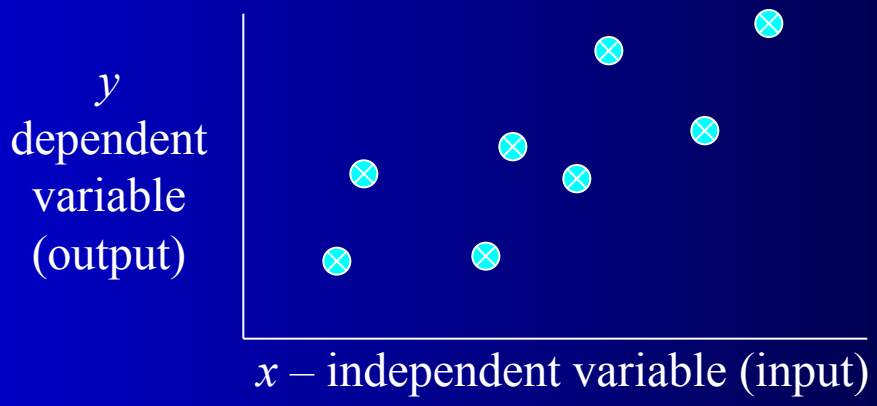
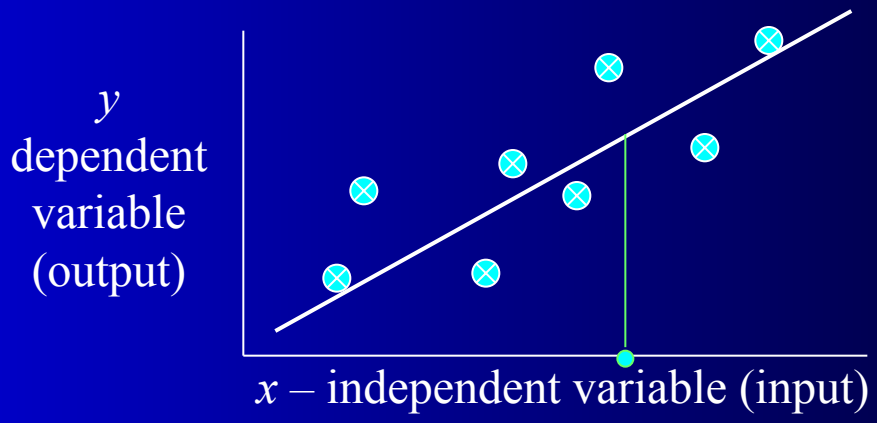
# Regression

- For classification the output(s) is nominal
- In regression the output is continuous
  - Function Approximation
- Many models could be used – Simplest is linear regression
  - Fit data with the best hyper-plane which "goes through" the points



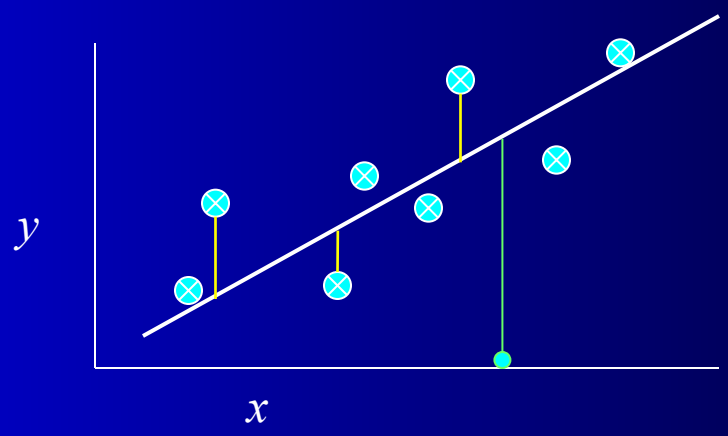
# Regression

- For classification the output(s) is nominal
- In regression the output is continuous
  - Function Approximation
- Many models could be used – Simplest is linear regression
  - Fit data with the best hyper-plane which "goes through" the points



# Regression

- For classification the output(s) is nominal
- In regression the output is continuous
  - Function Approximation
- Many models could be used – Simplest is linear regression
  - Fit data with the best hyper-plane which "goes through" the points
  - For each point the difference between the predicted point and the actual observation is the *residue*



# Simple Linear Regression

- For now, assume just one (input) independent variable $x$, and one (output) dependent variable $y$
  - Multiple linear regression assumes an input vector **x**
  - Multivariate linear regression assumes an output vector **y**
- We "fit" the points with a line (i.e. hyper-plane)
- Which line should we use?
  - Choose an objective function
  - For simple linear regression we choose sum squared error (SSE)
    - $\Sigma\ (predicted_i - actual_i)^2 = \Sigma\ (residue_i)^2$
  - Thus, find the line which minimizes the sum of the squared residues (e.g. least squares)
  - This exactly mimics the case assuming data points were sampled from the actual hyperplane with Gaussian noise added

