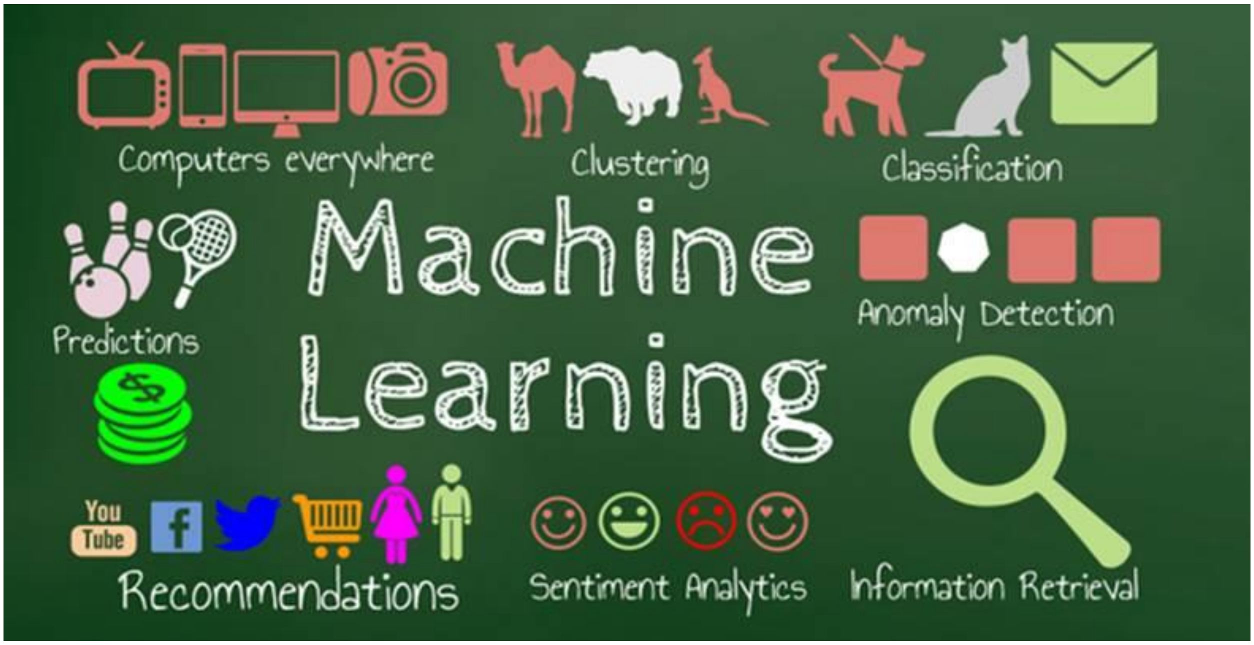
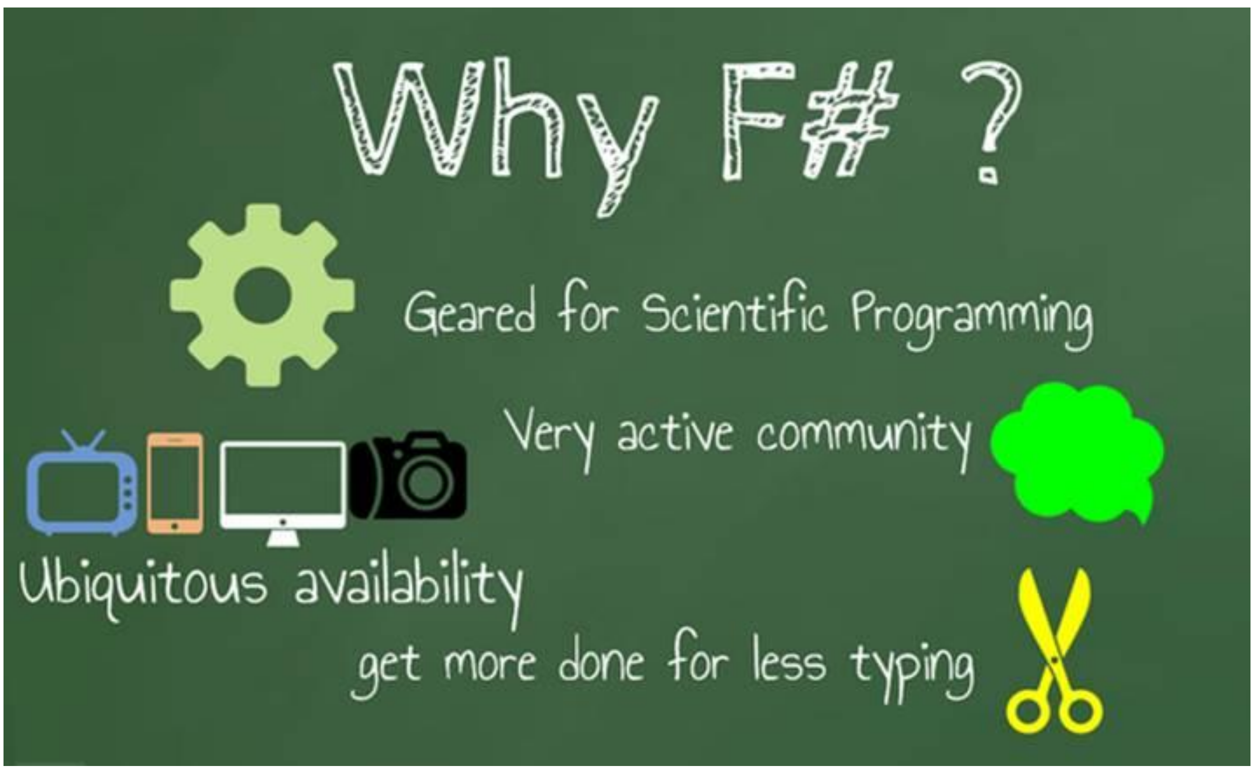
F# for Machine Learning

Chapter 1 - Introduction to Machine Learning

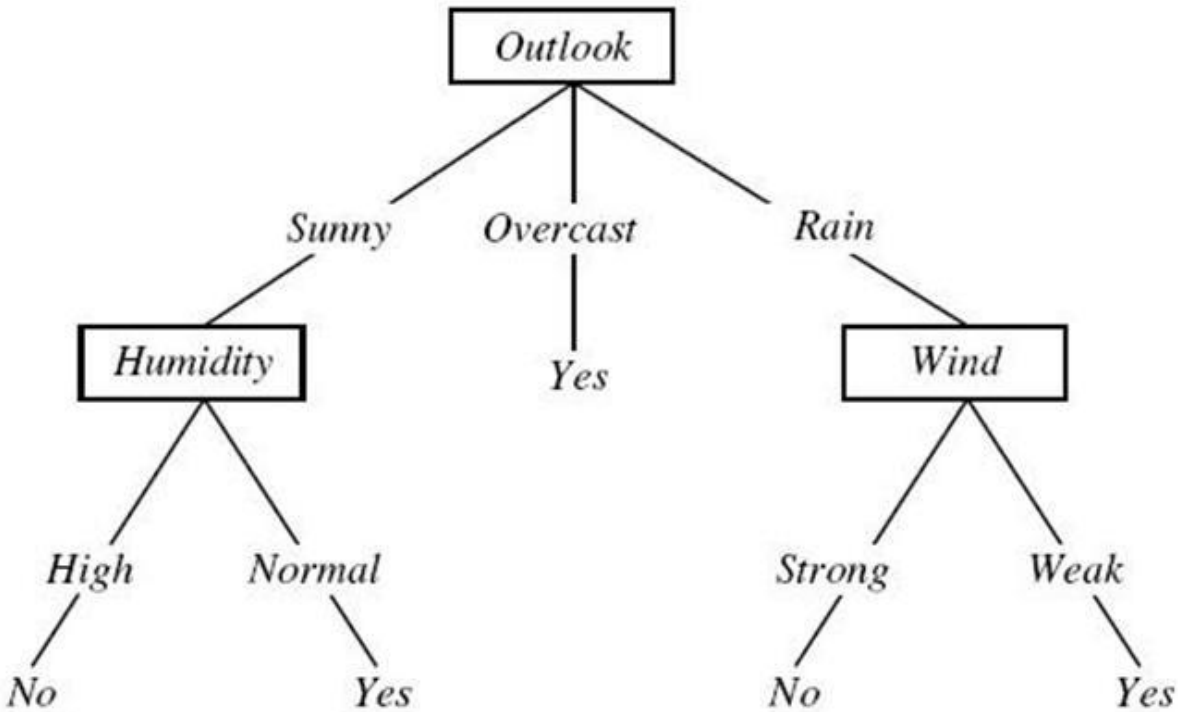




$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2} \cdot p_1 q_1 p_2 q_2 p_3 q_{3z}$

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i-p_i)^2}.$$



$$\mathrm{H}(X) = \sum_i \mathrm{P}(x_i)\,\mathrm{I}(x_i) = -\sum_i \mathrm{P}(x_i)\log_b \mathrm{P}(x_i), P(x_i) x_i I(x_i)$$

$$-(\frac{9}{14}\log_2\frac{9}{14} + \frac{5}{14}\log_2\frac{5}{14})$$

$$-(\frac{3}{5}\log_2\frac{3}{5} + \frac{2}{5}\log_2\frac{2}{5}) - (\frac{4}{4}\log_2\frac{4}{4})$$

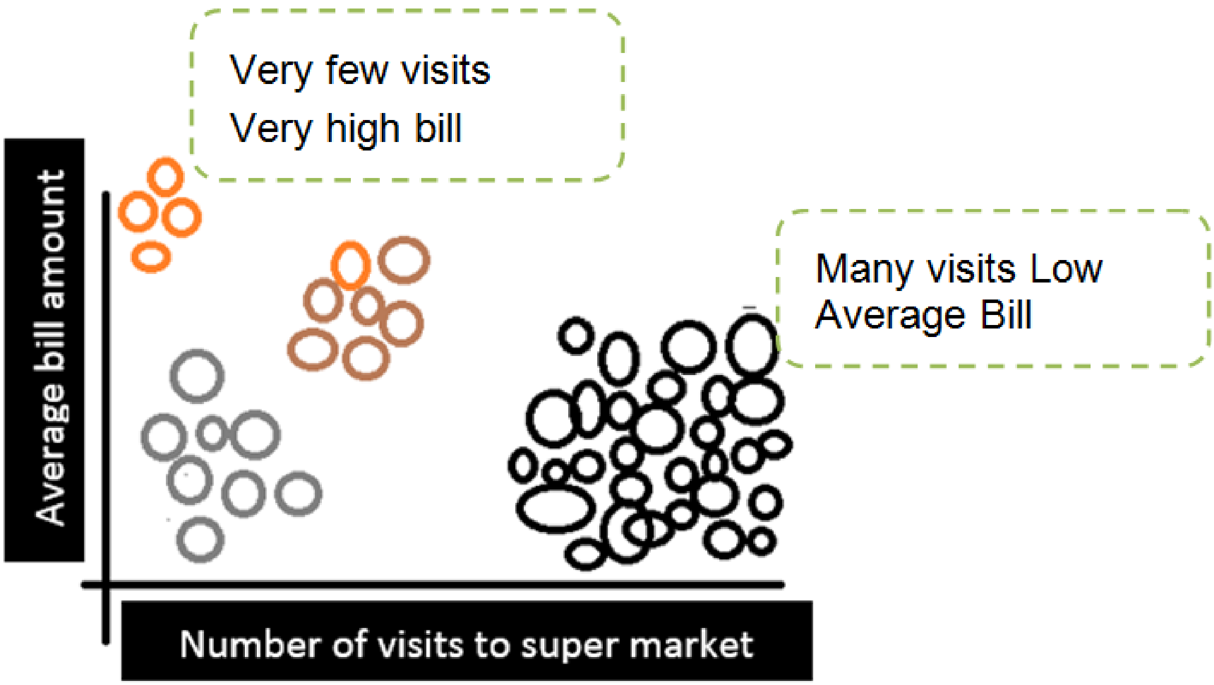
$$-(\frac{2}{5}\log_2\frac{2}{5} + \frac{3}{5}\log_2\frac{3}{5})$$

$\frac{4}{14} \times 0.0 + \frac{5}{14} \times 0.97 + \frac{5}{14} \times 0.97$ x y $x_1, x_2, x_3, \ldots, x_n | y$ x_1

