CPSC340A2

Qinglan Huang n6v9a Liu Yang v4d9

November 25, 2017

1 Random Forests

1.1 Implementation

1. If we change the depth parameters to Inf the training process will terminate not only when depth <1 but also when we reach baseSplit where the remaining objection of each leaf<1.

2.Because each random tree randomly pick only sqrt(d) features to train, it does not cover all data set and will never overfit no matter how deep the tree is.

3. check code at decisionTree.jl

```
function randomForest(X,y,depth,nTrees)
    subModels = Array{GenericModel}(nTrees)

    for i in 1: nTrees
        subModels[i] = randomTree(X,y,depth)
    end
    return subModels

end



function predictA(subModels, Xhat)
    (t,d) = size(Xhat)
    yhat = zeros(t)
    y = zeros(t,nTrees)

    for i in 1: nTrees
        model = subModels[i]

        y[:,i] = model.predict(Xhat)
    end
    for i in 1:t
        yhat[i] = mode(y[i,:])
    end

    return yhat
end
```

4. Test result is:

Train Error with depth-Inf decision tree:0.000
Test Error with depth-Inf decision tree:0.367
Train Error with depth-5 decision tree:0.311
Test Error with depth-5 decision tree:0.504
Train Error with depth-Inf decision forest:0.000
Test Error with depth-Inf decision forest:0.170

A decision tree cannot avoid overfitting, therefore it has a training error of 0.However, there is a relatively large test error. A single random tree, since only pick part of features, technically does not finish training process and has a bad performance on both train error and test error. But when we increase the number of random tree (random forest), we can avoid overfitting and cover all d features. Therefore the random forest has a low train error and test error as expected

1.2 Very-Short Answer Questions

1. It is going to be expensive because of the runtime.

2. b c f
b. Decreasing the depth is going to decrease the probability of the overfitting.
c. larger amount of data can improve accuracy
f.When we have many strong features, random tree can be correlated, the potential of overfitting increases .

3.Adding more dataset for random forests, and building more random tree, control depth of the each random tree in order to control number of feature for clear overall accuracy, avioding strong feature.

2 K-Means Clustering

2.1 Selecting among k-means Initialization

1. check code at kMeans.jl

```
function kMeansError(X,y,W)
    (n,d) = size(X)
    (k,d2) = size(W)
    assert(d == d2)
    temp =0
    for i in 1:n
        for j in 1:d
            temp += (X[i,j] - W[Int(y[i]),j]).^2
        end
    end
    return temp
end
```

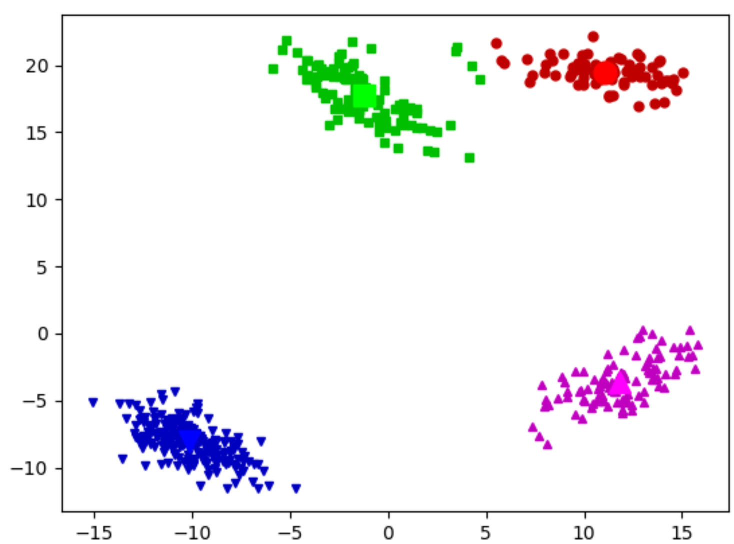
2.
kMeansError monotonically decreases until stabilization.

```
julia> include("example_Kmeans.jl")
Running k-means, changes = 500
Running k-means, kMeansError= 9609.749047512
Running k-means, changes = 32
Running k-means, kMeansError= 9466.879704072
Running k-means, changes = 20
Running k-means, kMeansError= 9404.445409588
Running k-means, changes = 11
Running k-means, kMeansError= 9386.083251910
Running k-means, changes = 10
Running k-means, kMeansError= 9376.066372056
Running k-means, changes = 5
Running k-means, kMeansError= 9371.944601192
Running k-means, changes = 3
Running k-means, kMeansError= 9370.487700096
Running k-means, changes = 0
Running k-means, kMeansError= 9370.487700096
```