# How do we "learn" parameters

- For the 2-*d* problem (line) there are coefficients for the bias and the independent variable (*y*-intercept and slope)

$$Y = \beta_0 + \beta_1 X$$

- To find the values for the coefficients which minimize the objective function we take the partial derivates of the objective function (SSE) with respect to the coefficients. Set these to 0, and solve.

$$\beta_1 = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - \left(\sum x\right)^2}$$

$$\beta_0 = \frac{\sum y - \beta_1 \sum x}{n}$$

# Multiple Linear Regression

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n$$

- There is a closed form for finding multiple linear regression weights which requires matrix inversion, etc.
- There are also iterative techniques to find weights
- One is the delta rule. For regression we use an output node which is not thresholded (just does a linear sum) and iteratively apply the delta rule – *For regression net is the output*

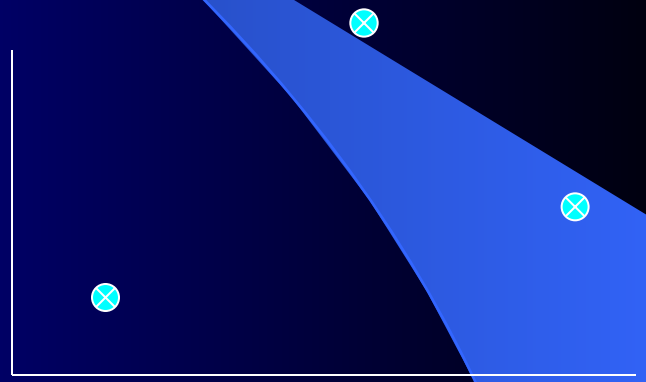
$$\Delta w_i = c(t - net)x_i$$

- Where $c$ is the learning rate and $x_i$ is the input for that weight
- Delta rule will update towards the objective of minimizing the SSE, thus solving multiple linear regression
- There are other regression approaches that give different results by trying to better handle outliers and other statistical anomalies

# SSE and Linear Regression

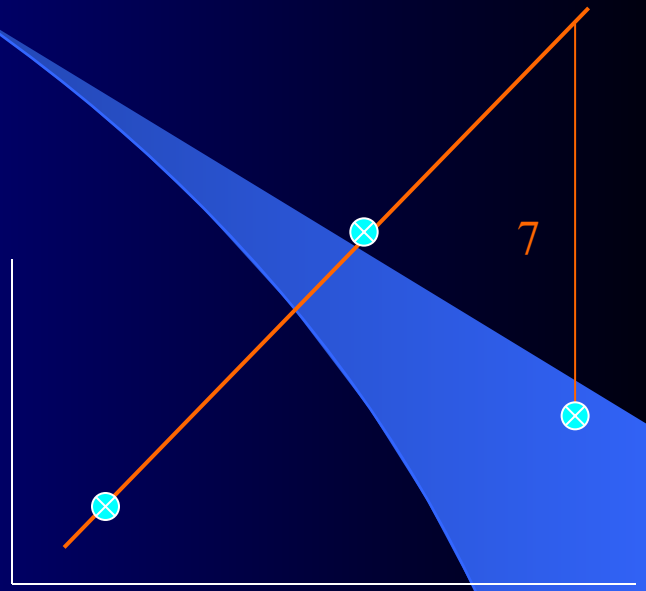
- SSE chooses to square the difference of the predicted vs actual.
- Don't want residues to cancel each other
- Could use absolute or other distances to solve problem

  $\Sigma\ |predicted_i - actual_i|$ : L1 vs L2
- SSE leads to a parabolic error surface which is good for gradient descent
- Which line would least squares choose?
  - There is *always one* "best" fit with SSE (L2)



