EECS445: Introduction to Machine Learning, Fall 2014
Homework #3

Due date: 6pm on 10/21 (Tuesday)

Reminder: While you are encouraged to think about problems in small groups, all written solutions must be independently generated. Please type or hand-write solutions legibly. While these questions require thought, please be as concise and clear in your answer as possible. Please address all questions to `http://piazza.com/class#fall2014/eecs445` with a reference to the specific question in the subject line (E.g. RE: Homework 3, Q2(c)).

Submission Instructions:
Please refer to previous homework problem sets for submission instructions.

1 [25 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 $K(\mathbf{x}, \mathbf{z})$ to be a valid kernel, it must correspond to an inner product in some higher dimensional space resulting from some feature mapping ϕ. Mercers theorem tells us that $K(\mathbf{x}, \mathbf{z})$ is a (Mercer) kernel if and only if for any finite set $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}\}$, the matrix K is symmetric and positive semidefinite, where the square matrix $K \in \mathbb{R}^N \times \mathbb{R}^N$ is given by $K_{ij} = K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$.

Now here comes the question: let K_1, K_2 be kernels over $\mathbb{R}^N \times \mathbb{R}^N$, let $a \in \mathbb{R}^+$ be a positive real number, let $f : \mathbb{R}^N \to \mathbb{R}$ be a real-valued function, let $\phi : \mathbb{R}^N \to \mathbb{R}^d$ be a function mapping from $\mathbb{R}^N$ to $\mathbb{R}^d$, let K_3 be a kernel over $\mathbb{R}^d \times \mathbb{R}^d$, 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, prove it; if you think it isnt, give a counter-example.

(a) $K(\mathbf{x}, \mathbf{z}) = K_1(\mathbf{x}, \mathbf{z}) + K_2(\mathbf{x}, \mathbf{z})$

(b) $K(\mathbf{x}, \mathbf{z}) = K_1(\mathbf{x}, \mathbf{z}) - K_2(\mathbf{x}, \mathbf{z})$

(c) $K(\mathbf{x}, \mathbf{z}) = aK_1(\mathbf{x}, \mathbf{z})$

(d) $K(\mathbf{x}, \mathbf{z}) = -aK_1(\mathbf{x}, \mathbf{z})$

(e) $K(\mathbf{x}, \mathbf{z}) = K_1(\mathbf{x}, \mathbf{z})K_2(\mathbf{x}, \mathbf{z})$

(f) $K(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$

(g) $K(\mathbf{x}, \mathbf{z}) = K_3(\phi(\mathbf{x}), \phi(\mathbf{z}))$

(h) $K(\mathbf{x}, \mathbf{z}) = p(K_1(\mathbf{x}, \mathbf{z}))$

[Hint: For part (e), the answer is that the K there is indeed a kernel. You still have to prove it, though. (This one may be harder than the rest.) This result may also be useful for another part of the problem.]

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

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

2 [25 points] Kernelizing the Perceptron

Let there be a binary classification problem with $t \in \{0, 1\}$. The perceptron uses hypotheses of the form $y(\mathbf{x}; \mathbf{w}) = f(\mathbf{w}^T\mathbf{x})$, where $f(\mathbf{z}) = \mathbf{I}[\mathbf{z} \geq \mathbf{0}]$. 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

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

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 $\phi(\mathbf{x})$ 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 dont 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) How you will (implicitly) represent the high-dimensional parameter vector $\mathbf{w}^{(i)}$, including how the initial value $\mathbf{w}^{(0)} = \mathbf{0}$ is represented (note that $\mathbf{w}^{(i)}$ is now a vector whose dimension is the same as the feature vectors $\phi(\mathbf{x})$);

(b) How you will efficiently make a prediction on a new input $\mathbf{x}^{(i+1)}$. I.e., how you will compute $y(\mathbf{x}^{(i+1)}; \mathbf{w}^{(i)}) = f(\mathbf{w}^{(i)T}\phi(\mathbf{x}^{(i+1)}))$, using your representation of $\mathbf{w}^{(i)}$; and

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

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

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

3 [32 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.

