EECS545 Machine Learning
Homework #3

Due date: 11:55pm, Tuesday 2/25/2020

Reminder: While you are encouraged to discuss problems in a small group (up to 5 people), you should write your own solutions and source codes independently. In case you worked in a group for the homework, please include the names of your collaborators in the homework submission. Your answers should be as concise and clear as possible. Please address all questions to `http://piazza.com/class#winter2020/eecs545` with a reference to the specific question in the subject line (E.g. RE: Homework 3, Q2(c)).
Submission Instruction: You should submit both **solution** and **source code**. We may inspect your source code submission visaully and run the code to check the results. Please be aware your points can be deducted if you don't follow the instructions listed below:

- Submit **solution** to **Gradescope**
    - Solution should contain your answers to **all** questions (typed or hand-written).
- Submit **source code** to **Canvas**
    - Source code should contain your codes to programming quesitons.
    - Source code should be **1 zip** file named **'HW3_yourUMID_yourfirstname_yourlastname.zip'**
    - The zip file should contain the original files in the **HW3.zip** that we provided; you should not change the file names and folder structure, so that your source code can be run after unzipping your zip file.

In summary, your source code submission for HW3 should be 1 zip file which includes

- q4.ipynb
- q5.ipynb
- q4_data
- q5_data

Source Code Instruction: Your source code should run under an environment with following libraries:

- Python 3.6+
- Numpy (for implementations of algorithms)
- Matplotlib (for plots)

Please do not use any other library unless otherwise instructed. You should be able to load the data and execute your code on someone else's computer with the environment described above. In other words, work with the setup we provide and do not change the directory structure. Use relative path instead of absolute path to load the data. Note, the outputs of your source code must match with your solution.

1 [15 points] MAP estimates and weight decay

Consider using a logistic regression model $p(y = 1|\mathbf{x}; \mathbf{w}) = g(\mathbf{w}^T\mathbf{x})$ where g is the sigmoid function, and let a training set $\{(\mathbf{x}^{(i)}, y^{(i)}); i = 1, ..., N\}$ be given as usual. The maximum likelihood estimate of the parameters $\mathbf{w}$ is given by

$$\mathbf{w}_{\text{ML}} = \arg\max_{\mathbf{w}} \prod_{i=1}^{N} p(y^{(i)}|\mathbf{x}^{(i)}; \mathbf{w}). \tag{1}$$

If we wanted to regularize logistic regression, then we might put a prior on the parameters. Suppose we chose the prior $\mathbf{w} \sim \mathcal{N}(0, \tau^2 I)$ (here, $\tau > 0$, and I is the $M + 1$-by-$M + 1$ identity matrix), and then found the MAP estimate of $\mathbf{w}$ as:

$$\mathbf{w}_{\text{MAP}} = \arg\max_{\mathbf{w}} p(\mathbf{w}) \prod_{i=1}^{N} p(y^{(i)}|\mathbf{x}^{(i)}; \mathbf{w}). \tag{2}$$

Prove that

$$\|\mathbf{w}_{\text{MAP}}\|_2 \leq \|\mathbf{w}_{\text{ML}}\|_2 \tag{3}$$

2 [25 + 4 points] Direct construction of valid kernels

In class, we saw that by choosing a kernel $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T\phi(\mathbf{z})$, we can implicitly map data to a high dimensional space, and have the SVM algorithm work in that space. One way to generate kernels is to explicitly define the mapping ϕ to a higher dimensional space, and then work out the corresponding k.

