Winter 2017: EECS 545 Homework 2

Due: 12 March 2017, 11:59 PM EST

Homework Policy: Working in groups is fine, but each member must submit their own writeup. Please write the members of your group on your solutions. There is no strict limit to the size of the group but we may find it a bit suspicious if there are more than 4 to a team. Questions labelled with **Challenge** are not strictly required, but you'll get some participation credit if you have something interesting to add, even if it's only a partial answer. For coding problems, please report your results (values, plots, etc.) in your written solution, and append the code in the end.

Late submission will NOT be accepted.

1 Positive (Semi-)Definite Matrix

(10 points) Assume $A \in \mathbb{S}^n$ ($\mathbb{S}^n$ denotes the set of symmetric $n \times n$ matrices) has the eigenvalue decomposition $A = Q\ \Lambda\ Q^\top$. If $\{\lambda_i\}_{i=1}^n$ are the eigenvalues of A, then prove that

(1) (3 points) A is PSD $\iff \lambda_i \geq 0,\ \forall\ i$

(2) (7 points) A is PD $\iff \lambda_i > 0,\ \forall\ i$

[**Hint**: Write the eigenvalue decomposition of A as a summation.]

2 Logistic Regression

(44 points) In the previous homework, we used closed-form solution to solve linear regression. Unfortunately, almost no real-life machine learning problems have closed-form solutions, e.g. logistic regression. In this problem, we explore more general methods: gradient descent, stochastic gradient descent, and Newton's method.

In logistic regression, we are given training data set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^m, y_i \in \{0, 1\}$. We want to find parameter $\mathbf{w} \in \mathbb{R}^m$ that minimizes the negative log-likelihood function (a.k.a cross entropy loss):

$$\ell(\mathbf{w}) = \sum_{i=1}^n \ell_i(\mathbf{w}) \tag{1}$$

$$= \sum_{i=1}^n -y_i \log h(\mathbf{x}_i) - (1 - y_i) \log(1 - h(\mathbf{x}_i)) \tag{2}$$

where $h(\mathbf{x}) = \hat{p}(y = 1|\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x})}$ is the predicted probability of $\mathbf{x}$ being from class 1, $\ell_i(\mathbf{w}) = -y_i \log h(\mathbf{x}_i) - (1 - y_i) \log(1 - h(\mathbf{x}_i))$ is loss of the i-th example.

(1) (3 points) Find the expression for the gradient $\nabla \ell(\mathbf{w})$.

(2) (10 points) Use the update rule of gradient descent (GD)

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla \ell(\mathbf{w}) \tag{3}$$

for optimizing $\ell(\mathbf{w})$. Implement GD (not a library function) and apply it to binary classification problem specified in files `q1x.dat` and `q1y.dat`. `q1x.dat` represent the inputs $\mathbf{x}_i$ and `q1y.dat` represents the outputs $t_i \in \{0, 1\}$. Each row is an example. `q1x.dat` has two columns; adding the intercept, each example can be represented using three features $[1, x_1, x_2]^\top$. Initialize $\mathbf{w}^{(0)} = \mathbf{0} =$

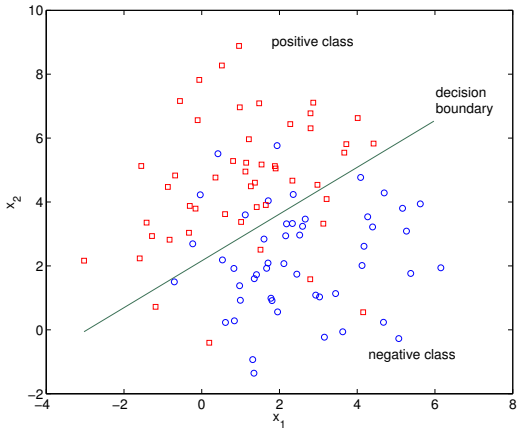

Figure 1: Binary classification decision boundary.