[1] maximally means increasing the geometric margin. Review lecture slides if you need a refresher. In addition, the construction of SVMs will be reviewed this Friday's discussion section

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

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \xi_i \; i = 1, \ldots, N \\ & \xi_i \geq 0 \; i = 1, \ldots, N \end{aligned}$$

The above is known as the *Primal Objective* of SVM[2]. 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:

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

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:

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

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

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

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

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

(b) **[10 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 $(\mathbf{w}, b)$ 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[3]. The pseudo code for optimizing $\mathbf{w}$ and b is given by:

Algorithm 1: SVM Batch Gradient Descent

$\mathbf{w}^* \leftarrow \mathbf{0}$;
$b^* \leftarrow 0$;
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 $\mathbf{w}^*$

[2] In EECS445, we don't dive deep into the *Dual Objective* of SVM. The Dual formulation is covered in depth in EECS545

[3] 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

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 `q3.mat` loads the variables `q3x_train`, `q3t_train`, `q3x_test`, and `q3t_test`. Run your implementation of SVM Batch Gradient Descent over the training data 6 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.

(c) **[4 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:

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

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

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

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

(d) **[8 points]** Implement the SVM algorithm using stochastic gradient descent (SGD). The pseudo-code looks like:

Algorithm 2: SVM Stochastic Gradient Descent

```
w* ← 0 ;
b* ← 0 ;
for j=1 to NumIterations do
    for i = 1 to N do
        w_grad ← ∇_w E^(i)(w*, b*) ;
        b_grad ← ∂/∂b E^(i)(w*, b*) ;

        w* ← w* − α(j) w_grad ;
        b* ← b* − 0.01 α(j) b_grad ;
    end
end
return w*
```

Use the same $\alpha(.), \eta_0, C$ as part (b). Run your implementation of SVM Stochastic Gradient Descent over the training data 6 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) **[4 points]** What can you conclude about the convergence rate of Stochastic gradient descent (SGD) versus gradient descent? How did you make this conclusion?

4 [18 points] Using Liblinear for SVM Classification

In this problem, we will use SVM to build a SPAM classifier. You have seen this dataset in the previous homework! Rather than building your implementation from scratch, you will use lib linear. [4].

In recent years, spam on electronic newsgroups has been an increasing problem. Here, we'll build a classifier to distinguish between "real" newsgroup messages, and spam messages. For this experiment, we obtained a set of spam emails, and a set of genuine newsgroup messages. Using only the subject line and body of each message, we'll learn to distinguish between the spam and non-spam.

The problem files live inside the folder `q4` in the data for this assignment. You can look at two sample spam emails in the files `spam_sample_original*`, and their preprocessed forms in the files `spam_sample_preprocessed*`. The first line in the preprocessed format is just the label and is not part of the message. The preprocessing ensures that only the message 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" rather than "terms," since some of the features will correspond to special values like EMAILADDR. You don't have to worry about the distinction.) The files `news_sample_original` and `news_sample_preprocessed` also give an example of a non-spam mail.

The work to extract feature vectors out of the documents has also been done for you, so you can just load in the design matrices (called document-term matrices in text classification) containing 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 j^{th} token in the i^{th} document.

For this problem, we've chosen as our set of tokens considered (that is, as our vocabulary) only the medium frequency tokens. The intuition is that 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 so many emails and are sufficiently content-free that they aren't worth modeling.) 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 term. For a list of the tokens used, see the file `TOKENS_LIST`.