error.png

3. If we just running k-means once, we potentially get a terrible clustering, with kMeansError stabilizes at 9370 (worser case is possible).However, when we run k-means 50 times, we can ensure a relatively small kMeansError about 3000, a relatively meaningful clustering.

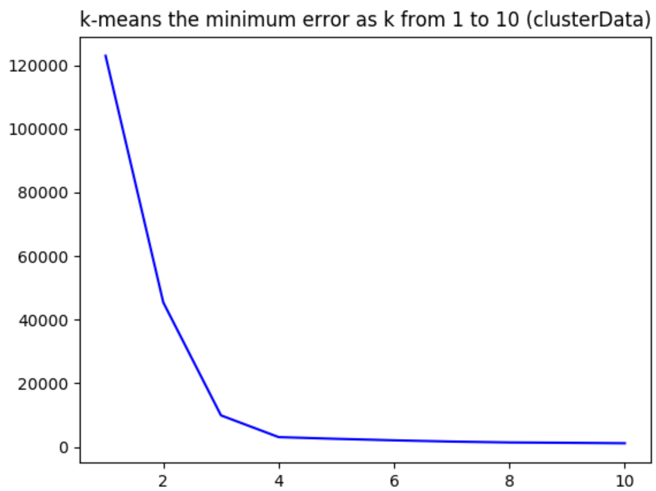
Here is the “best” clustering we obtained after running k-means 50 times.



2.2 Selecting k in k-means

1. Because the kMeansError decreases strictly monodically along with k increasing, kMeansError will lead us to choose k as big as the size of dataset but sacrificing running complexity (When k =n, the kMeansError =0)

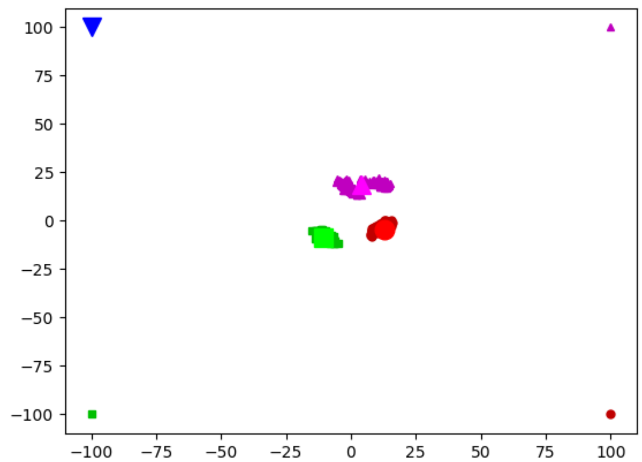
2.Violate golden rule.
3. The plot graph is:



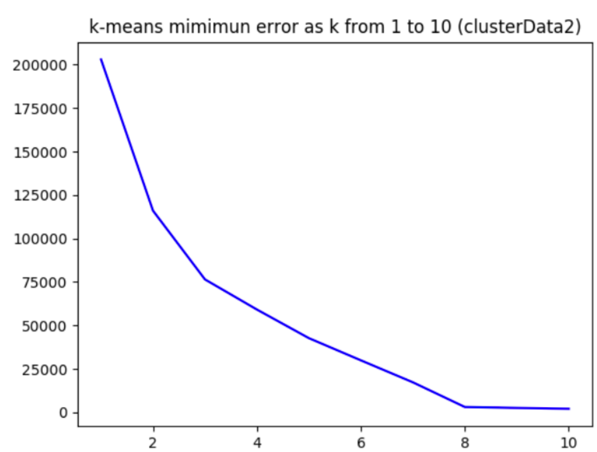
check code at example_kMeans2.jl
4. I would choose 4 in this case. There is tradeoff between run time and accuracy. Although, the sharpest "elbow" occurs at k =2, the minimum error a t k =2 is too high. But when we sacrifice runtime a little bit, I say a little bit because increasing k from 2 to 4 does not make a large difference on runtime ,but almost decreasing minimum error by 8 times, therefore there is huge difference