$[0, 0, 0]^\top$ and use learning rate $\eta = 0.001$ in GD. To determine convergence, we can update $\mathbf{w}$ until $\left|\ell(\mathbf{w})^{(t+1)} - \ell(\mathbf{w})^{(t)}\right| < \epsilon$, where ϵ is a very small number. In this problem, we set $\epsilon = 10^{-8}$.

What are the coefficients $\mathbf{w}$, including the intercept term, resulting from your fit? How many steps does your algorithm take until convergence?

(3) (1 points) Find the expression of gradient for the loss of the i-th example, $\nabla \ell_i(\mathbf{w})$.

(4) (10 points) Use the update rule of stochastic gradient descent (SGD)

$$\text{At each step: } i \sim \{1, \cdots, n\},$$
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla \ell_i(\mathbf{w}) \tag{4}$$

for optimizing $\ell(\mathbf{w})$. At each step, the example i is a uniformly random draw *with replacement* from the training set. Implement SGD (not a library function) and apply it to the same problem specified in part (2). Initialize $\mathbf{w}^{(0)} = \mathbf{0}$ and use learning rate

$$\eta = \frac{\eta_0}{(1 + \eta_0 t)^{3/4}}$$

with $\eta_0 = 1$. This learning rate formula[1] is recommended in [1]. As in part (2), set $\epsilon = 10^{-8}$ to determine convergence.

What are the coefficients $\mathbf{w}$, including the intercept term, resulting from your fit? How many steps does your algorithm take until convergence? (You can take average of 10 runs.)

(5) (5 points) Find the expression of Hessian matrix H for $\ell(\mathbf{w})$, and show that it is positive semi-definite and thus $\ell(\mathbf{w})$ is convex and has no local minima other than the global one. That is, show that

$$\mathbf{z}^\top H \mathbf{z} \geq 0$$

for any vector $\mathbf{z}$. [**Hint**: You might start by showing the fact that $\sum_i \sum_j z_i x_i x_j z_j = (\mathbf{x}^\top \mathbf{z})^2 \geq 0$.]

(6) (10 points) With the H you calculated in part (5), use the update rule of Newton's method

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - H^{-1} \nabla \ell(\mathbf{w}) \tag{5}$$

for optimizing $\ell(\mathbf{w})$. Implement Newton's method (not a library function) and apply it to the same problem specified in part (2). Initialize $\mathbf{w}^{(0)} = \mathbf{0}$. As in part (2), set $\epsilon = 10^{-8}$ to determine convergence. Note that we don't have to set learning rate for Newton's method.

What are the coefficients $\mathbf{w}$, including the intercept term, resulting from your fit? How many steps does your algorithm take until convergence?

[1] Advanced SGD learning algorithms are active research frontiers.

(7) (5 points) Plot the training data (your axes should be x_1 and x_2, corresponding to the two coordinates of the input, and you should use a different symbol for each point plotted to indicate whether that example had label 1 or 0). Also plot on the same figure the decision boundary fit by Newton's method. This should be a straight line showing the boundary separating the region where $h(\mathbf{x}) > 0.5$ from where $h(\mathbf{x}) \leq 0.5$, similar to Figure 1.

3 Naive Bayes and SVM for classifying SPAM

(31 points) In this problem, we will use the naive Bayes algorithm and an SVM to build a SPAM classifier[2].

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.

All the files for this problem are in `spam_classification.zip`. The text emails have been preprocessed to get them into a form usable by naive Bayes and SVM. 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 address (HTTPADDR), currency (DOLLAR) and numbers (NUMBER) were also replaced by special tokens to allow them to be considered properly in the classification process. The files `news_sample_original` and `news_sample_preprocessed` also give an example of a non-spam email.