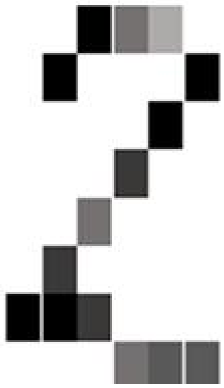
x_n y x_1 $h(x)$ $h(x) = \theta_0 + \theta_1 \times x_1 + \theta_2 \times x_2$ x_1 x_2 $\sum (h(x) - y)^2$ θ x_i

Visits	Average Bill
34	34.13
2	3400
3	2500
79	4.24
5	1200

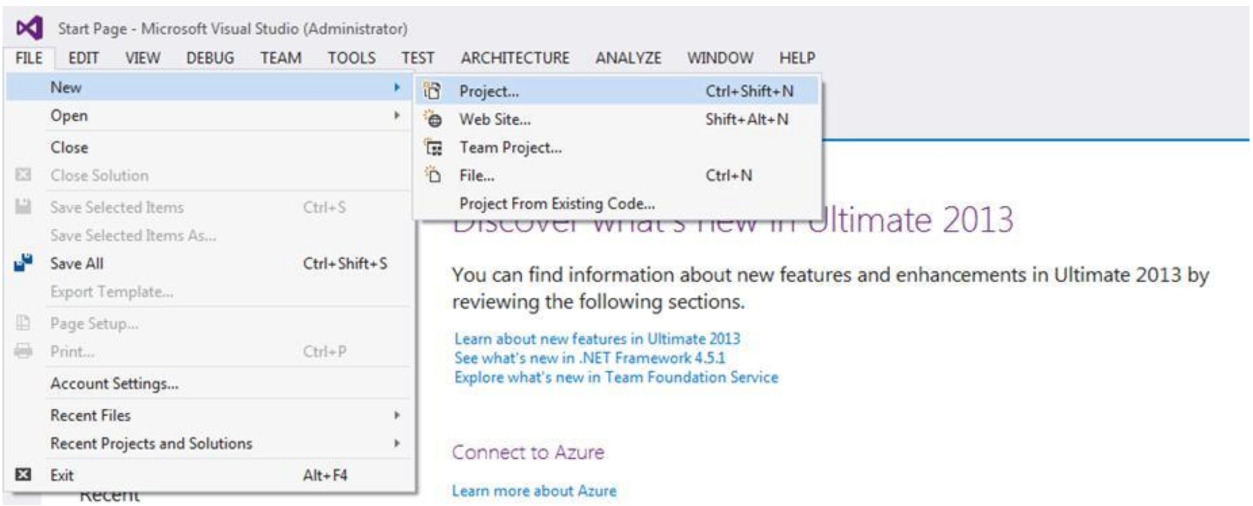
y_i $\prod_{i=1}^{n} [\frac{1}{1 + e^{-x_i \theta}}]^{y_i} \times [1 - \frac{1}{1 + e^{-x_i \theta}}]^{1 - y_i}$

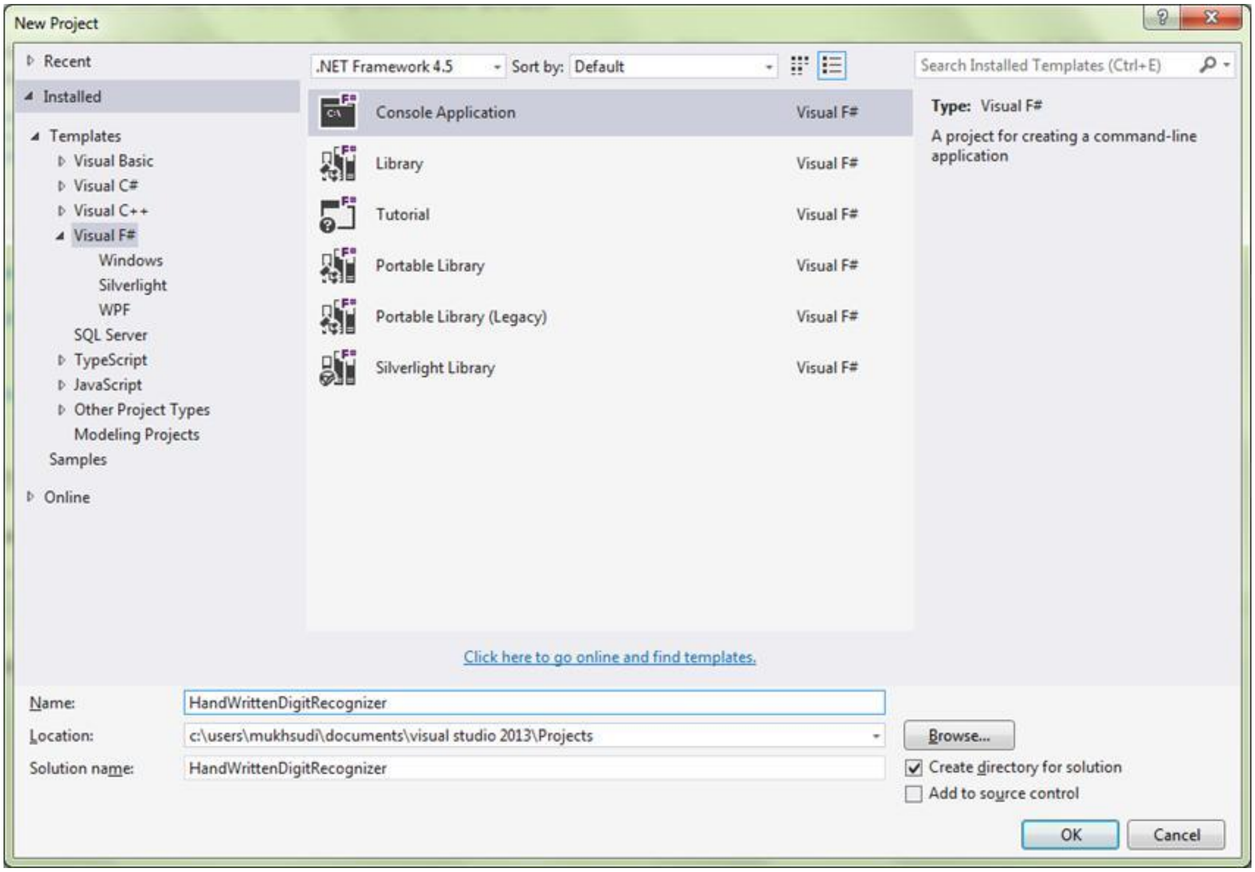


A few of our clients



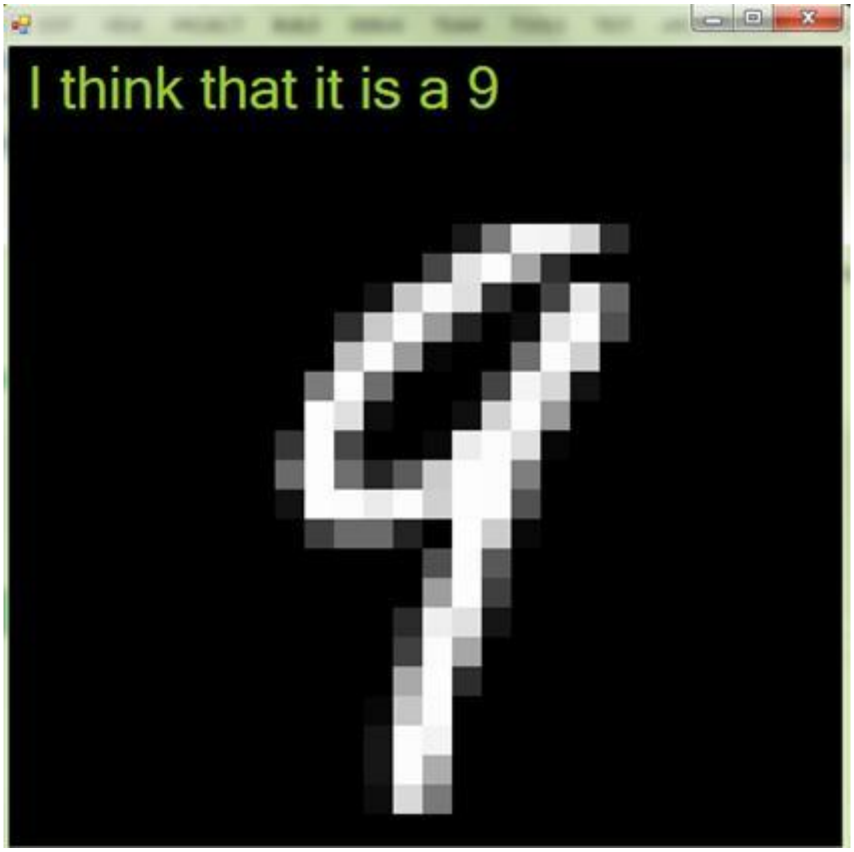
	A	B	C	D	E	F	G
1	Label	Pixel1	Pixel2	Pixel3	Pixel4	...	Pixel64
2	2	85.92679	0	0	16.50806	97.16278	31.14512
3	3	47.47406	50.22488	0	0	77.28356	14.00682
4							





Program.fs

```
// Learn more about F# at http://fsharp.net
// See the 'F# Tutorial' project for more help.

[<EntryPoint>]
let main argv =
    printfn "%A" argv
    0 // return an integer exit code
```



$d^2(p,q) = (p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2$.$pqp$

1	Label	Distance from Test Data
2	9	0.34
3	9	0.23
4	4	4.55
5	4	22.21
6	4	11.13
7	9	2.10
8	9	1.69

q

1	Label	Distance from Test Data
2	9	0.23
3	9	0.34
4	9	1.69
5	9	2.10
6	4	4.55
7	4	11.13
8	4	22.21

Chapter 2 - Linear Regression

FsPlot JavaScript charting library for F#

Accord Framework .NET

```
val velocities : Vector<float> = DenseVector 4-Double
23
 4
 5
 2
```

```
val y : Matrix<float> = DenseMatrix 3x2-Double
1  3
1  5
1  4
```

```
18.0   8   307.0      130.0      3504.      12.0   70  1        "chevrolet chevelle malibu"
15.0   8   350.0      165.0      3693.      11.5   70  1        "buick skylark 320"
18.0   8   318.0      150.0      3436.      11.0   70  1        "plymouth satellite"
16.0   8   304.0      150.0      3433.      12.0   70  1        "amc rebel sst"
17.0   8   302.0      140.0      3449.      10.5   70  1        "ford torino"
15.0   8   429.0      198.0      4341.      10.0   70  1        "ford galaxie 500"
14.0   8   454.0      220.0      4354.       9.0   70  1        "chevrolet impala"
```

```
  1. mpg:           continuous
  2. cylinders:     multi-valued discrete
  3. displacement:  continuous
  4. horsepower:    continuous
  5. weight:        continuous
  6. acceleration:  continuous
  7. model year:    multi-valued discrete
  8. origin:        multi-valued discrete
  9. car name:      string (unique for each instance)
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