2.3 K-Medians

1.



check code at example_kMedians.jl
The clustering2Dplot function gives a bad clustering because it cannot detect outliers, so has a high error.
2. I would choose k = 8 in this case.As we can see the picture down blew that. Also by choosing k =8 we can assign 4 outliers to a separate cluster ensuring the main data is not influenced by them.

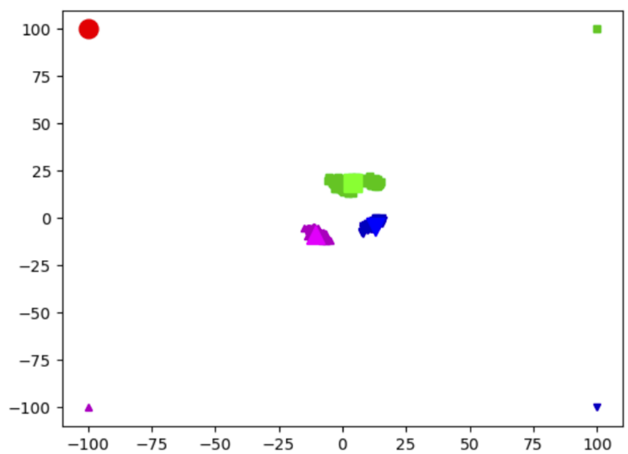


3. check code at kMedians.j or here:

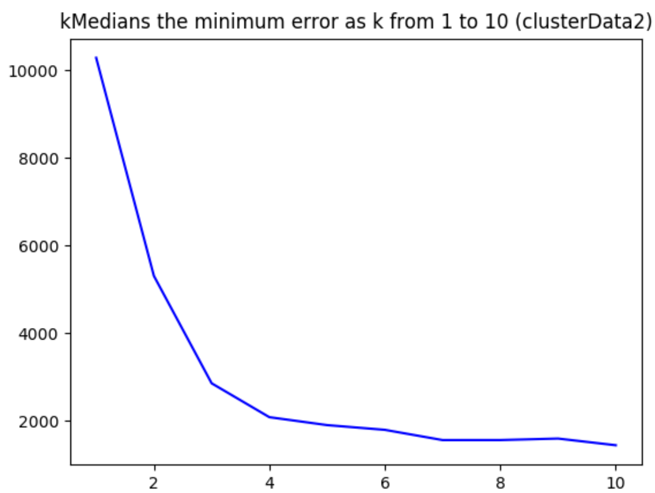
```julia
include("misc.jl")
include("clustering2Dplot.jl")

type PartitionModel
    predict # Function for clustering new points
    y # Cluster assignments
    W # Prototype points
end

function kMedian(X,k;doPlot=false)
    # K-means clustering
    (n,d) = size(X)

    # Choos random points to initialize means
    W = zeros(k,d)
    perm = randperm(n)

    for c = 1:k
        # initially pick points from X
        W[c,:] = X[perm[c],:]
    end

    # Initialize cluster assignment vector
    y = zeros(n)
    changes = n
    D=zeros(n,k)
    while changes != 0
        # Compute L1 norm distance between each point and each median
        for i in 1:k
            for j in 1:n
                D[j,i] = sum(abs.(X[j,:]-W[i,:]))
            end
        end

        # Assign each data point to closest mean (track number of changes labels)
```

```
        # Assign each data point to closest mean (track number of changes labels)
        changes = 0
        for i in 1:n
            #y_new is the nearest cluster
            (~,y_new) = findmin(D[i,:])
            changes += (y_new != y[i])
            #y records corresponding cluster of every datapoint
            y[i] = y_new
        end

        # Optionally visualize the algorithm steps
        if doPlot && d == 2
            clustering2Dplot(X,y,W)
            sleep(.1)
        end

        # Find median of each cluster

        for c in 1:k
            W[c,:] = median(X[y.==c,:],1)
        end

        # Optionally visualize the algorithm steps
        if doPlot && d == 2
            clustering2Dplot(X,y,W)
            sleep(.1)
        end

        #@printf("Running k-means, changes = %d\n",changes)
        #@printf("Running k-means, kMeansError= %.3f\n",kMeansError(X,y,W))
    end

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

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

        for i in 1:k
            for j in 1:n
                D[j,i] = sum(abs.(X[j,:]-W[i,:]))
            end
        end

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

    return PartitionModel(predict,y,W)
end


function kMeansError(X,y,W)
    (n,d) = size(X)
    (k,d2) = size(W)
    assert(d == d2)
    temp =0
    for i in 1:n
        for j in 1:d
            #temp += abs.(X[i,j] - W[Int(y[i]),j])
            temp += (X[i,j] - W[Int(y[i]),j]).^2
        end
    end
    return temp
end
```



