CPSC 340 Assignment 3 (due Monday October 17 at 11:55pm)

1 More Unsupervised Learning

1.1 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: the script *example_image.jl* shows how to read an image file using *Images* package, how to display an image using the *Plots* package, and how to convert images to/from the feature representation of their pixels.

1. Hand in your *quantizeImage* and *deQuantizeImage* functions.

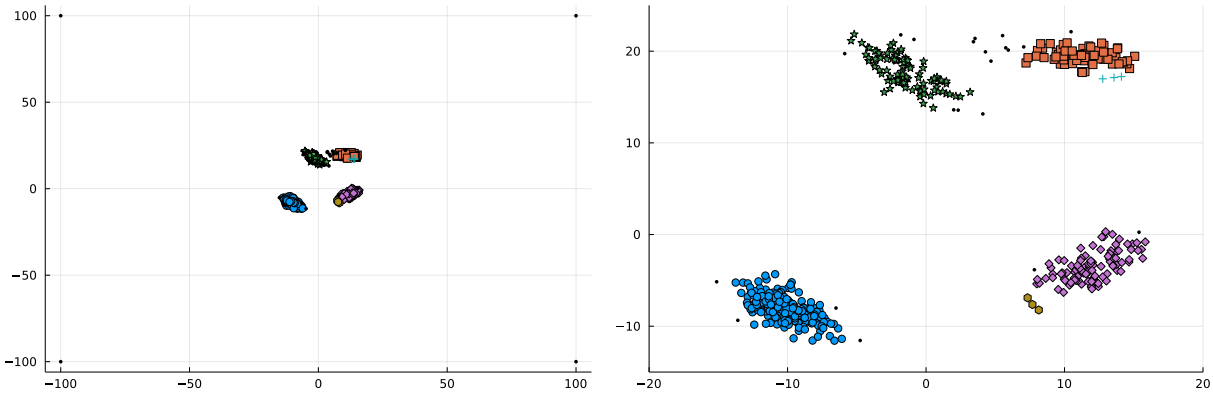
2. Show the image obtained if you encode the colours using 1, 2, 4, and 6 bits per pixel (instead of the original 24-bits).

3. Save the image with 6 bits per pixel to a .PNG file with *save* function. By going from 8-bits to 6-bits to store the colour of each pixel we would expect the PNG file to be 25% smaller than the original image. Explain why you think the actual PNG file is larger or smaller than this compression value.

   Answer: I would expect for the actual PNG to be smaller. As the magnitude of compression is on an exponential basis, the number of bits to represent each pixel actually decreases by 4x and not just 25%.

1.2 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 (points not assigned to any cluster and displayed as small black circles). 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).

2 Matrix Notation and Linear Regression

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

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

2.2 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 minimize the functions (but show your work in getting to this point).

1. $f(w) = \frac{1}{2}\|w - u\|^2$ (projection of u onto real space).
2. $f(w) = \frac{1}{2}\sum_{i=1}^{n} v_i(w^T x_i - y_i)^2 + \lambda w^T u$ (weighted and tilted least squares).
3. $f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w - w^0\|^2$ (least squares shrunk towards non-zero w^0).

Above we assume that u and w^0 are d by 1 vectors, that v is a n by 1 vector. You can use V as a diagonal matrix containing the v_i values 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. As a sanity check, make that the dimensions match for all quantities/operations. In order to make the dimensions match you may need to introduce an identity matrix. For example, $X^T Xw + \lambda w$ can be re-written as $(X^T X + \lambda I)w$.

2.3 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. $f(w) = \frac{1}{2}w^2 + w^{-1}$ with $w > 0$.
2. $f(w) = \max_i w_i$ with $w \in \mathbb{R}^n$ (maximum).
3. $f(y) = \max(0, 1 - t \cdot y)$ with $y \in \mathbb{R}$ and $t \in \{-1, +1\}$ (hinge loss).
4. $f(w) = \|Xw - y\|^2 + \lambda\|w\|_1$ with $w \in \mathbb{R}^d, \lambda \geq 0$ (L1-regularized least squares).
5. $f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$ with $w \in \mathbb{R}^d$ (logistic regression).

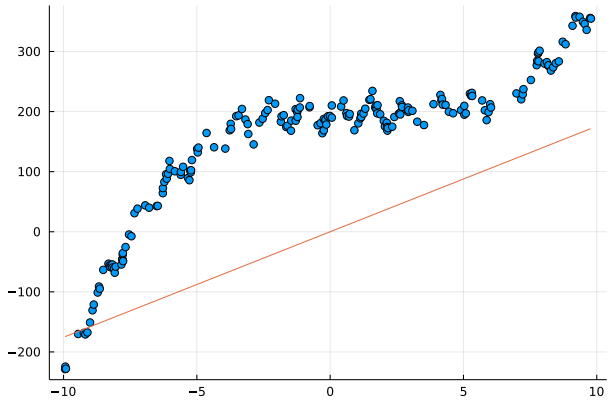
Hint for Part 5: this function may seem non-convex since it contains $\log(z)$ and log is concave, but there is a flaw in that reasoning: for example $\log(\exp(z)) = z$ is convex despite containing a log. To show convexity, it may be helpful to show that $\log(1+\exp(z))$ is convex, which can be done by computing the second derivative. It may simplify matters to note that $\frac{\exp(z)}{1+\exp(z)} = \frac{1}{1+\exp(-z)}$.