The work to extract feature vectors out of the documents has also been done for you, so you can just read in the design matrices (called document-word matrices in text classification) containing all the data. In a document-word matrix, the i^{th} row represents the i^{th} document/message, and the j^{th} column represents the j^{th} distinct word/token. The (i, j)-entry of this matrix represents the number of occurrences of the j^{th} word in the i^{th} document.

For this problem, we've chosen as our set of words/tokens considered (that is, as our *vocabulary*) only the medium frequency words. The intuition is that words that occur too often or too rarely do not have much classification value. (Example words that occur very often are words like "the", "and", and "of", which occur in so many emails and are sufficiently content-free that they aren't worth modeling.) Also, words 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 word. For a list of the tokens used, see the file `TOKENS_LIST`.

Since the document-word matrix is extremely sparse (has lots of zero entries), we have stored it in a sparse format to save space. These matrices are in `SPARSE.*` files. Each line of the file corresponds to a row in the document-word matrix, with the following format:

```
<label> <feature>:<value> <feature>:<value> ...
```

where `label = 1` means the document is a spam; `label = -1` means the document is not a spam. Each `feature` is the index of a word w_j in `TOKENS_LIST`; each `value` is the count of the word w_j in that document d_i, denoted as $c(d_i, w_j)$. The sparse format skips those entries where $c(d_i, w_j) = 0$. Note that the order of words is lost in this feature representation, hence the name "bag-of-words" representation.

(1) (10 points) Implement naive Bayes classifier for spam classification, using the multinomial event model and Laplace smoothing [2]. In naive Bayes classifier, the predicted probability of d_i being a spam follows the Bayes rule:

$$p(y_i = 1|d_i) = \frac{p(d_i|y_i = 1)p(y_i = 1)}{\sum_{y\in\{-1,1\}} p(d_i|y_i = y)p(y_i = y)} \ . \tag{6}$$

The predicted label is 1 ("spam") if $p(y_i = 1|d_i) > 0.5$, otherwise the predicted label is -1. The class prior probability $p(y_i = y)$ is simply the fraction of documents with label y:

$$p(y_i = y) = \frac{\sum_{i=1}^{n} \mathbf{1}_{\{y_i=y\}}}{n} \tag{7}$$

[2]Credit: question adopted from a previous version of Stanford CS229.

where $\mathbf{1}_{\{z\}}$ is the indicator function, $\mathbf{1}_{\{z\}} = 1$ if z is true; 0 otherwise. In the *multinomial event model*, the class-conditional probability $p(d_i|y_i = y)$ adopts the following assumption: each document is a sequence of words $d_i = [w_1, w_2, \cdots]$ that is drawn from a multinomial distribution of words, with as many independent trials as the length of d_i. That is,

$$p(d_i|y_i = y) = \prod_{j=1}^{m} p(w_j|y_i = y)^{c(d_i,w_j)} \tag{8}$$

where m is the size of our vocabulary. It encodes a naive Bayes assumption: "if the class label y is given, then the probability of seeing one word is independent of seeing another word". Each $p(w_j|y_i = y)$ is estimated as the chance of seeing word w_j if we concatenate all documents of label y:

$$p(w_j|y_i = y) = \frac{1 + \sum_{k \in I_y} c(d_k, w_j)}{\sum_{j=1}^{m} \left(1 + \sum_{k \in I_y} c(d_k, w_j)\right)} , \tag{9}$$

where $I_y = \{i : 1 \le i \le n, y_i = y\}$ is the index set of all documents with label y. In words, $\sum_{k \in I_y} c(d_k, w_j)$ is the total count of word w_j in all documents with label y. "1+" here is to prevent $p(w_j|y_i = y) = 0$ for any w_j, which is called the Laplace smoothing.

