HW 5, Submitted by: Aman Jain (aj644)

Answer 1

For each, state the input space X, the output space Y, describe the loss function and regularizer you would use for this problem (and, optionally, any feature transforma- tions), and explain why your choice of loss function and regularizer make sense for this problem.

Example 1: Logistics Loss

Problem description: Predicting risk category of a person (High, Medium, Low)

Input space X: Financial data of customer (income, credit score etc), social data (area of living, soci0 economic class), personal facotrs (age, working years etc), social media attributes etc.
Output space Y: Ordinal vector (High, Medium, Low)
Loss function: Logistics loss
Regularizer: NoRegularizer
Feature Transformation (Optional): Transforming output space to many hot vectors.

Explanation: Hinge loss is used for classification problems where output space is classes. We are more interested in if correct class is assigned or not - rather than margin by which

Example 2: Quantile Loss

Problem description: Any problem where measurement accuracy is uncertain,
OR we are more interested in some quantiles rather than mean
OR In addition to this, Quantile loss also useful in cases of multimodal input space (when we do not know it before hand).

Input space X: distance of star from earth, red shift in light, intensity of light etc.
Output space Y: Age of life of star
Loss function: Quantile loss
Regularizer: NoRegularizer
Feature Transformation (Optional):

Explanation: each of input space needs super accurate measurements. It may not be accurate all the time. While modelling we need to budget in for measurement errors. Also we may want to bias our prediction to certain section of the data- qauntile loss helps do that.

Answer 2

In [1]:
```
#import Pkg; Pkg.add("LowRankModels")
```

In [3]:
```
using Plots, Random, LinearAlgebra, Statistics, SparseArrays, DataFrames
include("proxgrad.jl")
```

Out[3]:
```
proxgrad_const
```

Solving ERM problems

The file `proxgrad.jl` contains code for solving regularized empirical risk minimization (ERM) problems. It provides the optimization function `proxgrad` together with a large number of predefined loss functions and regularizers.

The function `proxgrad` solves regularized ERM problems of the form

$$\text{minimize} \quad \sum_{i=1}^{n} \ell(y_i, w^T x_i) + r(w).$$

It solves these with the proximal gradient method, which we will learn shortly.

You can select from a range of losses. For real valued y, try:

- quadratic loss - `QuadLoss()`
- ℓ_1 loss - `L1Loss()`
- quantile loss (for α quantile) - `QuantileLoss(α)`

For Boolean y, try

- hinge loss - `HingeLoss()`
- logistic loss - `LogisticLoss()`
- weighted hinge loss - `WeightedHingeLoss()`

For nominal y, try

- multinomial loss - `MultinomialLoss()`
- one vs all loss - `OvALoss()`
  - (by default, it uses the logistic loss for the underlying binary classifier)

For ordinal y, try

- ordinal hinge loss - `OrdinalHingeLoss()`
- bigger vs smaller loss - `BvSLoss()`
  - (by default, it uses the logistic loss for the underlying binary classifier)

It also provides a few regularizers, including

- no regularization - `ZeroReg()`
- quadratic regularization - `QuadReg()`
- ℓ_1 regularization - `OneReg()`
- nonnegative constraint - `NonNegConstraint()`

Below, we provide some examples for how to use the proxgrad function to fit regularized ERM problems.

generate random data set

First (as usual), we'll generate some random data to try our methods on.

In [4]:
```
Random.seed!(0)
n = 50
d = 10
X = randn(n,d)
w♮ = randn(d)
y = X*w♮ + .1*randn(n);
```

Quadratic loss, quadratic regularizer

$$\text{minimize} \quad \frac{1}{n}||Xw - y||^2 + \lambda||w||^2$$

In [5]:
```
# we form \frac 1 n || · ||^2 by multiplying the QuadLoss() function by 
loss = 1/n*QuadLoss()

# we form λ|| · ||^2 by multiplying the QuadReg() function by λ
λ = .1
reg = λ*QuadReg()

# minimize 1/n ||Xw - y||^2 + λ||w||^2
#w = proxgrad(loss, reg, X, y, maxiters=5, c=.1, stepsize=1, max_inner_i
w = proxgrad(loss, reg, X, y, maxiters=5)

norm(X*w-y) / norm(y)
```

Out[5]: 0.08490911088603888

`maxiters` , the maximum number of iterations, controls how fully we converge. You can try increasing it to see if the error improves.

