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

On this assignment, we are only going to allow a maximum of 1 "late class" to be used. This is because the midterm is on October 17th, and we want to give you more than 1 day to look at the solutions before the actual exam.

1 More Unsupervised Learning

1.1 k-Medians

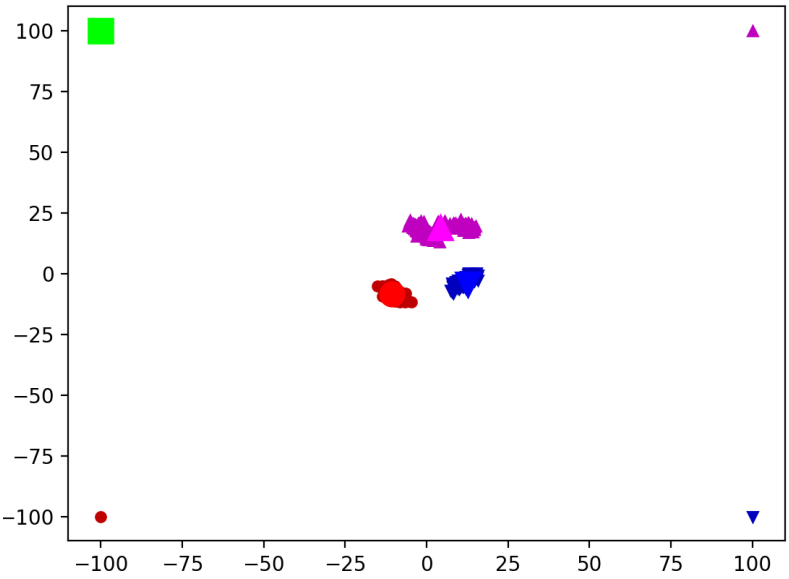
The data in *clusterData2.jl* is the exact same as *clusterData.jl* from a previous question, except it has 4 outliers that are far away from the data.

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 then 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 medians along each dimension). For this algorithm it makes sense to use the L1-norm version of the error (where y_i now represents the closest median in the L1-norm),

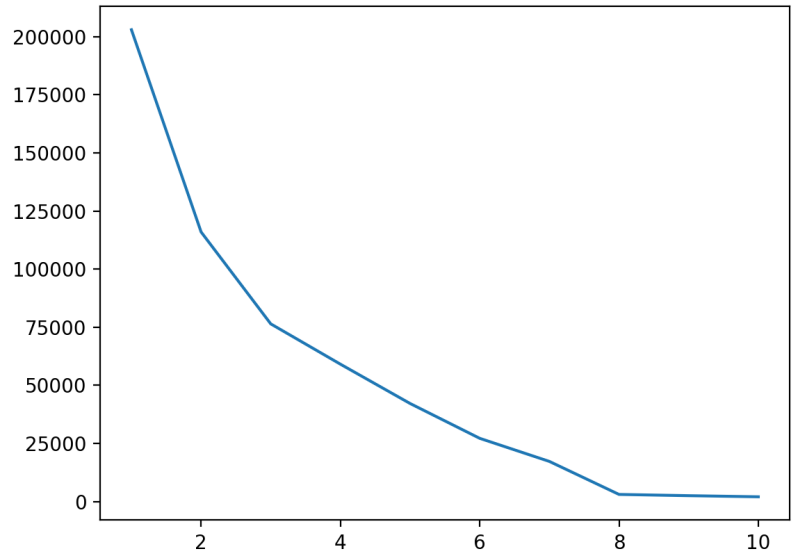
$$f(w_1, w_2, \ldots, w_k, y_1, y_2, \ldots, y_n) = \sum_{i=1}^{n} \|x_i - w_{y_i}\|_1 = \sum_{i=1}^{n} \sum_{j=1}^{d} |x_{ij} - w_{y_i j}|,$$

   Hand in your code and plot obtained by taking the clustering with the lowest L1-norm after using 50 random initializations for $k = 4$.
4. What value of k would be chosen by the elbow method if you k-medians and the L1-norm? Are you satisfied with this result?

