CPSC 340 Assignment 2 (due Friday October 13 ATE)

1 Random Forests

1.1 Implementation

Thefile *vowels.jld* contains a supervised learning dataset where we are trying to predict which of the 11 "steady-state" English vowels that a speaker is trying to pronounce.

You are provided with a `decisionTree` as well as a `randomTree` function in *decisionTree.jl* (both based on information gain). The random tree model differs from the decision tree model in two ways: it takes a bootstrap sample of the data before fitting and when fitting individual stumps it only considers $\lfloor\sqrt{d}\rfloor$ randomly-chosen features[1] In other words, `RandomTree` is the model we discussed in class that is combined to make up a random forest.

If you run *example_randomTree.jl*, it will fit both models to the dataset, and you will notice that it overfits badly.

1. If you set the *depth* parameter to *Inf*, why do the training functions terminate?
2. Why doesn't the random tree model with a depth of *Inf* have a training error of 0?
3. Create a function `randomForest` that takes in hyperparameters `depth` and `nTrees` (number of trees), and fits `nTrees` random trees each with maximum depth `depth`. For prediction, have all trees predict and then take the mode. Hand in your function. Hint: you can define an array for holding 10 *GenericModel* types using:
   `subModels = Array{GenericModel}(10)`.
4. Using 50 trees, and a depth of ∞, report the training and testing error. Compare this to what we got with a single `DecisionTree` and with a single `RandomTree`. Are the results what you expected? Discuss.

1.2 Very-Short Answer Questions

1. What is a a disadvantage of using a very-large number of trees in a random forest classifier?
2. Your random forest classifier has a training error of 0 and a very high test error. Which ones of the following could help performance?
   (a) Increase the maximum depth of the trees in your forest.
   (b) Decrease the maximum depth of the trees in your forest.
   (c) Increase the amout of data you consider for each tree (Collect more data and use 2n objects instead of n).
   (d) Decrease the amount of data you consider for each tree (Use 0.8n objects instead of n).

[1]The notation $\lfloor x\rfloor$ means the "floor" of x, or "x rounded down".

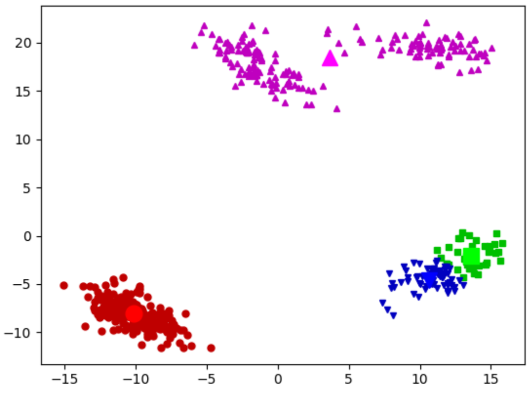
(e) Increase the number of features you consider for each tree.

(f) Decrease the number of features you consider for each tree.

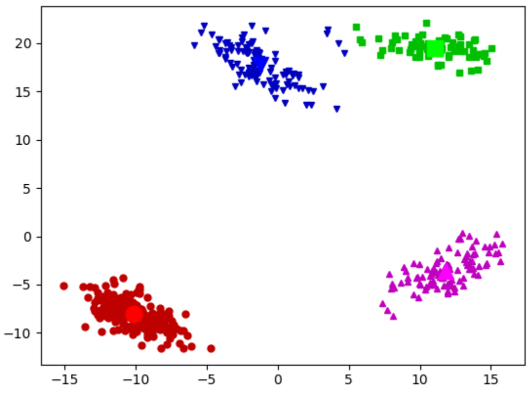
3. Suppose that you were training on raw audio segments and trying to recognize vowel sounds. What could you do to encourage the final classifier to be invariant to translation?

2 K-Means Clustering

If you run the function *example_Kmeans*, it will load a dataset with two features and a very obvious clustering structure. It will then apply the k-means algorithm with a random initialization. The result of applying the algorithm will thus depend on the randomization, but a typical run might look like this:



(Note that the colours are arbitrary due to the label switching problem.) But the 'correct' clustering (that was used to make the data) is something more like this:



2.1 Selecting among k-means Initializations

If you run the demo several times, it will find different clusterings. To select among clusterings for a *fixed* value of k, one strategy is to minimize the sum of squared distances between examples x_i and their means

w_{y_i},

$$f(w_1, w_2, \ldots, w_k, y_1, y_2, \ldots, y_n) = \sum_{i=1}^{n} \|x_i - w_{y_i}\|_2^2 = \sum_{i=1}^{n}\sum_{j=1}^{d}(x_{ij} - w_{y_ij})^2.$$

where y_i is the index of the closest mean to x_i. This is a natural criterion because the steps of k-means alternately optimize this objective function in terms of the w_c and the y_i values.

