Machine Learning Fall 2015 Homework 3

Make sure to explain you reasoning or show your derivations. Except for answers that are especially straightforward, you will lose points for unjustified answers, even if they are correct.

General Instructions

Submit your homework electronically on Canvas. We recommend using LaTeX, especially for the written problems. But you are welcome to use anything as long as it is neat and readable.

Include a README file that describes all other included files. Indicate which files you modified. You are welcome to create additional functions, files, or scripts, and you are also welcome to modify the included interfaces for existing functions if you prefer a different organization.

Since we will work on some of the homework in class, clearly indicate which parts of your submitted homework are work done in class and which are your own work.

Relatedly, cite all outside sources of information and ideas. **List any students you discussed the homework with.**

Written Problems

1. Machine learning methods can be viewed as function estimators. Consider the logical functions AND, OR, and XOR. Using a signed representation for Boolean variables, where input and output variables are in $\{+1, -1\}$, these functions are defined as

$$\mathrm{AND}(x_1, x_2) = \begin{cases} +1 & \text{if } x_1 = +1 \wedge x_2 = +1 \\ -1 & \text{otherwise} \end{cases} \tag{1}$$

$$\mathrm{OR}(x_1, x_2) = \begin{cases} +1 & \text{if } x_1 = +1 \\ +1 & \text{if } x_2 = +1 \\ -1 & \text{otherwise} \end{cases} \tag{2}$$

$$\mathrm{XOR}(x_1, x_2) = \begin{cases} +1 & \text{if } x_1 = +1 \wedge x_2 = -1 \\ +1 & \text{if } x_2 = +1 \wedge x_1 = -1 \\ -1 & \text{otherwise} \end{cases} \tag{3}$$

   (a) (4 points) Which of these three logical functions can be expressed as a linear classifier of the form

$$f(\boldsymbol{x}; \boldsymbol{w}) = \mathrm{sign}(w_1 x_1 + w_2 x_2 + b), \tag{4}$$

   and show weights w_1, w_2, and bias values b that mimic these logical functions. Which of these functions cannot be expressed as a linear classifier?

   (b) (5 points) For any logical functions that cannot be expressed as a linear classifier, show how it can be expressed as a two-layered perceptron of the form

$$f(\boldsymbol{x}; \boldsymbol{w}) = \mathrm{sign}(\boldsymbol{w}^{\mathrm{out}\top} \boldsymbol{h} + b^{\mathrm{out}}) \tag{5}$$

$$\boldsymbol{h} = [h_1, h_2]^\top \tag{6}$$

$$h_1 = \mathrm{sign}(\boldsymbol{w}^{(1)\top} \boldsymbol{x} + b^{(1)}) \tag{7}$$

$$h_2 = \mathrm{sign}(\boldsymbol{w}^{(2)\top} \boldsymbol{x} + b^{(2)}) \tag{8}$$

2. Derive the dual support vector machine (SVM) from first principles. To simplify the work, we will do the derivation without a bias and assuming the data is separable. The classification objective is

$$f(\boldsymbol{x}; \boldsymbol{w}) = \mathrm{sign}(\boldsymbol{w}^\top \boldsymbol{x}) \tag{9}$$

For each data point $\boldsymbol{x}_i$, a constraint that ensures that the point is correctly classified with a margin is

$$y_i \boldsymbol{w}^\top \boldsymbol{x}_i - 1 \geq 0. \tag{10}$$

A vector $\boldsymbol{w}$ that satisfies all these constraints for $i \in \{1, \ldots, n\}$ perfectly classifies all the training data.

(a) (5 points) For a vector $\boldsymbol{w}$ and for two data examples $\boldsymbol{x}_+$ and $\boldsymbol{x}_-$ whose labels are respectively $y_+ = +1$ and $y_- = -1$, what is the smallest possible distance between the points $\sqrt{(\boldsymbol{x}_+ - \boldsymbol{x}_-)^\top (\boldsymbol{x}_+ - \boldsymbol{x}_-)}$ if the two points satisfy the margin constraint, Equation (10)? Hint: the closest possible points will be aligned along the normal vector of the decision boundary, so $\boldsymbol{x}_+ = \boldsymbol{x}_- + \gamma \boldsymbol{w}$ for some γ.

(b) (3 points) The answer you derive in part (a) should reveal why the primal max-margin SVM objective is

$$\begin{aligned} \arg\min_{\boldsymbol{w}} \quad & \frac{1}{2} \boldsymbol{w}^\top \boldsymbol{w} \\ \text{s.t.} \quad & y_i \boldsymbol{w}^\top \boldsymbol{x}_i - 1 \geq 0, \ \forall i \in \{1, \ldots, n\} \end{aligned} \tag{11}$$

Convert this constrained optimization into a $\min \max$ of a Lagrangian function $L(\boldsymbol{w}, \boldsymbol{\alpha})$ by using Lagrange multipliers to penalize violations of the inequality constraints. For consistency, use $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_n]^\top$ as the symbol for your Lagrange multipliers. Write the saddle-point optimization over $\boldsymbol{w}$ and $\boldsymbol{\alpha}$, and indicate which variables are being minimized over and which are being maximized over. Your optimizations should be over $\boldsymbol{w} \in \mathbb{R}^d$ and $\boldsymbol{\alpha} \in \mathbb{R}^n_+$, where $\mathbb{R}_+$ is the set of nonnegative real numbers. Hint: be careful about the sign of the Lagrange multipliers and the constraint functions. Plug in some values to make sure the "adversary" can penalize constraint violations with nonnegative αs.