However in this question we are interested in direct construction of kernels. I.e., suppose we have a function $k(\mathbf{x}, \mathbf{z})$ that we think gives an appropriate similarity measure for our learning problem, and we are considering plugging k into a kernelized algorithm (like SVM) as the kernel function. However for $k(\mathrm{x}, \mathrm{z})$ to be a valid kernel, it must correspond to an inner product in some higher dimensional space resulting from some feature mapping ϕ. Mercer's theorem tells us that $k(\mathbf{x}, \mathbf{z})$ is a (Mercer) kernel if and only if for any finite set $\left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\right\}$, the matrix K is symmetric and positive semi-definite, where the square matrix $K \in \mathbb{R}^{N \times N}$ is given by $K_{ij} = k\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)$

Now here comes the question: Let k_1, k_2 be kernels over $\mathbb{R}^D \times \mathbb{R}^D$, let $a \in \mathbb{R}^+$ be a positive real number, let $f : \mathbb{R}^D \to \mathbb{R}$ be a real-valued function, let $\phi : \mathbb{R}^D \to \mathbb{R}^M$ be a function mapping from $\mathbb{R}^D$ to $\mathbb{R}^M$, let k_3 be a kernel over $\mathbb{R}^M \times \mathbb{R}^M$, and let $p : \mathbb{R} \to \mathbb{R}$ be a polynomial with positive coefficients.

For each of the functions k below, state whether it is necessarily a kernel. If you think it is, please prove it; if you think it is not, please give a counterexample.

(a) [3 points] $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) + k_2(\mathbf{x}, \mathbf{z})$

(b) [3 points] $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) - k_2(\mathbf{x}, \mathbf{z})$

(c) [3 points] $k(\mathbf{x}, \mathbf{z}) = ak_1(\mathbf{x}, \mathbf{z})$

(d) [3 points] $k(\mathbf{x}, \mathbf{z}) = -ak_1(\mathbf{x}, \mathbf{z})$

(e) [4 points] $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z})k_2(\mathbf{x}, \mathbf{z})$

(f) [3 points] $k(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$

(g) [3 points] $k(\mathbf{x}, \mathbf{z}) = k_3(\phi(\mathbf{x}), \phi(\mathbf{z}))$

(h) [3 points] $k(\mathbf{x}, \mathbf{z}) = p\left(k_1(\mathbf{x}, \mathbf{z})\right)$

(i) [4 points extra credit] Prove that the gaussian Kernel, $k(\mathbf{x}, \mathbf{z}) = \exp\left(-\|\mathbf{x} - \mathbf{z}\|^2/2\sigma^2\right)$ can be expressed as $\phi(\mathbf{x})^T\phi(\mathbf{z})$, where $\phi(.)$ is an infinite-dimensional vector.

[*Hint:* $\|\mathbf{x} - \mathbf{z}\|^2 = x^Tx + z^Tz - 2x^Tz$. *Consider using Power Series.*]

3 [20 points] Kernelizing the perceptron

Let there be a binary classification problem with $t \in \{0, 1\}$. The perceptron uses hypotheses of the form $y(\mathbf{x}; \mathbf{w}) = f\left(\mathbf{w}^T \mathbf{x}\right)$, where $f(z) = \mathbf{I}[\mathbf{z} \geq \mathbf{0}]$. In this problem we will consider a stochastic gradient descent-like implementation of the perceptron algorithm where each update to the parameters $\mathbf{w}$ is made using only one training example. However, unlike stochastic gradient descent, the perceptron algorithm will only make one pass through the entire training set. The update rule for this version of the perceptron algorithm is given by

$$\mathbf{w}^{(i+1)} := \mathbf{w}^{(i)} + \alpha \left[y^{(i+1)} - h\left(\mathbf{x}^{(i+1)}; \mathbf{w}^{(i)}\right) \right] \mathbf{x}^{(i+1)}$$

where $\mathbf{w}^{(i)}$ is the value of the parameters after the algorithm has seen the first i training examples. Prior to seeing any training examples, $\mathbf{w}(0)$ is initialized to $\mathbf{0}$.