1. Write a new function called *kMeansError* that a dataset X, a set of cluster assignments y, and a set of cluster means W, and computes this objective function. Hand in your code.
2. Instead of printing the number of labels that change on each iteration, what trend do you observe if you print the value of *kMeansError* after each iteration of the k-means algorithm?
3. Using the *clustering2Dplot* file, output the clustering obtained by running k-means 50 times (with $k = 4$) and taking the one with the lowest error. Note that the k-means training function will run much faster if you set `doPlot = false` or just remove this argument.

2.2 Selecting k in k-means

We now turn to the task of choosing the number of clusters k.

1. Explain why the *kMeansError* function should not be used to choose k.
2. Explain why even evaluating the *kMeansError* function on test data still wouldn't be a suitable approach to choosing k.
3. Hand in a plot of the minimum error found across 50 random initializations, as you vary k from 1 to 10.
4. The *elbow method* for choosing k consists of looking at the above plot and visually trying to choose the k that makes the sharpest "elbow" (the biggest change in slope). What values of k might be reasonable according to this method? Note: there is not a single correct answer here; it is somewhat open to interpretation and there is a range of reasonable answers.

2.3 k-Medians

The data in *clusterData2.mat* is the exact same as the above data, except it has 4 outliers that are very far away from the data.

1. Using the *clustering2Dplot* function, output the clustering obtained by running k-means 50 times (with $k = 4$) on *clusterData2.mat* and taking the one with the lowest error. Are you satisfied with the result?
2. What values of k might be chosen by the elbow method for this dataset?
3. Implement the *k-medians* algorithm, which assigns examples to the nearest w_c in the L1-norm and to updates the w_c by setting them to the "median" of the points assigned to the cluster (we define the d-dimensional median as the concatenation of the median of the points along each dimension). Hand in your code and plot obtained with 50 random initializations for $k = 4$.
4. Using the L1-norm version of the error (where y_i now represents the closest median in the L1-norm),

$$f(w_1, w_2, \ldots, w_k, y_1, y_2, \ldots, y_n) = \sum_{i=1}^{n} \|x_i - w_{y_i}\|_1 = \sum_{i=1}^{n}\sum_{j=1}^{d}|x_{ij} - w_{y_ij}|,$$

   what value of k would be chosen by the elbow method under this strategy? Are you satisfied with this result?

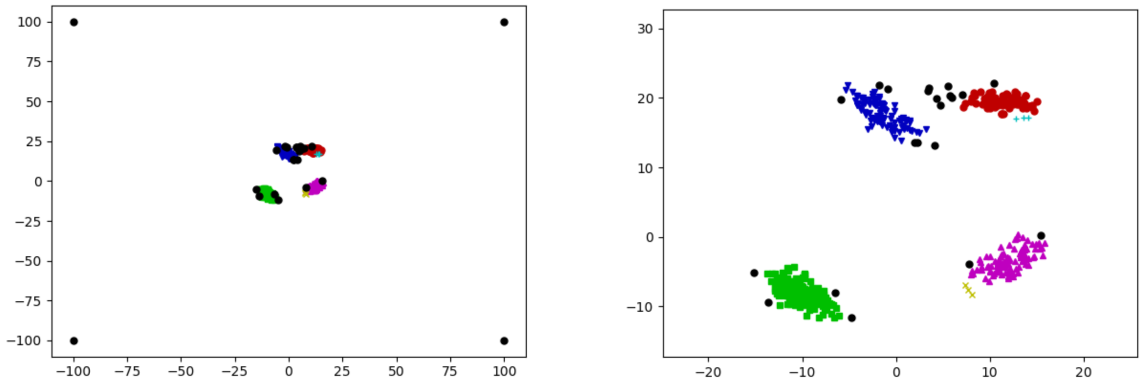
2.4 Very-Short Answer Questions

1. Does the standard k-means clustering algorithm always yield the optimal clustering solution for a given k?
2. If your set out to minimize the distance between each point and its mean in a k-means clustering, what value of k minimizes this cost? Is this value useful?
3. Describe a dataset with k clusters where k-means would not be able to find the true clusters.