4.



The graph has the sharpest change in slope between k =2 and k = 4, so the appropriate value for k could be 3.
It would give a relatively good clustering but cannot detect outliers

2.4 Very-Short Answer Questions

1.Nope, result depends on initial clusters
2.k=n, where n is the data size. when K = n, the distance is 0 no matter how.

3. clusters with different hierarchies

3 More Unsupervised Learning

3.1 Density-Based Clustering

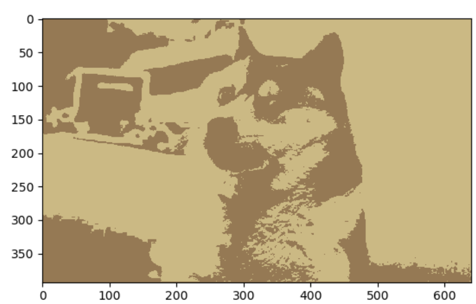
1.
radius = 2, minPoints = 2
2.
radius = 4, minPoints = 2
3.
radius = 15, minPoints =2
4.
radius = 20, minPoints =2

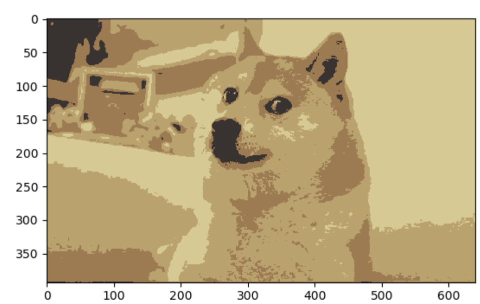
3.2 Vector Quantization

1. see at quantizeImage.jl and also can check here:

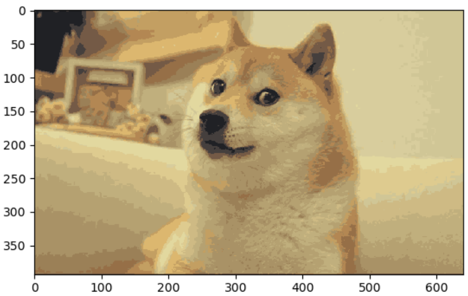
```julia
using PyPlot
include("kMeans.jl")
dog = imread("dog.png")

function quantizeImage(img,b)
    (nRows,nCols,a) = size(img)
    assert(a .== 3)
    ob = nRows * nCols
    X=reshape(img,ob, 3)

    model = kMeans(X,2.^b ,doPlot=false)
    y=model.predict(X)
    return deQuantizeImage(y,model.W,nRows,nCols)
end



function deQuantizeImage(y,W,nRows,nCols)
    y = reshape(y, nRows,nCols)
    biu = zeros(nRows,nCols,3)
    for i in 1:nRows
        for j in 1:nCols
            biu[i,j,:] = W[y[i,j],:]
        end
    end
    return biu
end

imshow(quantizeImage(dog,6))
```