In [6]:
```
w = proxgrad(loss, reg, X, y, maxiters=10)
norm(X*w-y) / norm(y)
```

Out[6]: 0.06951902408490718

Hinge loss, quadratic regularizer

$$\text{minimize} \quad \frac{1}{n}\sum_{i=1}^{n}(1 - y_i w^T x_i)_+ + \lambda||w||^2$$

In [7]:
```
ybool = Int.(sign.(y)) # form a boolean target

# we form \frac 1 n \sum_{i=1}^n (1 - · )_+ by multiplying the HingeLoss
loss = 1/n*HingeLoss()

# we form λ|| · ||^2 by multiplying the QuadReg() function by λ
λ = .1
reg = λ*QuadReg()

# minimize 1/n \frac 1 n \sum_{i=1}^n (1 - y_i w^T x_i)_+ + λ||w||^2
w = proxgrad(loss, reg, X, ybool, maxiters=10)

# misclassification error
(n - sum(sign.(X*w) .== ybool)) / n
```

Out[7]: 0.06

For nonsmooth problems (like the hinge loss), a smaller stepsize can also help:

In [8]:
```
w = proxgrad(loss, reg, X, ybool, maxiters=10, stepsize=.1)
# misclassification error
(n - sum(sign.(X*w) .== ybool)) / n
```

Out[8]: 0.04

Homework question Q2

Use the proxgrad function to fit the following objective

$$\text{minimize} \quad \frac{1}{n}\sum_{i=1}^{n}\log(1 + \exp(-\text{ybool}_i w^T x_i)) + \lambda||w||^2$$

for $\lambda = .5$

In [9]:
```
ybool = Int.(sign.(y)) # form a boolean target

# we form \frac 1 n \sum_{i=1}^n (1 - · )_+ by multiplying the HingeLoss
loss = 1/n*LogisticLoss()

# we form λ|| · ||^2 by multiplying the QuadReg() function by λ
λ = .5
reg = λ*QuadReg()

# minimize 1/n \frac 1 n \sum_{i=1}^n (1 - y_i w^T x_i)_+ + λ||w||^2
w = proxgrad(loss, reg, X, ybool, maxiters=10)

# misclassification error
(n - sum(sign.(X*w) .== ybool)) / n
```

Out[9]: 0.02

In [10]:
```
w = proxgrad(loss, reg, X, ybool, maxiters=10, stepsize=.1)
# misclassification error
(n - sum(sign.(X*w) .== ybool)) / n
```

Out[10]: 0.02

In [ ]:

Question 3

In [10]:
```
# import Pkg
# for pkg in ["DataFrames", "PyPlot", "ScikitLearn", "LowRankModels"]
#     if !(pkg in keys(Pkg.installed()))
#         Pkg.add(pkg)
#     end
# end
```

In [148]:
```
birth = CSV.read("birthSample.txt")
```

Out[148]: 9,322 rows × 4 columns

	MaritalStatus	PrenatalCare	Male	Weight
	Int64	Int64	Int64	Int64
1	0	1	0	3326
2	1	0	0	1340
3	0	0	0	3033

4	1	1	1	3884
5	0	0	0	3108
6	1	1	1	3912
7	1	1	1	2546
8	1	1	1	4545
9	0	1	0	3402
10	1	1	1	3884
11	0	1	1	3232
12	0	1	0	4000
13	0	0	1	2790
14	1	1	1	4139
15	0	0	1	3374
16	0	0	0	2778
17	1	1	1	4082
18	1	1	0	3751
19	1	0	1	3388
20	1	1	1	3480
21	1	1	1	3118
22	1	1	1	3130
23	0	0	1	3758
24	1	1	0	3515
25	1	1	1	3345
26	0	1	1	3544
27	0	0	1	3289
28	0	1	0	2777
29	1	1	0	3515
30	1	1	0	3612
⋮	⋮	⋮	⋮	⋮

a) Fit an ordinary least squares regression to the data. Interpret the coefficients that you find

In [149]:
```
X = convert(Matrix, birth[:,1:3])
y = birth[:,end]
Xoffset = [X ones(length(y))]
```