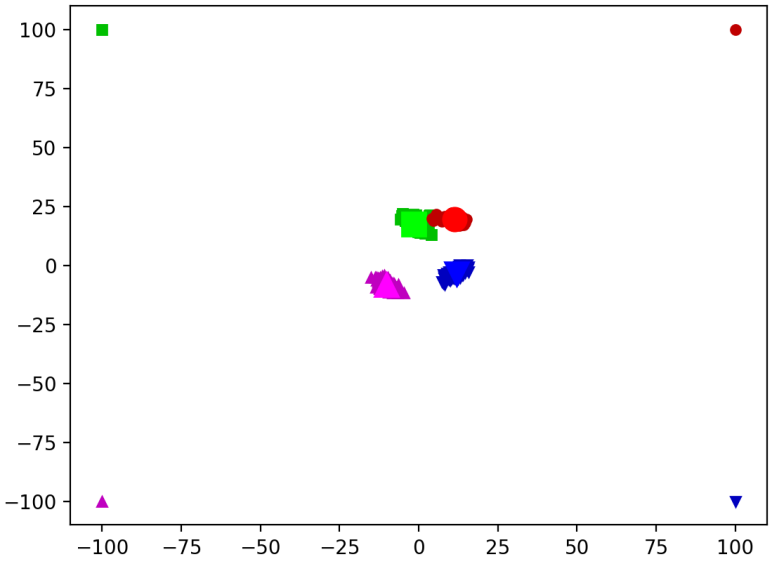
Answer: 1. the lowest error: 59116.295802958084;
not satisfied, the error is still too large since some outliers are included in some groups that are far from them.



Answer: 2. we had a plot of the minimum error found across 50 random initializations, as a function of k, taking k from 1 to 10.
From the graph, we choose k=8. Note: X-axis: k; Y-axis: minimum error



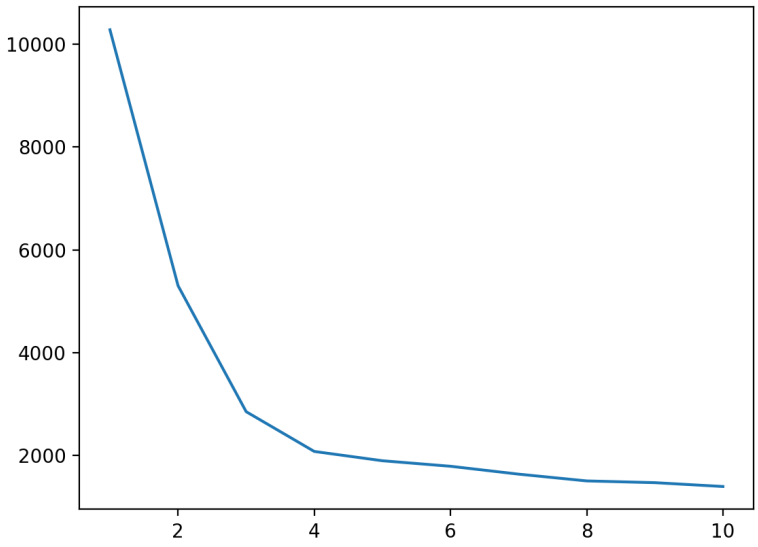
Answer: 3. lowest L1-norm: 2080.8964257454077




```
function kMedians(X,k;doPlot=false)
# K-medians clustering

(n,d) = size(X)

# Choose random points to initialize medians
W = zeros(k,d)
perm = randperm(n)
for c = 1:k
        W[c,:] = X[perm[c],:]
end

# Initialize cluster assignment vector
y = zeros(Int64, n)
changes = n
while changes != 0

        # Compute L1-norm between each point and each median
        D = distancesAbsolute(X,W)

        # Degenerate clusters will distance NaN, change to Inf
        # (since Julia thinks NaN is smaller than all other numbers)
        D[findall(isnan.(D))] .= Inf

        # Assign each data point to closest median (track number of changes labels)
        changes = 0
        for i in 1:n
                (~,y_new) = findmin(D[i,:])
                changes += (y_new != y[i])
                y[i] = y_new
        end
```

```
        # Find median of each cluster
        for c in 1:k
                W[c,:] = median(X[y.==c,:],dims=1)
        end
end

function predict(Xhat)
        (t,d) = size(Xhat)

        D = distancesAbsolute(Xhat,W)

        yhat = zeros(Int64,t)
        for i in 1:t
                (~,yhat[i]) = findmin(D[i,:])
        end
        return yhat
end

return PartitionModel(predict,y,W)
end

# Helper:
function distancesAbsolute(X1,X2)
        (n,d) = size(X1)
        (t,d2) = size(X2)
        @assert(d==d2)
        absDis = zeros(n,t)
        for i in 1:n
                for j in 1:t
                        absDis[i,j] = norm(X1[i,:]-X2[j,:], 1)
                end
        end
        return absDis
end

function kMediansError(X,y,W)
        (n,d) = size(X)

        f = 0
        for i in 1:n
                for j = 1:d
                        f += abs.(X[i,j] - W[y[i],j])
                end
        end
        return f
end
```