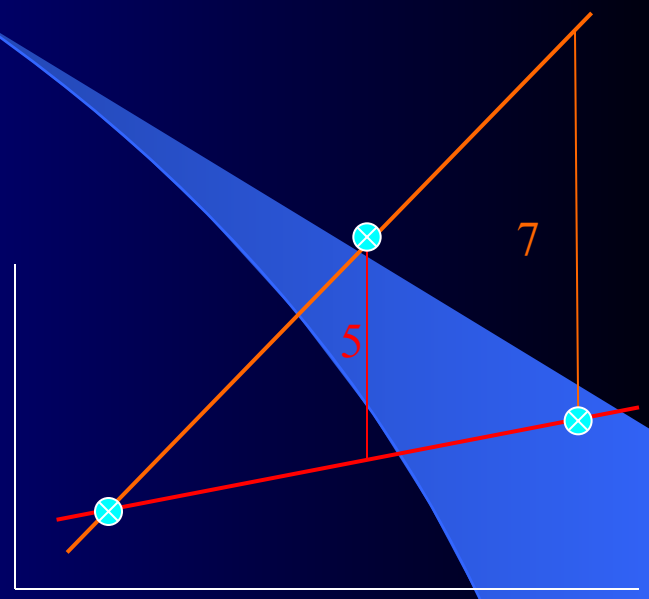
# SSE and Linear Regression

- SSE leads to a parabolic error surface which is good for gradient descent
- Which line would least squares choose?
  - There is always one "best" fit



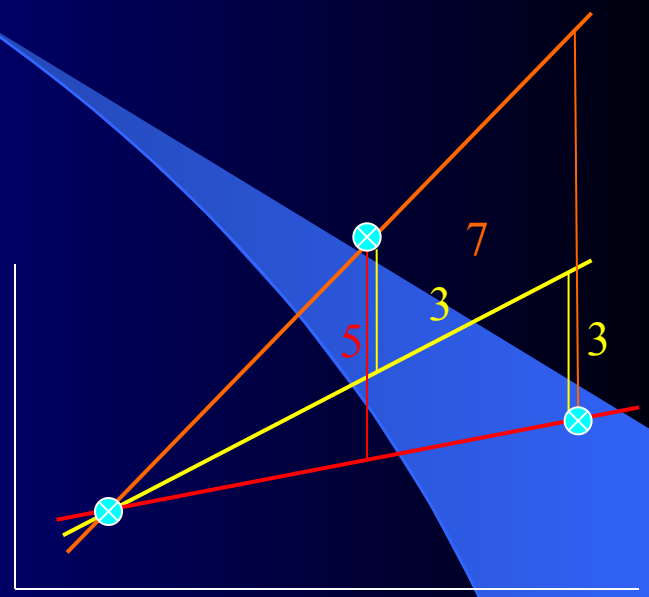
# SSE and Linear Regression

- SSE leads to a parabolic error surface which is good for gradient descent
- Which line would least squares choose?
  - There is always one "best" fit



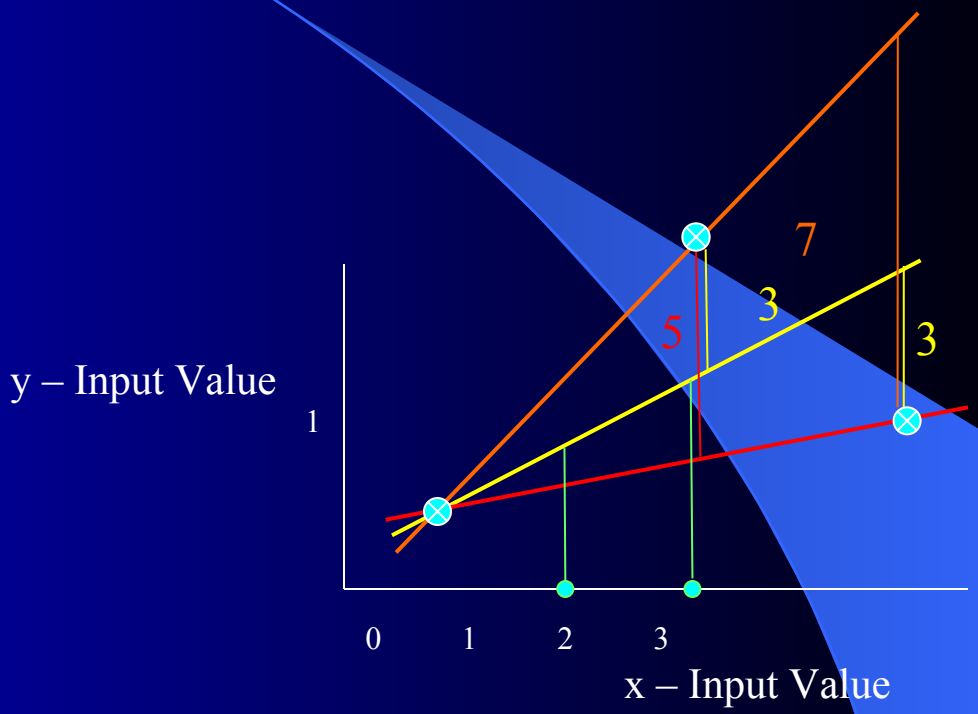
# SSE and Linear Regression

- SSE leads to a parabolic error surface which is good for gradient descent
- Which line would least squares choose?
  - There is always one "best" fit
- Note that the squared error causes the model to be more highly influenced by outliers
  - Though best fit assuming Gaussian noise error from true surface