(c) (5 points) Reversing the original outer minimization and the inner maximization, we obtain the *dual* objective function. We also obtain an inner minimization that has a closed form solution. For a fixed vector of Lagrange multipliers $\boldsymbol{\alpha}$, what is the value of $\boldsymbol{w}$ that minimizes the Lagrangian function?

(d) (6 points) Plug in your solution to $\boldsymbol{w}$ and simplify the new objective function. You should obtain a quadratic program over the $\boldsymbol{\alpha}$ variables where all instances of the data vectors $\boldsymbol{x}$ occur in inner products of the form $\boldsymbol{x}_i^\top \boldsymbol{x}_j$. Hint: an easier way to manage these equations is to substitute your solution for one of the $\boldsymbol{w}$ vectors in the $\boldsymbol{w}^\top \boldsymbol{w}$ term first, then cancel out as many terms as you can before plugging your solution into any remaining $\boldsymbol{w}$ terms. For example, if you derived $\boldsymbol{w} = \text{blah}$, start by simplifying the expression $\boldsymbol{w}^\top(\text{blah})$ and canceling with any similar terms from your Lagrangian.

(e) (2 points) Finally, replace all inner products of the form $\boldsymbol{x}_i^\top \boldsymbol{x}_j$ with a kernel function $K(x_i, x_j)$. Write the kernelized dual SVM optimization.

Programming Assignment

For this programming assignment, we have provided a lot of starter code. Your tasks will be to complete the code in a few specific places, which will require you to read and understand most of the provided code, but will only require you to write a small amount of code yourself.

In `syntheticData.mat`, there are 10 synthetic, two-dimensional data sets. All but the first data set are not linearly classifiable. We provided the main experiment script `runSyntheticExperiments.m` for you to use. For each model, the script uses cross-validation on the training data to choose learning parameters, trains on the full training set using those parameters, and evaluates the accuracy on the test set.

1. (8 points) Complete the backpropagation steps in `mlpObjective.m` for the multi-layered perceptron. Specifically, you will need to compute the gradient for the output-layer weights, then the error and gradients for the middle-layer weights, and finally the error and gradient for the input-layer. Once

you complete this step, the script `checkMLPDerivative.m` should run and report a derivative is *close* to the numerical approximation. (It won't pass the hard-coded 1e-6 threshold, so it will crash. But if the error is around 1e-6 in magnitude, it is correct.) Once the gradient computation is correct, the experiment using multi-layered perceptron on the synthetic data sets should run.

2. (8 points) Complete the kernel SVM `kernelSvmTrain.m` and `kernelSvmPredict.m`. In each function, there is a block of code for you to fill in. In `kernelSvmTrain`, set up the quadratic program to compute the dual SVM objective. We have provided the call to the quadprog function, the processing of the output alpha values, and the storage of the support vectors into the model struct. In `kernelSvmPredict`, compute the kernelized prediction score directly from the Gram matrix, the alpha dual variables, the support vector labels, and the bias.

   Once you complete this step, the linear kernel dual SVM should work.

3. (5 points) Complete the polynomial kernel function `polynomialKernel.m`. To see how a kernel function should work, look at `linearKernel.m` as a template. Calling `polynomialKernel(X, Y, 1)` should produce the exact same output as `linearKernel(X, Y)`. Once you complete this step and the previous step, the polynomial kernel experiment should run.

4. (5 points) Complete the Gaussian radial basis function kernel function `rbfKernel.m`. Make sure to use the trick for computing all pairwise distances between two sets of points that we used in the last homework.

5. (4 points) Once all four methods are working, run the experiment script in publish mode, which will iterate through each data set and plot the learned decision boundaries for each data set and each model type. Write a short report (one paragraph is sufficient, but feel free to write more) about the results. Which methods worked best on these synthetic data sets? Is there a pattern to when certain methods work well or not?

Table 1: Included files and brief descriptions. Unless indicated here in bold, the source files are just placeholders for code you should complete. You should not need to implement or modify bolded files.

File Path	**Description**
syntheticData.mat	Synthetic data
src/checkMLPDerivative.m	Script to check the gradient computation of multi-layer perceptron
src/crossValidate.m	Wrapper to run cross validation
src/kernelSvmPredict.m	Function **you will complete** that predicts labels using kernel SVM
src/kernelSvmTrain.m	Function **you will complete** that trains a kernel SVM
src/linearKernel.m	Function that computes the linear kernel between two sets of data
src/logistic.m	Function that computes the logistic of each dimension of its input
src/mlpFlatObjective.m	Wrapper function to help gradient checking of multi-layered perceptron
src/mlpObjective.m	Function **you will complete** that computes the regularized loss and gradient for a multi-layered perceptron
src/mlpPredict.m	Function that performs forward-propagation to predict labels using a multi-layered perceptron
src/mlpTrain.m	Function that runs vanilla gradient descent to train the weights of a multi-layered perceptron
src/nll.m	Function that computes the negative log likelihood of Bernoulli probabilities given the true labels and its gradient
src/plotData.m	Function that plots 2D, binary-class data
src/plotSurface.m	Function that plots the decision boundary for 2D inputs
src/polynomialKernel.m	Function **you will complete** that computes the polynomial kernel matrix between two sets of data points
src/rbfKernel.m	Function **you will complete** that computes the Gaussian radial-basis function kernel between two sets of data points
src/runSyntheticExperiments.m	Main experiment script that runs all four models on all ten data sets