Out[149]:
```
9322×4 Array{Float64,2}:
 0.0  1.0  0.0  1.0
 1.0  0.0  0.0  1.0
 0.0  0.0  0.0  1.0
 1.0  1.0  1.0  1.0
 0.0  0.0  0.0  1.0
 1.0  1.0  1.0  1.0
 1.0  1.0  1.0  1.0
 1.0  1.0  1.0  1.0
 0.0  1.0  0.0  1.0
 1.0  1.0  1.0  1.0
 0.0  1.0  1.0  1.0
 0.0  1.0  0.0  1.0
 0.0  0.0  1.0  1.0
 ⋮
 1.0  1.0  1.0  1.0
 1.0  1.0  1.0  1.0
 0.0  1.0  1.0  1.0
 0.0  1.0  1.0  1.0
 1.0  1.0  0.0  1.0
 0.0  0.0  0.0  1.0
 1.0  1.0  0.0  1.0
 1.0  1.0  0.0  1.0
 0.0  1.0  1.0  1.0
 0.0  1.0  0.0  1.0
 1.0  1.0  1.0  1.0
 0.0  1.0  1.0  1.0
```

Offset term has highest weight, 10 time higher than anyother feature (ParentalCare, martialStatus and Male) - which does not comprise of a good prediction model. This might be due to fact that all three features are binary in nature. Ordinary least square may not be the best method in this case.

In [150]:
```
d = zip(names(birth), Xoffset \ y)
for (n,v) in d
    println("$n: $v")
end
```

```
MaritalStatus: 101.36160671961902
PrenatalCare: 73.0584570783685
Male: 124.357238710163
Weight: 3138.9270414155408
```

b) Fit a quantile regression on the data with $q = 0.05$ and $q = 0.95$. Compare these coefficients to those you found in part a).

In [151]:
```
include("proxgrad.jl")
```

Out[151]:
```
proxgrad_const
```

In [152]:
```
using DataFrames,Random, ScikitLearn, LowRankModels, CSV, Plots, LinearA
```

In [153]:
```
n,d = size(Xoffset)
# Quantile loss function
loss = 1/n*QuantileLoss(quantile = 0.05)

# No Regularizer
λ = 0
reg = ZeroReg()

w = proxgrad(loss, reg, Xoffset, y, maxiters=10000)
```

Out[153]:
```
4-element Array{Float64,1}:
 292.71454623472334
 361.97972538082524
 248.41686333404468
 485.15146964169975
```

In [154]:
```
n,d = size(Xoffset)
# Quantile loss function
loss = 1/n*QuantileLoss(quantile = 0.95)

# No Regularizer
λ = 0
reg = ZeroReg()

w = proxgrad(loss, reg, Xoffset, y, maxiters=10000)
d = zip(names(birth), w)
for (n,v) in d
    println("$n: $v")
end
```

```
MaritalStatus: 971.8031538296428
PrenatalCare: 1208.9786526496955
Male: 962.1364514052751
Weight: 2629.056532932743
```

c) Fit quantile regressions for $q = 0.05, 0.10, \cdots, 0.95$.

In [155]:
```
q_range = range(0.05, 0.95, step=0.05)
for q in q_range
    print(q)
end
size(q_range)[1]
q_range[1]
```

```
0.050.10.150.20.250.30.350.40.450.50.550.60.650.70.750.80.850.90.95
```

Out[155]:
```
0.05
```

In [156]:
```
w_c = zeros((size(q_range)[1],4))
for i in range(1,size(q_range)[1])
    loss = 1/n*QuantileLoss(quantile = q_range[i])
    w_q = proxgrad(loss, reg, Xoffset, y, maxiters=10000)
    w_c[i,:] = w_q
end
print(w_c)
```

