**CIS 520: Machine Learning** *Due:* Feb 20, 2018 (11:59 AM)

# Problem Set 2

**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. (25 points) **Programming Exercise: Support Vector Machines.** In this problem, you will implement the SVM algorithm for binary classification and will apply it to both synthetic and real data. Write a piece of MATLAB code to learn an SVM classifier: your program should take as input a training 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 $\mathbf{X}$), parameter $C$, kernel type (which can be 'linear', 'polynomial' or 'rbf'), and kernel parameter (which you can take here to be a single real number $r$; this is used as the degree parameter $q$ in the polynomial kernel $K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}' + 1)^q$, and as the parameter $\gamma$ in the RBF kernel $K(\mathbf{x}, \mathbf{x}') = e^{-\gamma ||\mathbf{x} - \mathbf{x}'||_2^2}$); the program should return an SVM model as an output (consisting of the kernel type and parameters, the support vectors, the coefficients corresponding to the support vectors and the bias term). Use MATLAB's quadratic program solver `quadprog` to solve the SVM dual problem. You are provided with two data sets for this problem: a synthetic 2-dimensional data set (data set 1), and a real data set (data set 2); each data set is split into training and test sets.

   (a) **Data set 1 (synthetic 2-dimensional data).** This data set contains 200 training examples and 1800 test examples, all generated i.i.d. from a fixed probability distribution. (See folder `Synthetic-Dataset` under the folder for this problem for relevant files; labels $y$ are in the last column.)

      i. **Linear SVM.** Use your implementation of SVM to learn a linear classification model from the given training data, selecting $C$ from the range $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2\}$ via 5-fold cross-validation on the training set (the train and test data for each fold are provided in a separate folder in `Synthetic-Dataset/CrossValidation`). Show a plot of the training, test, and cross-validation errors as a function of $C$. Report the selected value of $C$, and the corresponding training and test error. Also show a plot of the resulting decision boundary (you can use the provided file `decision_boundary.m` for this).

      ii. **Kernel SVM: polynomial kernel.** Use your implementation of SVM to learn a polynomial kernel classifier from the given training data. Consider degree-$q$ polynomial kernels for $q \in \{1, 2, 3, 4, 5\}$. For each $q$, select $C$ from the range $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2\}$ via 5-fold cross-validation on the training set; further, also select $q$ using cross-validation. (In other words, for each $q$, select the best $C$ in terms of cross-validation error as above; then further select the best $q$ via cross-validation, i.e. select a value of $q$ for which the best value of $C$ gives the lowest cross-validation error). Use the same cross-validation folds as above. Show a plot of the training, test, and cross-validation errors as a function of $q$ (for each value of $q$, use the best value of $C$ based on cross-validation). Report the selected value of $q$ (and $C$), and the corresponding training and test error. Also show a plot of the resulting decision boundary (again, you can use the provided file `decision_boundary.m` for this).

      iii. **Kernel SVM: RBF kernel.** Use your implementation of SVM to learn an RBF kernel classifier from the given training data. Consider RBF kernels with parameter $\gamma$ in the range $\{10^{-2}, 10^{-1}, 1, 10, 10^2\}$. For each $\gamma$, select $C$ from the range $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2\}$ via 5-fold cross-validation on the training set; further, also select $\gamma$ using cross-validation. (In other words, for each $\gamma$, select the best $C$ in terms of cross-validation error as above; then further select the best $\gamma$ via cross-validation, i.e. select a value of $\gamma$ for which the best value of $C$ gives the lowest cross-validation error). Use the same cross-validation folds as above. Show a plot of the training, test, and cross-validation errors as a function of $\gamma$ (for each value of $\gamma$, use the best value of $C$ based on cross-validation). Report the selected value of $\gamma$ (and $C$), and the corresponding training and test error. Also show a plot of the resulting decision boundary (again, you can use the provided file `decision_boundary.m` for this).

   (b) **Data set 2 (real data).** This is a spam classification data set, 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).[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. (See folder `Spam-Dataset` under the folder for this problem for relevant files.) Repeat all steps (i)-(iii) in part (a) for this data set (except for plotting the decision boundaries, which is hard in 57 dimensions!); use the same parameter ranges as in part (a).

2. (15 points) **Programming Exercise: Nearest Neighbor.** In this problem, you will implement the nearest neighbor algorithm for binary classification and will apply it to both synthetic and real data. Write a piece of MATLAB code to implement the $k$-NN algorithm: your program should take as input a training 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 $\mathbf{X}$), and parameter $k$ indicating the number of neighbors to use; given a new instance $\mathbf{x}$, it should predict a binary label for $\mathbf{x}$. You are provided with two data sets for this problem: a synthetic 2-dimensional data set (data set 1), and a real data set (data set 2); each data set is split into training and test sets.