```
val myMat : Matrix<float> =
  DenseMatrix 3x3-Double
1    2  3
4    5  2
7  0.8  9

val myMat' : Matrix<float> =
  DenseMatrix 3x3-Double
-0.452083   0.1625   0.114583
 0.229167    0.125  -0.104167
  0.33125  -0.1375    0.03125
```

$$X = Q \begin{bmatrix} R \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} Q1 & Q2 \end{bmatrix}$$

$$\mathbf{M} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^*$$

$$X = U \begin{bmatrix} W \\ 0 \end{bmatrix} V'$$

$$\{(x_1, y_1)(x_2, y_2)(x_3, y_3), ..., (x_n, y_n)\}$$

x

y

x

y

x

y

$$\hat{y} = b_0 + b_1 \times x$$

$\hat{y}$

y

$$e_i = y_i - \hat{y}_i$$

b_0

b_1

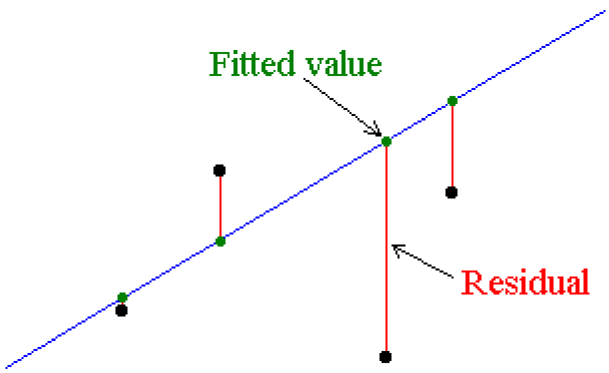
$$\sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2$$

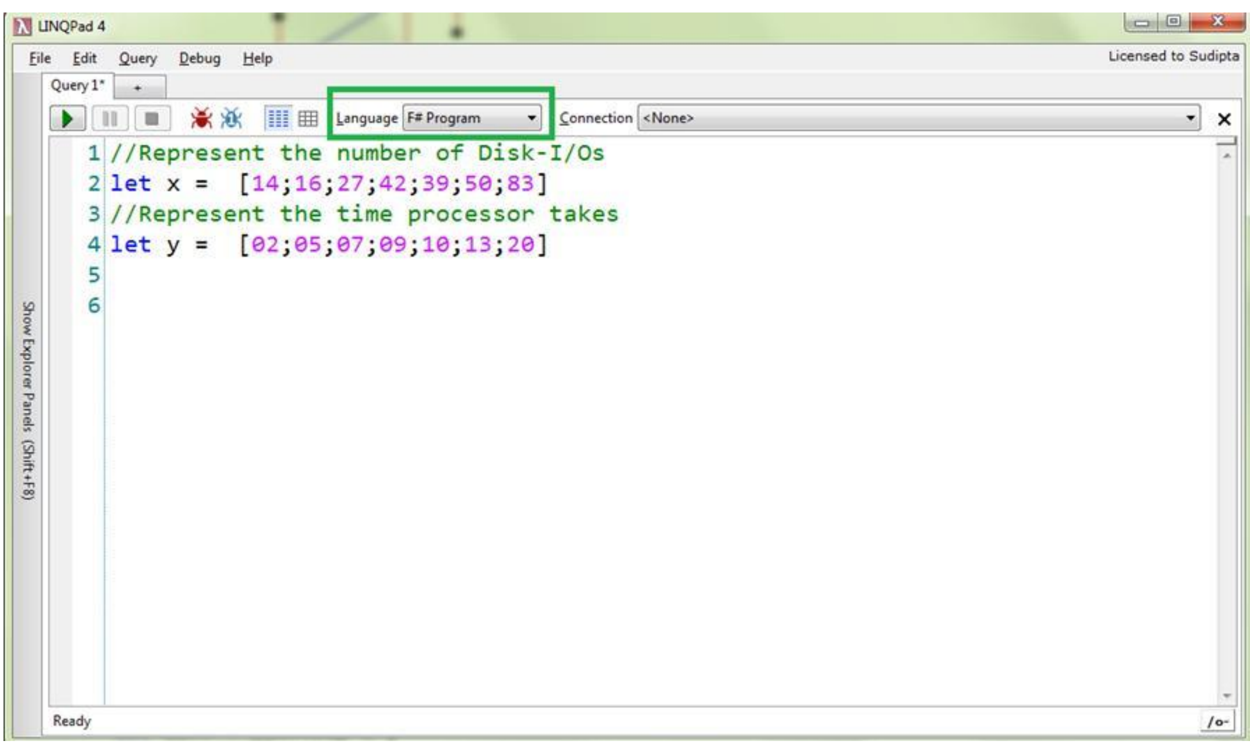
$$b_1 = \frac{\sum \bar{x}\bar{y} - n\bar{x}\bar{y}}{\sum x^2 - n\bar{x}^2}$$

$$b_0 = \bar{y} - b_1 x$$

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$\Sigma xy = \sum_{i=1}^{n} x_i y_i \qquad \Sigma x^2 = \sum_{i=1}^{n} x_i^2$$





b1

0.243756371049949

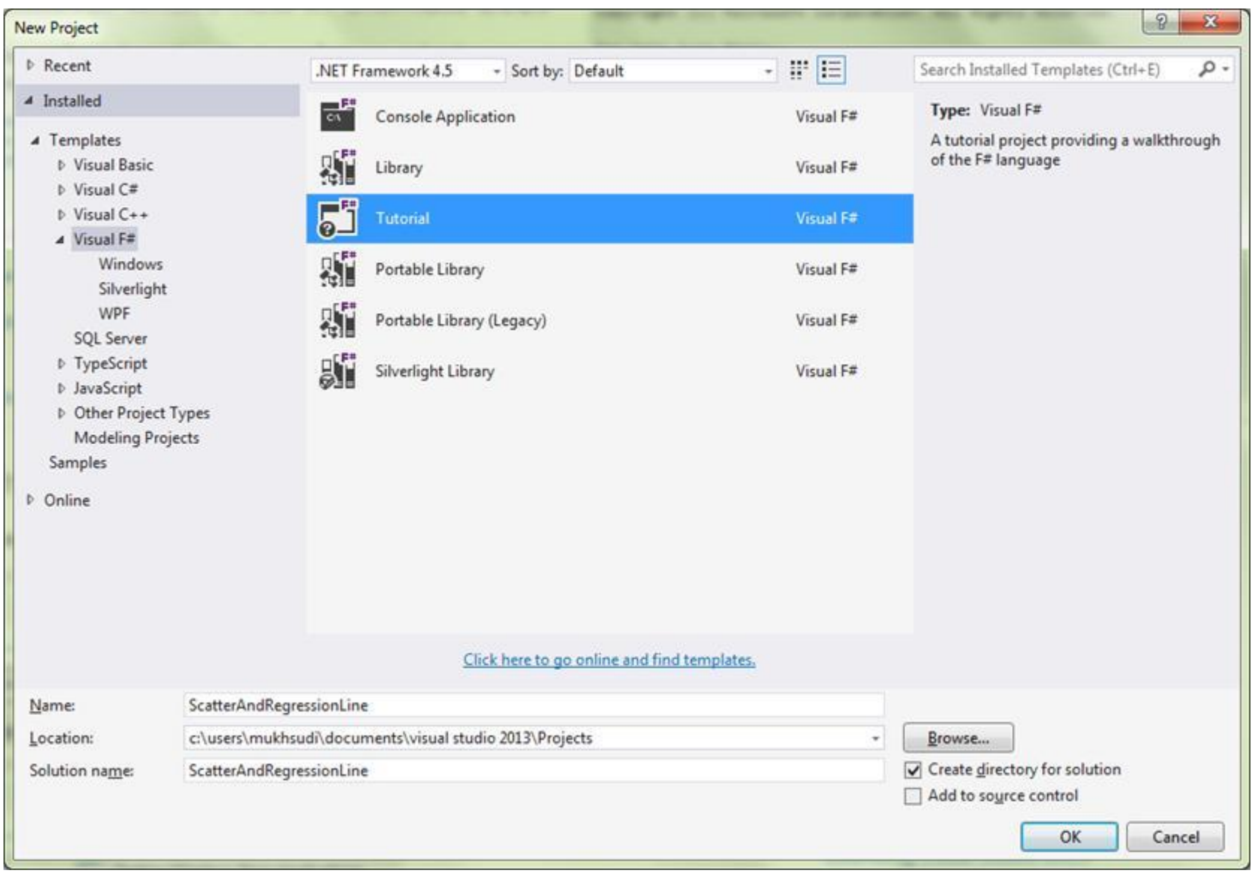
b0

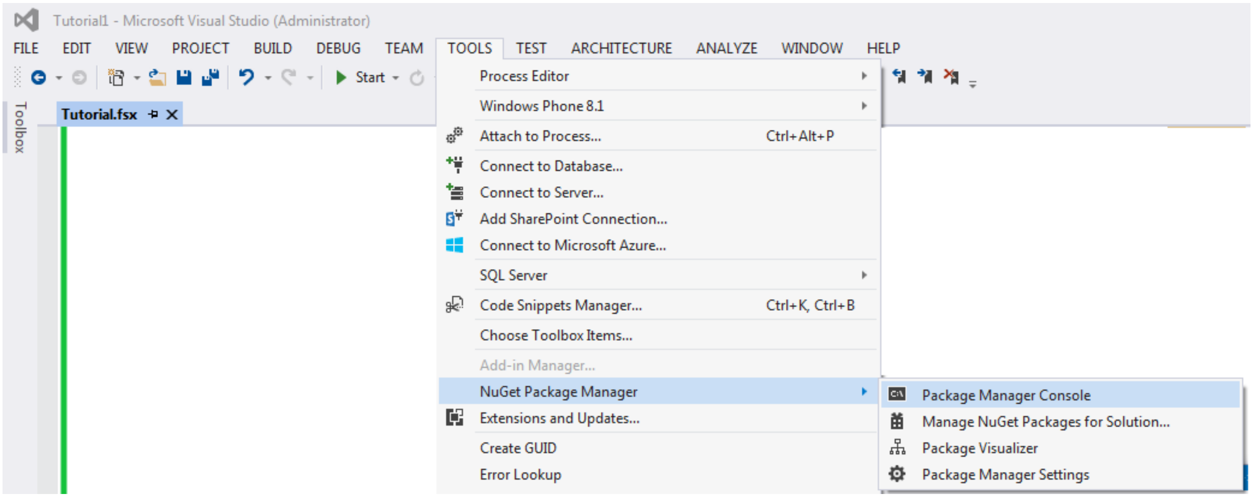
-0.00828236493374135

Result of the linear regression

FSharpList<IStructuralEquatable> (7 items)

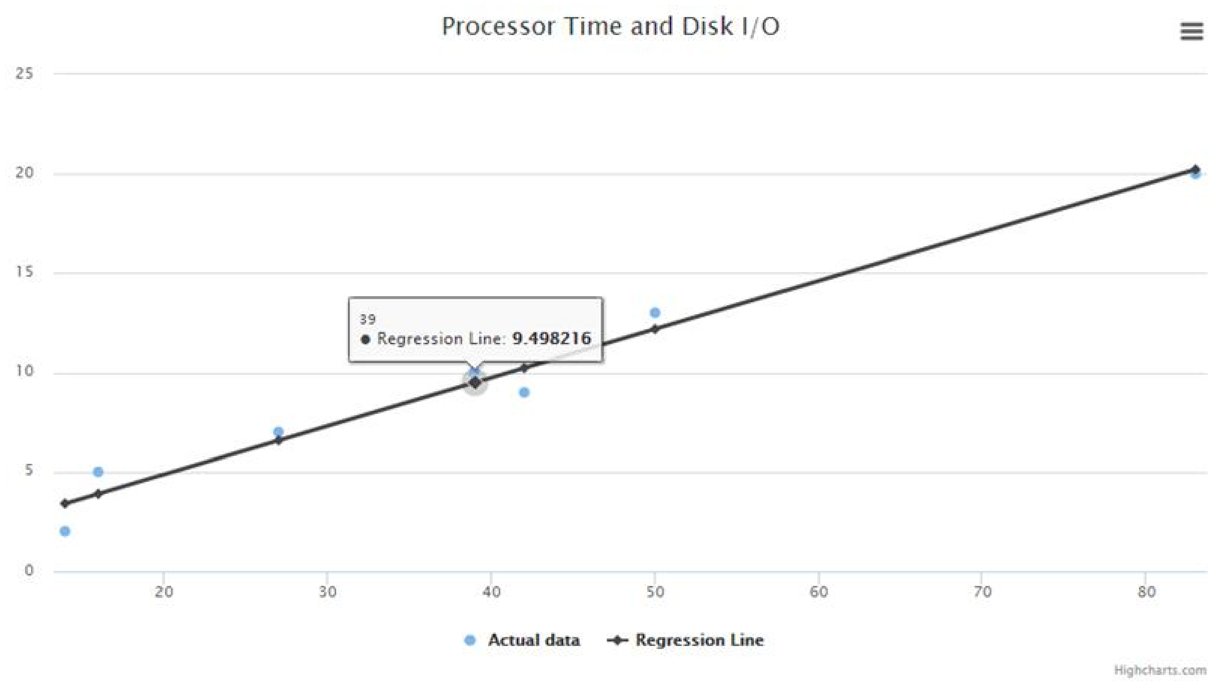
DiskIO	CPUTime	Estimate	Error	ErrorSquared
14	2	3.40430682976554	-1.40430682976554	1.97207767212615
16	5	3.89181957186544	1.10818042813456	1.22806386130049
27	7	6.57313965341488	0.426860346585118	0.182209755486767
42	9	10.2294852191641	-1.22948521916412	1.51163390414304
39	10	9.49821610601427	0.50178389398573	0.251787076263482
50	13	12.1795361875637	0.820463812436291	0.673160867517493
83	20	20.223496432212	-0.223496432212027	0.0499506552115052
271	66	65.9999999999993	0.000000000000012	5.8688837920489272




Tutorial1 - Microsoft Visual Studio (Administrator)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Start
Toolbox
Tutorial.fsx
Process Editor
Windows Phone 8.1
Attach to Process... Ctrl+Alt+P
Connect to Database...
Connect to Server...
Add SharePoint Connection...
Connect to Microsoft Azure...
SQL Server
Code Snippets Manager... Ctrl+K, Ctrl+B
Choose Toolbox Items...
Add-in Manager...
NuGet Package Manager
Extensions and Updates...
Create GUID
Error Lookup
Package Manager Console
Manage NuGet Packages for Solution...
Package Visualizer
Package Manager Settings

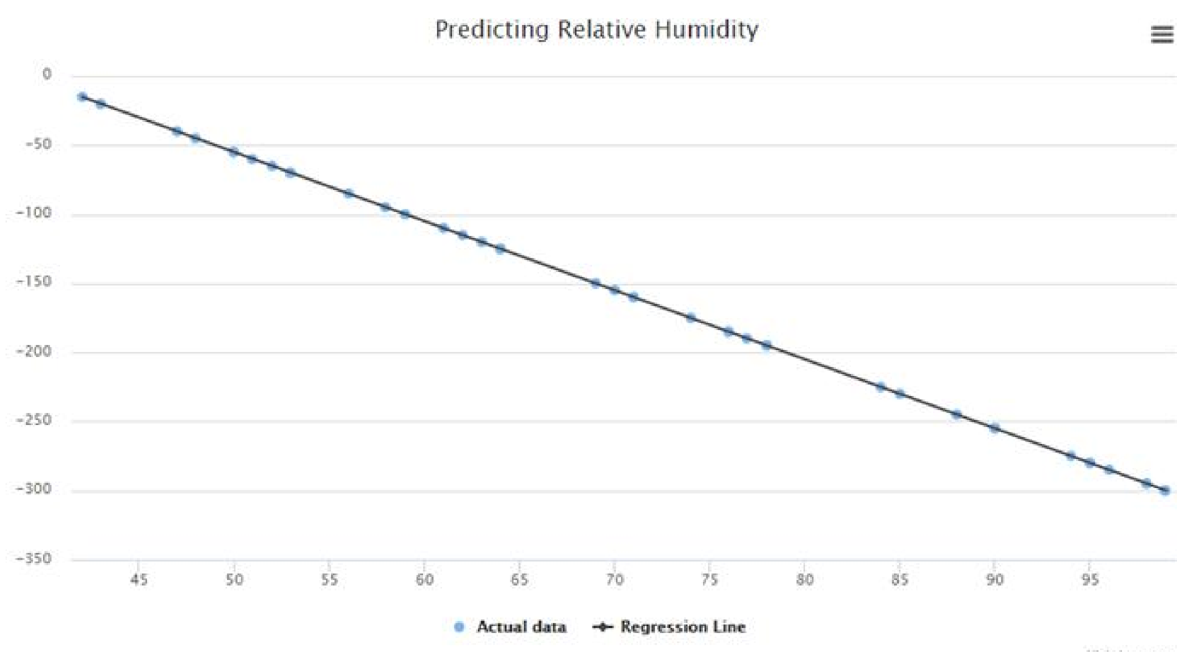
References

- FSharp.Core
- FsPlot
- FunScript
- FunScript.Interop
- FunScript.TypeScript.Binding.gapi
- FunScript.TypeScript.Binding.google_visualization
- FunScript.TypeScript.Binding.highcharts
- FunScript.TypeScript.Binding.jquery
- FunScript.TypeScript.Binding.lib
- FunScript.TypeScript.Binding.signalr
- Microsoft.AspNet.SignalR.Core
- Microsoft.Owin
- Microsoft.Owin.Cors
- Microsoft.Owin.Diagnostics
- Microsoft.Owin.Host.HttpListener
- Microsoft.Owin.Host.SystemWeb
- Microsoft.Owin.Hosting
- Microsoft.Owin.Security
- mscorlib
- Newtonsoft.Json
- Owin
- System
- System.Core
- System.Drawing
- System.Numerics
- System.Web.Cors
- System.Windows.Forms
- WebDriver




