Machine Learning Fall 2015 Homework 2

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. (Based on Murphy 8.5) Multiclass logistic regression has the form

$$p(y|\boldsymbol{x}; W) := \frac{\exp(\boldsymbol{w}_y^\top \boldsymbol{x})}{\sum_{c=1}^{C} \exp(\boldsymbol{w}_c^\top \boldsymbol{x})} \tag{1}$$

where W is a $d \times C$ weight matrix (for C classes and d dimensional data). This formulation is natural, but it includes some redundancy. For C classes, we can arbitrarily define $\boldsymbol{w}_C = \vec{0}$, without loss of generality for the last class C, since $p(y = C|\boldsymbol{x}; W) = 1 - \sum_{c=1}^{C-1} p(y = c|\boldsymbol{x}; W)$. In this case, the model has the form

$$p(y|\boldsymbol{x}; W) := \frac{\exp(\boldsymbol{w}_y^\top \boldsymbol{x})}{1 + \sum_{c=1}^{C-1} \exp(\boldsymbol{w}_c^\top \boldsymbol{x})}. \tag{2}$$

If we don't "clamp" one of the vectors to some constant value, the parameters will be *unidentifiable*, meaning that multiple parameters can yield the same model. The extra vector creates an extra degree of freedom for each dimension. However, suppose we don't clamp $\boldsymbol{w}_c = 0$, so we are using Equation (1), but we add ℓ_2 regularization by solving

$$\hat{W} = \arg\min_{W} \frac{\lambda}{2} \sum_{c=1}^{C} ||\boldsymbol{w}_c||_2^2 - \sum_{i=1}^{n} \log p(y_i|\boldsymbol{x}_i; W) \tag{3}$$

Prove that this regularization removes the extra degree of freedom by showing that at the optimum, the weights for all classes along any single dimension will always sum to zero,

$$\sum_{c=1}^{C} w_c[j] = 0, \ \forall j \in \{1, \ldots, d\}. \tag{4}$$

2. For binary classification, the perceptron prediction rule is

$$f(x) = \begin{cases} +1 & \text{if } \boldsymbol{w}^\top \boldsymbol{x} > 0 \\ -1 & \text{if } \boldsymbol{w}^\top \boldsymbol{x} \leq 0 \end{cases}, \tag{5}$$

And the perceptron learning update for example $(y, \boldsymbol{x})$, where $y \in \{-1, 1\}$ is

$$\boldsymbol{w} \leftarrow \begin{cases} \boldsymbol{w} & \text{if } f(x) = y \\ \boldsymbol{w} + y\boldsymbol{x} & \text{if } f(x) \neq y \end{cases}. \tag{6}$$

(a) The perceptron update is often argued to be a gradient step. But what objective function is it optimizing? Extrapolate the function J for which the perceptron update is a gradient step $\boldsymbol{w} \leftarrow \boldsymbol{w} - \nabla_{\boldsymbol{w}} J$.

(b) This function should look like a loss function: a penalty on the misclassification of a single data example. We can easily use the same loss function on a batch of data, summing up the total penalty for all examples, and do batch learning with the perceptron loss function. Write a batch-learning objective function using the perceptron loss.

(c) If you are given linearly separable data, and a $\boldsymbol{w}$ that correctly classifies all data, what is the value of your objective from part (b)? Are there other vectors $\boldsymbol{w}$ that achieve the same objective score?

(d) If you have data that is not linearly separable, that is, no vector $\boldsymbol{w}$ can correctly classify every point, what is the optimal $\boldsymbol{w}$ for the objective from part (b)?

(e) Why does the perceptron algorithm still work despite the degeneracies these exercises reveal about perceptron loss?

Programming Assignment

For this assignment, you will run an experiment comparing different versions of linear classifiers for multi-class classification. You are welcome to use as much or as little of the starter code as you prefer. But *make sure to examine all the starter code before you start*, in case there is something we have provided that you don't necessarily need to build yourself.