   (a) **Data set 1 (synthetic 2-dimensional data)**. This data set contains 200 training examples and 1800 test examples, all generated i.i.d. from a fixed probability distribution. (See folder `Synthetic-Dataset` under the folder for this problem for relevant files; labels $y$ are in the last column.)

      i. **1-NN.** Implement 1-NN. Report the training and test error. Show a plot of the decision boundary (you can use the provided file `decision_boundary.m` for this).

      ii. **$k$-NN.** Implement $k$-NN. Consider $k$ in the range $\{1, 5, 9, 49, 99\}$, and perform 5-fold cross validation to select $k$ from this range (the train and test data for each fold are provided in a separate folder in `Synthetic-Dataset/CrossValidation`). Show a plot of the training, test, and cross-validation errors as a function of $k$. Report the selected value of $k$, and the corresponding training and test error. Also show a plot of the resulting decision boundary (again, you can use the provided file `decision_boundary.m` for this).

   (b) **Data set 2 (real data)**. This is a spam classification data set, 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).[2] 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. (See folder `Spam-Dataset` under the folder for this problem for relevant files.) Repeat step (ii) in part (a) for this data set (except for plotting the decision boundaries, which is hard in 57 dimensions!); use the same parameter ranges as in part (a).

3. (20 points) **Programming Exercise: Neural Networks for Classification.** In this problem, you will implement a neural network learning algorithm for binary classification (with one hidden layer) and will apply it to a real data set. Write a piece of MATLAB code to implement a one-hidden-layer neural network for binary classification using batch gradient descent: your program should take as input a training 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 $\mathbf{X}$), a step size (learning rate) parameter $\eta$, and the desired number of iterations; it should output a neural network model that can be used to make predictions on new instances. For this problem, you may find it helpful to convert the labels $y_i \in \{\pm 1\}$ to labels $\widetilde{y}_i \in \{0, 1\}$ via $\widetilde{y}_i = (y_i + 1)/2$.

   Recall that a neural network model for binary classification produces class probability estimates $\widehat{\eta}(\mathbf{x})$; a one-hidden-layer model with $d_1$ hidden units can be written as

$$\widehat{\eta}(\mathbf{x}) = g_\sigma(f_{\mathbf{W},\mathbf{b}}(\mathbf{x}))$$

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

with

$$f_{\mathbf{W},\mathbf{b}}(\mathbf{x}) = \mathbf{W}^{(2)} g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + b^{(2)}\,,$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{d_1 \times d}$ is a matrix of weights connecting the input layer to the hidden layer; $\mathbf{W}^{(2)} \in \mathbb{R}^{1 \times d_1}$ is a row vector of weights connecting the hidden layer to the output layer; $\mathbf{b}^{(1)} \in \mathbb{R}^{d_1}$ is a vector of bias terms associated with the hidden units; $b^{(2)}$ is a bias term associated with the output unit; $g$ is the activation function used by the hidden units; and $g_\sigma$ is the logistic sigmoid function. The optimization objective is given by

$$J(\mathbf{W},\mathbf{b}) = \frac{1}{m}\sum_{i=1}^{m}\big(-\widetilde{y}_i\,\ln(\widehat{\eta}(\mathbf{x}_i)) - (1-\widetilde{y}_i)\,\ln(1-\widehat{\eta}(\mathbf{x}_i))\big)\,.$$

Denoting

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$

$$\mathbf{a}^{(1)} = g(\mathbf{z}^{(1)})\,,$$

the derivatives of the objective with respect to the parameters can be written as

$$\begin{aligned}
\frac{\partial J(\mathbf{W},\mathbf{b})}{\partial w_{jk}^{(1)}} &= \frac{1}{m}\sum_{i=1}^{m}\Big(g_\sigma\big(f_{\mathbf{W},\mathbf{b}}\mathbf{x}_i\big) - \widetilde{y}_i\Big)\cdot w_{1j}^{(2)}\cdot g'(z_{ij}^{(1)})\cdot x_{ik}\\
\frac{\partial J(\mathbf{W},\mathbf{b})}{\partial b_{j}^{(1)}} &= \frac{1}{m}\sum_{i=1}^{m}\Big(g_\sigma\big(f_{\mathbf{W},\mathbf{b}}(\mathbf{x}_i)\big) - \widetilde{y}_i\Big)\cdot w_{1j}^{(2)}\cdot g'(z_{ij}^{(1)})\\
\frac{\partial J(\mathbf{W},\mathbf{b})}{\partial w_{1j}^{(2)}} &= \frac{1}{m}\sum_{i=1}^{m}\Big(g_\sigma\big(f_{\mathbf{W},\mathbf{b}}(\mathbf{x}_i)\big) - \widetilde{y}_i\Big)\cdot a_{ij}^{(1)}\\
\frac{\partial J(\mathbf{W},\mathbf{b})}{\partial b^{(2)}} &= \frac{1}{m}\sum_{i=1}^{m}\Big(g_\sigma\big(f_{\mathbf{W},\mathbf{b}}(\mathbf{x}_i)\big) - \widetilde{y}_i\Big)\,.
\end{aligned}$$

You are provided with a spam classification data set, 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).[3] 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. (See folder `Spam-Dataset` under the folder for this problem for relevant files.)