```
val xV : float [] = [|14.0; 16.0; 27.0; 42.0; 39.0; 50.0; 83.0|]
val yV : float [] = [|2.0; 5.0; 7.0; 9.0; 10.0; 13.0; 20.0|]
val b1 : float = 0.243756371
val b0 : float = -0.008282364934
```

$$RH \approx 100 - 5(t - t_d)$$



1	Size	Bedrooms	Bathrooms	Distance_From_Scool	Distance_From_Grocery_Store	Price
2	2143	2	1	0.98344108	0.293775301	208
3	2410	2	2	0.304425464	0.076821232	208
4	1339	1	1	0.380983271	0.547134073	161
5	1822	5	2	0.271166043	0.396839619	190
6	1230	3	2	0.298155285	0.442560643	210
7	1733	2	2	0.268317226	0.959010297	72

$$y(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$$

$$Y = \theta' X$$

$$\theta = (X'X)^{-1} X' Y$$

```
val qrlTheta : Vector<float> =
  DenseVector 5-Double
   2.57806
-0.0650069
  0.137941
-0.0064542
   1.66903
```

$$\theta = (X'WX)^{-1} X'WY$$

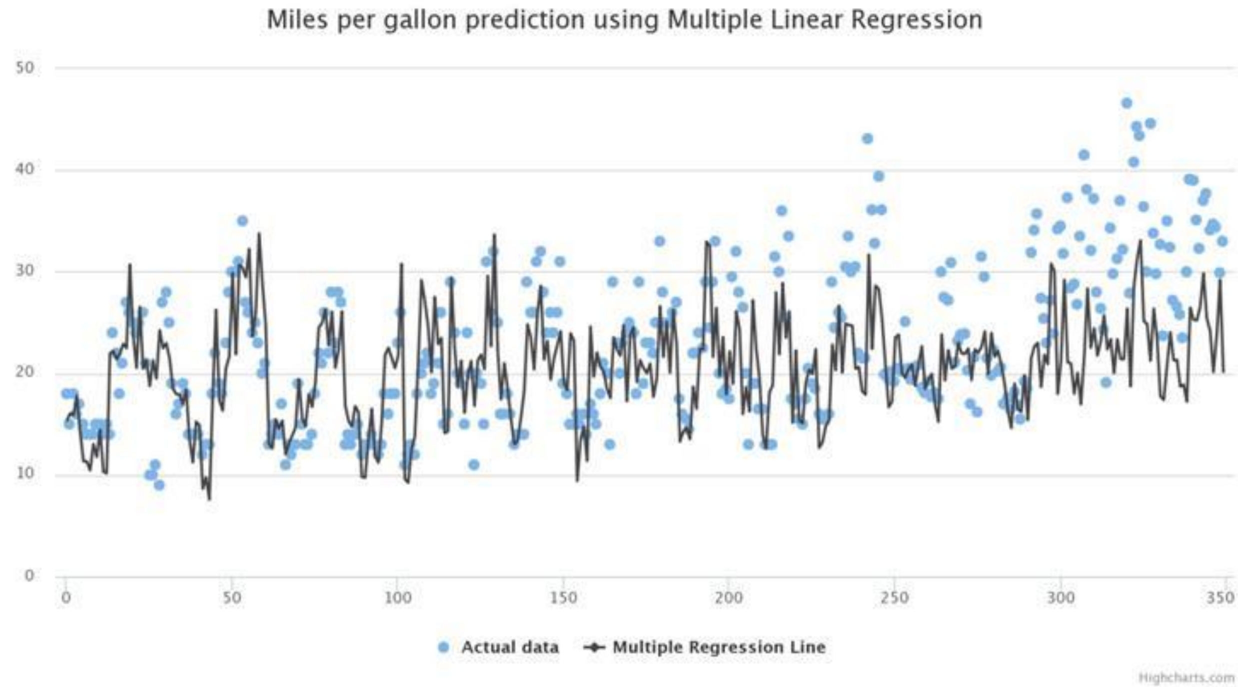
$$W_i = \exp\left(\frac{-(x^i - x)^2}{2\tau^2}\right)$$

```
val W : Matrix<float> =
  DiagonalMatrix 4x4-Double
27.0453          0          0          0
        0  60.3146          0          0
        0          0  29.1453          0
        0          0          0  29.2378
```

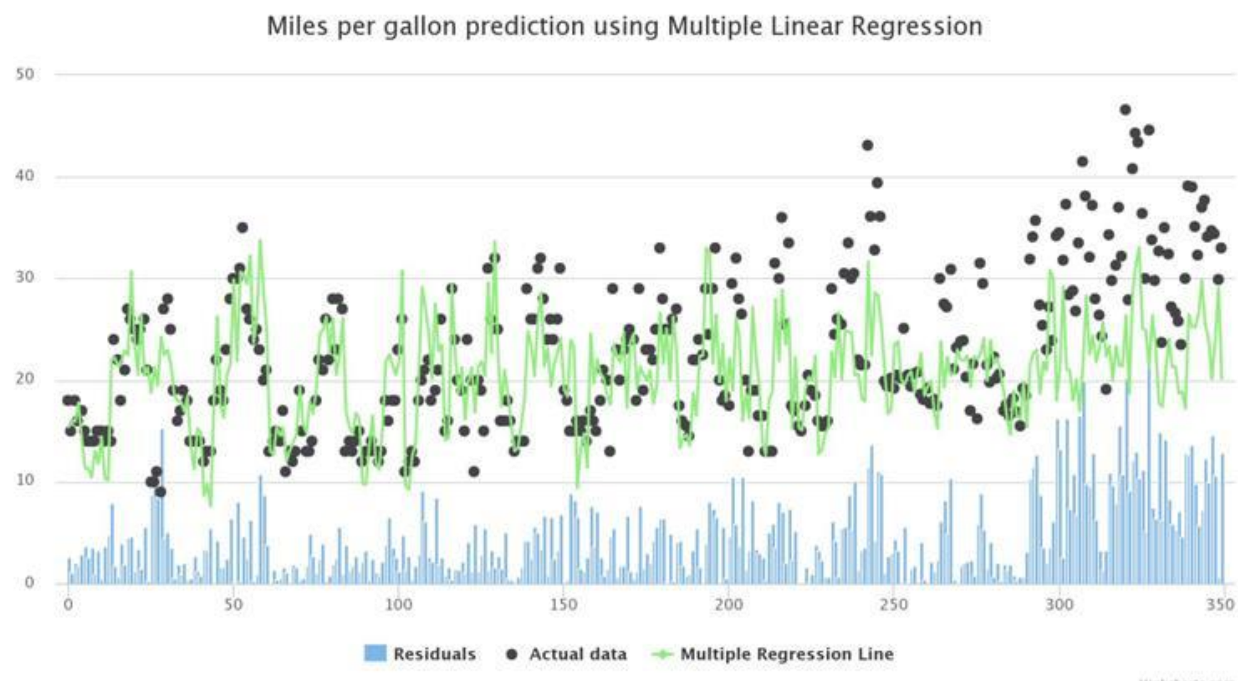
τ

τ

τ




```
val mpgResiduals : (float * float * float) [] =
  [|(18.0, 15.37701898, 2.622981024); (15.0, 16.05735903, -1.057359031);
    (18.0, 15.88952375, 2.110476246); (16.0, 17.78011233, -1.780112326);
    (17.0, 14.19868013, 2.801319873)|]
```



λ

$$\theta = (X'X + \lambda I)^{-1} X'Y$$

λ
λ
λ
λ
λ
λ

$$x_i = \frac{x_i - \mu}{S}$$

μ