Estimate parameters $\{p(y_i = y), p(w_j|y_i = y), \forall y \in \{-1, 1\}\,,\ \forall w_j, j = 1, \cdots, m\}$ using `SPARSE.TRAIN`, and then report the error rate on test set `SPARSE.TEST`.

$$\text{error rate} = \frac{\#\text{ of wrongly classified documents in } \texttt{SPARSE.TEST}}{\#\text{ of documents in } \texttt{SPARSE.TEST}} \times 100\% \ . \tag{10}$$

Remark. If you implement naive Bayes in the straightforward way, you'll notice that the product in (8) often equals zero. This is because the product of many numbers less than one is a very small number. The standard computer representation of real numbers cannot handle numbers that are too small, and instead rounds them off to zero. (This is called "underflow".) You'll have to find a way to compute naive Bayes' predicted class labels without explicitly representing very small numbers such as $p(d_i|y_i = y)$. [**Hint**: Think about taking logarithms.]

(2) (5 points) Intuitively, some tokens may be particularly indicative of an email being in a particular class. We can try to get an informal sense of how indicative token j is for the SPAM class by looking at:

$$\log\left(\frac{p(w_j|y = 1)}{p(w_j|y = -1)}\right) \ . \tag{11}$$

Using the parameters learned in part (1), find the 5 tokens that are most indicative of the SPAM class (i.e. have the highest positive value on the measure above). The numbered list of tokens in the file `TOKENS_LIST` should be useful for identifying the tokens.

(3) (5 points) Repeat part (1), but with training set of size ranging from 50, 100, 200, ..., up to 1400, by using the files `SPARSE.TRAIN.*`. Plot the test error rate each time (using `SPARSE.TEST`) to obtain a learning curve (test error rate vs. training set size). Which training set size gives the lowest test error?

(4) (10 points) Train an SVM on this data set using the LIBLINEAR software package, available for download from `http://www.csie.ntu.edu.tw/~cjlin/liblinear/` or GitHub `https://github.com/cjlin1/liblinear`. This implements an SVM using a linear kernel.

See `README.txt` of the package for instructions of downloading and running LIBLINEAR. Similar to part (3), train an SVM with training set sizes 50, 100, 200, ..., up to 1400, by using the files `SPARSE.TRAIN.*`. The sparse format of these files is compatible with LIBLINEAR, so you don't need further processing. Plot the test error rate each time using `SPARSE.TEST`. 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. Using command-line programs (`train` and `predict` for Linux and Mac, `train.exe` and `predict.exe` for Windows) can be more stable. Please include the programs or commands that you run in your solution. The manual for using the command line programs is available from `http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf`

(5) (1 point) How do naive Bayes and Support Vector Machines compare (in terms of generalization error) as a function of training set size?

4 Maximum Entropy Distributions

(20 points) Let X be a discrete random variable taking values in a finite support set $S = \{x_1, \cdots, x_K\}$. Its probability mass function is specified by K values, $\{p(X = x_k) = p_k, k = 1, \cdots, K\}$. Its entropy is defined as $H(X) = \sum_{k=1}^{K} -p_k \log p_k$.

(1) (10 points) Find the distribution of X (the set of p_k's) that achieves the maximum entropy $H(X)$.

(2) (10 points) **Challenge:** Suppose we know the support $S = \{1, 2, 3\}$ *and* the expectation $\mathbb{E}[X] = 2.5$. What is the distribution of X that achieves the maximum entropy? Show steps to justify your answer.

[**Hint**: formulate it as an constrained optimization problem, and solve it using Lagrangian multipliers. What properties of probability can you use to decide the constraints?]

5 Properties of Kernels

(15 points) Use the properties of kernels to prove the following:

(1) (5 points) Prove that if k is an inner product kernel, then k is a symmetric, positive definite kernel.

(2) (5 points) If $k(\mathbf{x}, \mathbf{x}')$ is a valid kernel, verify that $q(k(\mathbf{x}, \mathbf{x}'))$ is also a valid kernel if $q(\cdot)$ is a polynomial with non-negative coefficients.

(3) (5 points) If k_1 and k_2 are valid inner-product kernels and $a_1, a_2 \geq 0$, then show that $a_1 k_1 + a_2 k_2$ is also a valid inner product kernel.

