CIS 520: Machine Learning *Due:* Feb 5, 2019 (10:29 AM)

Problem Set 1

Submission Instructions:

Unless otherwise announced, all problem sets require submission of two components on Canvas:

- **PDF Submission:** This is your main submission, and must contain your solutions to **all problems that you are attempting, including both mathematical problems and programming problems.** It must be submitted as a **single PDF file**, compiled in LaTeX using the LaTeX template provided on Canvas. For programming problems, in addition to including any plots or observations as requested, be sure to include a snippet of your code in your LaTeX solution; for this, we encourage you to use `mcode` or `listings` packages for LaTeX.
- **Code Submission:** This should contain all your (Matlab) code in a **single zip folder**, and should be submitted under the problem set's code submission component on Canvas. Note that this is **in addition** to writing your code inline in the PDF file above.

Collaboration Policy:

You are allowed to discuss problems in groups, but you must write all your solutions and code **individually, on your own**. All collaborators with whom problems were discussed **must** be listed in your PDF submission.

1. (10 points) **Cost-Sensitive Binary Classification.** Consider a cost-sensitive binary classification task in which misclassifying a negative instance as positive (false positive prediction) incurs a cost of $c \in (0,1)$, and misclassifying a positive instance as negative (false negative prediction) incurs a cost of $1-c$. This can be formulated as a learning task with instance space $\mathcal{X}$, label and prediction spaces $\mathcal{Y} = \widehat{\mathcal{Y}} = \{\pm 1\}$, and cost-sensitive loss $\ell_c : \{\pm 1\} \times \{\pm 1\} \to \mathbb{R}_+$ defined as

$$\ell_c(y, \widehat{y}) = \begin{cases} c & \text{if } y = -1 \text{ and } \widehat{y} = +1 \\ 1-c & \text{if } y = +1 \text{ and } \widehat{y} = -1 \\ 0 & \text{if } \widehat{y} = y. \end{cases}$$

Equivalently, the loss function can be written in the form of a loss matrix as follows:

$$\begin{array}{cc|cc} & & \multicolumn{2}{c}{\widehat{y}} \\ & & -1 & +1 \\ \hline y & -1 & 0 & c \\ & +1 & 1-c & 0 \end{array}$$

Let D be a joint probability distribution on $\mathcal{X} \times \{\pm 1\}$, with conditional label probabilities given by $\eta(x) = \mathbf{P}(Y = +1 | X = x)$, and for any classifier $h : \mathcal{X} \to \{\pm 1\}$, define the ℓ_c-generalization error of h as

$$\text{er}_D^c[h] = \mathbf{E}_{(X,Y)\sim D}\left[\ell_c(Y, h(X))\right].$$

Derive a Bayes optimal classifier in this setting, i.e. a classifier $h^* : \mathcal{X} \to \{\pm 1\}$ with

$$\text{er}_D^c[h^*] = \inf_{h:\mathcal{X}\to\{\pm 1\}} \text{er}_D^c[h].$$

Explain your reasoning. How does your derivation change if the loss incurred on misclassifying a positive instance as negative is a and that for misclassifying a negative instance as positive is b for some arbitrary $a, b > 0$?

2. (10 points) **Binary Classification with a Reject Option.** Consider a binary classification task with a 'reject' option, in which a classifier can choose to reject instances about which it is unsure, with a corresponding cost of $c \in \left(0, \frac{1}{2}\right)$; misclassifications incur a cost of 1 as usual. This can be formulated as a learning task with instance space $\mathcal{X}$, label space $\mathcal{Y} = \{\pm 1\}$, prediction space $\widehat{\mathcal{Y}} = \{-1, +1, \mathsf{reject}\}$, and loss $\ell_{\mathsf{reject}(c)} : \{\pm 1\} \times \{-1, +1, \mathsf{reject}\} \to \mathbb{R}_+$ defined as

$$\ell_{\mathsf{reject}(c)}(y, \widehat{y}) = \begin{cases} c & \text{if } \widehat{y} = \mathsf{reject} \\ 1 & \text{if } \widehat{y} \in \{\pm 1\} \text{ and } \widehat{y} \neq y \\ 0 & \text{if } \widehat{y} = y. \end{cases}$$