```
val it : Matrix<float> =
  DenseMatrix 6x3-Double
    0.1875   999.313     0.0625
 -0.619792  0.571647  -0.620317
  0.166667     847.5        0.5
         2      1150          2
         2      1220          2
         1       734          1
```

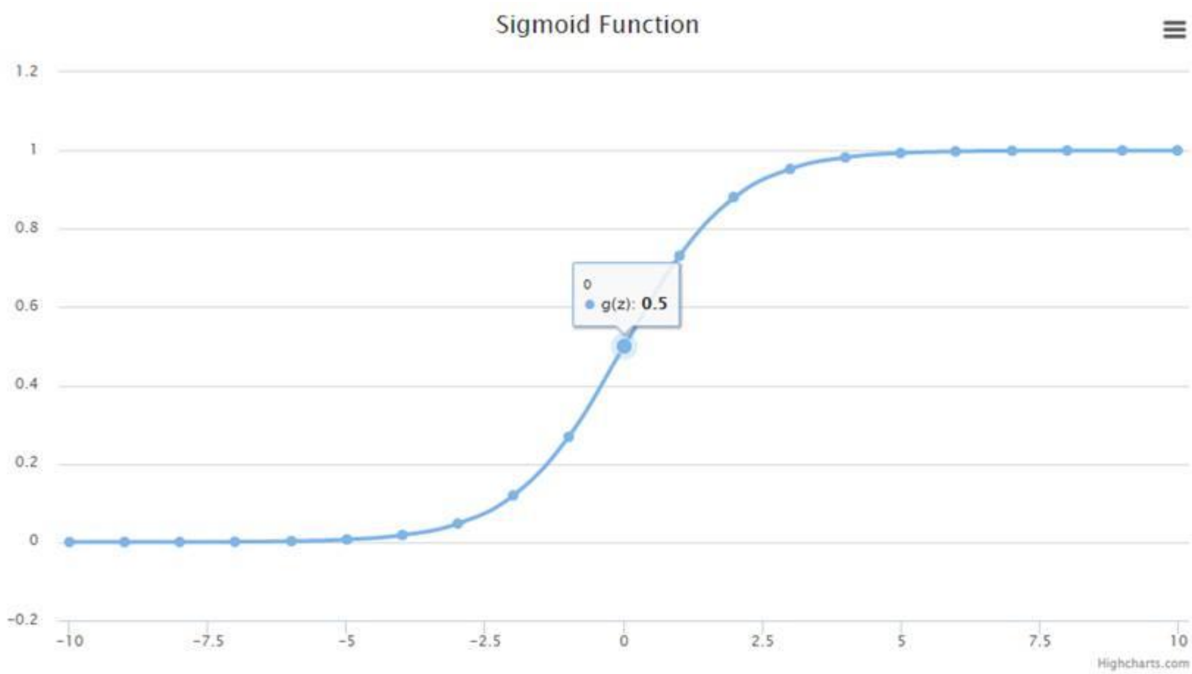
x

s

Chapter 3 - Classification Techniques

```
842302,M,17.99,10.38,122.8,1001,0.1184,0.2776,0.3001,0.1471,0.2
842517,M,20.57,17.77,132.9,1326,0.08474,0.07864,0.0869,0.07017,
```

$$g(z) = \frac{1}{(1 + e^{-z})}$$



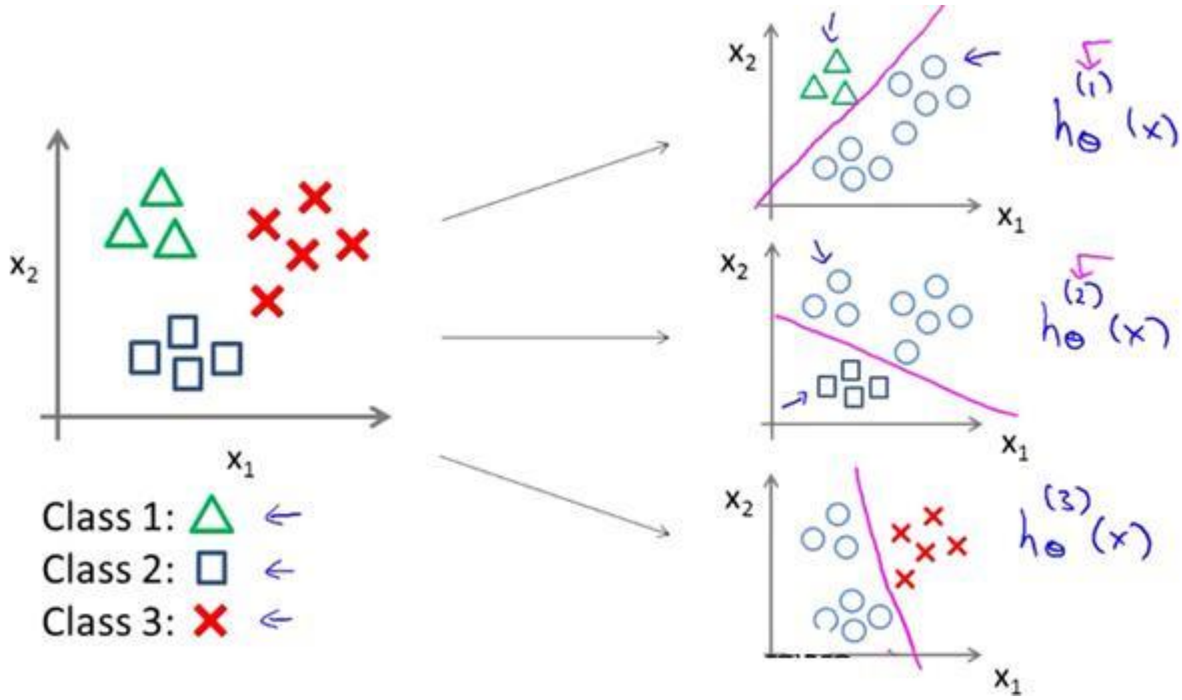
$$h_\theta(x) = g(\theta^T x) = \frac{1}{(1 + e^{-\theta^T x})}$$

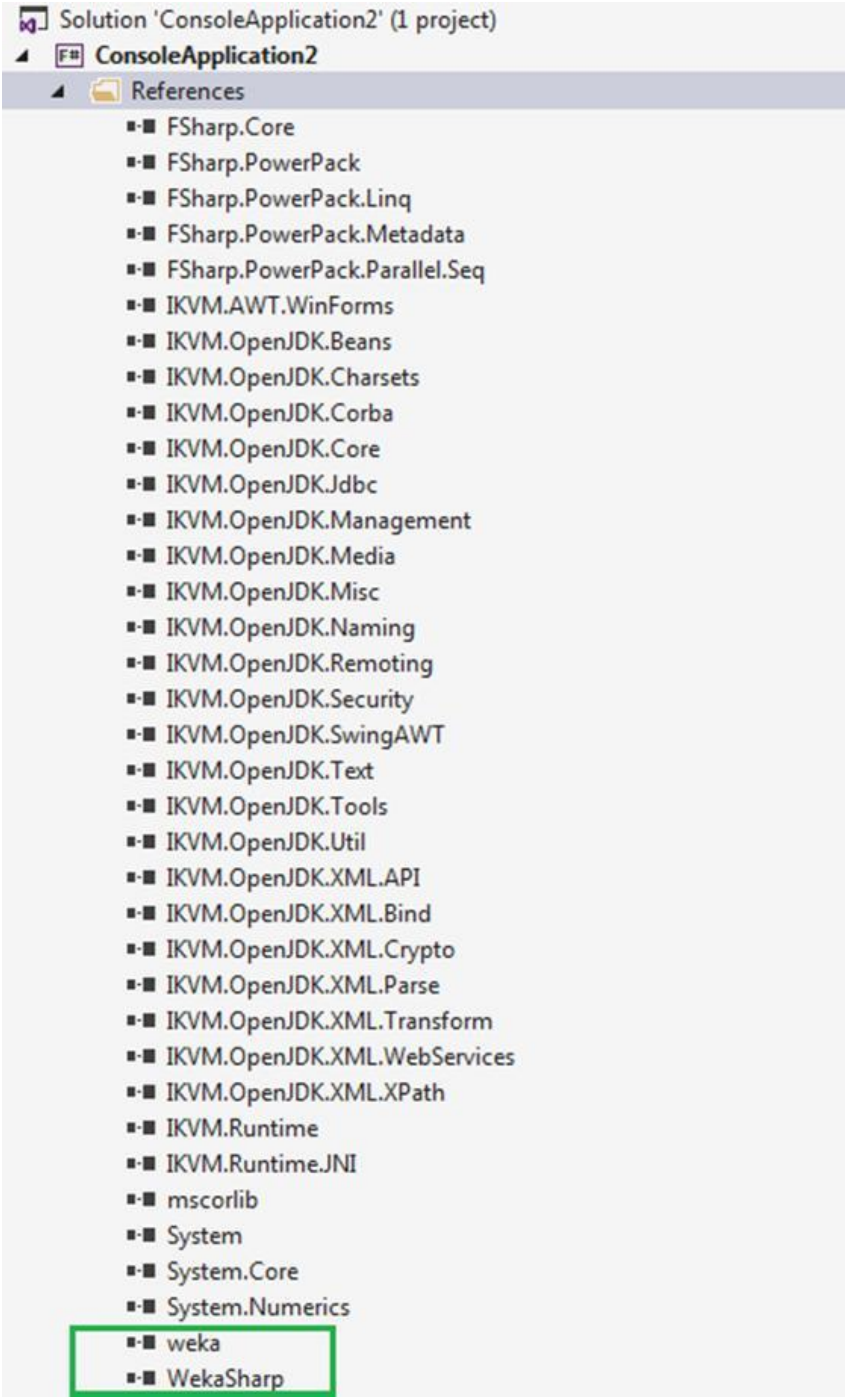
$$\theta_m = \theta - (\alpha/m) * (X^T * (h^T - Y))$$

$$\alpha$$

$$m$$

$$Prediction = \theta^T * X_{unknown}$$


x2
x1
Class 1:
Class 2:
Class 3:
x2
x1
$h_\theta^{(1)}(x)$
x2
x1
$h_\theta^{(2)}(x)$
x2
x1
$h_\theta^{(3)}(x)$


Solution 'ConsoleApplication2' (1 project)
ConsoleApplication2
References
FSharp.Core
FSharp.PowerPack
FSharp.PowerPack.Linq
FSharp.PowerPack.Metadata
FSharp.PowerPack.Parallel.Seq
IKVM.AWT.WinForms
IKVM.OpenJDK.Beans
IKVM.OpenJDK.Charsets
IKVM.OpenJDK.Corba
IKVM.OpenJDK.Core
IKVM.OpenJDK.Jdbc
IKVM.OpenJDK.Management
IKVM.OpenJDK.Media
IKVM.OpenJDK.Misc
IKVM.OpenJDK.Naming
IKVM.OpenJDK.Remoting
IKVM.OpenJDK.Security
IKVM.OpenJDK.SwingAWT
IKVM.OpenJDK.Text
IKVM.OpenJDK.Tools
IKVM.OpenJDK.Util
IKVM.OpenJDK.XML.API
IKVM.OpenJDK.XML.Bind
IKVM.OpenJDK.XML.Crypto
IKVM.OpenJDK.XML.Parse
IKVM.OpenJDK.XML.Transform
IKVM.OpenJDK.XML.WebServices
IKVM.OpenJDK.XML.XPath
IKVM.Runtime
IKVM.Runtime.JNI
mscorlib
System
System.Core
System.Numerics
weka
WekaSharp

```
sepal length,sepal width,petal length,petal width,species
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
```