# SSE and Linear Regression Generalization

- In generalization all $x$ values map to a $y$ value on the chosen regression line



# Linear Regression - Challenge Question

$$\Delta w_i = c(t - net)x_i$$

- Assume we start with all weights as 1 (don't use bias weight though you usually always will – else forces the line through the origin)
- Remember for regression we use an output node which is not thresholded (just does a linear sum) and iteratively apply the delta rule – *thus the net is the output*
- What are the new weights after one iteration through the following training set using the delta rule with a learning rate $c = 1$
- How does it generalize for the novel input (-.3, 0)?

| $x_1$ | $x_2$ | *Target y* |
|---|---|---|
| .5 | -.2 | 1 |
| 1 | 0 | -.4 |

- After one epoch the weight vector is:
  - A. 1 .5
  - B. 1.35 .94
  - C. 1.35 .86
  - D. .4 .86
  - E. None of the above

# Linear Regression - Challenge Question

$$\Delta w_i = c(t - net)x_i$$

- Assume we start with all weights as 1
- What are the new weights after one iteration through the training set using the delta rule with a learning rate $c = 1$
- How does it generalize for the novel input (-.3, 0)?

| $x_1$ | $x_2$ | *Target* | *Net* | $w_1$ | $w_2$ |
|---|---|---|---|---|---|
| | | | | 1 | 1 |
| .5 | -.2 | 1 | | | |
| 1 | 0 | -.4 | | | |

$w_1 = 1 +$

# Linear Regression - Challenge Question

$$\Delta w_i = c(t - net)x_i$$

- Assume we start with all weights as 1
- What are the new weights after one iteration through the training set using the delta rule with a learning rate $c = 1$
- How does it generalize for the novel input (-.3, 0)?
  - -.3*-.4 + 0*.86 = .12

| $x_1$ | $x_2$ | *Target* | *Net* | $w_1$ | $w_2$ |
|---|---|---|---|---|---|
| | | | | 1 | 1 |
| .5 | -.2 | 1 | .3 | 1.35 | .86 |
| 1 | 0 | -.4 | 1.35 | -.4 | .86 |