Equivalently, the loss function can be written in the form of a loss matrix as follows:

$$\begin{array}{cc|ccc} & & \multicolumn{3}{c}{\widehat{y}} \\ & & -1 & +1 & \mathsf{reject} \\ \hline y & -1 & 0 & 1 & c \\ & +1 & 1 & 0 & c \end{array}$$

Let D be a joint probability distribution on $\mathcal{X} \times \{\pm 1\}$, with conditional label probabilities given by $\eta(x) = \mathbf{P}(Y = +1 | X = x)$, and for any classifier $h : \mathcal{X} \to \{-1, +1, \mathsf{reject}\}$, define the $\ell_{\mathsf{reject}(c)}$-generalization error of h as

$$\text{er}_D^{\mathsf{reject}(c)}[h] = \mathbf{E}_{(X,Y)\sim D}\left[\ell_{\mathsf{reject}(c)}(Y, h(X))\right].$$

Derive a Bayes optimal classifier in this setting, i.e., a classifier $h^* : \mathcal{X} \to \{-1, +1, \mathsf{reject}\}$ with

$$\mathrm{er}_D^{\mathrm{reject}(c)}[h^*] = \inf_{h:\mathcal{X}\to\{-1,+1,\mathsf{reject}\}} \mathrm{er}_D^{\mathrm{reject}(c)}[h]\,.$$

Explain your reasoning. What happens if $c \geq \frac{1}{2}$?

3. (15 points) **Gaussian Naïve Bayes.** Consider a binary classification task with instance space $\mathcal{X} = \mathbb{R}^d$ and label and prediction spaces $\mathcal{Y} = \widehat{\mathcal{Y}} = \{\pm 1\}$. Assume that given the label $y \in \{\pm 1\}$, the features are conditionally independent, and moreover, are normally distributed, so that the class-conditional densities $p(\mathbf{x}|y)$ can be written as

$$p(\mathbf{x}|y) = \prod_{j=1}^{d} p(x_j|y),$$

with

$$p(x_j|y) = \frac{1}{\sqrt{2\pi}\sigma_{y,j}} \exp\Big(- \frac{(x_j - \mu_{y,j})^2}{2\sigma_{y,j}^2} \Big)\,,$$

where $\mu_{y,j}$, $\sigma_{y,j}^2$ are the mean and variance parameters of the (univariate) normal distributions. Let $p_{+1} = \mathbf{P}(Y = +1)$.

   (a) Find an expression for the conditional class probability function $\eta(\mathbf{x})$ in this setting. Use this to construct a Bayes optimal classifier (with respect to the usual 0-1 loss).

   (b) How would you proceed if the parameters $\mu_{y,j}$, $\sigma_{y,j}$, p_{+1} are unknown, but you are given a training sample $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)) \in (\mathcal{X} \times \{\pm 1\})^m$ where the examples $(\mathbf{x}_i, y_i)$ are drawn i.i.d. from the above model?

4. (25 points) **Programming Exercise: Logistic Regression.** Write a piece of MATLAB code to implement logistic regression on a given binary classification data set $(\mathbf{X}, \mathbf{y})$, where $\mathbf{X}$ is an $m \times d$ matrix (m instances, each of dimension d) and $\mathbf{y}$ is an m-dimensional vector (with $y_i \in \{\pm 1\}$ being a binary label associated with the i-th instance in $\mathbf{X}$): your program should take the training data $(\mathbf{X}, \mathbf{y})$ as input and output a d-dimensional weight vector $\widehat{\mathbf{w}}$ and bias (threshold) term $\widehat{b}$ representing the learned classifier. For this problem, you are provided a spam classification data set[1], where each instance is an email message represented by 57 features and is associated with a binary label indicating whether the email is spam ($+1$) or non-spam (-1). The data set is divided into training and test sets; there are 250 training examples and 4351 test examples. The goal is to learn from the training set a classifier that can classify new email messages in the test set.

   (a) **Unregularized logistic regression.** Use your implementation of logistic regression to learn a classifier from 10% of the training data, then 20% of the training data, then 30% and so on up to 100% (separate files containing r% of the training examples are provided under the folder for this problem with file names `Spambase/Train-subsets/X_train_r%.txt`, and the corresponding labels are provided with file names `y_train_r%.txt` in the same folder). In each case, measure the classification error on the training examples used, as well as the error on the given test set. Plot a curve showing both the training error[2] and the test error (on the y-axis) as a function of the number of training examples used (on the x-axis). Such a plot is often called a **learning curve**. At the end, report the training and test error obtained when training on the full 100% data set.