(a) **Sigmoid units.** Use your implementation of the neural network learning algorithm to learn a one-hidden-layer neural network classification model from the given training data. Take all $d_1$ hidden units to have a **logistic sigmoid** activation function. Consider $d_1$ in the range $\{1, 5, 10, 15, 25, 50\}$, and perform 5-fold cross validation to select $d_1$ from this range (the train and test data for each fold are provided in a separate folder in `Spam-Dataset/CrossValidation`). In each case, when running your gradient descent algorithm, use the following settings: use the provided `.mat` files in `Spam-Dataset/setting-files` folder to initialize $\mathbf{W}, \mathbf{b}$ parameters; fix the step size (learning rate) $\eta$ to 0.1; and fix the number of iterations to 20,000. Show a plot of the training, test, and cross-validation errors as a function of the number of hidden units $d_1$. Report the selected value of $d_1$, and the corresponding training and test error.

(b) **Comparison of different algorithms.** Summarize the results you have obtained on the spam classification data set using different algorithms in the table below, and write down any observations you may have.

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

| **Algorithm** | **Training Error** | **Test Error** |
|---|---|---|
| Linear logistic regression (from PS1) | | |
| Linear SVM | | |
| Kernel SVM, polynomial kernel $(\mathbf{x}^\top \mathbf{x}' + 1)^q$ | | |
| Kernel SVM, RBF kernel $e^{-\gamma \|\mathbf{x}-\mathbf{x}'\|_2^2}$ | | |
| Neural network (sigmoid units) | | |
| 1-NN | | |
| $k$-NN | | |

4. (25 points) **Programming Exercise: Neural Networks for Regression.** In this problem, you will implement a neural network learning algorithm for regression (with one hidden layer) and will apply it to both synthetic and real data sets. Write a piece of MATLAB code to implement a one-hidden-layer neural network for regression using batch gradient descent: your program should take as input a training 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 $y_i \in \mathbb{R}$ being a real-valued label associated with the $i$-th instance in $\mathbf{X}$), a step size (learning rate) parameter $\eta$, and the desired number of iterations; it should output a neural network model that can be used to make predictions on new instances.

   Recall that a one-hidden-layer neural network regression model $d_1$ hidden units can be written as

$$f_{\mathbf{W},\mathbf{b}}(\mathbf{x}) = \mathbf{W}^{(2)} g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + b^{(2)} ,$$

   where $\mathbf{W}^{(1)} \in \mathbb{R}^{d_1 \times d}$ is a matrix of weights connecting the input layer to the hidden layer; $\mathbf{W}^{(2)} \in \mathbb{R}^{1 \times d_1}$ is a row vector of weights connecting the hidden layer to the output layer; $\mathbf{b}^{(1)} \in \mathbb{R}^{d_1}$ is a vector of bias terms associated with the hidden units; $b^{(2)}$ is a bias term associated with the output unit; and $g$ is the activation function used by the hidden units. The optimization objective is given by

$$J(\mathbf{W}, \mathbf{b}) = \frac{1}{m} \sum_{i=1}^{m} (f_{\mathbf{W},\mathbf{b}}(\mathbf{x}) - y_i)^2 .$$

   Denoting

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$
$$\mathbf{a}^{(1)} = g(\mathbf{z}^{(1)}) ,$$

   the derivatives of the objective with respect to the parameters can be written as

$$\begin{aligned}
\frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial w_{jk}^{(1)}} &= \frac{2}{m} \sum_{i=1}^{m} \big(f_{\mathbf{W},\mathbf{b}}(\mathbf{x}_i) - y_i\big) \cdot w_{1j}^{(2)} \cdot g'(z_{ij}^{(1)}) \cdot x_{ik} \\
\frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial b_{j}^{(1)}} &= \frac{2}{m} \sum_{i=1}^{m} \big(f_{\mathbf{W},\mathbf{b}}(\mathbf{x}_i) - y_i\big) \cdot w_{1j}^{(2)} \cdot g'(z_{ij}^{(1)}) \\
\frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial w_{1j}^{(2)}} &= \frac{2}{m} \sum_{i=1}^{m} \big(f_{\mathbf{W},\mathbf{b}}(\mathbf{x}_i) - y_i\big) \cdot a_{ij}^{(1)} \\
\frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial b^{(2)}} &= \frac{2}{m} \sum_{i=1}^{m} \big(f_{\mathbf{W},\mathbf{b}}(\mathbf{x}_i) - y_i\big) .
\end{aligned}$$