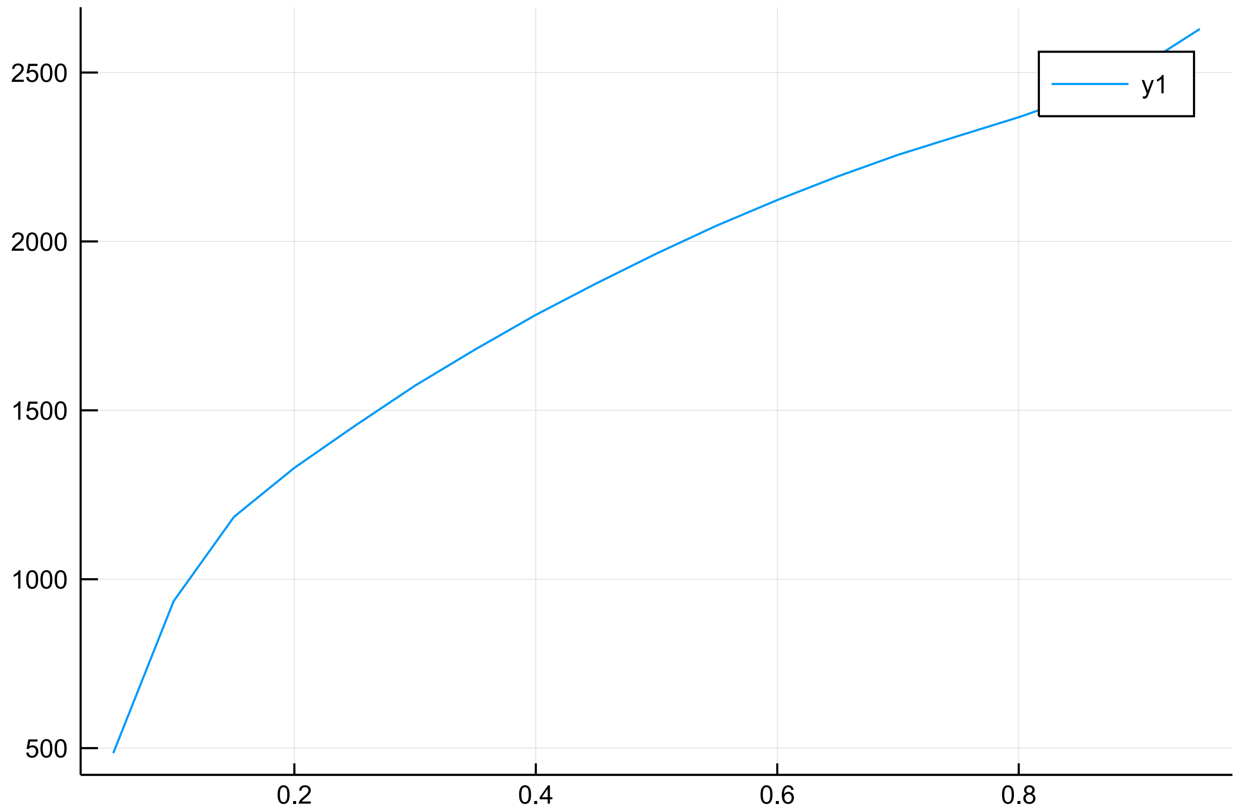
```
┌ Warning: `range(start, stop)` (with neither `length` nor `step` give
n) is deprecated, use `range(start, stop=stop)` instead.
│   caller = top-level scope at In[156]:2
└ @ Core ./In[156]:2
```

```
[292.71454623472334 361.97972538082524 248.41686333404468 485.15146964
169975; 559.6834370306774 692.1751770006242 470.83941214332435 934.971
7871701563; 643.7324608453055 827.705857112201 531.456232568129 1184.6
5533147394; 630.1200386183451 859.3235357219518 517.6112422227116 1329
.7073589358074; 605.1037331044903 869.3501394550785 502.2523063719841
1453.811950225321; 584.3549667453466 876.7282771937784 490.85228491738
45 1573.220446256203; 565.1700278910453 872.8742759064619 489.00708002
57451 1680.6561896588312; 546.7568118428981 866.7426517914759 492.2460
8453121965 1782.559643853224; 538.8475649003113 853.7582063934426 497.
9779017377859 1875.6257240935035; 532.3960523492682 843.0960094399558
509.7679682471836 1964.1217549882185; 532.3476721733788 834.4523707359
173 522.8617249517438 2047.200064363892; 539.3673031538287 829.9434670
671149 544.0451619823989 2122.5508474575854; 549.8545376529312 827.665
7369663058 578.7237717228055 2192.3292211971316; 566.7941428877882 838
.5074018450272 607.4932417935779 2256.3596867624583; 598.1574769362791
856.2747264535899 643.7083243938852 2312.285882857834; 646.92458699844
63 891.4756490022679 695.1263677323091 2367.7497318172996; 716.5523492
813394 955.8457412572769 765.6756060931011 2429.7722591718048; 831.397
7687191025 1057.372130444126 857.6501823643165 2514.624329542971; 971.
8031538296428 1208.9786526496955 962.1364514052751 2629.056532932743]
```

d) Create an intercept plot that plots quantiles against the intercept coefficient from that quantile regression. Create coefficient plots for MaritalStatus, Male, and PrenatalCare coefficients.