   (b) **L_2-regularized logistic regression.** Incorporate an L_2-regularizer in your implementation of logistic regression (taking the regularization parameter λ as an additional input). Run the regularized version of logistic regression on the entire training set for different values of λ in the

[1]UCI Machine Learning Repository: `http://archive.ics.uci.edu/ml/datasets/Spambase`

[2]Note: the training error should be calculated only on the subset of examples used for training, not on all the training examples available in the given data set.

range $\{10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$. Plot the training and test error achieved by each value of λ (λ on the x-axis and the classification error on the y-axis). Report the average cross-validation error (across the five folds that are provided in `Spambase/Cross-validation`)[3] for each value of λ, the value of λ selected by this cross-validation procedure and the corresponding training and test error. How does the test error achieved with regularization compare with the unregularized version?

Hints:

(a) *You can use the provided code* `classification_error.m` *to compute the errors*.

(b) *The* `glmfit` *function in MATLAB, which implements (unregularized) logistic regression, can be used as a sanity check in this problem; note that the input labels to this function must be in* $\{0,1\}$ *and not in* $\{\pm 1\}$*. For implementing your own version of logistic regression, the* `fminunc` *MATLAB function for unconstrained optimization will be useful. Do not forget to include the bias (threshold) term in the logistic regression model.*

5. (15 points) **Regression on the Unit Square.** Consider a regression task in which instances contain two features, each taking values in $[0,1]$, so that the instance space is $\mathcal{X} = [0,1]^2$, and with label and prediction spaces $\mathcal{Y} = \widehat{\mathcal{Y}} = \mathbb{R}$. Suppose examples $(\mathbf{x}, y)$ are drawn from a joint probability distribution D on $\mathcal{X} \times \mathbb{R}$ whose marginal density on $\mathcal{X}$ is given by

$$\mu(\mathbf{x}) = 2x_1 \quad \forall \mathbf{x} = (x_1, x_2) \in \mathcal{X}$$

and conditional distribution of Y given $\mathbf{x}$ is given by

$$Y|X = \mathbf{x} \ \sim \ \mathcal{N}(x_1 - x_2 + 1, 1)\,.$$

(a) Find an optimal regression model for D (under squared loss). What is the minimum achievable squared error for D, $\mathrm{er}_D^{\mathrm{sq},*}$?

(b) Suppose you give your friend a training sample $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$ containing m examples drawn i.i.d. from D, and your friend learns a regression model given by

$$f_S(\mathbf{x}) = x_1 - x_2 \quad \forall \mathbf{x} = (x_1, x_2) \in \mathcal{X}\,.$$

Find the squared error of f_S w.r.t. D, $\mathrm{er}_D^{\mathrm{sq}}[f_S]$.

6. (25 points) **Programming Exercise: Least Squares Regression.** Write a small piece of MATLAB code to implement (linear) least squares regression (with and without regularization). The input to your code is a training data set $(\mathbf{X}, \mathbf{y})$, where $\mathbf{X}$ is an $m \times d$ matrix of training examples and $\mathbf{y}$ is an m-dimensional vector of real-valued labels associated with the instances in $\mathbf{X}$. You are provided with two data sets for this problem: a synthetic 1-dimensional data set (data set 1), and a real 8-dimensional data set (data set 2); each data set is split into training and test sets.

(a) **Data set 1 (synthetic 1-dimensional data).** This data set contains 100 training examples and 1000 test examples, all generated i.i.d. from a fixed probability distribution. For this data set, you will run (unregularized) least squares regression.

i. **Learning curve.** Use your implementation of (unregularized) least squares regression to learn a regression model from 10% of the training data, then 20% of the training data, then 30% and so on up to 100% (separate files containing r% of the training examples are provided under the folder for this problem with file names `Data-set-1/Train-subsets/X_train_r%.txt`,