3 Linear and Nonlinear Regression

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:



3.1 Linear Regresion with Bias Variable

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.

3.2 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 you should assume that you only have one feature ($d = 1$). We will 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.

3.3 Gaussian RBFs

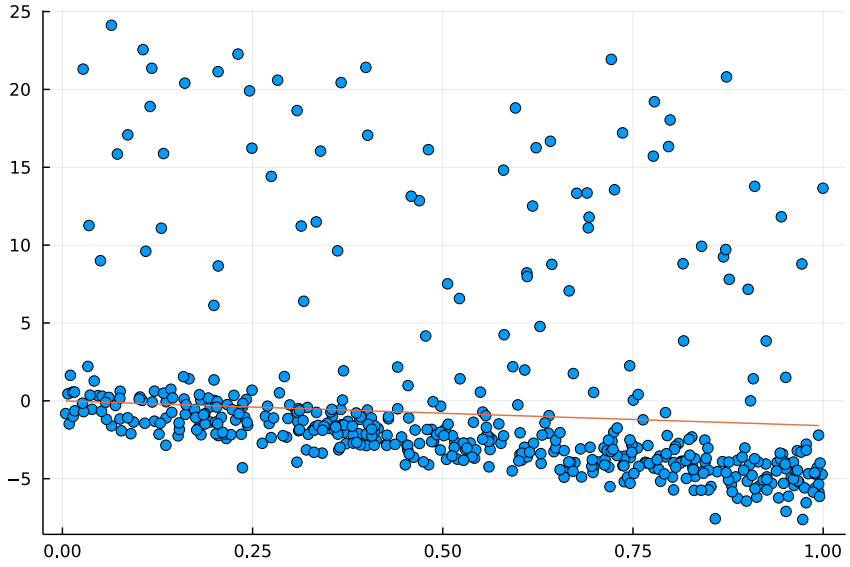
A popular alternative to using polynomials is to use Gaussian radial basis functions, with one basis function centered on each training example. For a generic feature $\tilde{x}_i$, the transformed feature vector with this choise is given by

$$\tilde{z}_i = \left[\underbrace{\exp\left(-\frac{(\tilde{x}_i - x_1)^2}{2\sigma^2}\right)}_{\text{feature 1}} \quad \underbrace{\exp\left(-\frac{(\tilde{x}_i - x_2)^2}{2\sigma^2}\right)}_{\text{feature 2}} \quad \cdots \quad \underbrace{\exp\left(-\frac{(\tilde{x}_i - x_n)^2}{2\sigma^2}\right)}_{\text{feature } n}\right],$$

where σ^2 is a hyper-parameter. With n training examples, this generates a set of n features based on a transformation of the distance between the example and each training example. Hand in a function implementing least squares under this basis, and the plots obtained by $\sigma = 10$, $\sigma = 1$, and $\sigma = 0.1$. How does the value of σ^2 affect the fundamental trade-off?

4 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:



4.1 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 this 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.

4.2 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 Huber loss,

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

where

$$h(r_i) = \begin{cases} \frac{1}{2} r_i^2 & \text{for } |r_i| \leq \epsilon \\ \epsilon(|r_i| - \frac{1}{2}\epsilon) & \text{otherwise} \end{cases}.$$

This is less sensitive to outliers than least squares, although it can no longer be minimized by solving a linear system. 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. Hint: you can start by computing the derivative of h with respect to r_i and then get the gradient using the chain rule. You can use $\text{sgn}(r_i)$ as a function that returns 1 if r_i is positive and -1 if it is negative.

4.3 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.[1]

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 Huber regression model. 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 Huber loss (from the previous section). Hand in your code, as well as the plot obtained using this robust regression approach with $\epsilon = 1$.

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

5 Very-Short Answer Questions

1. Describe a dataset with k clusters where k-means cannot find the true clusters.
2. Why do we need random restarts for k-means but not for density-based clustering?
3. Why is it not a good idea to create an ensemble out of multiple k-means runs with random restarts and, for each example, output the mode of the label assignments (voting)?
4. For each outlier detection method below, list an example method and a problem with identifying outliers using this method:
   - Model-based outlier detection.
   - Graphical-based outlier detection.
   - Supervised outlier detection.
5. Why do we minimize $\frac{1}{2}\sum_{i=1}^{n}(wx_i - y_i)^2$ instead of the actual mean squared error $\frac{1}{n}\sum_{i=1}^{n}(wx_i - y_i)^2$ in (1D) least squares?
6. Give an example of a feature matrix X for which the least squares problem *cannot* be solved as $w = (X^\top X)^{-1}(X^\top y)$.
7. Why do we typically add a column of 1 values to X when we do linear regression? Should we do this if we're using decision trees?
8. 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?
9. If a function is convex, what does that say about stationary points of the function? Does convexity imply that a stationary points exists?
10. For robust regression based on the L1-norm error, why can't we just set the gradient to 0 and solve a linear system? In this setting, why we would want to use a smooth approximation to the absolute value?
11. What is the problem with having too small of a learning rate in gradient descent?
12. What is the problem with having too large of a learning rate in gradient descent?