$$w_1 = 1 + 1(1 - .3).5 = 1.35$$

# Linear Regression Homework

$$\Delta w_i = c(t - net)x_i$$

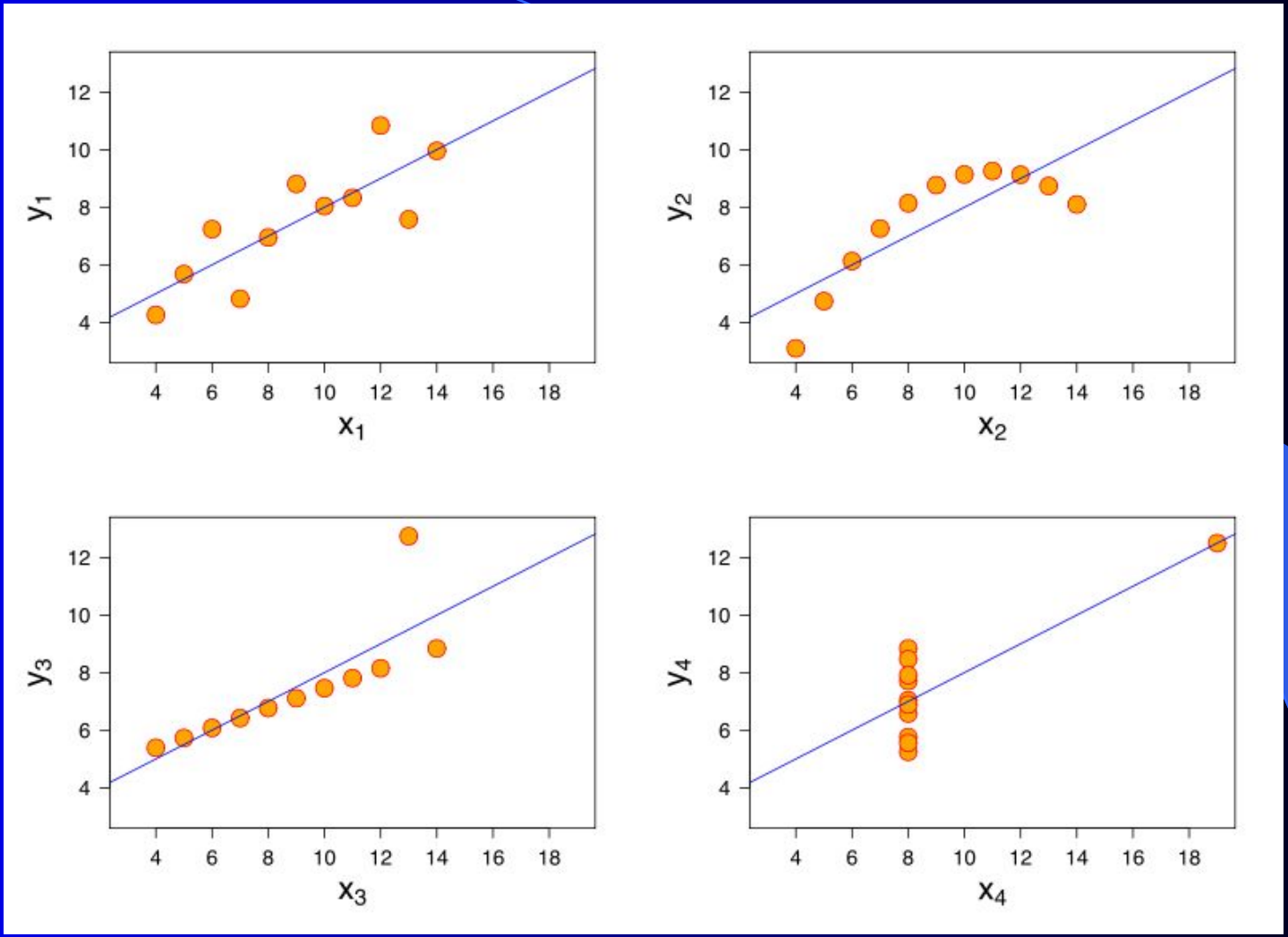
- Assume we start with all weights as 0 (Include the bias!)
- What are the new weights after one iteration through the following training set using the delta rule with a learning rate $c = .2$
- How does it generalize for the novel input (1, .5)?

| $x_1$ | $x_2$ | *Target* |
|---|---|---|
| .3 | .8 | .7 |
| -.3 | 1.6 | -.1 |
| .9 | 0 | 1.3 |

# Intelligibility (Interpretable ML, Transparent)

- One advantage of linear regression models (and linear classification) is the potential to look at the coefficients to give insight into which input variables are most important in predicting the output
- The variables with the largest magnitude have the highest correlation with the output
  - A large positive coefficient implies that the output will increase when this input is increased (positively correlated)
  - A large negative coefficient implies that the output will decrease when this input is increased (negatively correlated)
  - A small or 0 coefficient suggests that the input is uncorrelated with the output (at least at the $1^{st}$ order)
- Linear regression/classification can be used to find best "indicators"
  - Be careful not to confuse correlation with causality
  - Linear cannot detect higher order correlations!! The power of more complex machine learning models.