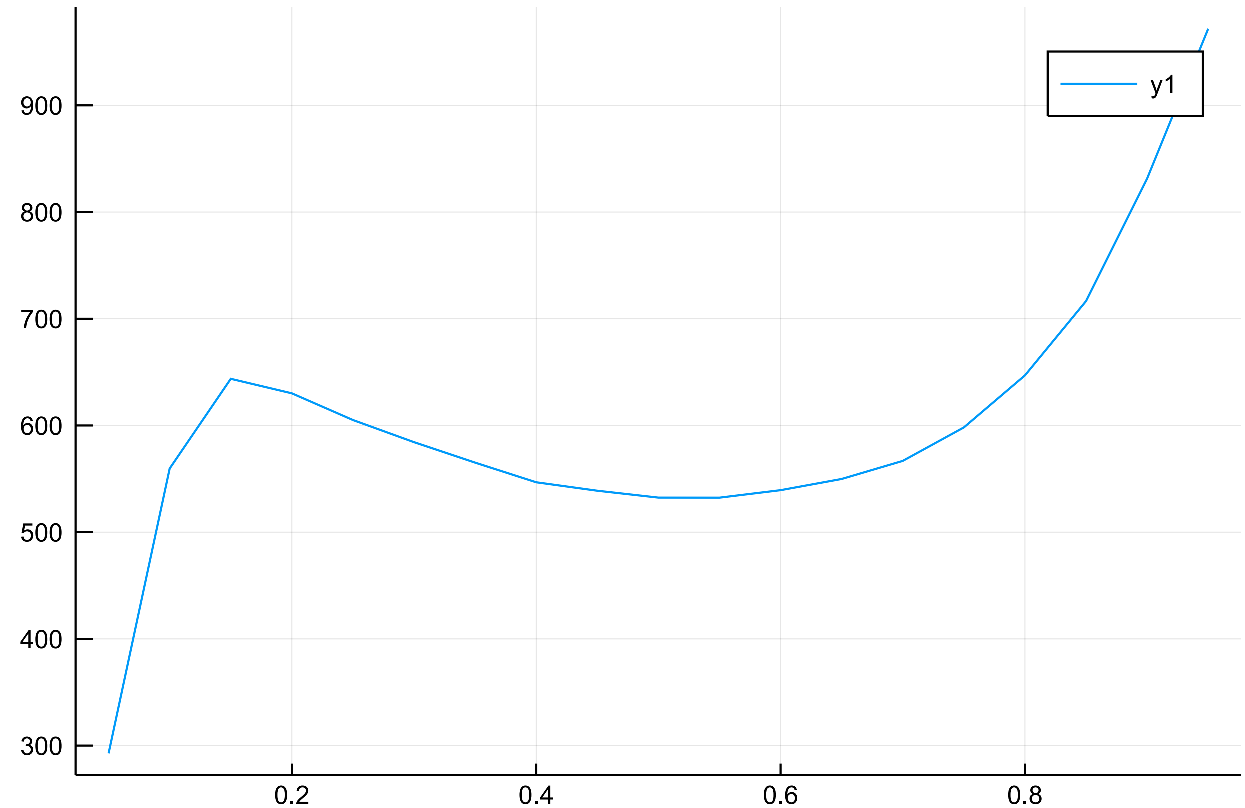
In [160]:
```
# Graph for Quantile vs offset term
plot(LinRange(q_range),w_c[:,4])
```

Out[160]:



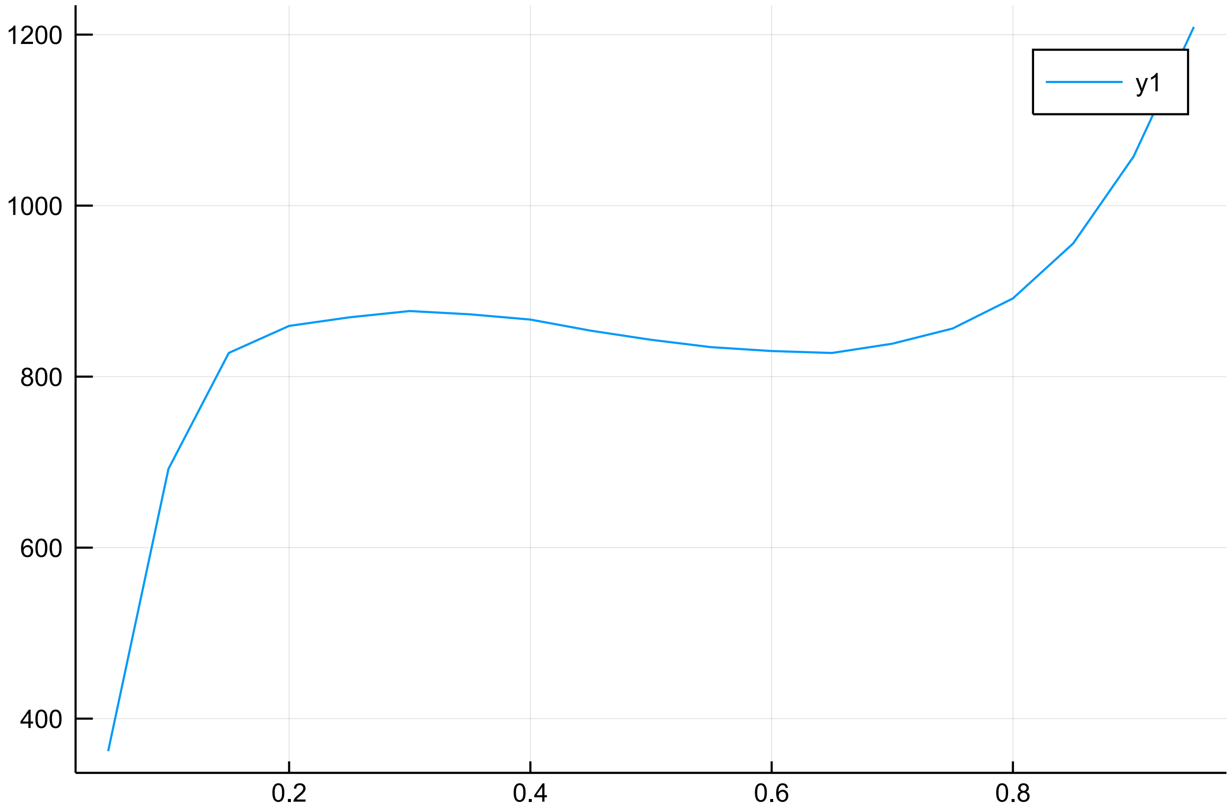
In [157]:
```
# Graph for Quantile vs MartialStatus
plot(LinRange(q_range),w_c[:,1])
```

Out[157]:



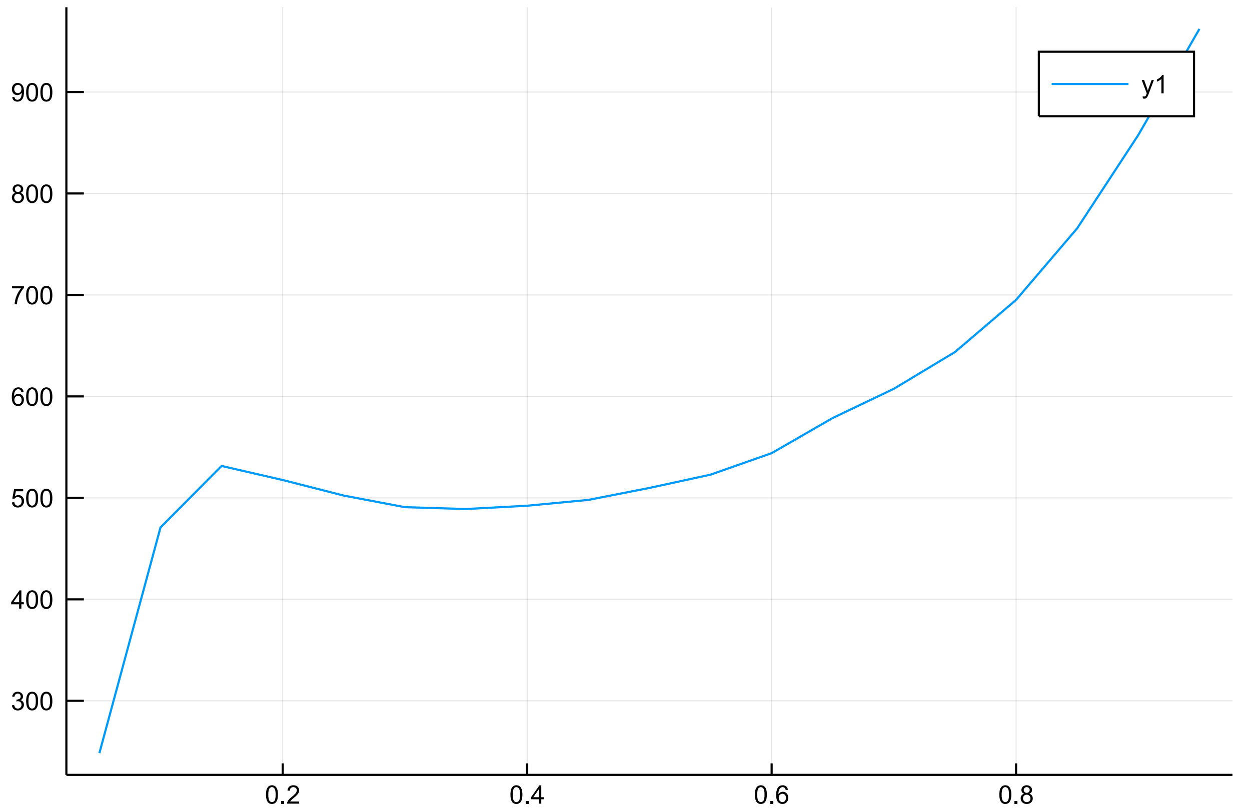
In [158]:
```
# Graph for Quantile vs Male
plot(LinRange(q_range),w_c[:,2])
```