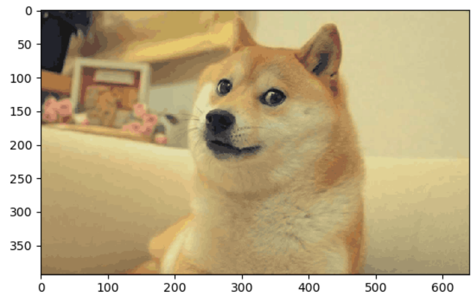
2.
value =1 and value = 2





value = 4 and value =6





3.3 Very-Short Answer Questions

1.The cluster will be decided mainly by the distance of larger scale.
2.Advantage: We can label outliers in supervised model. Therefore it can predict similar outliers very well.
Disadvantage:If there is outliers in dataset which are different from training set, there it cannot detect this outlier and will behave horribly.
3. The cost of finding all rows i in X is going to be like: O(n)
The cost of runtime for finding all rows after given us a hash table that assigns rows of X to keys that divide the space into a 2D grid of squares with radius r, and we are using k to demote the maximum number of pointed to hashed to same key value: O(k)

4 Matrix Notation and Linear Regression

4.1 Converting to Matrix/Vector/Norm Notation

1.$\sum_{i=1}^{n} \|w^T x_i - y_i\| = \|Xw - y\|_1$

2.$\max_{1\leq i\leq n} \|w^T x_i - y_i\| + \frac{\lambda}{2} \sum_{j=1}^{n} w_j^2 = \|Xw - y\|_{\infty} + \frac{\lambda}{2} \|w^2\|$

3.$\sum_{i=1}^{n} v_i (w^T x_i - y_i)^2 + \lambda \sum_{j=1}^{d} \|wj\| = v_i \|w^T x_i - y_i\|^2{}_2 + \lambda \|w_j\|_1$

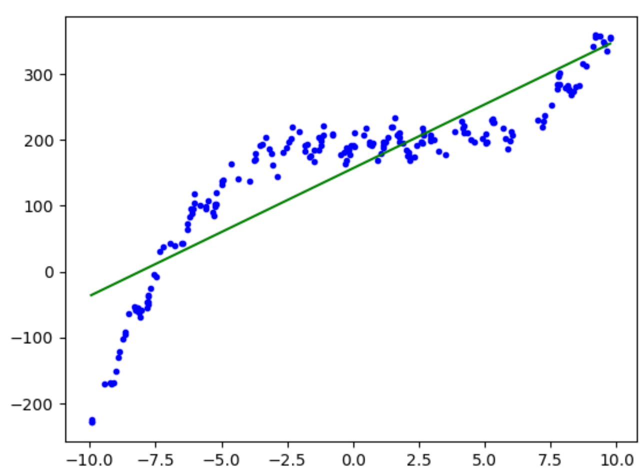
4.2 Minimizing Quadratic Functions as Linear Systems

1.
f(w) $= \frac{1}{2}\|w - u\|^2$
f(w)$= \frac{1}{2}\|w\|^2 + \frac{1}{2}\|u\|^2$- $w^T u$

$\bigtriangledown f(w) = w - u$
Thus, w = u
2.
f(w) = $\frac{1}{2}\|w\|^2$+w^TX^Ty
f(w) = $\frac{1}{2}w^TIw$+w^TX^Ty
$\bigtriangledown f(w)$=$w + X^Ty$
Thus, w = $-X^Ty$
3.
f(w) =$\frac{1}{2}\|Xw - y\|^2$ +$\frac{1}{2}w^T\Lambda w$
f(w) =$\frac{1}{2}\|X\|^2\|w\|^2$+$\frac{1}{2}\|y\|^2$-$w^TX^Ty$+$\frac{\Lambda}{2}\|w\|^2$
$\bigtriangledown f(w)$ =$\|X\|^2w$-X^Ty+Λw =0
4.
f(w) = $\frac{1}{2}\sum_{i=1}^{n} v_i(w^Tx_i - y_i)^2$
f(w) = $\frac{1}{2}(Xw - y)^Tv_i(Xw - y)$
f(w) =$\frac{1}{2}w^TX^TvXw$ - w^Tx^Tvy +$\frac{1}{2}y^Tvy$
$\bigtriangledown f(w)$ =$X^TvXw - X^Tvy$
Thus, $X^TvXw = X^Tvy$