# Anscombe's Quartet



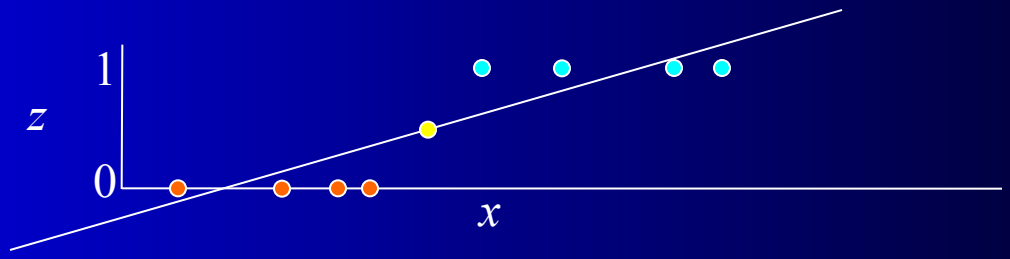

What lines "really" best fit each case? – different approaches

# Delta rule natural for regression, not classification

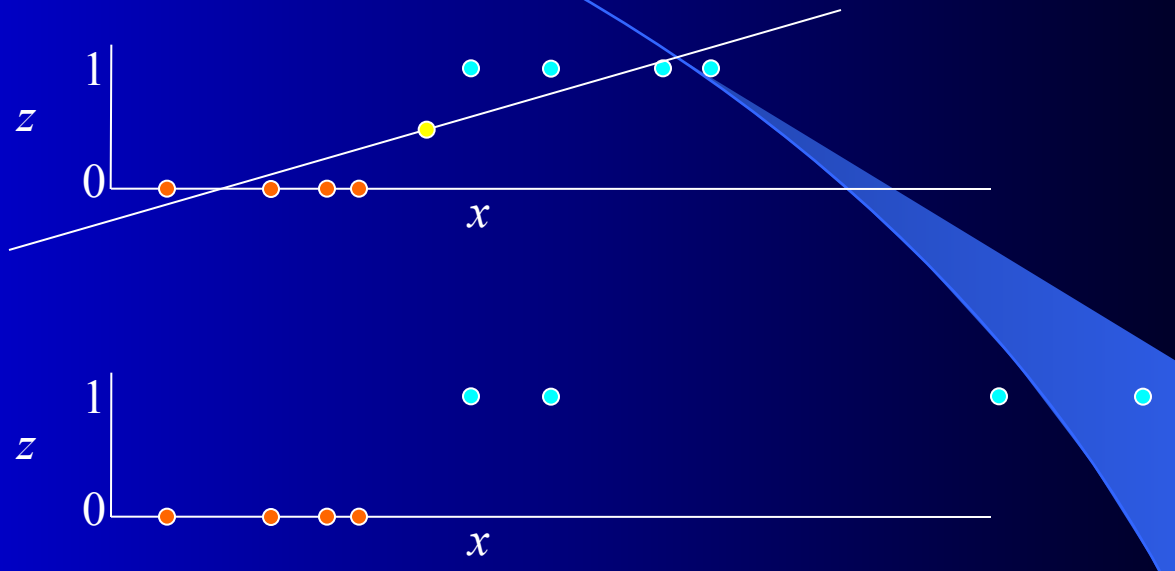
$$\Delta w_i = c(t - net)x_i$$

- First consider the one-dimensional case
- The decision surface for the perceptron would be any (first) point that divides instances


- Delta rule will try to fit a line through the target values which minimizes SSE and the decision point is where the line crosses .5 for 0/1 targets. Looking down on data for perceptron view. Now flip it on its side for delta rule view.
- Will converge to the one optimal line (and dividing point) for this objective