```
rain_chance,rush_hour,weekday,time,traffic
0.142727454259399,no,no,507,no
0.282271440738007,no,no,520,no
0.585435714379622,yes,yes,515,yes
0.529794960995109,no,yes,545,no
0.832959234636724,yes,yes,436,yes
0.136123508278338,yes,no,506,no
```

X

X

X

X

X

$$tfidf("example", d_2) = tf("example", d_2) \times idf("example", D)$$

$$tfidf("example", d_2) = 3 \times 0.3010 \approx 0.9030$$

$$\log \frac{N}{|\{d \in D : t \in d\}|}$$

D

d_2

P_i

Q_i

H

N

P_i

$H(i)/N$

P

Q

P_i

Q_i

$$d_{Euc} = \sqrt{\sum_{i=1}^{d} | P_i - Q_i |^2}$$

$$\sum_{i=1}^{d} | P_i - Q_i |$$

$$\max_i | P_i - Q_i |$$

$$d_{sor} = \frac{\sum_{i=1}^{d} | P_i - Q_i |}{\sum_{i=1}^{d} (P_i + Q_i)}$$

$$d_{gow} = \frac{1}{d} \sum_{i=1}^{d} \frac{| P_i - Q_i |}{R_i}$$

$$= \frac{1}{d} \sum_{i=1}^{d} | P_i - Q_i |$$

$$d_{sg} = \frac{\sum_{i=1}^{d} | P_i - Q_i |}{\sum_{i=1}^{d} \max(P_i, Q_i)}$$

$$d_{kul} = \frac{\sum_{i=1}^{d} | P_i - Q_i |}{\sum_{i=1}^{d} \min(P_i, Q_i)}$$

$$d_{Can} = \sum_{i=1}^{d} \frac{| P_i - Q_i |}{P_i + Q_i}$$

$$s_{IS} = \sum_{i=1}^{d} \min(P_i, Q_i)$$

$$d_{WH} = \sum_{i=1}^{d} \left(1 - \frac{\min(P_i, Q_i)}{\max(P_i, Q_i)}\right)$$

$$s_{Cze} = \frac{2\sum_{i=1}^{d} \min(P_i, Q_i)}{\sum_{i=1}^{d} (P_i + Q_i)}$$

$$s_{Mot} = \frac{\sum_{i=1}^{d} \min(P_i, Q_i)}{\sum_{i=1}^{d} (P_i + Q_i)}$$

$$s_{Ruz} = \frac{\sum_{i=1}^{d} \min(P_i, Q_i)}{\sum_{i=1}^{d} \max(P_i, Q_i)}$$

$$s_{IP} = P \bullet Q = \sum_{j=1}^{d} P_i Q_i$$

$$s_{HM} = 2\sum_{i=1}^{d} \frac{P_i Q_i}{P_i + Q_i}$$

$$s_{Cos} = \frac{\sum_{i=1}^{d} P_i Q_i}{\sqrt{\sum_{i=1}^{d} P_i^2}\sqrt{\sum_{i=1}^{d} Q_i^2}}$$

$$s_{Jac} = \frac{\sum_{i=1}^{d} P_i Q_i}{\sum_{i=1}^{d} P_i^2 + \sum_{i=1}^{d} Q_i^2 - \sum_{i=1}^{d} P_i Q_i}$$

$$s_{Dice} = \frac{2\sum_{i=1}^{d} P_i Q_i}{\sum_{i=1}^{d} P_i^2 + \sum_{i=1}^{d} Q_i^2}$$

$$s_{Fid} = \sum_{i=1}^{d} \sqrt{P_i Q_i}$$

$$d_B = -\ln \sum_{i=1}^{d} \sqrt{P_i Q_i}$$

$$d_H = 2\sqrt{1 - \sum_{i=1}^{d} \sqrt{P_i Q_i}}$$

$$d_M = \sqrt{2 - 2\sum_{i=1}^{d} \sqrt{P_i Q_i}}\ )$$

$$d_{sqc} = \sum_{i=1}^{d} (\sqrt{P_i} - \sqrt{Q_i})^2$$

$$d_{sqe} = \sum_{i=1}^{d} (P_i - Q_i)^2$$

$$d_{SqChi} = \sum_{i=1}^{d} \frac{(P_i - Q_i)^2}{P_i + Q_i}$$

$$d_P(P,Q) = \sum_{i=1}^{d} \frac{(P_i - Q_i)^2}{Q_i}$$

$$d_N(P,Q) = \sum_{i=1}^{d} \frac{(P_i - Q_i)^2}{P_i}$$

$$d_{PChii} = 2\sum_{i=1}^{d} \frac{(P_i - Q_i)^2}{P_i + Q_i}$$

$$d_{Div} = 2\sum_{i=1}^{d} \frac{(P_i - Q_i)^2}{(P_i + Q_i)^2}$$

$$d_{Clk} = \sqrt{\sum_{i=1}^{d} \left(\frac{|P_i - Q_i|}{P_i + Q_i} \right)^2}$$

$$d_{AdChi} = \sum_{i=1}^{b} \frac{(P_i - Q_i)^2 (P_i + Q_i)}{P_i Q_i}$$

$$d_{KL} = \sum_{i=1}^{d} P_i \ln \frac{P_i}{Q_i}$$

$$d_J = \sum_{i=1}^{d} (P_i - Q_i) \ln \frac{P_i}{Q_i}$$

$$d_{Kdiv} = \sum_{i=1}^{d} P_i \ln \frac{2P_i}{P_i + Q_i}$$

$$d_{Top} = \sum_{i=1}^{d} \left(P_i \ln\left(\frac{2P_i}{P_i + Q_i} \right) + Q_i \ln\left(\frac{2Q_i}{P_i + Q_i} \right) \right)$$

$$d_{JS} = \frac{1}{2} \left[\sum_{i=1}^{d} P_i \ln\left(\frac{2P_i}{P_i + Q_i} \right) + \sum_{i=1}^{d} Q_i \ln\left(\frac{2Q_i}{P_i + Q_i} \right) \right]$$

$$d_{JD} = \sum_{i=1}^{b} \left[\underbrace{\frac{P_i \ln P_i + Q_i \ln Q_i}{2}}_{\text{left}} - \underbrace{\left(\frac{P_i + Q_i}{2} \right) \ln\left(\frac{P_i + Q_i}{2} \right)}_{\text{right}} \right]$$

$$d_{TJ} = \sum_{i=1}^{d} \left(\frac{P_i + Q_i}{2} \right) \ln\left(\frac{P_i + Q_i}{2\sqrt{P_i Q_i}} \right)$$

$$d_{KJ} = \sum_{i=1}^{d} \left(\frac{(P_i^{\,2} - Q_i^{\,2})^2}{2(P_i Q_i)^{3/2}} \right)$$

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

$$S(X,Y) = \frac{|X \cap Y|}{|X \cap Y| + \alpha|X - Y| + \beta|Y - X|}$$

	$y = 1$		
$x = 1$	$\boldsymbol{a}$	$\boldsymbol{b}$	
$x = 0$	$\boldsymbol{c}$	$\boldsymbol{d}$	
		$b + d$	p

$$s_{SS1} = \frac{a}{a + 2b + 2c}$$

$$s_{SS2} = \frac{2a + 2d}{p + a + d}$$

$$s_{SS3} = \frac{1}{4} \cdot \left[\frac{a}{a+b} + \frac{a}{a+c} + \frac{d}{b+d} + \frac{d}{c+d} \right]$$

$$s_{SS4} = \frac{a}{\sqrt{(a+b)(a+c)}} \cdot \frac{d}{\sqrt{(b+d)(c+d)}}$$

$$SMC = \frac{\text{Number of Matching Attributes}}{\text{Number of Attributes}} = \frac{M_{00} + M_{11}}{M_{00} + M_{01} + M_{10} + M_{11}}$$

M_{11}

M_{01}

M_{10}

M_{00}

P

Q

Chapter 5 - Collaborative Filtering

$b_{u,i}$

u

i

$b_{u,i} = \mu$

$$b_{u,i} = \bar{r}_u$$

$$b_{u,i} = \bar{r}_i$$

u

i

$$b_{u,i} = \mu + b_u + b_i$$

$$b_u$$

$$b_i$$

$$b_u = \frac{1}{\mid I_u \mid} \sum_{i \in I_u} (r_{u,i} - \mu)$$

$$b_i = \frac{1}{\mid U_i \mid} \sum_{u \in U_i} (r_{u,i} - b_u - \mu)$$

```
//Non personalized baseline predictor
let baseline (ratings:(float list)list) =
    let mu = ratings |> List.map ( fun ra -> [for i in 0 .. ra.Length - 1 -> ra.[i]]
                                              |> List.filter (fun t -> t <> 0.0)
                                              |> List.average)
                     |> List.average
    let mutable bu = ratings |> List.sumBy (fun rating -> [for i in 0 .. rating.Length - 1 -> rating.[i]]
                                                           |> List.filter (fun ri -> ri <> 0.0)
                                                           |> List.sumBy (fun ri -> ri - mu))

    bu <- bu / float ratings.[0].Length
    let mutable bi = ratings |> List.sumBy (fun ra -> [for i in 0 .. ra.Length - 1 -> ra.[i]]
                                                       |> List.filter (fun t -> t <> 0.0)
                                                       |> List.sumBy (fun z -> z - bu - mu))

    bi <- bi / float ratings.Length
    mu + bu + bi
```

x

y

$$r = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2}\sqrt{\sum_i (y_i - \overline{y})^2}}$$

$\bar{x}$

$\bar{x}$

$\bar{x}$

$\bar{y}$

$\bar{y}$

$\bar{y}$

x_i

$\bar{x}$

y_i

$\bar{y}$

u

```
let x = [1.;2.;3.;4.;5.]
let y = [1.;2.;2.;3.;4.]
let z = [3;4]

let x_bar = List.average x
let y_bar = List.average y

let numerator =
    List.zip x y
    |> List.sumBy (fun item -> (fst item - x_bar)*(snd item - y_bar))

let d1 = x |> List.sumBy(fun xi -> (xi - x_bar) ** 2.0)
let d2 = y |> List.sumBy(fun yi -> (yi - y_bar) ** 2.0)

let denominator = sqrt d1 * sqrt d2

let pearsons = numerator / denominator

printfn "Pearsons Correlation Coefficient is %f" pearsons
```

$$w_{u,v} = \frac{\sum_{i \in I}(r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I}(r_{u,i} - \bar{r}_u)^2}\sqrt{\sum_{i \in I}(r_{v,i} - \bar{r}_v)^2}}$$

$r_{u,i}$

i

$r_{v,i}$

i

v

u

v

v

$\bar{r}_u$

$\bar{r}_v$

	I_1	I_2	I_3	I_4
U_1	4	?	5	5
U_2	4	2	1	
U_3	3		2	4
U_4	4	4		
U_5	2	1	3	5

U_1

i

a

$$P_{a,i} = \bar{r}_a + \frac{\sum_{u \in U}(r_{u,i} - \bar{r}_u) \cdot w_{a,u}}{\sum_{u \in U}|w_{a,u}|}$$

$\bar{r}_a$

i

```
//Average rating for all other rated items by the user
//except the item "except"
let rBaru(u:float list)(except:int)=
    let filtered = u |> List.mapi(fun i j -> if i <> except then j else 0.0)
                     |> List.filter(fun t -> t <> 0.0)
    float ( List.sum filtered ) / float filtered.Length

//The following function finds the common item indices
let commonItemIndices (ratings:(float list)list)(a:int)(u:int)=
        List.zip  ratings.[a]  ratings.[u]
           |> List.mapi (fun index rating ->
                            if fst rating <> 0.0
                                   && snd rating <> 0.0 then index else -1 )
           |> List.filter ( fun index -> index <> -1)

//The following function returns the average of user a and u
let mu_au (ratings:(float list)list)(a:int)(u:int)=
    let com = commonItemIndices ratings a u
    let mu_a = com |> List.map (fun index ->  ratings.[a].[index]) |> List.average
    let mu_u = com |> List.map (fun index ->  ratings.[u].[index]) |> List.average
    (mu_a,mu_u)

//Calculates User-User similarity using Pearson's Correlation Coefficient
let Simu (ratings:(float list)list) (a:int)(u:int)=
        //Indices of the items rated by both user a and u
        let common = commonItemIndices ratings a u
        let averages = mu_au ratings a  u
        let ra = fst averages
        let ru = snd averages
        let num = common |> List.sumBy (fun index -> (ratings.[a].[index] - ra)*
                                                     (ratings.[u].[index] - ru))
        let d1 = common |> List.sumBy (fun index -> (ratings.[a].[index] - ra)** 2.0)
        let d2 = common |> List.sumBy (fun index -> (ratings.[u].[index] - ru)** 2.0)
        //If either d1 or d2 is 0 then we shall hit a divide by zero case
        //to avoid that we must return 0.
        if d1 = 0.0 || d2 = 0.0 then 0.0 else num  / ((sqrt d1) * (sqrt d2 ))
```

i

a

```
//User-User Basic Collaborative Filtering - basic
let Predictu(ratings:(float list)list)(a:int)(i:int) =
    let rb = rBaru ratings.[a] i
    let neighborIndices = ratings
                          |> List.mapi(fun index rating ->
                             if rating.[i] <> 0.0 then index else -1)
                          |> List.filter(fun index -> index <> -1)
    //Rating of neighbors are obtained
    let neighbors  = neighborIndices
                         |> List.map (fun index -> ratings.[index])
    let gaps = neighbors |> List.map (fun neighbor -> neighbor.[i] - (rBaru neighbor i))
    let simis = neighborIndices |> List.map (fun index -> Simu ratings a index)
    let num = List.zip gaps simis |> List.sumBy (fun t -> fst t * snd t)
    let den = simis |>  List.sumBy (fun similarity -> abs similarity)
    if den <> 0.0 then
       let div = num / den
       let predicted = rb + div
       //Sometimes the value of "predicted" can be beyond the range. [1-5]
       //so having a 7 is same as 5.0 in practice (meaning the user might love the item)
       //so is having a -1 which is same as 1 (meaning the user might hate the item)
       if predicted > 5.0 then 5.0 elif predicted < 1.0 then 1.0 else  predicted
    else
       0.0 //We don't know what it is.
```

