Logistic Perceptron Learner

First Probability and Logarithms

Probability is the study of randomness and uncertainty. For the sake of a mathemetical interpretation, it is a measure of the likelihood that an event will occur. This is ratio of the event to the sample set. This sample set is the set of all possible outcomes.

- $\mathcal{S}$: sample set
- $\mathcal{E}$: set of an event such that $\mathcal{E} \subseteq \mathcal{S}$
- $p = \frac{|\mathcal{E}|}{|\mathcal{S}|}$: the probability of an event and $0 \leq p \leq 1$
- $1 - p$: the probability of an event not occurring is.

This particular probability distribution would be a Bernouli distribution.

- $\mathcal{X} \sim Bern(p)$:is this a special case of the binomial distribution where n = 1. $\mathcal{X} \sim Bin(n, p)$ Both of these ditributions are discrete which is favorable for our classification outcomes.
- $f(k;p) = \begin{cases} p & \text{if } k = 1 \\ q = 1 - p & \text{if } k = 0 \end{cases}$

With this we can now define odds which is simply the ratio the probability of an event occurring to the probability of the event not occuring.

- $odds_p = \frac{p}{1-p}$ some examples of odds would be:

odds of a fair coin flip = 0.5/0.5 = 1 or 1:1

odds of a fair coin die the roll = 0.333/0.666 = 1/1 or 1:2

In logistic perceptron learner we are learning the unkown p for any given linear combination of features

- $x \in \mathbb{R}^{\mathrm{d}+1}$

In the classic perceptron model we are learning the weights for a linear combination of features

- $\hat{y} = \sum_{i=1}^{d} w_i x_i + b = \sum_{i=1}^{d+1} w_i x_i = w^T x$ to get $y \in \{-1, 1\}$ or $\{0, 1\}$

we then update the weights if miscalssified via

- $w_{t+1} = w_t - \alpha \nabla f_t$.

From the loss function we get the gradient

- $h(w) = \frac{1}{2} \Sigma (\hat{y} - y)^2$
- $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial w} = \nabla f_t = (\hat{y} - y) x^i$

  x^i = $\begin{pmatrix} x_{1}\

```
x_{2} \\
\vdots \\
x_{n}
\end{pmatrix}$
```

And we are also learning

- p which is $\hat{p}$: since our algorithm only gets $g \approx f$ The function that links the linear combination of our variables, (features) and the Bernoulis probability distrbution is the logit. In more formal terms a function that maps our linear

In [5]:
```
#In this data set se will be looking at nba players how are still in the league aft
er 5yrs worth of stats
#gathered about them


using Plots, CSV

nba = CSV.read("nba_logreg.csv")
```

Out[5]: 1,340 rows × 21 columns (omitted printing of 13 columns)

	Name	GP	MIN	PTS	FGM	FGA	FG%	3P Made
	String⍰	Int64⍰	Float64⍰	Float64⍰	Float64⍰	Float64⍰	Float64⍰	Float64⍰
1	Brandon Ingram	36	27.4	7.4	2.6	7.6	34.7	0.5
2	Andrew Harrison	35	26.9	7.2	2.0	6.7	29.6	0.7
3	JaKarr Sampson	74	15.3	5.2	2.0	4.7	42.2	0.4
4	Malik Sealy	58	11.6	5.7	2.3	5.5	42.6	0.1
5	Matt Geiger	48	11.5	4.5	1.6	3.0	52.4	0.0
6	Tony Bennett	75	11.4	3.7	1.5	3.5	42.3	0.3
7	Don MacLean	62	10.9	6.6	2.5	5.8	43.5	0.0
8	Tracy Murray	48	10.3	5.7	2.3	5.4	41.5	0.4
9	Duane Cooper	65	9.9	2.4	1.0	2.4	39.2	0.1
10	Dave Johnson	42	8.5	3.7	1.4	3.5	38.3	0.1
11	Corey Williams	35	6.9	2.3	0.9	2.4	36.5	0.0
12	Sam Mack	40	6.7	3.6	1.2	3.0	39.8	0.1
13	Lorenzo Williams	27	6.6	1.3	0.6	1.3	47.2	0.0
14	P.J. Hairston	45	15.3	5.6	1.9	6.0	32.3	1.1
15	Elmore Spencer	44	6.4	2.4	1.0	1.9	53.7	0.0
16	John Crotty	40	6.1	2.6	0.9	1.8	51.4	0.1
17	Stephen Howard	49	5.3	2.1	0.7	1.9	37.6	0.0
18	Randy Woods	41	4.2	1.7	0.6	1.6	34.8	0.1
19	Larry Johnson	82	37.2	19.2	7.5	15.3	49.0	0.1
20	Larry Johnson	82	37.2	19.2	7.5	15.3	49.0	0.1
21	Billy Owens	80	31.4	14.3	5.9	11.1	52.5	0.0
22	Stacey Augmon	82	30.5	13.3	5.4	11.0	48.9	0.0
23	Mark Macon	76	30.3	10.6	4.4	11.7	37.5	0.1
24	Steven Smith	61	29.6	12.0	4.9	10.7	45.4	0.7
25	Mitch McGary	32	15.2	6.3	2.8	5.2	53.3	0.0
26	Larry Stewart	76	29.3	10.4	4.0	7.8	51.4	0.0
27	Mike Iuzzolino	52	24.6	9.3	3.1	6.8	45.1	1.1
28	Doug Smith	76	22.5	8.8	3.8	9.2	41.5	0.0
29	Paul Graham	78	22.0	10.1	3.9	8.7	44.7	0.7
30	Donald Hodge	51	20.7	8.4	3.2	6.4	49.7	0.0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