Since the document-term matrix is extremely sparse (has lots of zero entries), we have stored it in our own efficient format to save space. Similar to the previous homework, the function `full()` is used to convert the sparse matrix into a normal matlab matrix. This is already done for you! Modify the indicated sections of `svm_train.m` and `svm_test.m`

(a) **[15 points]** Train an SVM on this dataset using the LIBLINEAR SVM library, available for download from
`http://www.csie.ntu.edu.tw/~cjlin/liblinear/`. This implements an SVM using a linear kernel. Like the Naive Bayes implementation, an outline for your code is provided in `svm_train.m` and `svm_test.m`.

See `README.txt` for instructions for downloading and installing LIBLINEAR. Train an SVM with training set sizes 50, 100, 200, . . . , 1400, by using the file `MATRIX.TRAIN`. E.g. to use a training set of 50, use the first 50 points from MATRIX.TRAIN. Plot the test error each time, using `MATRIX.TEST` as the test data. Use the LIBLINEAR default options when training and testing. You don't need to try different parameter values.

HINT: Running LIBLINEAR in Matlab on Windows can be buggy, depending on which version of Windows you run. However, there are command line programs (`train` and `predict`) that you can run (without using MATLAB) which are located in `liblinear-1.94/windows` for Windows and `liblinear-1.94/`

[4] Credit: Question adopted from `http://www.stanford.edu/class/cs229/ps/ps2.pdf`

for Linux/Unix. If you do it this way, please include the commands that you run from the command line in your solution. The manual for usage of the command line programs is available from: `http://www.csie.ntu.edu.tw/ cjlin/papers/liblinear.pdf`

Note that you must still convert the data into SVM form (using -1,1 instead of 0,1).

(b) **[3 points]** How do naive Bayes (from previous assignment) and Support Vector Machines compare (in terms of generalization error) as a function of the training set size?

5 [Extra Credit 8 points, Optional] Naive Bayes Classifier Using NLTK

In this problem you will be using the Naive Bayes classifier from the NLTK package. You can find detailed information about NLTK at `http://www.nltk.org/`

1 Installing NLTK

a You can find instructions of how to install NLTK in different OS here at: `http://www.nltk.org/install.html`

b Notice if you are using Mac OS you may run into a situation where you numpy and pandas packages conflict. In this case you will need to uninstall your pandas by typing (in a terminal) : `pip uninstall pandas` and then reinstall it using a different version by typing (also in a terminal): `pip install pandas==0.13.1`

c The commands in step 2 may need administrator's authorization to execute, you may need to use the `s`udo command.

2 Run the code

a Switch directory to the folder of `NLTKNaiveBayes` which is included in the data package on CTools.

b A template is named `NLTKNB_template.py`. There are three lines of code that need to be added in near the end of the file. Your first task will be to fill in the missing code. After that please rename the template file following the naming convention we use for homework programs.

c After adding in the code, you can either run the file in any IDE or type in command line `python` to start python first, then type `execfile(`⟨`filename`⟩`)` to run your code, where ⟨`filename`⟩ is the name of the file with your code added in.

d After your code executes successfully, you will see the results of accuracies of testing spam emails and ham(non-spam) emails and a list of the 20 (can be changed to any other number less than the total number of features, 1448) most informative words.

3 What this code does
This code takes use of the NLTK functions and assembles a Naive Bayes classifier. It uses the same data set we used for problem 3 in homework 2 and problem 4 in homework 3. However, rather than binary matrices, the data we used in this problem is a collection of text files which contains only the feature words from the `token list`. The text files are generated according to the feature matrix you downloaded from CTools, `MATRIX.TRAIN` and `MATRIX.TEST`.

4 What to turn in

a The three lines of code you added in and a screenshot of your program's OUTPUT. (3 points)

b Comment on the performance of the code using NLTK. (You may compare it to your own code from problem 3 in homework2). (2 points)

c Explain the term "most informative words" based on your output. (2 points)

Helpful commands:

1. command line commands: `pip install ⟨package_name⟩`, `sudo pip install ⟨package_name⟩`, `pip uninstall ⟨package_name⟩`, `pip install ⟨package_name⟩==⟨version_number⟩`
2. python commands: `import`, `execfile`