Let K be a Mercer kernel corresponding to some very high-dimensional feature mapping ϕ. Suppose ϕ is so high-dimensional (say, infinite-dimensional) that it's infeasible to ever represent ϕ(x) explicitly. Describe how you would apply the "kernel trick" to the perceptron to make it work in the high-dimensional feature space ϕ, but without ever explicitly computing $\phi(\mathbf{x})$. [Note: You don't have to worry about the intercept term. If you like, think of ϕ as having the property that $\phi_0(\mathbf{x}) = 1$ so that this is taken care of.] Your description should specify:

(a) [6 points] How you will represent the high-dimensional parameter vector $\mathbf{w}^{(i)}$ using the feature vectors $\phi\left(\mathbf{x}^{(i)}\right)$ including how the initial value $\mathbf{w}^{(0)} = \mathbf{0}$ is represented;

(b) [7 points] How you will efficiently make a prediction on a new input $\mathbf{x}^{(i+1)}$. I.e., how you will compute $h\left(\mathbf{x}^{(i+1)}; \mathbf{w}^{(i)}\right) = f\left(\mathbf{w}^{(i)T} \phi\left(\mathbf{x}^{(i+1)}\right)\right)$, using your representation of $\mathbf{w}^{(i)}$; [Note: In Kernelizing, we should show that all the occurrences of feature $\phi(\mathbf{x})$ are in the form of inner products. And then we can replace them with a kernel to avoid explicit computation on the features when learning model parameters as well as making predictions.]

(c) [7 points] How you will modify the update rule given above to perform an update to w on a new training example $\left(\mathbf{x}^{(i+1)}, y^{(i+1)}\right)$; i.e., using the update rule corresponding to the feature mapping ϕ :

$$\mathbf{w}^{(i+1)} := \mathbf{w}^{(i)} + \alpha \left[y^{(i+1)} - h\left(\phi\left(\mathbf{x}^{(i+1)}\right); \mathbf{w}^{(i)}\right) \right] \phi\left(\mathbf{x}^{(i+1)}\right)$$

[Note: If you prefer, you are also welcome to do this problem using the convention of labels $y \in \{-1, 1\}$, and $f(z) = \text{sign}(z) = 1$ if $z \geq 0, -1$ otherwise.

4 [25 points] Implementing Soft Margin SVM by Optimizing Primal Objective

Support Vector Machines (SVM) is a discriminative model for classification. Although it is possible to develop SVMs that do K class classifications, we are going to restrict ourselves to binary classification in this question, where the class label is either $+1$ (positive) or -1 (negative). SVM is not a probabilistic algorithm. In other words, in its usual construction, it does not optimize a probability measure as a likelihood. SVM tries to find the "best" hyperplane that *maximally*[1] separates the positive class from the negative class.

Recall that the objective function for maximizing the soft margin is equivalent to the following minimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \tag{4}$$

$$\text{subject to } y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \geq 1 - \xi_i,\ \forall i = 1,\ldots,N \tag{5}$$

$$\xi_i \geq 0,\ \forall i = 1,\ldots,N \tag{6}$$

The above is known as the Primal Objective of SVM. Notice the two constraints on ξ_i. It means that ξ_i must satisfy both of those conditions, while minimizing the sum of ξ_i 's times C. The constrained minimization is equivalent to following minimization:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\max\left(0, 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right)\right)$$

You will be working with the above minimization in this problem. However, please think to yourself why combining the minimization of ξ_i' s and the two constraints became a max as shown above. To shorten notation, lets define our loss function $E(\mathbf{w}, b)$ as:

$$E(\mathbf{w},b) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\max\left(0, 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right)\right) \tag{7}$$

(a) [6 points] Find the derivatives of the loss function with respect to our parameters. Show that:

$$\nabla_{\mathbf{w}} E(\mathbf{w},b) = \mathbf{w} - C\sum_{i=1}^{N}\mathbf{I}\left[y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) < 1\right] y^{(i)}\mathbf{x}^{(i)} \tag{8}$$

$$\frac{\partial}{\partial b} E(\mathbf{w},b) = -C\sum_{i=1}^{N}\mathbf{I}\left[y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) < 1\right] y^{(i)}, \tag{9}$$

where $\mathbf{I}[.]$ is the indicator function.

