    1.1555
0.1816|    0.1924|    1.0260|    1.0490
0.1702|    0.1785|    0.9463|    0.9697
0.1670|    0.1754|    0.8826|    0.9072
0.1613|    0.1681|    0.8334|    0.8576
0.1559|    0.1653|    0.7909|    0.8160
```

[3]The word used here was originally "output" but that has been changed to "print" to avoid confusion regarding what the function should return.

[4]Clarified that all the four values are averaged over the train and test sets and must be reported in the PDF.

Stochastic gradient descent Implement the stochastic gradient descent algorithm for linear softmax regression. Recall from the notes that the SGD algorithm is:

function $\theta = \text{SGD}(\{(x^{(i)}, y^{(i)})\}, h_\theta, \ell, \alpha)$
  Initialize: $\theta \leftarrow 0$
  For $t = 1, \ldots, T$:
    For $i = 1, \ldots, m$:
      $\theta \leftarrow \theta - \alpha \nabla_\theta \ell(h_\theta(x^{(i)}), y^{(i)})$
  return θ

That is, we take small gradient steps on *each* example, rather than computing the average gradient over all the examples.

You'll implement the following function:

```
def softmax_sgd(X, y, Xt, yt, epochs=10, alpha = 0.5):
"""
Run stochastic gradient descent to solve linear softmax
regression.

Inputs:
X: numpy array of training inputs
y: numpy array of training outputs
Xt: numpy array of testing inputs
yt: numpy array of testing outputs
epochs: number of passes to make over the whole training set
alpha: step size

Outputs:
Theta: 10 x 785 numpy array of trained weights
"""
```

You should print[5] the same information (train/test error/loss) as for the above function, but importantly only print this information *once* per outer iteration, before you iterate over all the examples. Report these values in your PDF.

5 Submitting to Autolab

Create a tar file containing your writeup and the completed `cls.py` modules for the programming problems. Make sure that your tar has these files at the root and not in a subdirectory. Use the following commands from a directory with your files to create a `handin.tgz` file for submission.

```
$ ls
cls.py  writeup.pdf
$ tar cvzf handin.tgz writeup.pdf cls.py
a writeup.pdf
```

[5] The word used here was originally "output" but that has been changed to "print" to avoid confusion regarding what the function should return.

```
a cls.py
$ ls
handin.tgz cls.py writeup.pdf
```