3 More Unsupervised Learning

3.1 Density-Based Clustering

If you run the function *example_dbCluster*, it will apply the basic density-based clustering algorithm to the dataset from the previous part. The final output should look like this:



(The right plot is zoomed in to show the non-outlier part of the data.) Even though we know that each object was generated from one of four clusters (and we have 4 outliers), the algorithm finds 6 clusters and does not assign some of the original non-outlier objects to any cluster. However, the clusters will change if we change the parameters of the algorithm. Find and report values for the two parameters (*radius* and *minPts*) such that the density-based clustering method finds:

1. The 4 "true" clusters.
2. 3 clusters (merging the top two, which also seems like a reasonable interpretaition).
3. 2 clusters.
4. 1 cluster (consisting of the non-outlier points).

3.2 Vector Quantization

Discovering object groups is one motivation for clustering. Another motivation is *vector quantization*, where we find a prototype point for each cluster and replace points in the cluster by their prototype. If our inputs are images, we could use vector quantization on the set of RGB pixel values as a simple image compression algorithm.

Your task is to implement this simple image compression algorithm by writing a `quantizeImage` and a `deQuantizeImage` function. The `quantizeImage` function should take the name of an image file (like "dog.png" for the provided image) and a number b as input. It should use the pixels in the image as examples and the 3 colour channels as features, and run k-means clustering on this data with 2^b clusters. The code should store the cluster means and return four arguments: the cluster assignments y, the means W, the number of rows in the image $nRows$, and the number of columns $nCols$. The `deQuantizeImage` function should take these four arguments and return a version of the image (the same size as the original) where each pixel's original colour is replaced with the nearest prototype colour.

To understand why this is compression, consider the original image space. Say the image can take on the values $0, 1, \ldots, 254, 255$ in each colour channel. Since $2^8 = 256$ this means we need 8 bits to represent each colour channel, for a total of 24 bits per pixel. Using our method, we are restricting each pixel to only take on one of 2^b colour values. In other words, we are compressing each pixel from a 24-bit colour representation to a b-bit colour representation by picking the 2^b prototype colours that are "most representative" given the content of the image. So, for example, if $b = 6$ then we have 4x compression.

Note: you can read the image using "imread" function in the PyPlot package (it takes a file name and returns a $nRows$ by $nCols$ by 3 array containing the images RGB values). Similarly, the "imshow" function can display an image represented in this format. You may find it help to use the "reshape" function.

1. Hand in your *quantizeImage* and *deQuantizeImage* functions.
2. Show the image obtained if you encode the colours using 1, 2, 4, and 6 bits per pixel (instead of the original 24-bits).

3.3 Very-Short Answer Questions

1. Suppose that you had only two features and that they have very-different scales (like kilograms vs. milligrams). How would this affect the result of density-based clustering?
2. Name a key advantage and drawback of using a supervised outlier detection method rather than an unsupervised method?
3. Given an $n \times 2$ matrix X and a test query $\hat{x}$, what is the cost of finding all rows i in X where $\|x_i - \hat{x}\| \leq r$ for some $r > 0$? How does this cost change if I give you a hash table that assigns rows of X to keys that divide the space into a 2D grid of squares with radius r, if we use k to denote the maximum number of points hashed to the same key value?

4 Matrix Notation and Linear Regression

4.1 Converting to Matrix/Vector/Norm Notation

Using our standard supervised learning notation (X, y, w) express the following functions in terms of vectors, matrices, and norms (there should be no summations or maximums).

1. $\sum_{i=1}^{n} |w^T x_i - y_i|$.
2. $\max_{i \in \{1,2,\ldots,n\}} |w^T x_i - y_i| + \frac{\lambda}{2} \sum_{j=1}^{d} w_j^2$.
3. $\sum_{i=1}^{n} v_i (w^T x_i - y_i)^2 + \lambda \sum_{j=1}^{d} |w_j|$.

You can use V to denote a diagonal matrix that has the (non-negative) values v_i along the diagonal. The "regularization parameter" λ is a non-negative scalar.

4.2 Minimizing Quadratic Functions as Linear Systems

Write finding a minimizer w of the functions below as a system of linear equations (using vector/matrix notation and simplifying as much as possible). Note that all the functions below are convex so finding a w with $\nabla f(w) = 0$ is sufficient to minimiize the functions (but show your work in getting to this point).

1. $f(w) = \frac{1}{2}\|w - u\|^2$.
2. $f(w) = \frac{1}{2}\|w\|^2 + w^T X^T y$.
3. $f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{1}{2}w^T \Lambda w$.
4. $f(w) = \frac{1}{2}\sum_{i=1}^{n} v_i(w^T x_i - y_i)^2$.

Above we assume that u is a d by 1 vector, and Λ is a d by d diagonal matrix with positive entries along the diagonal.