[3]Specifically, for each value of λ, train 5 models, each time leaving out one of the 5 folds and training on the remaining 4, and measuring the error on the left-out fold; average this error over the 5 models to obtain the cross-validation error for that value of λ, and select the value of λ with smallest cross-validation error.

and the corresponding labels are provided with file names `y_train_r%.txt` in the same folder). In each case, measure the mean squared error on the training examples used, as well as the error on the given test set. Plot a curve showing both the training error[4] and the test error (on the y-axis) as a function of the number of training examples used (on the x-axis).

ii. **Analysis of model learned from full training data.** Write down the weight and bias terms, $\widehat{w}$ and $\widehat{b}$, in the linear model learned from the full training data. Also write down the training and test error of this model. In a single figure, draw a plot of the learned linear function (input instance on the x-axis and the predicted value on the y-axis), along with a scatter plot depicting the true label associated with each test instance.

(b) **Data set 2 (real 8-dimensional data).** This is a real data set that involves predicting concrete compressive strength from 8 properties of concrete.[5] The data set contains 721 training examples and 309 test examples. For this data set, you will implement both unregularized least squares regression and L_2-regularized least squares regression (ridge regression).

i. **Unregularized least squares regression.** Use your implementation of unregularized least squares regression to learn two models: one from 10% of the training data and the second from the full training data (separate files containing r% of the training examples are provided under the folder for this problem with file names `Data-set-2/Train-subsets/X_train_r%.txt`, and the corresponding labels are provided with file names `y_train_r%.txt` in the same folder). In each case (10% and 100%), write down the learned weight vector and bias terms, as well as the training and test errors.

ii. **Ridge regression.** Use your implementation of L_2-regularized least squares regression to learn two models: one from 10% of the training data, and the second from the full training data. In each case (10% and 100%), select the regularization parameter λ from the range $\{0.1, 1, 10, 100, 500, 1000\}$ using 5-fold cross-validation on the relevant training data (you must use the folds provided in `Data-set-2/Cross-validation`).[6] In each case, draw a plot showing λ on the x-axis and the training, test, and cross-validation errors on the y-axis (3 curves in each plot), and write down the chosen value of λ , together with the weight vector and bias terms learned using this value of λ, as well as the corresponding training and test errors.[7]

iii. **Comparison of models learned by the two methods.** For each of the two training sets considered above (10% and 100%), compare the training and test errors of the models learned using unregularized least squares regression and ridge regression. What can you conclude from this about the value of regularization for small and large training sets?

Hints:

(a) *You can use the provided code* `mean_squared_error.m` *to compute the errors.*

(b) *Do not forget to include a bias (threshold) term b in the model. This can be done as follows: add an extra column of 1's to (the right of) the training data matrix $\mathbf{X}$ and create an augmented matrix $\widetilde{\mathbf{X}}$ of dimension $m \times (d+1)$; then the solution to the unregularized least squares regression (with the bias term included) is given by $[\widehat{\mathbf{w}}; \widehat{b}] = (\widetilde{\mathbf{X}}^\top \widetilde{\mathbf{X}})^{-1} \widetilde{\mathbf{X}}^\top \mathbf{y}$. In the case of the L_2-regularized least squares regression (with the bias term included), the solution takes the form $[\widehat{\mathbf{w}}; \widehat{b}] = (\widetilde{\mathbf{X}}^\top \widetilde{\mathbf{X}} + \lambda m \mathbf{R})^{-1} \widetilde{\mathbf{X}}^\top \mathbf{y}$, where $\mathbf{R}$ is a $(d+1) \times (d+1)$ matrix containing the $d \times d$ identity matrix $\mathbf{I}_d$ at the top left, augmented with an extra column of 0's to the right and an extra row of 0's at the bottom. Use built-in MATLAB functions for matrix operations such as computing inverses etc.*

[4]Note: the training error should be calculated only on the subset of examples used for training, not on all the training examples available in the given data set.

[5]UCI Machine Learning Repository: `https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength`

[6]Specifically, for each value of λ, train 5 models, each time leaving out one of the 5 folds and training on the remaining 4, and measuring the error on the left-out fold; average this error over the 5 models to obtain the cross-validation error for that value of λ, and select the value of λ with smallest cross-validation error.

[7]Cross-validation errors should be computed as described above; for training and test errors, also train a model on the full relevant (10% or 100%) training data.