6 The Kernel Trick

(20 points)

(1) (10 points) **Kernel perceptron** (PRML Exercise 6.2): Read Section 4.1.7 in Bishop PRML (pages 192-196, downloadable in Canvas directory of HW2. Do not distribute.) In this problem, we develop a dual formulation of the perceptron learning algorithm. Using the perceptron learning rule (4.55), show that the learned weight vector $\mathbf{w}$ can be written as a linear combination of the vectors $t_n \boldsymbol{\phi}(\mathbf{x}_n)$ where $t_n \in \{+1, -1\}$. Denote the coefficients of this linear combination by α_n and derive a formulation of the perceptron learning algorithm, and the predictive function for the perceptron, in terms of α_n. Show that the feature vector $\boldsymbol{\phi}(\mathbf{x})$ enters only in the form of the kernel function $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{x}')$.

(2) (10 points) **Kernel nearest neighbor classifier** (PRML Exercise 6.3): The nearest-neighbor classifier assigns a new input vector $\mathbf{x}$ to the same class as that of the nearest input vector $\mathbf{x}_n$ from the training set, where in the simplest case, the distance is defined by the Euclidean metric $\|\mathbf{x} - \mathbf{x}_n\|^2$. By expressing this rule in terms of scalar products and then making use of kernel substitution, formulate the nearest-neighbor classifier for a general nonlinear kernel $k(\mathbf{x}, \mathbf{x}')$.

7 Asymmetric Cost SVM

(20 points) Suppose you have a supervised learning problem where, it turns out, the cost of labeling data as a false positive (i.e., labeling it as '+1' when the true label is '−1') is different from the cost of labeling it as a false negative (i.e., labeling it as '−1' when the true labeling '+1'). You can imagine such a situation arises when we are dealing with medical diagnoses. The asymmetric cost SVM models these situations by posing the following optimization problem:

$$\begin{aligned} \text{P1:} \quad \min_{\mathbf{w}, b, \boldsymbol{\xi}} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 \; + \; C_{-1} \sum_{i: y_i = -1} \xi_i \; + \; C_{+1} \sum_{i: y_i = +1} \xi_i \\ \text{subject to:} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i, \; \forall \; i = 1, 2, \cdots, N \\ & \xi_i \geq 0, \; \forall \; i = 1, 2, \cdots, N \end{aligned}$$

Here, C_{-1} is the cost of a false positive; C_{+1} is the cost of a false negative. Both C_{+1} and C_{-1} are known, fixed constants.

(1) (2 points) We will find the dual optimization problem. First, write down the Lagrangian for this problem. Use a_i and r_i to denote the Lagrange multipliers corresponding to the two constants in the primal optimization problem above (P1).

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{r}) =$$

(2) (3 points) Find the derivatives with respect to the primal variables.

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{r}) =$$
$$\frac{\partial \mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{r})}{\partial b} =$$
$$\nabla_{\boldsymbol{\xi}} \mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{r}) =$$

(3) (10 points) Find the dual optimization problem. Simplify the dual as much as you can. In particular, to obtain full credit, the Lagrange multipliers r_i should not appear in your simplified form. [**Hint**: From the primal problem, can you check whether strong duality holds? Would the KKT conditions apply?]

$$\text{P2:} \quad \max_{\boldsymbol{a}} \; W(\boldsymbol{a}) =$$
$$\text{subject to:} \quad \cdots$$

(4) (5 points) Kernelize the dual problem. [**Hint**: when can you use the kernel trick?]

References

[1] Léon Bottou. Stochastic Gradient Descent Tricks. *Neural networks: Tricks of the trade*, Springer Berlin Heidelberg, 2012. 421–436.

[2] Andrew McCallum and Kamal Nigam. A comparison of event models for naive bayes text classification. *AAAI-98 workshop on learning for text categorization*, Vol. 752. 1998.