```
//The above rating matrix is represented as (float list)list in F#
let ratings = [[4.;0.;5.;5.];[4.;2.;1.;0.];[3.;0.;2.;4.];[4.;4.;0.;0.];[2.;1.;3.;5.]]
//Finding the predicted rating for user 1 for item 2
let p12 = Predictu  ratings  0 1
```

N

a

u

	I_1	I_2	I_3	I_4
U_1	4	?	5	5
U_2	4	2	1	
U_3	3		2	4
U_4	4	4		
U_5	2	1	3	5

u

a

u

$$w_{u,v}$$

i

$$(r_{u,i} - \bar{r}_u)$$

$$w_{a,u}$$

Z

Z

$$z = \frac{x - \mu}{\sigma}$$

μ

σ

Z

Z

$$p_{u,i} = \bar{r}_u + \sigma_u \frac{\sum_{u' \in N} s(u,u')(r_{u',i} - \bar{r}_{u'})/\sigma_{u'}}{\sum_{u' \in N} |s(u,u')|}$$

$s(u,v)$

u

v

σ_u

u

```
//Calculates the standard deviation
let stddev(list:float list)=
    sqrt (List.fold (fun acc elem -> acc + (float elem - List.average list) ** 2.0 ) 0.0
                     list / float list.Length)
```

Z

```
//Calculates the z-score
let zscore (ratings:(float list)list)(userIndex:int)(itemIndex:int)=
        let rBar = rBaru ratings.[userIndex] itemIndex
        let sigma = stddev ratings.[userIndex]
        ratings.[userIndex].[itemIndex] - rBar / sigma
```

```
let PredictuZ(ratings:(float list)list)(a:int)(i:int) =
        let rb = rBaru ratings.[a] i
        let neighborIndices = ratings
                                |> List.mapi(fun index rating ->
                                                if rating.[i] <> 0.0
                                                      then index else -1)
                                |> List.filter(fun index -> index <> -1)
        let neighbors  = neighborIndices|> List.map (fun index -> ratings.[index])
        //This line is changed to use Z-Score instead of just the differences.
        let gaps = neighbors |> List.map (fun neighbor ->  zscore ratings a i)
        let simis = neighborIndices |> List.map (fun index -> Simu ratings a index)
        let num = List.zip gaps simis |> List.sumBy (fun t -> fst t * snd t)
        let den = simis |>  List.sumBy (fun t -> abs t)
        if den <> 0.0 then
            let div = num / den
            let predicted = rb + div * stddev ratings.[a]
            //Sometimes the value can be beyond the range.
            //so having a 7 is same as 5.0 in practice
            //so is having a -1 which is same as zero
            if predicted > 5.0 then 5.0 elif predicted < 1.0 then 1.0 else  predicted
        else
            0.0 //Else we don't know
```

Z

```
let SimuDot (ratings :(float list)list) (a:int)(u:int)=
        let num =  List.zip ratings.[a] ratings.[u]
                   |> List.sumBy (fun item -> fst item * snd item)
        let d1 = ratings.[a] |> List.sumBy (fun item -> item  * item )
        let d2 = ratings.[u] |> List.sumBy (fun item -> item  * item )

        if d1 = 0.0 || d2 = 0.0 then 0.0 else num / (sqrt d1 * sqrt d2)
```

$$p_{u,i} = \frac{\sum_{j \in S} s(i,j) r_{u,j}}{\sum_{j \in S} |s(i,j)|}$$

S

i

```
let SimiDot (ratings :(float list)list)(i:int)(j:int)=
        let li = ratings |> List.map(fun rating -> rating.[i])
        let lj = ratings |> List.map(fun rating -> rating.[j])
        let num = List.zip li lj |> List.sumBy  (fun item -> fst item * snd item)
        let d1 = li |> List.sumBy (fun item -> item  * item )
        let d2 = lj |> List.sumBy (fun item -> item  * item )

        if d1 = 0.0 || d2 = 0.0 then 0.0 else num / (sqrt d1 * sqrt d2)
```

i

```
//Item based collaborative filtering - basic
let Predicti (ratings:(float list)list)(userIndex:int)(itemIndex:int)=
    let rated = ratings.[userIndex]
                           |> List.mapi (fun i t ->
                                        if t <> 0.0 then
                                                  i else -1)
                           |> List.filter (fun k -> k <> -1)
    let num = rated |> List.sumBy (fun i -> ratings.[userIndex].[i] *
                                       SimiDot ratings itemIndex i)
    let den = rated |> List.sumBy ( fun i -> abs (SimiDot ratings itemIndex i) )
    let predicted = num / den
    //Predicting something as bad as -1.34 is same as predicting it as 1
    //Similarly predicting something as good as 7.5 is same as predicting it as 5
    //on a 1-5 rating scale.
    //Other than that the ranking might
    if predicted < 0.0 then 1. elif predicted > 5. then 5. else predicted

 let ratings = [[4.;0.;5.;5.];[4.;2.;1.;0.];[3.;0.;2.;4.];[4.;4.;0.;0.];[2.;1.;3.;5.]]
 //pre01 stands for the prediction for user 0 for item 1
 let pre01 = Predicti ratings 0 1
```

```
let ratings = [[4.;0.;5.;5.];[4.;2.;1.;0.];[3.;0.;2.;4.];[4.;4.;0.;0.];[2.;1.;3.;5.]]
//pre01 stands for the prediction for user 0 for item 1
//pre01 stands for the prediction for user 0 and item 1
let pre01i = Predicti ratings 0 1
let pre13i = Predicti ratings 1 3
let pre21i = Predicti ratings 2 1
let pre32i = Predicti ratings 3 2
let pre33i = Predicti ratings 3 3
printfn " Item - Item Collaborative Filtering "
printfn "pre01  = %A" pre01i
printfn "pre13  = %A" pre13i
printfn "pre21  = %A" pre21i
printfn "pre32  = %A" pre32i
printfn "pre33  = %A" pre33i
let pre01u = Predictu ratings 0 1
let pre13u = Predictu ratings 1 3
let pre21u = Predictu ratings 2 1
let pre32u = Predictu ratings 3 2
let pre33u = Predictu ratings 3 3
printfn " User - User Collaborative Filtering "
```

$p_{u,i}$

u

i

$r_{u,i}$

$$\frac{1}{n}\sum_{u,i}|p_{u,i} - r_{u,i}|$$

```
let mae (ratings:float list)(predictions:float list) =
        (List.zip ratings predictions |> List.sumBy (fun t -> abs (fst t - snd t)))
                                /float ratings.Length
```

$$\frac{1}{n(r_{\text{high}} - r_{\text{low}})}\sum_{u,i}|p_{u,i} - r_{u,i}|$$

```fsharp
//Normalized Mean Absolute Error
let nmae (ratings:float list)(predictions:float list) =
    let rMax = ratings |> List.max
    let rMin = ratings |> List.min
    (mae ratings predictions )/(rMax - rMin)
```

$$\sqrt{\frac{1}{n}\sum_{u,i}(p_{u,i} - r_{u,i})^2}$$

```fsharp
//Root mean squared error
let rmse(ratings:float list)(predictions:float list) =
     sqrt( ( List.zip ratings predictions
                  |> List.map (fun t -> fst t - snd t)
                         |> List.sum )
           /(float predictions.Length ))
```

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$$

$$SPC = \frac{TN}{N} = \frac{TN}{FP + TN}$$

$$PPV = \frac{TP}{TP + FP}$$

$$NPV = \frac{TN}{TN + FN}$$

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} = 1 - SPC$$

$$FDR = \frac{FP}{FP + TP} = 1 - PPV$$

$$FNR = \frac{FN}{P} = \frac{FN}{FN + TP}$$

$$ACC = \frac{TP + TN}{P + N}$$

$$F1 = \frac{2TP}{2TP + FP + FN}$$

$$\frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

```
module confusion =
    let TP (matches : int [] [])  =
        matches |> Array.mapi( fun i j -> matches.[i].[i]) |> Array.sum

    //True Positive for entries correctly identified of type "thisOne"
    let TP_for (thisOne : int) (matches : int [] []) =
        matches.[thisOne].[thisOne]

    //False positive for
    let FP_for (thisOne : int) (matches : int [] []) =
        let all = [for i in 0 .. matches.Length-1 -> matches.[i].[thisOne]]
        let allSum = all |> List.sum
        allSum - (TP_for thisOne matches)

    //False negative for
    let FN_for (thisOne : int) (matches : int [] []) =
        let all = [for  i in 0 .. matches.[thisOne].Length - 1 -> matches.[thisOne].[i]]
        let allSum = all |> List.sum
        allSum - matches.[thisOne].[thisOne]

    //F1
    let F1(thisOne : int) (matches : int [] []) =
        2. * float (TP_for thisOne matches) /
        ( 2. * float (TP_for thisOne matches) + float( FP_for thisOne matches )+ float (FN_for thisOne matches))
```

```
open confusion
let matches = [|[|5;3;0|];[|2;3;1|];[|0;2;11|]|]
let cat = 0
let dog = 1
let rabbit = 2

let catF1 = F1 cat matches
let dogF1 = F1 dog matches
let rabbitF1 = F1 rabbit matches
```

$s_1, \ldots, s_n$

$u_1, \ldots, u_n$

$$u_1^p, \ldots, u_n^p$$

$$u_i$$

$$u_i^p$$

$$\bar{u}$$

$$u_1, \ldots, u_n$$

$$\bar{u}^p$$

$$u_1^p, \ldots, u_n^p$$

$$\rho = \frac{\sum_{i=1} \quad (u_i - \bar{u})(u_i^p - \bar{u^p})}{n \cdot stdev(u) \cdot stdev(u^p)}.$$

```
let trueRanks = [1.;2.;3.;4.;5.;5.;7.;8.;9.;10.]
let predictedRanks = [1.;2.;3.;4.;6.;7.;5.;8.;10.;9.]
let uBar  =  trueRanks  |> List.average

let upBar = predictedRanks |> List.average

let stdevTrue  = stddev trueRanks
let stdevPredicted = stddev predictedRanks

let n = 10.

let numerator = List.zip trueRanks predictedRanks
                    |> List.map (fun item -> (float (fst item) - uBar) * (float (snd item) - upBar))
                    |> List.sum

let denominator = n * stdevTrue * stdevPredicted

let p = numerator /denominator
```

$$N$$

$$N$$

Chapter 6 - Sentiment Analysis

```
type SentiWordNetEntry = {POS:string; ID:string; PositiveScore:string; NegativeScore:string; Words:string}
```

```
let sentiWordList = System.IO.File.ReadAllLines(@"SentiWordNet_3.0.0_20130122.txt")
                                 |> Array.filter (fun line -> not (line.StartsWith("#")))
                                 |> Array.map (fun line -> line.Split '\t')
                                 |> Array.map (fun lineTokens -> {POS = lineTokens.[0];
                                                                  ID = lineTokens.[1];
                                                                  PositiveScore = lineTokens.[2].Trim();
                                                                  NegativeScore = lineTokens.[3].Trim();
                                                                  Words = lineTokens.[4]})

                                |> Array.map(fun item ->  [item.Words.Substring(0,item.Words.LastIndexOf('#')+1);
                                                            item.PositiveScore;item.NegativeScore])


let getPolarity (sentiWordNetList:string list[]) (word:string) =
    let matchedItem = sentiWordNetList
                      |> Array.filter(fun item -> item.[0].Contains (word))

    match matchedItem.Length  with
        | 0 -> (0.0,0.0)//No value found
        //There can be multiple match; picking the first one (i.e: matchedItem.[0])
        | _ -> (float matchedItem.[0].[1], float matchedItem.[0].[2])


let getPolarityScore (sentence:string) (sentiWordNetList:string list[]) =
    let words = sentence.Split ' '
    let mutable totalPositivity = 0.0
    let mutable totalNegativity = 0.0
    let polarities  = words
                      |> Array.map(fun word -> getPolarity sentiWordNetList word)
    polarities
            |> Array.map (fun polarity -> totalPositivity <- totalPositivity + fst polarity)
            |> ignore

    polarities
            |> Array.map (fun polarity -> totalNegativity <- totalNegativity + snd polarity)
            |> ignore

    if totalPositivity > totalNegativity then 1  //Positive polarity
    elif totalNegativity = totalPositivity then 0  //Neutral polarity
    else -1 //Negative polarity

//Finding polarities of the sentences using SentiWordNet
getPolarityScore "I am loving this product.I thought that the camera will be much better" sentiWordList
getPolarityScore "don't buy this drug . it gave me a bummer" sentiWordList //negative

let allPositiveWords (sentiWordNetList:string list[])=
    sentiWordNetList
      |> Array.filter(fun sentiWord ->  float sentiWord.[1] > float sentiWord.[2])
      |> Array.map (fun sentiWord -> sentiWord.[0])

let allNegativeWords (sentiWordNetList:string list[])=
    sentiWordNetList
      |> Array.filter(fun sentiWord ->  float sentiWord.[1] < float sentiWord.[2])
      |> Array.map (fun sentiWord -> sentiWord.[0])
```