4.3 Linear Regresion with Bias Variable

The graph plot is going to be like



The function code is:

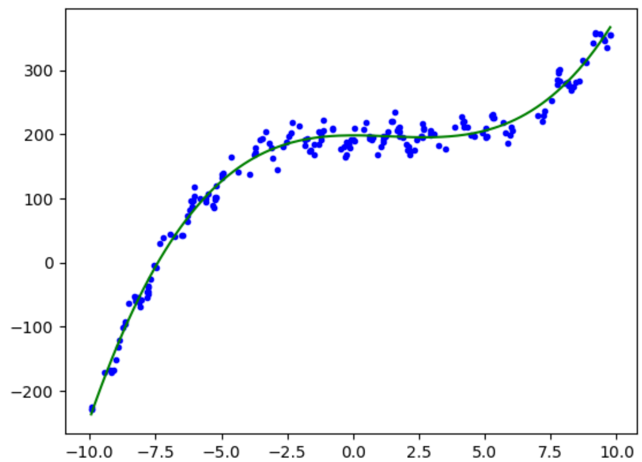
```
function leastSquaresBias(X,y)

    # Find regression weights minimizing squared error
    (n,d) = size(X)
    #w = (X'*X)\(X'*y)

    X_0 = ones(n)
    biaX = hcat(X_0 , X)
    biaw = (biaX'*biaX)\(biaX'*y)

    # Make linear prediction function

    function predict(Xpredict)
        (n2,) = size(Xpredict)
        X_1 = ones(n2)
        Xpredict = hcat(X_1,Xpredict)
    return Xpredict * biaw
end


    # Return model
    return GenericModel(predict)
end
```

The test result is:

```
julia> include("example_nonLinear.jl")
Squared train Error with least squares: 3551.346
Squared test Error with least squares: 3393.869
1-element Array{Any,1}:
 PyObject <matplotlib.lines.Line2D object at 0x12d9e7ed0>
```

4.4 Linear Regression with Polynomial Basis

The plot graph is:



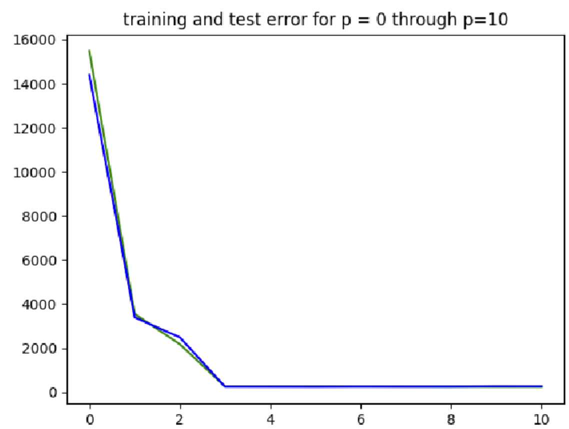
The code is:

```
function polyBasis(x,p)
    (n,) = size(x)
    Z = ones(n,p+1)
    for i in 2:(p+1)
        Z[:,i] = x.^(i-1)
    end
    return Z
end


function leastSquaresBasis(x,y,p)
    Z = polyBasis(x,p)
    w = (Z' * Z) \ (Z' * y)

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

    return GenericModel(predict)
end
```