Models

The three models you will implement are perceptron, logistic regression, and Gaussian naive Bayes (with spherical covariance). The multiclass forms of these models are summarized here.

Multiclass Perceptron The multiclass perceptron uses a weight vector for each class, which can conveniently be represented with a matrix $W = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_C\}$. The prediction function is

$$f_{\text{perc}}(\boldsymbol{x}) := \operatorname*{arg\,max}_{c \in \{1,\ldots,C\}} \boldsymbol{w}_c^\top \boldsymbol{x} = \operatorname*{arg\,max}_{c \in \{1,\ldots,C\}} \left[W^\top \boldsymbol{x}\right]_c . \tag{7}$$

The multiclass perceptron update rule when learning from example $\boldsymbol{x}_t$, ground-truth label y_t is.

$$\boldsymbol{w}_{y_t} \leftarrow \boldsymbol{w}_{y_t} + \boldsymbol{x}_t \tag{8}$$

$$\boldsymbol{w}_{(f_{\text{perc}}(\boldsymbol{x}))} \leftarrow \boldsymbol{w}_{(f_{\text{perc}}(\boldsymbol{x}))} - \boldsymbol{x}_t \tag{9}$$

Multiclass Logistic Regression Multiclass logistic regression also uses a weight vector for each class, and in fact has the same prediction formula as perceptron.

$$f_{\text{lr}}(\boldsymbol{x}) := \operatorname*{arg\,max}_{c \in \{1,\ldots,C\}} \boldsymbol{w}_c^\top \boldsymbol{x} = \operatorname*{arg\,max}_{c \in \{1,\ldots,C\}} \left[W^\top \boldsymbol{x}\right]_c . \tag{10}$$

The key difference is that it is built around a probabilistic interpretation:

$$p_{\text{lr}}(y|\boldsymbol{x}; W) := \frac{\exp(\boldsymbol{w}_y^\top \boldsymbol{x})}{\sum_{c=1}^{C} \exp(\boldsymbol{w}_c^\top \boldsymbol{x})} . \tag{11}$$

For data set $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, the regularized negative log likelihood is

$$L(D) = \frac{\lambda}{2} ||W||_{\text{F}}^2 + \sum_{i=1}^{n} \log \left(\sum_c \exp(\boldsymbol{w}_c^\top \boldsymbol{x}_i) \right) - \sum_{i=1}^{n} \boldsymbol{w}_{y_i}^\top \boldsymbol{x}_i \tag{12}$$

where $||W||_{\mathrm{F}}^2$ is the squared Frobenius norm $\sum_{ij} \boldsymbol{w}_i[j]^2$, and the gradient of the log likelihood is

$$\nabla_{\boldsymbol{w}_c} L = \lambda \boldsymbol{w}_c + \sum_{i=1}^{n} \boldsymbol{x}_i \left(\frac{\exp(\boldsymbol{w}_c^\top \boldsymbol{x}_i)}{\sum_{c'} \exp(\boldsymbol{w}_{c'}^\top \boldsymbol{x}_i)} - I(y_i = c) \right) \tag{13}$$

$$= \lambda \boldsymbol{w}_c + \sum_{i=1}^{n} \boldsymbol{x}_i \left(p_{\mathrm{lr}}(y|\boldsymbol{x}_i; W) - I(y_i = c) \right) \tag{14}$$

Gaussian Naive Bayes Naive Bayes with uniform, spherical-covariance Gaussian feature distributions are also linear classifiers. Though it's possible to derive a similar weight matrix as the previous two methods, it's much more natural to keep a Gaussian naive Bayes (GNB) model its standard form, with a mean for each class. For simplicity, and to provide some control, we will set the covariance parameter σ manually, so it can act as a regularization parameter. Then the prediction function is

$$f_{\mathrm{gnb}}(\boldsymbol{x}) := \underset{c \in \{1,\ldots,C\}}{\arg\max}\; p(c; \boldsymbol{\theta}) \prod_{j=1}^{d} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2} (\boldsymbol{x}[j] - \boldsymbol{\mu}_c[j])^2 \right) \tag{15}$$

$$= \underset{c \in \{1,\ldots,C\}}{\arg\max}\; \log p(c; \boldsymbol{\theta}) - \frac{1}{2\sigma^2} \sum_{j=1}^{d} (\boldsymbol{x}[j] - \boldsymbol{\mu}_c[j])^2 \tag{16}$$

$$= \underset{c \in \{1,\ldots,C\}}{\arg\max}\; \log p(c; \boldsymbol{\theta}) - \frac{1}{2\sigma^2} (\boldsymbol{x} - \boldsymbol{\mu}_c)^\top (\boldsymbol{x} - \boldsymbol{\mu}_c) \tag{17}$$

And training the parameters for the model involve fitting the mean vectors for each class $\boldsymbol{\mu}_c$ to the observed means of each class,

$$\boldsymbol{\mu}_c \leftarrow \frac{1}{n_c} \sum_{i : y_i = c} \boldsymbol{x}_i \;, \tag{18}$$

where n_c is the number of examples from class c, and fitting the multinomial distribution for the classes

$$\theta_c = \frac{n_c + \alpha}{n + \alpha C}. \tag{19}$$

Using this formulation of a GNB model, you will need to consider two regularization parameters: α and σ. Both control how sensitive the model components are to the observed data.

Tasks

1. Construct two types of two-dimensional, synthetic data sets:

   (a) Linearly separable data: data that the perceptron prediction rule would be able to correctly classify into its correct classes. In other words, each data point is labeled by some actual linear weight matrix W.

   (b) Non-separable data: data that no perceptron prediction rule should be able to classify. One way to create this type of data is to create linearly separable data and randomly change some labels, or alternatively add random noise to the features after they have been cleanly labeled.

   For each type, create data sets with 100 two-dimensional training points and 100 test points, with some points in each of four classes (some imbalance is fine, but try to make sure each class has at least roughly 10 points). It can help to first create a predictor function for the linear models.

2. Implement perceptron. Implement online training by iterating through the training set one example at a time. *After each update*, compute the training accuracy and testing accuracy for each iteration and plot the results after seeing 1,000 examples (10 sweeps through the data, or *epochs*). Discuss the behavior of perceptron you see for separable and non-separable data.

3. Implement multiclass logistic regression. This task will likely be the most amount of work for the assignment. We have provided a training function `logRegTrain.m` that you are welcome to use, which uses MATLAB's optimization toolbox function `fminunc`[1] to minimize the negative log likelihood. To use this training function and its gradient-based optimizer, implement the function `logRegNLL`, which takes the weight matrix, the data, and model parameters and outputs the negative log likelihood and the gradient. We have included useful utility for computing the log likelihood in `logsumexp.m`, which computes `log(sum(exp(x)))` in a numerically stable way.

   If you want to write your own calls to the optimizer, an example of how to do this is as follows:

   ```
   [nll, gradient] = logRegNLL(W, trainData, trainLabels, params)
   ```

   Then for a training set `trainData` and `trainLabels`, pass an anonymous function to `fminunc` with

   ```
   objective = @(x) logRegNLL(x, trainData, trainLabels, params);
   options = optimoptions(@fminunc, ...
       'DerivativeCheck','off',...
       'GradObj','on',...
       'Display','final',...
       'Algorithm', 'quasi-newton',...
       'MaxIter', 2000);
   W0 = zeros(numDimensions, numClasses);
   W = fminunc(objective, W0, options);
   ```

   where the options struct contains options for the optimizer. `DerivativeCheck` will numerically estimate the derivative of your objective function and compare it to your computed derivative. This numerical derivative checking can become very slow in high dimensions, but should help you debug for the 2D data. `GradObj` tells the optimizer that your function provides the gradient, `Display` determines how much console feedback the optimizer gives, `Algorithm` chooses between different optimization algorithms (I recommend keeping this set to `quasi-newton`), and `MaxIter` is the iteration at which the optimizer will give up on solving for the optimization. A useful tool is to also add the option pair `'PlotFcns', @optimplotfval`, which will plot the objective value and allow you to interactively and gracefully stop the optimization if it's taking too long.

   Note that these optimizers take initial points (e.g., `W0` in the example code above). One useful strategy is to take advantage of this functionality when sweeping over parameters. E.g., once you've solved for the optimal W for some λ, if you make a small change to λ, you should expect the new solution to be close, so using the previous solution as the initial point will make the optimization a lot faster than starting from the origin. The included `logRegTrain.m` takes in an optional starting point for this reason.

   For various values of λ, train logistic regression and report the training and testing accuracy of each model for each data set type. Try to find values of λ that allow you to discern a trend. Discuss your findings.

4. Implement Gaussian naive Bayes. Choose a reasonable range of values for parameters α and σ. Compare training and testing accuracy for all combinations of α and σ parameters.

5. Compare and plot the best (post hoc) observed performance among the three linear classifiers.[2]

6. Run analogous experiments (steps 2-6) on the cardiotocography data. The data includes measurements from fetal cardiotocograms that medical experts diagnosed into ten possible diagnosis classes.[3]

[1]You can also use Mark Schmidt's `minFunc` `http://www.cs.ubc.ca/~schmidtm/Software/minFunc.html`, which is free and usually much faster. Or you can implement your own basic gradient descent function.

[2]We aren't doing cross-validation here to save time, but the best possible test performance is a reasonable approximation of the best possible performance if we had tuned the regularizers ideally. The best thing to do would be to do cross-validation and testing, but for this homework, the post-hoc comparison is sufficient.

[3]`https://archive.ics.uci.edu/ml/datasets/Cardiotocography`

Compare how the perceptron model behavior changes as it sees more examples, how logistic regression and Gaussian naive Bayes behave as you vary the regularization parameters. You may want to adjust the granularity of your measurements. For example, you may only measure testing and training performance on perceptron weights only once per epoch, or you may use a coarser set of regularization parameter values.

Write a report on your findings. Include plots of the performance as well as example plots of the 2D data and classification labels to help you understand and explain the behavior of the models. Use either plots or tables for comparing performance. Look for similarities between the model behavior as well as differences and hypothesize about why there are similarities and differences.

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
Cardiotocography.mat	Original data loaded manually from Excel spreadsheet (yuck)
processedCardio.mat	Processed data with values centered and scaled to unit variance
src/gnbPredict.m	Predictor function for Gaussian naive Bayes (GNB) model, which takes a GNB model and a data set and outputs a label vector.
src/gnbTrain.m	Training function for GNB model, which takes a training set with labels and parameters and outputs a GNB model.
src/linearPredict.m	Linear classifier prediction function, which takes a matrix of linear weights and a data set and outputs a label vector.
src/logRegNLL.m	Objective function for the logistic regression negative log likelihood. The function takes in a weight matrix, a data matrix, a label vector, and parameters and outputs the negative log likelihood and the gradient with respect to the weights.
src/logRegTrain.m	Training function for logistic regression, which takes a training set with labels and parameters and outputs a model. **This function is complete and will work once you implement** `logRegNLL.m`.
src/logsumexp.m	Utility function that computes `log(sum(exp(x)))` in a numerically stable way that avoids floating point underflow. **This function is complete and should be useful for implementing** `logRegNLL.m`.
src/perceptronUpdate.m	Weight update function that takes a data point, a model containing a weight matrix, and a label and outputs the updated model according to the perceptron online learning rule.
src/plotPredictions.m	Utility that plots 4-class 2D data with color-coded correct and incorrect predictions. **This function is included to give you an example of how to visualize this kind of data.**
src/processData.m	Script that loads the raw cardiotogography data and centers and scales the data. **This script is complete and you should not need to modify it.**
src/runCardioExperiments.m	Main driver script for experiments. The provided version contains the outline of the tasks.
src/runSyntheticExperiments.m	Main driver script for experiments. The provided version contains the outline of the tasks.