(b) [8 points] Implement the SVM algorithm using batch gradient descent. In previous assignments, you have implemented gradient descent while minimizing over one variable. Minimizing over two variables (w, b) is not different. Both gradients are computed from current parameter values, and then parameters are simultanously updated. Note: it is a common mistake to implement gradient descent over multiple parameters by updating the first parameter, then computing the derivative w.r.t second parameter using the updated value of the first parameter[2]. The pseudo code for optimizing $\mathbf{w}$ and b is given by:

[1] maximally means increasing the geometric margin. Review lecture slides if you need a refresher.

[2] In fact, updating one parameter then computing the gradient of the second parameter using the updated value of the first parameter, is a different optimization algorithm, known as Coordinate Descent.

Algorithm 1: SVM Batch Gradient Descent

$\mathbf{w}^* \leftarrow 0$;
$b^* \leftarrow 0$;
for $j = 1$ *to* $NumIterations$ **do**
  $\mathbf{w}_{grad} \leftarrow \nabla_{\mathbf{w}} E(\mathbf{w}^*, b^*)$;
  $b_{grad} \leftarrow \frac{\partial}{\partial b} E(\mathbf{w}^*, b^*)$;
  $\mathbf{w}^* \leftarrow \mathbf{w}^* - \alpha(j)\mathbf{w}_{grad}$;
  $b^* \leftarrow b^* - 0.01\alpha(j) b_{grad}$;
end
return $\mathbf{w}^*$

The learning rate $\alpha(.)$ is now a function of time (iteration number) rather than a constant. This allows us to define a decaying learning rate as:

$$\alpha(j) = \frac{\eta_0}{1 + j\eta_0}$$

Throughout this question, set $\eta_0 = 0.5$ and the slack cost $C = 5$. Loading the file `q4_data.npy` loads the arrays `q4x_train`, `q4y_train`, `q4x_test`, and `q4y_test`. Run your implementation of SVM Batch Gradient. Descent over the training data 4 times, once for each NumIterations= $\{5, 50, 100, 1000\}$ For each run, report your trained parameters $(\mathbf{w}, b)$ and the test classification accuracy.

(c) [3 points] In Stochastic Gradient Descent, we compute the gradients and update our parameters for every training example (rather than batch updating). To do stochastic gradient descent properly, we need to define a loss function per example (the loss function $E(\mathbf{w}, b)$, above, is defined over the entire training set). We can rewrite the loss function above as:

$$\begin{aligned} E(\mathbf{w}, b) &= \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N} \max\left(0, 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right)\right) \\ &= \sum_{i=1}^{N}\left[\frac{1}{2N}\|\mathbf{w}\|^2 + C\max\left(0, 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right)\right)\right] \end{aligned}$$

Expressing the error function as one summation allows us to define an error term per training example like:

$$E^{(i)}(\mathbf{w}, b) = \frac{1}{2N}\|\mathbf{w}\|^2 + C\max\left(0, 1 - y^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right)\right)$$

$$\text{then }, E(\mathbf{w}, b) = \sum_{i=1}^{N} E^{(i)}(\mathbf{w}, b)$$

Find the expressions for $\nabla_{\mathrm{w}} E^{(i)}(\mathrm{w}, b)$ and $\frac{\partial}{\partial b} E^{(i)}(\mathrm{w}, b)$

(d) [5 points] Implement the SVM algorithm using stochastic gradient descent (SGD). The pseudo-code

looks like:

Algorithm 2: SVM Batch Gradient Descent

$\mathbf{w}^* \leftarrow 0$;
$b^* \leftarrow 0$;
for $j = 1$ *to* $NumIterations$ **do**
  for $i = 1$ *to* N **do**
    $\mathbf{w}_{grad} \leftarrow \nabla_{\mathbf{w}} E^{(i)}(\mathbf{w}^*, b^*)$;
    $b_{grad} \leftarrow \frac{\partial}{\partial b} E^{(i)}(\mathbf{w}^*, b^*)$;
    $\mathbf{w}^* \leftarrow \mathbf{w}^* - \alpha(j)\mathbf{w}_{grad}$;
    $b^* \leftarrow b^* - 0.01\alpha(j)b_{grad}$;
  end
end
return $\mathbf{w}^*$