```
let getPolarity (sentiWordNetList:string list[]) (word:string) =
    let wordWithHash = String.concat "" [word; "#"]
    let wordWithLeadingBlankAndHash = String.concat "" [" ";wordWithHash]
    let matchedItem = sentiWordNetList
                      |> Array.filter(fun item -> item.[0].ToString().StartsWith(wordWithHash)
                                                  || item.[0].ToString().Contains wordWithLeadingBlankAndHash)

    match matchedItem.Length  with
        | 0 -> if word = "Negative_detected" then (0.0,0.675)
               elif word = "Ok_detected" then (0.125,0.0)
               else (0.0,0.0)//No value found
        //There can be multiple match
        | _ -> (float matchedItem.[0].[1], float matchedItem.[0].[2])
```

```
let negations = ["no";"not";"never";"seldom";"neither";"nor"]
let badCombos = negations
                |> List.collect (fun x -> posList |> List.map (fun y -> x + " " + y))
let okCombos = negations
                |> List.collect (fun x -> negList |> List.map (fun y -> x + " " + y))
```

```
let mutable sen = "the camera of the phone was not amazing"
badCombos |> List.map (fun badWordCombo -> sen <- Regex.Replace (sen, badWordCombo,"Negative_detected"))
          |> ignore
okCombos |> List.map (fun badWordCombo -> sen <- Regex.Replace (sen, badWordCombo,"Ok_detected"))
         |> ignore
```

$$SO(w) = \sum_{w_p \in positive-words} A(w, w_p) - \sum_{w_n \in negative-words} A(w, w_n)$$

$$A(word1, word2)$$

$$W_p$$

$$W_n$$

$$A(word1, word2)$$

$$A\,()$$

$$\mathrm{PMI}\,(word_1, word_2) = \log_2\left(\frac{\mathrm{p}(word_1 \;\&\; word_2)}{\mathrm{p}(word_1)\;\mathrm{p}(word_2)}\right)$$

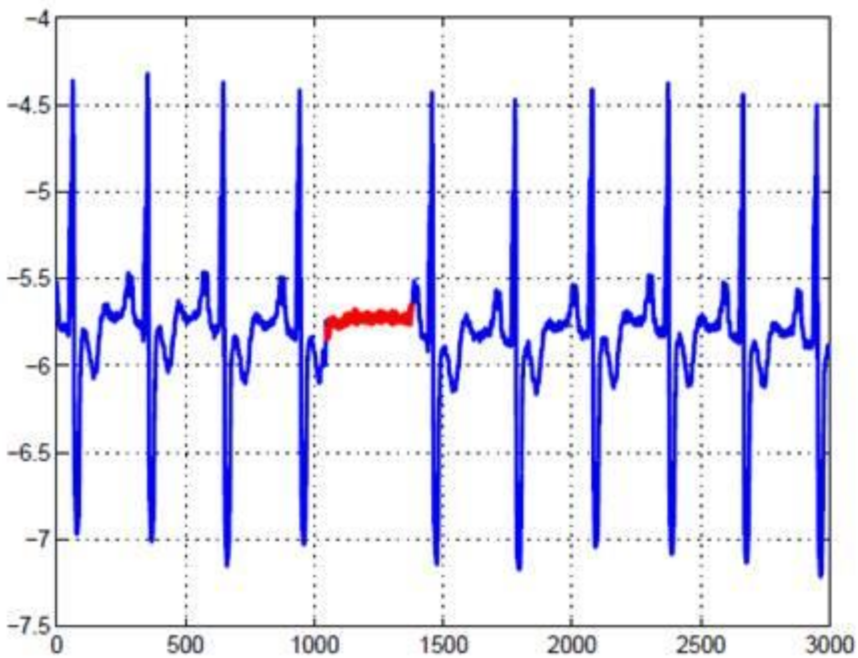
$$p\,(word)$$

$$p(word_1 \& word_2)$$

```
let prob list word =
    let matchCount =  list |> List.filter (fun z -> z |> List.contains word)
                                                      |> List.length |> float
    matchCount / float list.Length


let probBoth list w1 w2 =
    let matchCount = list |>List.filter (fun z -> z |> List.contains w1 && z |> List.contains w2 )
                          |>List.length |> float
    matchCount / float list.Length

let pmi docs w1 w2 =
    let numerator = probBoth docs w1 w2
    let denominator = (prob docs w1) * (prob docs w2)
    if denominator > 0.0 && numerator > 0.0 then log (numerator / denominator) else 0.0


//List of positive words
let pWords = ["good"; "nice"; "excellent"; "positive"; "fortunate";
              "correct"; "superior"]

//List of negative words
let nWords = ["bad"; "nasty"; "poor"; "negative"; "unfortunate"; "wrong"; "inferior"]
let mutable posi = 0.0 //Total positive semantic orientation
let mutable negi = 0.0 //Total negative semantic orientation

let docs =[
           [["positive";"outlook"];["good";"service"];["nice";"people"];["bad";"location"]];//Bank1
           [["nasty";"behaviour"];["unfortunate"; "outcome"];["poor";"quality"]]//Bank2
          ]
for i in 0 .. docs.Length - 1 do
    for j in 0 .. docs.[i].Length - 1    do
        for pw in pWords do
             posi <- posi + pmi docs docs.[i].[j] pw

for i in 0 .. docs.Length - 1 do
    for j in 0 .. docs.[i].Length - 1    do
        for pw in nWords do
            negi <- negi + pmi docs docs.[i].[j] pw

let so_pmi = posi - negi //Calculating semantic orientation's value

let calculateSO (docs:string list list)(words:string list)=
    let mutable res  = 0.0
    for i in 0 .. docs.Length - 1 do
        for j in 0 .. docs.[i].Length - 1    do
            for pw in words do
                 res <- res + pmi docs docs.[i].[j] pw
    res
```

```
let soPMI ( reviews : string list list list )=
    let mutable posi = 0.0
    let mutable negi = 0.0
    reviews |> List.map (fun docs ->
                        posi <- calculateSO docs pWords
                        negi <- calculateSO docs nWords
                        (docs,   posi - negi))
```

Chapter 7 - Anomaly Detection



```
//Finds the median
let median numbers =
     let sorted =  List.sort numbers
     let n = float numbers.Length
     let x = int (n/2.)
     let mutable result = 0.0
     if (float numbers.Length) % 2. = 0.0 then result <- float (numbers.[x] +
                                                        numbers.[x-1]) / 2.0
                                          else result <- float numbers.[x]
     result
//Finds the inter quartile range
let getIQRRange numbers  =
    let med = median numbers
    let smaller = numbers |> List.filter (fun item -> item < med)
    let bigger  = numbers |> List.filter (fun item -> item > med)
    let q1 = median smaller
    let q3 = median bigger
    let iqr = q3 - q1
    (q1-1.5 * iqr, q3 + 1.5*iqr )

//Find the indices where the outliers occur
let findOutliers numbers =
    let iqrRange = getIQRRange numbers
    numbers |> List.mapi (fun index item -> if item < fst iqrRange || item > snd iqrRange
                                            then index else -1)
            |> List.filter (fun index -> index <> -1)
```

x

z

$$\mathbf{z} = \frac{|\mathbf{x} - \bar{\mathbf{x}}|}{S}$$

$\bar{\mathbf{x}}$

S

z

```
let stdDevList list =
    let avg = List.average list
    sqrt (List.fold (fun acc elem -> acc + (float elem - avg) ** 2.0 ) 0.0 list
                         / float list.Length)

let zScores xs =
    let x_bar  = List.average xs
    let s = stdDevList xs
    let scores = xs |>List.map ( fun x -> abs (x - x_bar) / s )
    scores
```

$$z > \frac{N-1}{\sqrt{N}} \sqrt{\frac{t^2_{\alpha/(2N),N-2}}{N-2+t^2_{\alpha/(2N),N-2}}}$$

$$t_{\alpha/(2N),N-2}$$

z

$$t_{\alpha/(2N),N-2}$$

```
let findAnomalies (xs:float list) t =
    let n = float xs.Length
    let threshold = (( n - 1.)/(sqrt  n)) * sqrt ( t ** 2. / ( n - 2. + t ** 2.))
    let z_scores = zScores xs
    xs |> List.mapi (fun i x -> if z_scores.[i] > threshold then i else - 1 )
       |> List.filter (fun z -> z <> -1)
```

$$y^2 = (\mathbf{x} - \bar{\mathbf{x}})' S^{-1} (\mathbf{x} - \bar{\mathbf{x}})$$

S

x

x

```
//Converting multivariate data to univariate data
//so that Grubb's test can be used.
let toUnivariate (xs:(float list)list) =
    let s = getCovarianceMatrix xs
    let x_bar = meanOf xs
    let mats = xs |> List.map (fun x -> (x, DenseMatrix.ofRowList[x] -
                                           DenseMatrix.ofRowList [x_bar]))
    mats |> List.map (fun elem -> (fst elem, (((snd elem) * s.Inverse()) *
                                            (snd elem).Transpose()).At(0,0)))
```

```
#load "...\packages\MathNet.Numerics.FSharp.3.10.0\MathNet.Numerics.fsx"
open MathNet.Numerics.LinearAlgebra

//Returns the mean value of each column
let meanOf(x:(float list)list)=
    let k = x.[0].Length - 1
    let n = x.Length - 1
    let revs = [for i in 0 .. n -> [0 .. k] |> List.map(fun t -> x.[i].[t])]
    [0 .. k]|>List.map (fun k ->  List.average revs.[k])

//Gets the covariance matrix of the given matrix
let getCovarianceMatrix (x:(float list)list)=
    let n = x.Length //Number of rows
    let k = x.[0].Length//Number of columns
    let mean = meanOf(x)//Mean of the rows returns a vector of k elements
    //repmats is the repetition of mean row n times
    let repmats = DenseMatrix.ofRowList [for i in 0 .. n - 1 -> mean]
    let xC = (DenseMatrix.ofRowList x) - repmats
    let covMat = (xC.Transpose() * xC).DivideByThis(float n)
    covMat
```

```
let ys = toUnivariate [[2.;2.];[2.;5.];[6.;5.];[100.;345.]]
printfn "ys = %A" ys
```

x

x_k

$\hat{x}$

$\hat{x}$

$$\Sigma := \frac{1}{m}\sum_{k=1}^{m}(x_k - \hat{x})(x_k - \hat{x})^T.$$

$(x_k - \hat{x})$

χ^2

χ^2

$$\chi^2 = \sum_{i=1}^{n} \frac{(X_i - E_i)^2}{E_i}$$

X_i

E_i

χ^2

χ^2

```
let chiSquareStatistic xs es =
    List.zip xs es
      |> List.map (fun elem -> (fst elem,( (fst elem - snd elem ) ** 2.0)
                                        /(fst elem)))
```

χ^2

x

m

j

x

x

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$$

$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$

$$p(x) = \prod_{j=1}^{n} p(x_j;\mu_j,\sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j-\mu_j)^2}{2\sigma_j^2}\right)$$

$p\left(x\right)$

```
//Calculates mu j
let mu(x:(float list)list)(j:int)=
    x |> List.map ( fun xrow -> xrow.[j])
      |> List.average

//The following function finds the square of the standard deviation
//of the jth feature: Calculates sigma squared j
let sigmaSqr(x:(float list)list)(j:int)=
     x |> List.map (fun xrow -> (xrow.[j] - mu x j) ** 2.0)
       |> List.average

//Calculates the product of the probabilities
//for each feature.
let px (trainingSet:(float list)list)(xtest:float list)=
    let n = trainingSet.Length
    let root2pi = sqrt ( 2.0 * 3.14159)

    let probs  = [for i in 0 .. n - 1 -> (1./root2pi * sqrt(sigmaSqr trainingSet i))
                                          (2.0 * sigmaSqr trainingSet i))]
    let mutable pxValue = 1.0
    probs |> List.map (fun z -> pxValue <- pxValue * z) |> ignore
    pxValue

let data = [1;45;1;3;54;1;45;24;5;23;5;5]
let windowSize = 3
let series  = [for i in 0 .. data.Length-windowSize ->
                 data |> Seq.skip i |> Seq.take 3 |> Seq.toList]
```

N