Out[158]:



In [159]:
```
# Graph for Quantile vs PrenatalCare
plot(LinRange(q_range),w_c[:,3])
```

Out[159]:



For small quantiles (upto 15%ile), all four cofficent (Offset term, martial status, male and PrenatalCare) increases steadily. Coffiecient for offset terms remains dominant through out.

In mid quantiles (15%ile-80%ile) martial status and prenatal takes a U shape, male remains roughly flat and offset cofficients keeps increasing.

In larger quantiles (80%ile+), all four cofficents increase steadly (similar to lower quantile).

e) How should you interpret the intercepts of the quantile regressions?

Quantile regression intercepts can be interpreted to least square intercepts with the difference that least square tries to model (aka tries to achieve) mean of dependent variable) while quantile regression models (or tries to achieve) mentioned quantile of dependent variable.

f) What does the coefficient plot tell you about the effect of prenatal care for infants with low birth weight compared to those with average birth weights?

prenatal is increasingly important factors in the lower quantiles (low birth weights), however in the average brith weights, there importance reduces. In temrs of impact, data suggests, after a weight threashhold, prenatal care loses effect on birth weights.

Question 4

<Sbmitted seperate PDF (Handwritten)>

Question 5

In [48]:
```
data = CSV.read("breast-cancer.csv", header = false)

data_mat = convert(Matrix, data)
data_mat = data_matrix[shuffle(1:end), :]
X = data_mat[:,2:end]
y = data_mat[:,1]
X = [X ones(length(y))];
```

In [61]:
```
train_proportion = 0.5

Xtrain = X[1:Int(ceil(length(y)*train_proportion)),:]
Xtest = X[Int(ceil(length(y)*train_proportion))+1:end,:]

ytrain = y[1:Int(ceil(length(y)*train_proportion))]
ytest = y[Int(ceil(length(y)*train_proportion))+1:end]
```

Out[61]:
```
341-element Array{Int64,1}:
 -1
  1
  1
 -1
  1
  1
 -1
 -1
  1
 -1
  1
  1
 -1
  ⋮
 -1
 -1
 -1
 -1
  1
  1
  1
 -1
  1
  1
 -1
 -1
```

In [62]:
```
n = length(ytrain)
ybool = Int.(sign.(ytrain)) # form a boolean target

# we form \frac 1 n \sum_{i=1}^n (1 - · )_+ by multiplying the HingeLoss
loss = 1/n*HingeLoss()

# we form λ|| · ||^2 by multiplying the QuadReg() function by λ
λ = 1
reg = λ*QuadReg()

# minimize 1/n \frac 1 n \sum_{i=1}^n (1 - y_i w^T x_i)_+ + λ||w||^2
w_hinge = proxgrad(loss, reg, Xtrain, ybool, maxiters=20)

# misclassification error
(n - sum(sign.(Xtrain*w_hinge) .== ybool)) / n
```