Use the same $\alpha(.), \eta_0, C$ as part (b). Run your implementation of SVM Stochastic Gradient Descent over the training data 4 times, once for each NumIterations= $\{5, 50, 100, 1000, 5000, 6000\}$ For each run, report your trained parameters $(\mathbf{w}, b)$ and the test classification accuracy.

(e) [3 points] What can you conclude about the convergence rate of Stochastic gradient descent (SGD) versus gradient descent? How did you make this conclusion?

5 [15 points] Scikit-learn SVM for classifying SPAM

Recall the Q4 in HW2. We will repeat it using scikit-learn SVM, and compare Naive-Bayes and SVM.
NOTE: If you are already familiar with the data used for Question 4 in HW2, you can skip the paragraphs below and start from (a).

In our data, the text emails have been pre-processed so that they can be used for naive Bayes. The pre-processing ensures that only the email body and subject remain in the dataset; email addresses (*EMAILADDR*), web addresses (HTTPADDR), currency (DOLLAR) and numbers (NUMBER) were also replaced by the special tokens to allow them to be considered properly in the classification process. (In this problem, we'll going to call the features "tokens".)

We have done the feature extraction works for you, so you can just load the data matrices (called document-term matrices in text classification) which contain all the data. In a document-term matrix, the i^{th} row represents the i^{th} document/email, and the j^{th} column represents the j^{th} distinct token. Thus, the (i, j) entry of this matrix represents the number of occurrences of the jth token in the ith document.

For this problem, we chose the set of tokens (vocabulary) to only contain the medium frequency tokens, as the tokens that occur too often or too rarely do not have much classification value. (Examples: tokens that occur very often are terms like "the," "and," and "of," which occur in any spam and non-spam emails.) Also, terms were stemmed using a standard stemming algorithm; basically, this means that "price," "prices" and "priced" have all been replaced with "price," so that they can be treated as the same token.

Since the document-term matrix is sparse (has lots of zero entries), we store it in an efficient format to save space. We provide a starter code **q5.ipynb** which contains the **readMatrix** function that reads in the document-term matrix, the correct class labels for all emails, and the full list of tokens.

(a) **[7 points]** Implement an SVM classifier with **linear kernel** for spam classification using a python package named *scikit-learn*[3]. You can install this package by running the first code cell in the **q5.ipynb** file. You should use the **LinearSVC** class[4] in this package to train your parameters with the training dataset **MATRIX.TRAIN**. Then, use these parameters to classify the test dataset **MATRIX.TEST**

[3] See `https://scikit-learn.org/stable/` for more information about the package.
[4] See `https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html` for more information about the **LinearSVC** class.

and report the error using the **evaluate** function. Implement your code in **q5.ipynb**.
Note: Make sure to use a **linear kernel-SVM**; use the *LinearSVC* class already imported in **q5.ipynb**.

(b) **[4 points]** Repeat part (a), but with training sets of size ranging from 50, 100, 200, . . . , up to 1400, by using the files `MATRIX.TRAIN.*`. Plot the test error each time (use `MATRIX.TEST` as the test data) to obtain a learning curve (test set error vs. training set size). You may need to change the call to `readMatrix` in `nb_train.m` to read the correct file each time. Which training-set size gives the best test set error?

(c) **[4 points]** Recall the result you got for Q4 in HW2. How do naive Bayes and Support Vector Machines compare (in terms of generalization error) as a function of the training set size?

Credits

Some questions adopted/adapted from http://www.stanford.edu/class/cs229/ps/ps3.pdf and from Bishop PRML.