In [7]:
```
# zero is if they are not in the nba after 5yrs and 1 is there are in the nba after 5yrs
nba[21]
```

Out[7]:
```
1340-element Array{Union{Missing, Float64},1}:
 0.0
 0.0
 0.0
 1.0
 1.0
 0.0
 1.0
 1.0
 0.0
 0.0
 0.0
 1.0
 1.0
 ⋮
 0.0
 0.0
 1.0
 1.0
 1.0
 0.0
 0.0
 0.0
 1.0
 0.0
 1.0
 1.0
```

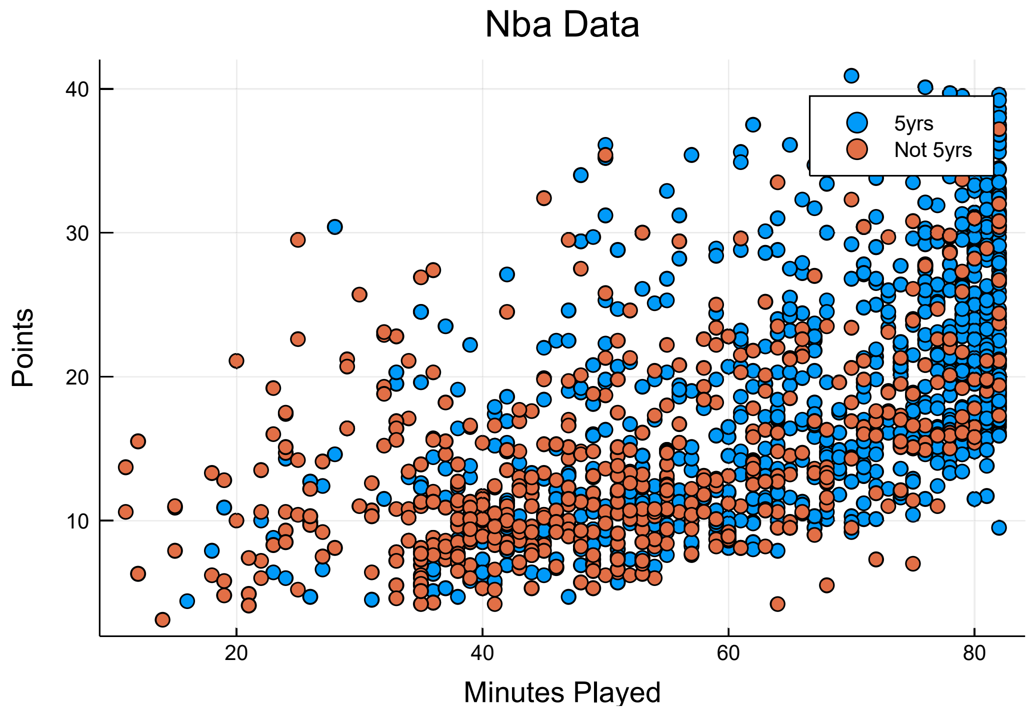
In [9]:
```
#we want to see if we can predict if a playe makes it past 5yrs or not.
data = [x for x in zip(nba[2], nba[3], nba[21])]
```

Out[9]:
```
1340-element Array{Tuple{Int64,Float64,Float64},1}:
 (36, 27.4, 0.0)
 (35, 26.9, 0.0)
 (74, 15.3, 0.0)
 (58, 11.6, 1.0)
 (48, 11.5, 1.0)
 (75, 11.4, 0.0)
 (62, 10.9, 1.0)
 (48, 10.3, 1.0)
 (65, 9.9, 0.0)
 (42, 8.5, 0.0)
 (35, 6.9, 0.0)
 (40, 6.7, 1.0)
 (27, 6.6, 1.0)
 ⋮
 (73, 18.9, 0.0)
 (40, 15.4, 0.0)
 (82, 18.9, 1.0)
 (82, 18.3, 1.0)
 (50, 16.3, 1.0)
 (79, 16.1, 0.0)
 (80, 15.8, 0.0)
 (80, 15.8, 0.0)
 (68, 12.6, 1.0)
 (43, 12.1, 0.0)
 (52, 12.0, 1.0)
 (47, 11.7, 1.0)
```

In [10]:
```
#We can not fit a linear model here

scatter([x[1:2] for x in data if x[3] == 1.0], label = "5yrs")
scatter!([x[1:2] for x in data if x[3] != 1.0], label = "Not 5yrs")
plot!(title = "Nba Data", xlabel = "Minutes Played", ylabel = "Points")
```

Out[10]:



In [39]:
```
data[101:150]
```

Out[39]:
```
50-element Array{Tuple{Int64,Float64,Float64},1}:
 (73, 11.0, 0.0)
 (15, 10.9, 0.0)
 (64, 10.9, 1.0)
 (56, 10.6, 0.0)
 (64, 10.6, 0.0)
 (68, 10.1, 1.0)
 (47, 10.0, 0.0)
 (52, 9.8, 1.0)
 (64, 9.5, 0.0)
 (34, 10.8, 0.0)
 (42, 9.5, 1.0)
 (50, 9.3, 1.0)
 (52, 7.3, 1.0)
 ⋮
 (75, 17.9, 1.0)
 (72, 17.0, 0.0)
 (79, 16.8, 1.0)
 (25, 10.4, 0.0)
 (81, 15.9, 1.0)
 (63, 14.0, 1.0)
 (81, 13.8, 1.0)
 (61, 13.2, 1.0)
 (43, 13.0, 0.0)
 (55, 12.9, 1.0)
 (59, 12.4, 0.0)
 (66, 11.3, 1.0)
```

In [43]:
```
#trianing data
train_X, train_Y = [[x[1], x[2]] for x in data[1:100]], [x[3] for x in data[1:100]]
```

Out[43]:
```
(Array{Float64,1}[[36.0, 27.4], [35.0, 26.9], [74.0, 15.3], [58.0, 11.6], [48.0,
11.5], [75.0, 11.4], [62.0, 10.9], [48.0, 10.3], [65.0, 9.9], [42.0, 8.5]  …  [4
7.0, 14.3], [37.0, 13.6], [62.0, 13.3], [62.0, 12.5], [51.0, 11.9], [36.0, 11.8]
, [74.0, 11.6], [63.0, 11.6], [33.0, 10.8], [54.0, 11.2]], [0.0, 0.0, 0.0, 1.0,
1.0, 0.0, 1.0, 1.0, 0.0, 0.0  …  0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.
0])
```

In [68]:
```
#testing data
test_X, test_Y = [[x[1], x[2]] for x in data[101:150]], [x[3] for x in data[101:150
]]
```

Out[68]:
```
(Array{Float64,1}[[73.0, 11.0], [15.0, 10.9], [64.0, 10.9], [56.0, 10.6], [64.0,
10.6], [68.0, 10.1], [47.0, 10.0], [52.0, 9.8], [64.0, 9.5], [34.0, 10.8]  …  [7
9.0, 16.8], [25.0, 10.4], [81.0, 15.9], [63.0, 14.0], [81.0, 13.8], [61.0, 13.2]
, [43.0, 13.0], [55.0, 12.9], [59.0, 12.4], [66.0, 11.3]], [0.0, 0.0, 1.0, 0.0,
0.0, 1.0, 0.0, 1.0, 0.0, 0.0  …  1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.
0])
```

In [57]:
```
train_X
```

Out[57]:
```
100-element Array{Array{Float64,1},1}:
 [36.0, 27.4]
 [35.0, 26.9]
 [74.0, 15.3]
 [58.0, 11.6]
 [48.0, 11.5]
 [75.0, 11.4]
 [62.0, 10.9]
 [48.0, 10.3]
 [65.0, 9.9]
 [42.0, 8.5]
 [35.0, 6.9]
 [40.0, 6.7]
 [27.0, 6.6]
 ⋮
 [76.0, 15.4]
 [55.0, 14.9]
 [47.0, 14.3]
 [37.0, 13.6]
 [62.0, 13.3]
 [62.0, 12.5]
 [51.0, 11.9]
 [36.0, 11.8]
 [74.0, 11.6]
 [63.0, 11.6]
 [33.0, 10.8]
 [54.0, 11.2]
```

In [52]:
```
h(w,train_X[9])
```

Out[52]:
```
1
```

In [ ]:

In [46]:
```
#predictor function
function predict(x,w)
    x_new = copy(x)
    push!(x_new,1.0)
    return w'*x_new
end
```

Out[46]: predict (generic function with 1 method)

In [61]:
```
w = rand(3)

function σ(s)
    return 1/(1 +exp(-s))
end

function logistic_perception!(x, y, w, α)
    # to make a 1 to correspond to the bais
    new_x = copy(x)
    new_x = [1.0, x[1], x[2]]
    #in vector form w'*new_x
    z = w' * new_x
    y_hat = σ(z)
    if y_hat < 0.5 && y == 1
        w-= α* (y-y_hat)*σ(z)*(1 - σ(z))*new_x
    end
    return w
end
```

Out[61]: logistic_perception! (generic function with 1 method)

In [60]:
```
for _ in 1:10000
    for i in 1 :100
       logistic_perception!(train_X[i], train_Y[i] ,w,0.45)
    end
end
```

In [64]:
```
σ(predict(train_X[2],w))
```

Out[64]: 0.999999998350017

In [69]:
```
σ(predict(test_X[15],w))
```

Out[69]: 0.9999982477976892

In [70]:
```
σ(predict(test_X[3],w))
```

Out[70]: 0.9999999995470104