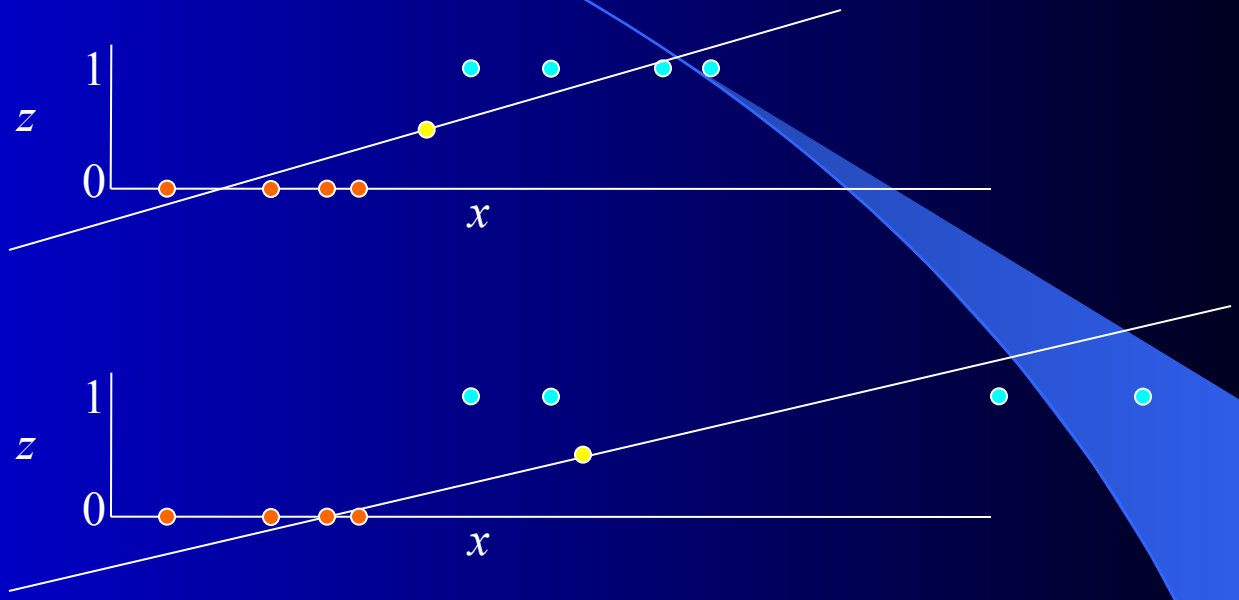


# Delta Rule for Classification?



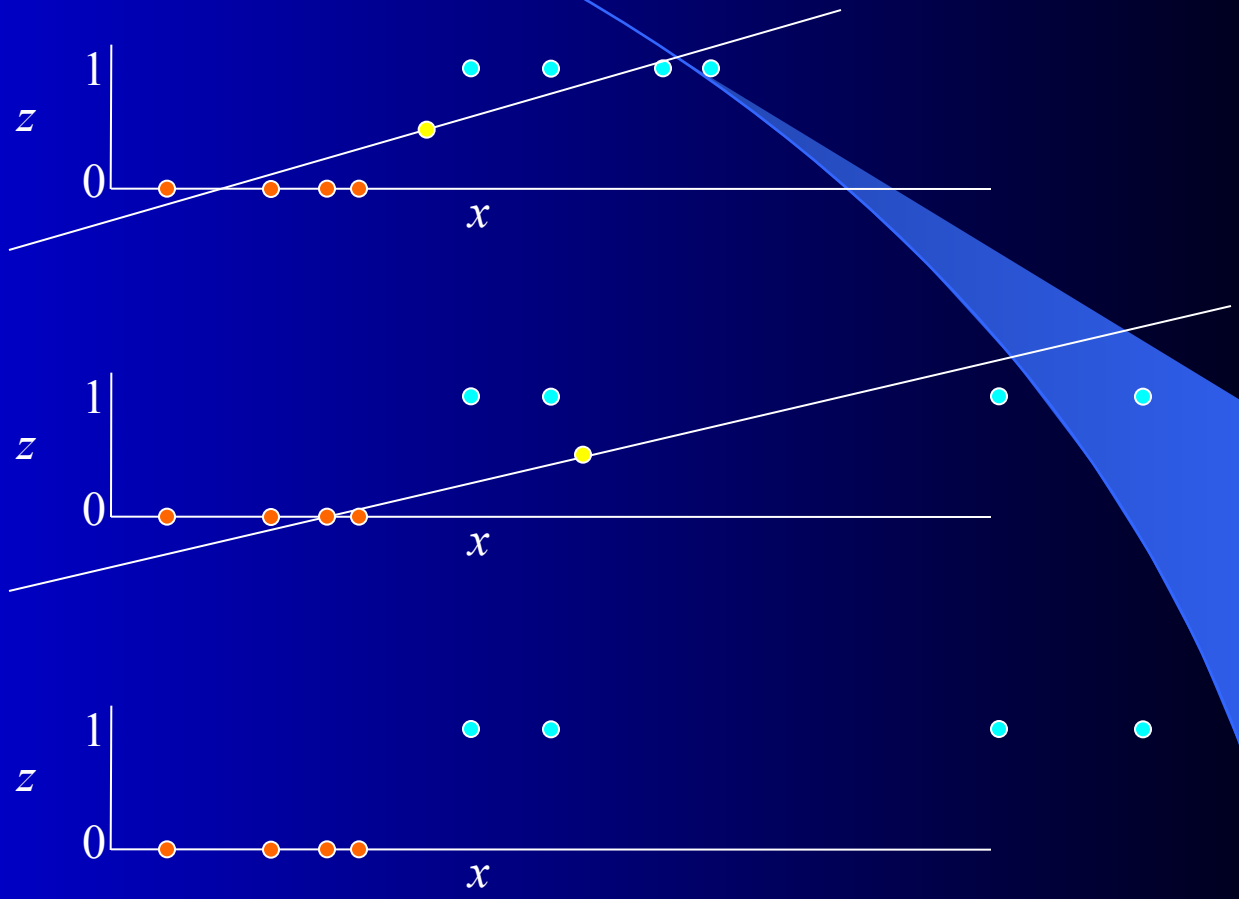

- What would happen in this adjusted case for perceptron and delta rule and where would the decision point (i.e. .5 crossing) be?