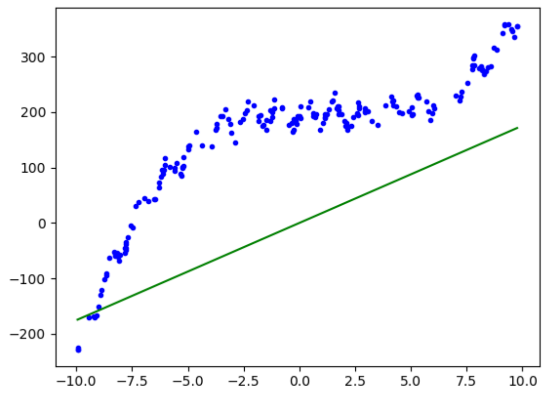
Hint: Once you convert to vector/matrix notation, you can use the results from class to quickly compute these quantities term-wise. As a sanity check for your derivation, make sure that your results have the right dimensions.

4.3 Linear Regresion with Bias Variable

If you run the script *example_nonLinear*, it will:

1. Load a one-dimensional regression dataset.
2. Fit a least-squares linear regression model.
3. Report the training error.
4. Report the test error (on a dataset not used for training).
5. Draw a figure showing the training data and what the linear model looks like.

Unfortunately, this is an awful model of the data. The average squared training error on the data set is over 28000 (as is the test error), and the figure produced by the demo confirms that the predictions are usually nowhere near the training data:



The y-intercept of this data is clearly not zero (it looks like it's closer to 200), so we should expect to improve performance by adding a *bias* variable, so that our model is

$$y_i = w^T x_i + w_0.$$

instead of

$$y_i = w^T x_i.$$

Write a new function, *leastSquaresBias*, that has the same input/model/predict format as the *leastSquares* function, but that adds a *bias* variable w_0. Hand in your new function, the updated plot, and the updated training/test error.

Hint: recall that adding a bias w_0 is equivalent to adding a column of ones to the matrix X. Don't forget that you need to do the same transformation in the *predict* function.

4.4 Linear Regression with Polynomial Basis

Adding a bias variable improves the prediction substantially, but the model is still problematic because the target seems to be a *non-linear* function of the input. Write a new function, *leastSquaresBasis(x,y,p)*, that takes a data vector x (i.e., assuming we only have one feature) and the polynomial order p. The function should perform a least squares fit based on a matrix Z where each of its rows contains the values $(x_i)^j$ for $j = 0$ up to p. E.g., *leastSquaresBasis(x,y,3)* should form the matrix

$$Z = \begin{bmatrix} 1 & x_1 & (x_1)^2 & (x_1)^3 \\ 1 & x_2 & (x_2)^2 & (x_2)^3 \\ \vdots & & & \\ 1 & x_n & (x_n)^2 & (x_N)^3 \end{bmatrix},$$

and fit a least squares model based on it. Hand in the new function, and report the training and test error for $p = 0$ through $p = 10$. Explain the effect of p on the training error and on the test error.

Note: for this question we'll assume $d = 1$ (we'll discuss polynomial bases with more input features later in the course).

Hints: To keep the code simple and reduce the chance of having errors, you may want to write a new function *polyBasis* that you can use for transforming both the training and testing data.

4.5 Manual Search for Optimal Basis

Polynomials are a flexible class of functions, but there is structure in this data that is not well-modelled by polynomials. Try to find a nonlinear basis that gives the best performance on this dataset in terms of test error. Report the basis that you use and the training/test score that you achieve.

Hint: the data seems to have periodic behaviour, and it's possible to obtain training and test errors below 60.