The test is:

```julia
# Fit a least squares model
include("leastSquares.jl")

trainError =[]
testError = []

for p in 0:10
  model = leastSquaresBasis(X,y,p)

  # Evaluate training error
  yhat = model.predict(X)
  push!(trainError, mean((yhat - y).^2))
  #@printf("Squared train Error with least squares Basis: %.3f\n",trainError)

  # Evaluate test error
  yhat = model.predict(Xtest)
  push!(testError, mean((yhat - ytest).^2))
  #@printf("Squared test Error with least squares Basis: %.3f\n",mean((yhat - ytest).^2))
end

using PyPlot
figure()
title("training and test error for p = 0 through p=10")
plot(0:10,trainError,"g")
plot(0:10,testError,"b")


model = leastSquaresBasis(X,y,10)
# Plot model
figure()
plot(X,y,"b.")
Xhat = minimum(X):.1:maximum(X)
yhat = model.predict(Xhat)
plot(Xhat,yhat,"g")
```

The test graph from p=0 to p=10 is:



As in the graph that we can see when p value increase, test error and train error decreasing and getting more stable and we also notice that test error and train error is almost same.

4.5 Manual Search for Optimal Basis

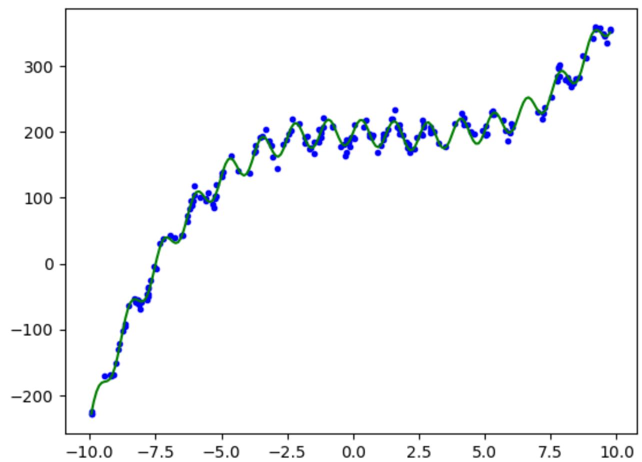
The code is like:

```
function manualSearch(X,p)
    (n,) = size(X)
    Z = ones(n,1+p*2)
    for i in 1:p
        Z[:,2*i] = X.^(i)
        Z[:,2*i+1] =sin.(i*X)
    end
    return Z
end

function  leastSquaresManual(X,y,p)
    Z = manualSearch(X,p)
    w = (Z' * Z) \ (Z' * y)
    function predict(Xhat)
        bang = manualSearch(Xhat,p)
        return bang * w
     end

    return GenericModel(predict)
end
```

The result is:

```
julia> include("example_manualSearch.jl")
Squared train Error with least squares Basis: 46.077
Squared test Error with least squares Basis: 50.865
1-element Array{Any,1}:
 PyObject <matplotlib.lines.Line2D object at 0x14e6af050>
```

The plot graph is:



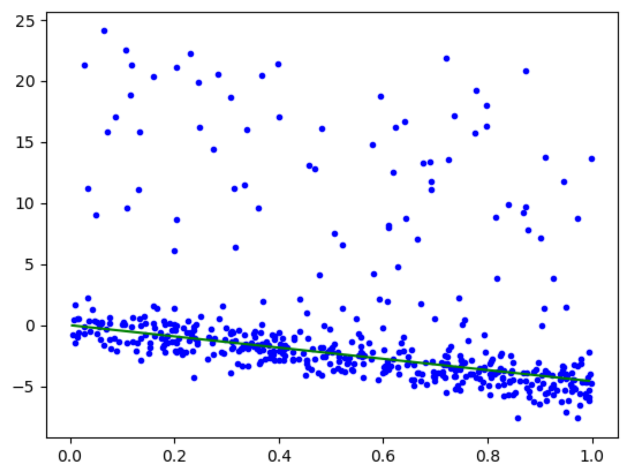
4.6 Very-Short Answer Questions

1. Because y_i is continuous variable in this case
2.When there is a perfect linear relationship between two features, then least squares estimate is not going to be unique.
3.construct Z:$\mathcal{O}(np)$
calculate model(w):$n * p^2 + p^3$
predict: construct +predict: np +np
overall=$\mathcal{O}(np^2 + p^3)$
4. When data hase piecewise linear relationship