   (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. (See folder `Data-set-1` under the folder for this problem for relevant files; labels $y$ are in the last column.) For this data set, you will traing neural network models with both sigmoid units and ReLU units.

i. **Sigmoid units.** Use your implementation of the neural network learning algorithm to learn a one-hidden-layer neural network regression model with 15 hidden units, from increasing fractions of the training data (use the training subsets provided). Take all hidden units to have a **logistic sigmoid** activation function. When running your gradient descent algorithm, use the following settings: use the provided `.mat` files to initialize $\mathbf{W}, \mathbf{b}$ parameters; fix the step size (learning rate) $\eta$ to 0.1; and fix the number of iterations to 20,000. Plot a learning curve showing the training and test error (in terms of square loss) corresponding to training on 10% of the training data, 20%, and so on up to 100%. Report the training and test error of the model learned from the entire training set. Also show a plot of the (nonlinear) function learned from the entire training set (input instance on the x-axis and the predicted value on the y-axis), together with a scatter plot depicting the true label associated with each test instance.

ii. **ReLU units.** Repeat the steps in part (i) above using the **ReLU** activation function.

iii. **Comparison of different algorithms.** Summarize the results you have obtained on this data set using different algorithms in the table below, and write down any observations you may have, particularly regarding the comparison between linear and nonlinear methods.

| **Algorithm** | **Training Error** | **Test Error** |
|---|---|---|
| Linear least squares regression (from PS1) | | |
| Neural network (sigmoid units) | | |
| Neural network (ReLU units) | | |

(b) **Data set 2 (real data).** This is a real data set that involves predicting concrete compressive strength from 8 properties of concrete.[4] The data set contains 721 training examples and 309 test examples. (See folder `Data-set-2` under the folder for this problem for relevant files; labels $y$ are in the last column.) For this data set, you will train a neural network model with ReLU units.

i. **ReLU units.** Use your implementation of the neural network learning algorithm to learn a one-hidden-layer neural network regression model from the given training data. Take all $d_1$ hidden units to have a **ReLU** activation function. Consider $d_1$ in the range $\{7, 10, 15, 17, 20\}$, and perform 5-fold cross validation to select $d_1$ from this range (use the provided cross-validation folds). In each case, when running your gradient descent algorithm, use the following settings: use the provided `.mat` files initialize $\mathbf{W}, \mathbf{b}$ parameters; fix the step size (learning rate) $\eta$ to $3.5 \times 10^{-6}$; and fix the number of iterations to 20,000. Show a plot of the training, test, and cross-validation errors as a function of the number of hidden units $d_1$. Report the selected value of $d_1$, and the corresponding training and test error.

ii. **Comparison of different algorithms.** Summarize the results you have obtained on this data set using different algorithms in the table below, and write down any observations you may have.

| **Algorithm** | **Training Error** | **Test Error** |
|---|---|---|
| Linear least squares regression (from PS1) | | |
| Linear ridge regression (from PS1) | | |
| Neural network (ReLU units) | | |

*Hint: To speed up the execution time of your code, we recommend using* `repmat` *in MATLAB. You might find this function helpful in vectorizing the code.*

5. (15 points) **Decision Trees.** Consider a binary classification task on $\mathcal{X} = \mathbb{R}^2$ with the following set of 8 training examples:

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

| $i$ | $\mathbf{x}_i$ | $y_i$ |
|---|---|---|
| 1 | $(3, 9)$ | $+1$ |
| 2 | $(7, 11)$ | $-1$ |
| 3 | $(4, 6)$ | $+1$ |
| 4 | $(7, 2)$ | $+1$ |
| 5 | $(9, 9)$ | $-1$ |
| 6 | $(10, 2)$ | $+1$ |
| 7 | $(6, 12)$ | $-1$ |
| 8 | $(6, 1)$ | $+1$ |

You would like to construct a decision tree classifier based on the above training examples and are trying to decide what split to use at the root node. Suppose you are considering the following two splits: (1) $x_1 > 5$ and (2) $x_2 > 8$.

(a) What is the entropy associated with a single leaf node containing all the above examples? What is the Gini index associated with such a leaf node? Show your calculations.

(b) For each of the above two splits, calculate the information gain that would result from using that split at the root node. Show your calculations. Which of these splits would you choose based on the entropy criterion?

(c) For each of the above two splits, calculate the Gini reduction that would result from using that split at the root node. Show your calculations. Which of these splits would you choose based on the Gini index criterion?