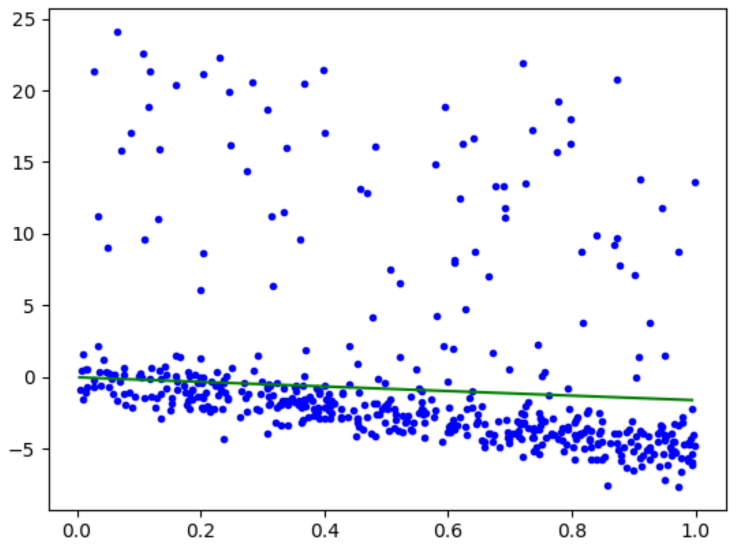
4.6 Very-Short Answer Questions

1. In this question, why are we computing the squared error $(y_i - \hat{y}_i)^2$ and not testing the equality $(y_i = \hat{y}_i)$?
2. Describe a simple 2-feature ($d = 2$) case where the least squares estimate would not be unique.

3. What is the computational complexity of computing the closed-form (exact) solution to a linear least squares problem where we have one feature ($d = 1$) and use polynomial basis of degree p?
4. In what circumstance would a regression tree with linear regressions at the leaves be a better choice than a linear least squares regression model?

5 Robust Regression and Gradient Descent

The script *example_outliers* loads a one-dimensional regression dataset that has a non-trivial number of 'outlier' data points. These points do not fit the general trend of the rest of the data, and pull the least squares model away from the main downward trend that most data points exhibit:



5.1 Weighted Least Squares in One Dimension

One of the most common variations on least squares is *weighted* least squares. In this formulation, we have a weight v_i for every training example. To fit the model, we minimize the weighted squared error,

$$f(w) = \frac{1}{2}\sum_{i=1}^{n} v_i(w^T x_i - y_i)^2.$$

In this formulation, the model focuses on making the error small for examples i where v_i is high. Similarly, if v_i is low then the model allows a larger error.

Write a model function, *weightedLeastSquares(X,y,v)*, that implements this model (note that a previous question asks you to show how this formulation can be solved as a linear system). Apply this model to the data containing outliers, setting $v_i = 1$ for the first 400 data points and $v_i = 0.1$ for the last 100 data points (which are the outliers). Hand in your function and the updated plot.

5.2 Smooth Approximation to the L1-Norm

Unfortunately, we typically do not know the identities of the outliers. In situations where we suspect that there are outliers, but we do not know which examples are outliers, it makes sense to use a loss function that is more robust to outliers. In class, we discussed using the sum of absolute values objective,

$$f(w) = \sum_{i=1}^{n} |w^T x_i - y_i|.$$

This is less sensitive to outliers than least squares, but it is non-differentiable and harder to optimize. Nevertheless, there are various smooth approximations to the absolute value function that are easy to optimize. One possible approximation is to use the log-sum-exp approximation of the max function[2]

$$|r| \approx \log(\exp(r) + \exp(-r)).$$

Using this approximation, we obtain an objective of the form

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

which is smooth but less sensitive to outliers than the squared error. Derive the gradient ∇f of this function with respect to w. You should show your work but you do not have to express the final result in matrix notation.

5.3 Robust Regression

The function *example_gradient* is the same as *example_outlier*, except that it fits the least squares model using a *gradient descent* method. You'll see that it produces the same fit as we obtained using the normal equations.

The typical input to a gradient method is a function that, given w, returns $f(w)$ and $\nabla f(w)$. See *funObj* in the *leastSquaresGradient* function for an example. Note that *leastSquaresGradient* also has a numerical check that the gradient code is approximately correct, since implementing gradients is often error-prone.[3]

An advantage of gradient-based strategies is that they are able to solve problems that do not have closed-form solutions, such as the formulation from the previous section. The function *robustRegression* has most of the implementation of a gradient-based strategy for fitting the robust regression model under the log-sum-exp approximation. The only part missing is the function and gradient calculation inside the *funObj* code. Modify this function to implement the objective function and gradient based on the smooth approximation to the absolute value function (from the previous section). Hand in your code, as well as the plot obtained using this robust regression appraoch.

5.4 Very-Short Answer Questions

1. In class we considered 4 general strategies for outlier detection (model-based, graph-based, cluster-based, distance-based). Pick two of these and describe whether they would be effective for detecting the outliers in this dataset.

[2]Other possibilities are the Huber loss, $|r| \approx \sqrt{r^2 + \epsilon}$ for some small ϵ.

[3]Though sometimes the numerical gradient checker itself can be wrong. For a lot more on numerical differentiation you can take CPSC 303.

2. When should we consider using gradient descent to approximate the solution to the least squares problem instead of exactly solving it with the closed form solution?
3. Why are we smoothing the absolute value? Why can't we just set the gradient to 0 and solve a linear system?