5 Robust Regression and Gradient Descent

5.1 Weighted Least Squares in One Dimension

The plot graph is:



The code is :

```
function weightedLeastSquares(X,y,v)

    w = (X'*v*X)\(X'*v*y)
    predict(Xhat) = Xhat*w

    return GenericModel(predict)
end
```

The test is like:

```
# Load X and y variable
using JLD
data = load("outliersData.jld")
(X,y,Xtest,ytest) = (data["X"],data["y"],data["Xtest"],data["ytest"])

# Fit a least squares model
include("leastSquares.jl")
#model = weightedLeastSquares(X,y,0)
v1 = ones(400)
v2 = ones(100)

V = vcat(v1,0.1*v2)
V = Diagonal(V)
model = weightedLeastSquares(X,y,V)
# Evaluate training error
yhat = model.predict(X)
trainError = mean((yhat - y).^2)
@printf("Squared train Error with least squares: %.3f\n",trainError)

# Evaluate test error
yhat = model.predict(Xtest)
testError = mean((yhat - ytest).^2)
@printf("Squared test Error with least squares: %.3f\n",testError)
# Plot model
using PyPlot
figure()
plot(X,y,"b.")
Xhat = minimum(X):.01:maximum(X)
yhat = model.predict(Xhat)
plot(Xhat,yhat,"g")
```

5.2 Smooth Approximation to the L1-Norm

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

we make value r like:

$r_i = w^T x_i - y_i$

First we convert function to matrix notation.

$f(w) = log(exp(Xw - y) + ex(y - Xw))$

$\frac{d}{dw} f(w) = \frac{\frac{d}{dw}(exp(Xw-y)+exp(y-Xw))}{exp(Xw-y)+exp(y-Xw)}$

$\frac{d}{dw} f(w) = \frac{(\frac{d}{dw}(Xw-y))*exp(Xw-y)+\frac{d}{dw}(Xw-y))*exp(y-Xw)}{exp(Xw-y)+exp(y-Xw)}$

$\frac{d}{dw} f(w) = \frac{X^T*exp(Xw-y)-X^T*exp(y-Xw)}{exp(Xw-y)+exp(y-Xw)}$

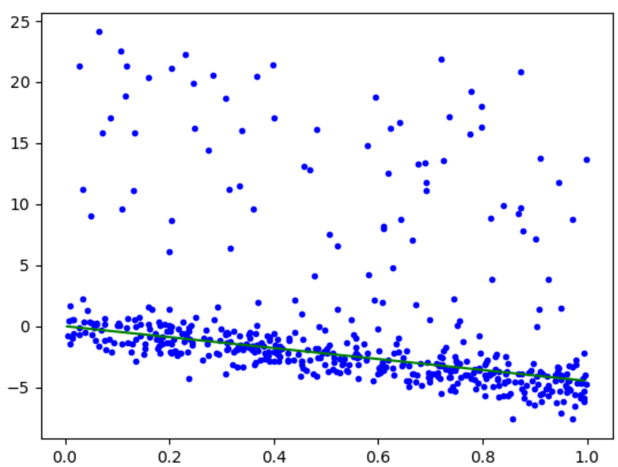
$\frac{d}{dw} f(w) = X^T \frac{exp(Xw-y)-exp(y-Xw)}{exp(Xw-y)+exp(y-Xw)}$

5.3 Robust Regression

The code is like:

```
function robustRegressionObj(w,X,y)

    a = exp.(X*w -y)
    b = exp.(y-X*w)

    f = sum(log.(a+b))

    r = (a-b) ./ (a+b)

    g = X' * r

    return (f,g)
end
```

The dataset in graph is like:



5.4 Very Short Answer Questions

1. graph-based and distance-based Cluster-based cannot work because these outliers are disperse.

Model-based cannot work because we have too many outliers here and strongly influence our mean and standard deviatiol therefore the model we obtained is not accurate.
Graph-based and distance-based can detect outliers more effective in this dataset.
2.If there are lots of the features, then we need to use gradient descent to solve the problem.
3. Gradient is problem prone. Because model can be non-invertible or can not be differentible.