Answer: 4. we had a plot of the minimum error found across 50 random initialization, as a function of k, taking k from 1 to 10.
From the graph, we choose k=3 or 4. Note: X-axis: k; Y-axis: minimum error



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. 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).

Answer:
1. radius: 3; minPts: 3;
2. radius: 4; minPts: 3;
3. radius: 13; minPts: 3;
4. radius: 30; minPts: 3

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.

Answer:
1. $||Xw - y||_1 + \lambda||w||_1$
2. $(Xw - y)^T V(Xw - y) + ||\Lambda^{1/2} w||^2$
3. $||Xw - y||_\infty^2 + \frac{1}{2}||\Lambda w||_1$

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 v 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 $(X^T X + \lambda I)w$.

Answer:
1. $f(w) = \frac{1}{2}||w||^2 - w^T u + \frac{1}{2}||u||^2$
$\nabla f(w) = w - u$, set it to 0 $\Rightarrow$ $w = u$
2. $f(w) = \frac{1}{2}(Xw - y)^T V(Xw - y) + \lambda w^T u$
$\nabla f(w) = X^T VXw - X^T Vy + \lambda u$, we need to solve the linear equation $(X^T VX)w = X^T Vy - \lambda u$
3. $\nabla f(w) = X^T Xw - X^T y + \lambda w - \lambda w^0$, we need to solve the linear equation $(X^T X + \lambda)w = X^T y + \lambda w^0$

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. $f(w) = \alpha w^2 - \beta w + \gamma$ 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. $f(w) = \sum_{i=1}^{n} [\max\{0, |w^T x_i - y_i|\} - \epsilon] + \frac{\lambda}{2}\|w\|_2^2$ with $w \in \mathbb{R}^d, \epsilon \geq 0, \lambda \geq 0$ (support vector regression).

General hint: for the first two you can check that the second derivative is non-negative since they are one-dimensional. For the last 3 you'll have to use some of the results regarding how combining convex functions can yield convex functions which can be found in the lecture slides.

Hint for part 4 (logistic regression): 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, you can reduce the problem to showing that $\log(1 + \exp(z))$ is convex, which can be done by computing the second derivative. It may simplify matters to note that $\frac{\exp(z)}{1+\exp(z)} = \frac{1}{1+\exp(-z)}$.

Answer:
1. $f'(w) = 2\alpha w - \beta$; $f''(w) = 2\alpha$, since $\alpha >= 0$, so $f''(w) >= 0$, it's convex.

2. $f'(w) = log(w) + 1$, $f''(w) = 1/w$, since $w > 0$, so $f''(w) > 0$, it's convex.

3. For $||Xw-y||^2$, L2-Norms are convex functions and $Xw-y$ is a linear function, we know that composition of a convex function with a linear function is also convex, so $||Xw-y||^2$ is convex. For $\lambda||w||_1$, we know that a convex function multiplied by a non-negative constant is also convex, so that $\lambda||w||_1$ is convex. Last $f(w)$ is the sum of two convex functions, and sum of convex functions is also convex function, hence f(w) is convex.

4. reduced the problem into showing $log(1 + exp(z))$ is convex:

$f'(z) = exp(z)/(1 + exp(z)) = 1/(1 + exp(-z))$,

$f''(z) = exp(-z)/(1 + exp(-z)^2) >= 0$, so that $log(1 + exp(z))$ is convex; since we know that $-y_i w^T x_i$ is a linear function, and the composition of a linear function with a convex function is also convex function, hence $f(w)$ is convex.

5. For the first part, $w^T x_i - y_i$ is a linear function and $f(x) = ||x||$ is convex, $w^T x_i - y_i$ is the composition of a convex function with a linear function, so that it's convex. Also 0 and ϵ are convex since they are constant (second derivative is 0). Since the max of convex functions is also convex, then $max0, |w^T x_i - y_i|$ is convex. Since the sum of convex functions is also convex, so that $\sum_{i=1}^{n}[\max\{0, |w^T x_i - y_i|\} - \epsilon]$ is also convex. For the second part, we know that L2-Norms function is convex, so $\frac{\lambda}{2}\|w\|_2^2$ is convex because it is a convex function multiplied by a non-negative constant. Last, the sum of convex functions is also convex, hence f(w) is convex.

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

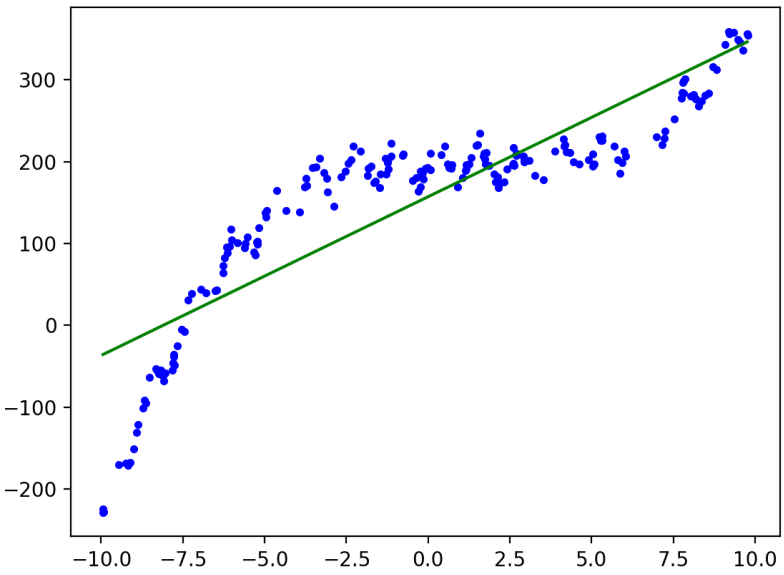
$$y_i = w^T x_i.$$

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.

Answer: updated training error: 3551.346; updated test error: 3393.869

```
function leastSquaresBias(X,y)
        (n,d) = size(X)
        w0 = ones(n,1)
        Xtemp = hcat(w0, X)
        w = (Xtemp'Xtemp)\(Xtemp'y)

        function predict(Xhat)
                (t,) = size(Xhat)
                w0 = ones(t,1)
                yhat = hcat(w0, Xhat)*w
                return yhat
        end

        return GenericModel(predict)
end
```



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 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.

Answer: training error goes down as p increases, test error goes down first and then goes up a bit as p increases, because of overfitting problem as the model gets more complicated.

```
function leastSquaresBasis(X,y,p)

        function polyBasis(X,p)
                (n,) = size(X)
                if p == 0
                        return ones(n,1)
                end
                if p == 1
                        return hcat(ones(n,1), X)
                end
                Xtemp = hcat(ones(n,1), X)
                for i in 2:p
                        Xtemp2 = X
                        Xtemp = hcat(Xtemp, Xtemp2.^i)
                end
                return Xtemp
        end

        Xtemp = polyBasis(X,p)
        w = (Xtemp'Xtemp)\(Xtemp'y)

        function predict(Xhat)
                Xhattemp = polyBasis(Xhat,p)
                return Xhattemp*w
        end

        return GenericModel(predict)
end
```

p	training error	test error
0	15480.520	14390.763
1	3551.346	3393.869
2	2167.992	2480.725
3	252.046	242.805
4	251.462	242.126
5	251.143	239.545
6	248.583	246.005
7	247.011	242.888
8	241.306	245.966
9	235.762	259.296
10	235.074	256.300

3.3 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.

Answer:
basis: $(1\ X\ X^2\ X^3\ sin(4X)\ sin(5X)\ sin(6X)\ sin(7X))$
training error: 46.778;
test error: 49.785.

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,

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

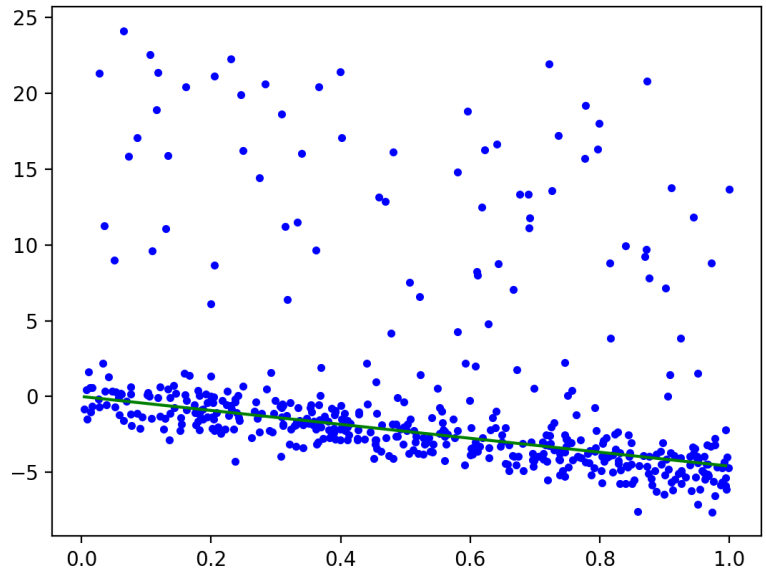
In this formulation, the model focuses on making the error small for examples i where v_i is high. Similarly, if v_i is low then the model allows a larger error.

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.

Answer:

```
function weightedLeastSquare(X,y,v)
        w = (X'*v*X)\(X'*v*y)
```

```
        predict(Xhat) = Xhat*w

        return GenericModel(predict)
end

# in example_outliers, with setted v, we can get new model
v = Diagonal(vcat(fill(1,400), fill(0.1,100)))
model = weightedLeastSquare(X,y,v)
```



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.

Answer:

$$h'(r_i) = \begin{cases} r_i & \text{for } |r_i| \leq \epsilon \\ \epsilon sgn r_i & \text{for } |r_i| > \epsilon \end{cases}.$$

then we have

$$h'(r_i) = \begin{cases} -\epsilon & \text{for } r_i < -\epsilon \\ r_i & \text{for } -\epsilon \leq r_i \leq \epsilon \\ \epsilon & \text{for } r_i > \epsilon \end{cases}.$$

so $\nabla f(w)$, we have

$$\nabla f(w) = \sum_{i=1}^{n} x_i h'(w^T x_i - y_i),$$

where

$$h'(r_i) = \begin{cases} -\epsilon & \text{for } r_i < -\epsilon \\ r_i & \text{for } -\epsilon \leq r_i \leq \epsilon \\ \epsilon & \text{for } r_i > \epsilon \end{cases}.$$

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.

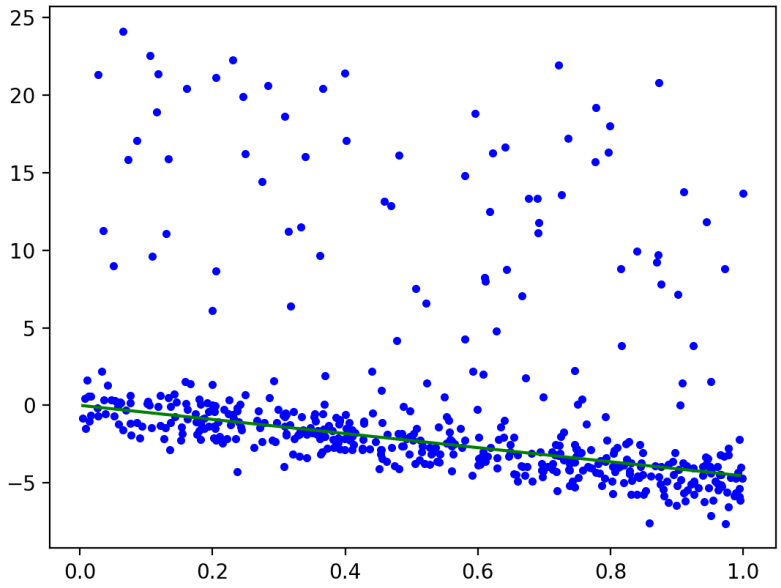
The typical input to a gradient method is a function that, given w, returns $f(w)$ and $\nabla f(w)$. See *funObj* in the *leastSquaresGradient* function for an example. Note that *leastSquaresGradient* also has a numerical check that the gradient code is approximately correct, since implementing gradients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

```
function robustRegressionObj(w,X,y)
        (n,d) = size(X)
        f = 0
        for i in 1:n
                temp = (X[i,:]*w)[1] - y[i]
                if (abs(temp) <= 1)
                        f += 0.5*((temp)^2)
                else
                        f += (abs(temp)-0.5)
                end
        end
        g = zeros(size(w))
        for i in 1:n
                temp = (X[i,:]*w)[1] - y[i]
                if (temp < -1)
```

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

```
                g .-= X[i,:]
            elseif (temp > 1)
                g .+= X[i,:]
            else
                g .+= X[i,:] * temp
            end
        end
        return (f,g)
end
```



5 Very-Short Answer Questions

1. Describe a dataset with k clusters where k-means cannot find the true clusters.

   Answer: a circle cluster surround by a ring of points, where the ring points are kind of far away from the circle. it's non-convex.

2. Why do we need random restarts for k-means but not for density-based clustering?

   Answer: since K-means need to assign points to its cloest mean, while DESCAN doesn't need to do that.

3. Can hierarchical clustering find non-convex clusters?

   Answer: Yes

4. For each outlier detection method below, list an example method and a problem with identifying outliers using this method:

   - Model-based outlier detection.

     Answer: z-score, problem: fail for multi-modal, outlier will be in middle of different modals

   - Graphical-based outlier detection.

Answer: scatterplot, problem: can only see two dimensions at one time, fail if you wanna see more dimensions at a time.

- Supervised outlier detection.

  Answer: decision tree, problem: may overfit when model gets more complcated.

5. In linear regression, why do we compute the squared error $(y_i - \hat{y}_i)^2$ and not test the equality $(y_i = \hat{y}_i)$?

   Answer: it is unlikely to find a line where $(\hat{y}_i = y_i)$ exactly for many points, may due to noise, relationship not quite being linear.

6. Describe a simple 2-feature $(d = 2)$ case where the least squares estimate would not be unique.

   Answer: two features are collinear, then they are identical to all examples, if we increase weight for one feature and decrease weight for the other feature, then we will get another solution, which makes LS estimate not unique.

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?

   Answer: to make y intercept not forced at 0; it has nothing to od with decision tree.

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?

   Answer: when the number of features is large (d), gradient descent will be faster.

9. If a function is convex, what does that say about stationary points of the function? Does convexity imply that a stationary points exists?

   Answer: all stationary points are global minimum; nope, it's possible that convex functions having no stationary point.

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?

    Answer: because usually the equation is non-differential at their minimum value point, so there's no normal equations to solve; use smooth approximation, we can apply gradient descent to find the global mimimum.

11. What is the problem with having too small of a learning rate in gradient descent?

    Answer: the convergence will be too slow.

12. What is the problem with having too large of a learning rate in gradient descent?

    Answer: the optimization will not be able to converge/diverge.