Out[62]:
```
0.14619883040935672
```

In [63]:
```
w_hinge
```

Out[63]:
```
10-element Array{Float64,1}:
 -0.09241961388283453
  0.06398574676200873
  0.06742788102635296
  0.04015398475562908
 -0.06844541956515239
  0.11994639504862761
 -0.04800963391216968
  0.06838924242083942
 -0.03699383861174235
 -0.10709237614491854
```

In [75]:
```
n = length(ytrain)
ybool = Int.(sign.(ytrain)) # form a boolean target

# we form \frac 1 n \sum_{i=1}^n (1 - · )_+ by multiplying the HingeLoss
loss = 1/n*LogisticLoss()

# we form λ|| · ||^2 by multiplying the QuadReg() function by λ
λ = 1
reg = λ*QuadReg()

# minimize 1/n \frac 1 n \sum_{i=1}^n (1 - y_i w^T x_i)_+ + λ||w||^2
w_log = proxgrad(loss, reg, Xtrain, ybool, maxiters=20)

# misclassification error
(n - sum(sign.(Xtrain*w_logistic) .== ybool)) / n
```

Out[75]:
```
0.14035087719298245
```

In [76]:
```
w_log
```

Out[76]:
```
10-element Array{Float64,1}:
 -0.08000965853379985
  0.06310334494366261
  0.06624563385820195
  0.04081769676777572
 -0.06599429228753907
  0.10632021166709396
 -0.0413584185444494
  0.06914415687044588
 -0.03221309140962072
 -0.09757440550214655
```

Part B

In [77]:
```
# misclassification error
n = length(ytest)
ybool = Int.(sign.(ytest))
(n - sum(sign.(Xtest*w_hinge) .== ytest)) / n ##For hinge
```

Out[77]:
```
0.11730205278592376
```

In [78]:
```
(n - sum(sign.(Xtest*w_log) .== ytest)) / n ##For logistic
```

Out[78]:
```
0.11143695014662756
```

Part C

Let L be for logisitics loss function, L = log(1+exp(-ywX))

L(x,y=1;w) = log(1+exp(-wX))

L(x,y=-1;w) = log(1+exp(wX))

Normalization constant, z(x; w) = exp (–L(x,y=1;w)) + exp (–L(x,y=-1;w))

z(x; w) = exp (–log(1+exp(-wX)))) + exp (–log(1+exp(wX))

z(x; w) = 1/(1+exp(-wX)) + 1/(1+exp(wX))

z(x; w) = exp(wX)/(1+exp(wX)) + 1/(1+exp(wX))

z(x; w) = (1+exp(wX))/(1+exp(wX))

z(x; w) = 1

hence for logistic loss function, normalization constant is always 1.

Part D

In [121]:
```
function P_log(x,y,w)
    z1 = y*dot((w_log),x)
    z2 = 1/(1+exp(-1*z1[1]))
    return z2
end
```

Out[121]: `P_log (generic function with 1 method)`

In [172]:
```
function hinge_loss(x,y,w)
    z1 = y*dot((w_log),x)
    #println("z1", z1)
    z2 = 1-z1[1]
    #println("z2", z2)
    return (max(0,z2))
end
```

Out[172]: `hinge_loss (generic function with 1 method)`

In [173]:
```
function P_hinge(x,y,w)
    pos = exp(-1*hinge_loss(x,1,w))
    #print(pos)
    neg = exp(-1*hinge_loss(x,-1,w))
    #print(neg)
    if(y == 1)
        return(pos/(pos+neg))
        else
        return(neg/(pos+neg))
    end
end
```

Out[173]: P_hinge (generic function with 1 method)

In [174]:
```
#Test
x = Xtest[1,:]
y = ytest[1,:]
P_hinge(x,y,w_hinge)
```

Out[174]: 0.518355419645159

In [142]:
```
total_P_log = 0
total_P_hinge = 0

for i in length(ytest)
    x = Xtest[i,:]
    y = ytest[i,:]
    total_P_log = total_P_log + log(P_log(x,y,w_log))
    total_P_hinge = total_P_hinge + log(P_hinge(x,y,w_hinge))
end

println("loglikelihood for logistics is:", total_P_log)
println("loglikelihood for hinge is:", total_P_hinge)
```

```
loglikelihood for logistics is:-0.91236292560698
loglikelihood for hinge is:-1.169597260496795
```

Loglikelyhood is larger for logistics loss function.

In [ ]: