CPSC 340 Assignment 3 (due November 3 ATE)

1 Convex Functions

Recall that convex loss functions are typically easier to minimize than non-convex functions, so it's important to be able to identify whether a function is convex.

Show that the following functions are convex:

1. $f(w) = \alpha w^2 - \beta w + \gamma$ with $w \in \mathbb{R}, \alpha \geq 0, \beta \in \mathbb{R}, \gamma \in \mathbb{R}$ (1D quadratic).
2. $f(w) = w \log(w)$ with $w > 0$ ("neg-entropy")
3. $f(w) = \|Xw - y\|^2 + \lambda \|w\|_1$ with $w \in \mathbb{R}^d, \lambda \geq 0$ (L1-regularized least squares).
4. $f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$ with $w \in \mathbb{R}^d$ (logistic regression).
5. $f(w, w_0) = \sum_{i=1}^{N} [\max\{0, w_0 - w^T x_i\} - w_0] + \frac{\lambda}{2} \|w\|_2^2$ with $w \in R^d, w_0 \in \mathbb{R}, \lambda \geq 0$ ("1-class" SVM).

Hint: for the first two you can use the second-derivative test since they are one-dimensional. For the last 3 you'll have to use some of the results in class regarding how combining convex functions can yield convex functions. The logistic regression case may not seem obvious with $\log(x)$ is concave, but a similar case would be showing that $\log(\exp(x))$ is convex despite this same issue.

2 Gaussian RBFs and Regularization

Unfortunately, in practice we often don't know what basis to use. However, if we have enough data then we can make up for this by using a basis that is flexible enough to model any reasonable function. These may perform poorly if we don't have much data, but can perform almost as well as the optimal basis as the size of the dataset grows. In this question you will explore using Gaussian radial basis functions (RBFs), which have this property. These RBFs depend on a parameter σ, which (like p in the polynomial basis) can be chosen using a validation set. In this question, you will also see how cross-validation allows you to tune parameters of the model on a larger dataset than a strict training/validation split would allow.

2.1 Regularization

If you run the demo *example_RBF.jl*, it will load a dataset and split the training examples into a "train" and a "validation" set (it does this randomly since the data is sorted). It will then search for the best value of σ for the RBF basis. Once it has the "best" value of σ, it re-trains on the entire dataset and reports the training error on the full training set as well as the error on the test set.

A strange behaviour appears: if you run the script more than once it might choose different values of σ. Sometimes it chooses a large value of σ that follows the general trend but misses the oscillations. Other times it sets $\sigma = 1$ or $\sigma = 2$, which fits the oscillations better but overfits and gives a much higher test

error.[1] Modify the *leastSquaresRBF* function so that it allows a regularization parameter λ and it fits the model with L2-regularization. Hand in your code, and report and describe how the performance changes if you use a regularized estimate with $\lambda = 10^{-12}$ (a very small value).

2.2 Cross-Validation

While the method rarely performs too badly with regularization, but it's clear that the randomization of the training/validation sets has an effect on the value of σ that we choose. This variability would be reduced if we had a larger "train" and "validation" set, and one way to simulate this is with *cross-validation*. Modify the training/validation procedure to use 10-fold cross-validation to select σ, and hand in your code. How does this change the performance when fixing $\lambda = 10^{-12}$?[2]

2.3 Cost of Non-Parametric Bases

When dealing with larger datasets, an important issue is the dependence of the computational cost on the number of training examples n and the number of features d.

1. What is the cost in big-O notation of training a linear regression model with Gaussian RBFs on n training examples with d features (for fixed σ and λ)?
2. What is the cost of classifying t new examples with this model?
3. When is it cheaper to train using Gaussian RBFs than using the basic linear basis?
4. When is it cheaper to test using Gaussian RBFs than using the basic linear basis?

3 Logistic Regression with Sparse Regularization

If you run the function *example_logistic.jl*, it will:

1. Load a binary classification dataset containing a training and a validation set.
2. "Standardize" the columns of X and add a bias variable.
3. Apply the same transformation to $Xvalidate$.
4. Fit a least squares model, using the sign of $w^T x_i$ to make predictions.
5. Report the number of features selected by the model (number of non-zero regression weights).
6. Report the error on the training and validation sets.

Least squares does ok as a binary classifier on this dataset, but it uses all the features (even though only the prime-numbered features are relevant) and the validation error is above the minimum achievable for this model (which is 1 percent, if you have enough data and know which features are relevant). In this question, you will modify this demo to use different forms of regularization to improve on these aspects.

[1] This behaviour seems to be dependent on your exact setup. Because the $X^T X$ matrix with the RBF matrix is really-badly behaved numerically, different floating-point and matrix-operation implementations will handle this in different ways: in some settings it will actually regularizer for you!

[2] In practice, we typically use cross-validation to choose both σ and λ

3.1 Logistic Regression

Instead of least squares, modify the script to use logistic regression. You can use the *logReg.jl* file, which implements the training and prediction function for a logistic regresion classifier (using a version of the *findMin* function that does derivative checking for you and that uses more-clever choices of step-sizes). When you switch to using logistic regression, report how the following quantities change: the training error, validation error, and number of features.

3.2 L2-Regularization

Make a new function, *logRegL2*, that takes an input parameter λ and fits a logistic regression model with L2-regularization. Specifically, while *logReg* computes w by minimizing

$$f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i)),$$

your new function *logRegL2* should compute w by minimizing

$$f(w) = \sum_{i=1}^{n} \left[\log(1 + \exp(-y_i w^T x_i))\right] + \frac{\lambda}{2}\|w\|^2.$$

Hand in the objective function that your updated code minimizes, and using $\lambda = 1.0$ report how the following quantities change: the training error, the validation error, the number of features used, and the number of gradient descent iterations.

3.3 L1-Regularization

Make a new function, *logRegL1*, that takes an input parameter λ and fits a logistic regression model with L1-regularization,

$$f(w) = \sum_{i=1}^{n} \left[\log(1 + \exp(-y_i w^T x_i))\right] + \lambda\|w\|_1.$$

Hand in your *logRegL1* code. Using this new code and $\lambda = 1$, report the following quantities: the training error, the validation error, and the number of features the model uses.

You should use the function *findMinL1*, which implements a proximal-gradient method to minimize the sum of a differentiable function g and $\lambda\|w\|_1$,

$$f(w) = g(w) + \lambda\|w\|_1.$$

This function has a similar interface to *findMin*, except that you (a) only provide the code to compute the function/gradient of the differentiable part g and (b) need to provide the value λ.

3.4 L0-Regularization

The function *logRegL0* contains part of the code needed to implement the *forward selection* algorithm, which approximates the solution with L0-regularization,

$$f(w) = \sum_{i=1}^{n} \left[\log(1 + \exp(-y_i w^T x_i))\right] + \lambda\|w\|_0.$$

The 'for' loop in this function is missing the part where we fit the model using the subset Sj, then compute the score and updates the *minScore/minS*. Modify the 'for' loop in this code so that it fits the model using only the features Sj, computes the score above using these features, and updates the *minScore/minS* variables (if you want to turn off the diagonistics generated by*findMin*, you can use *verbose* = *false*).[3] Hand in your updated code. Using this new code, set $\lambda = 1$ and report: the training error, the validation error, and the number of features used.

Note that the code differs a bit from what we discussed in class, since we assume that the first feature is the bias variable and assume that the bias variable is always included. Also, note that for this particular case using the L0-norm with $\lambda = 1$ is equivalent to what is known as the Akaike information criterion (BIC) for variable selection.

4 Very-Short Answer Questions

1. Why would you use a score BIC instead of a validation error for feature selection?
2. Why do we use forward selection instead of exhaustively search all subsets in search and score methods?
3. In L2-regularization, how does λ relate to the two parts of the fundamental trade-off?
4. Give one reason why one might chose to use L1 regularization over L2 and give one reason for the reverse case.
5. If we have a feature selection method that tends to have many false negatives (it misses many relevant variables), describe an ensemble feature selection method that could decrease the number of false negatives.
6. What is the main problem with using least squares to fit a linear model for binary classification?
7. For a linearly-separable binary classification problem, how does an SVM classifier differ from a classifier found using the perceptron algorithm?
8. When $d >> n$, why do we use the polynomial kernel to implement the polynomial basis?
9. Which of the following methods produce linear classifiers? (a) binary least squares as in Question 3, (b) the perceptron algorithm, (c) SVMs, and (d) logistic regression.

Hints: we're looking for short and concise 1-sentence answers, not long and complicated answers. Also, there is roughly 1 question per lecture.

[3]Note that Julia doesn't like when you re-define functions, but if you change the variable *Xs* it will actually change the behaviour of the *funObj* that is already defined.