# Delta Rule for Classification?



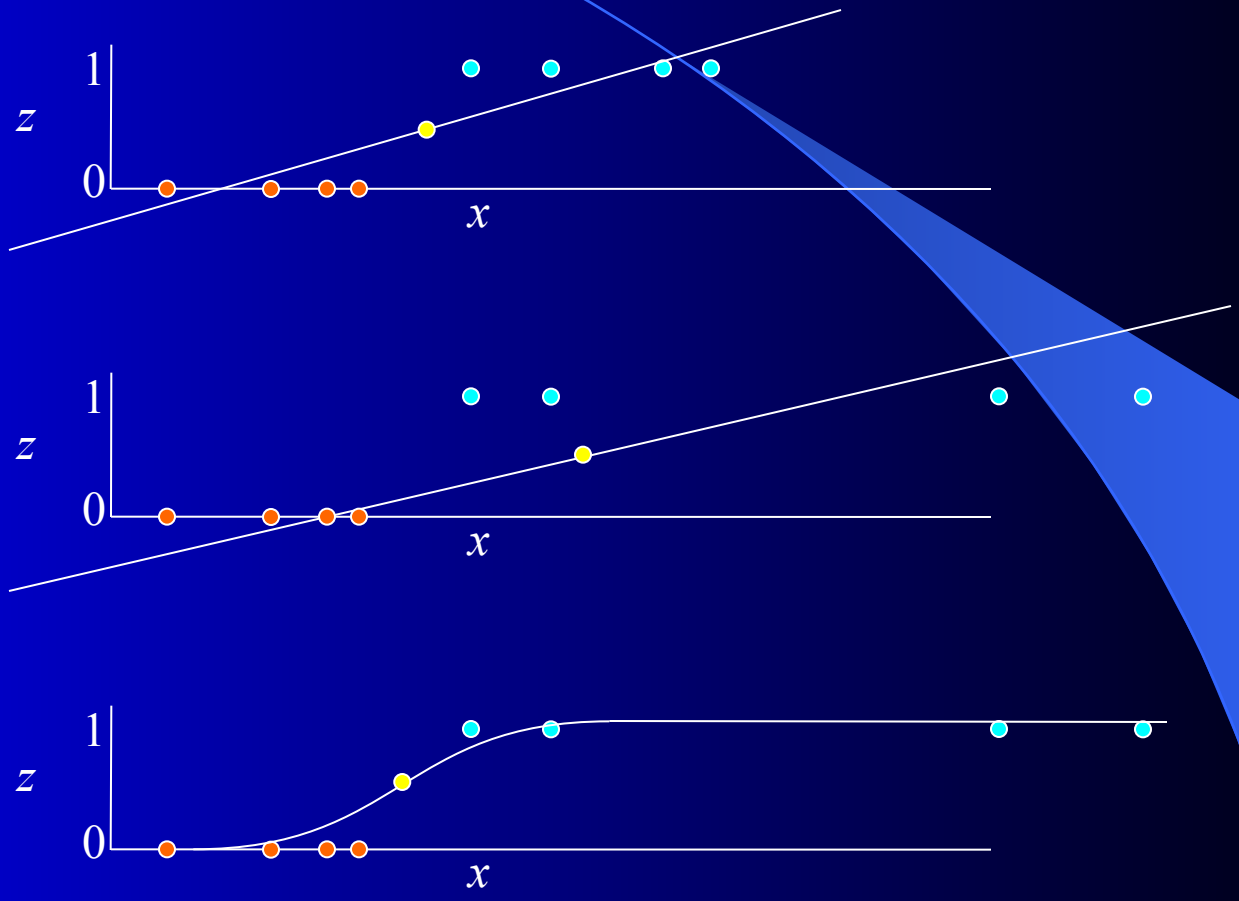

- Leads to misclassifications even though the data is linearly separable
- For Delta rule the objective function is to minimize the regression line SSE, not maximize classification

# Delta Rule for Classification?




- What would happen if we were doing a regression fit with a sigmoid/logistic curve rather than a line?

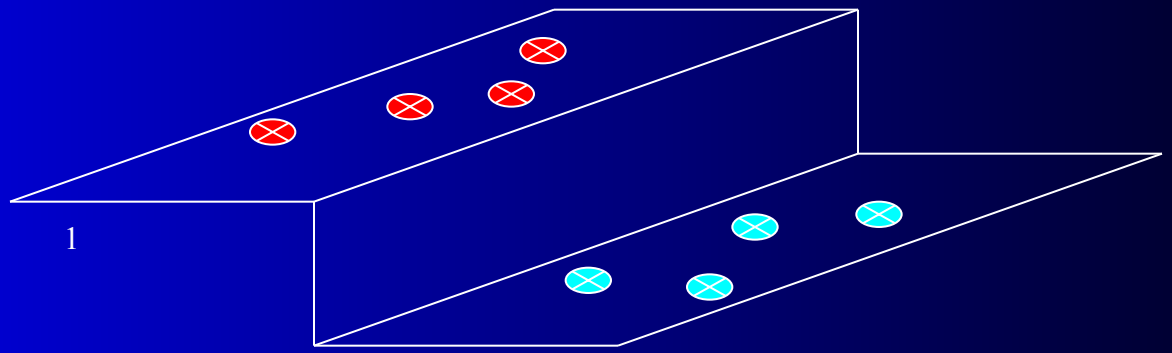
# Delta Rule for Classification?

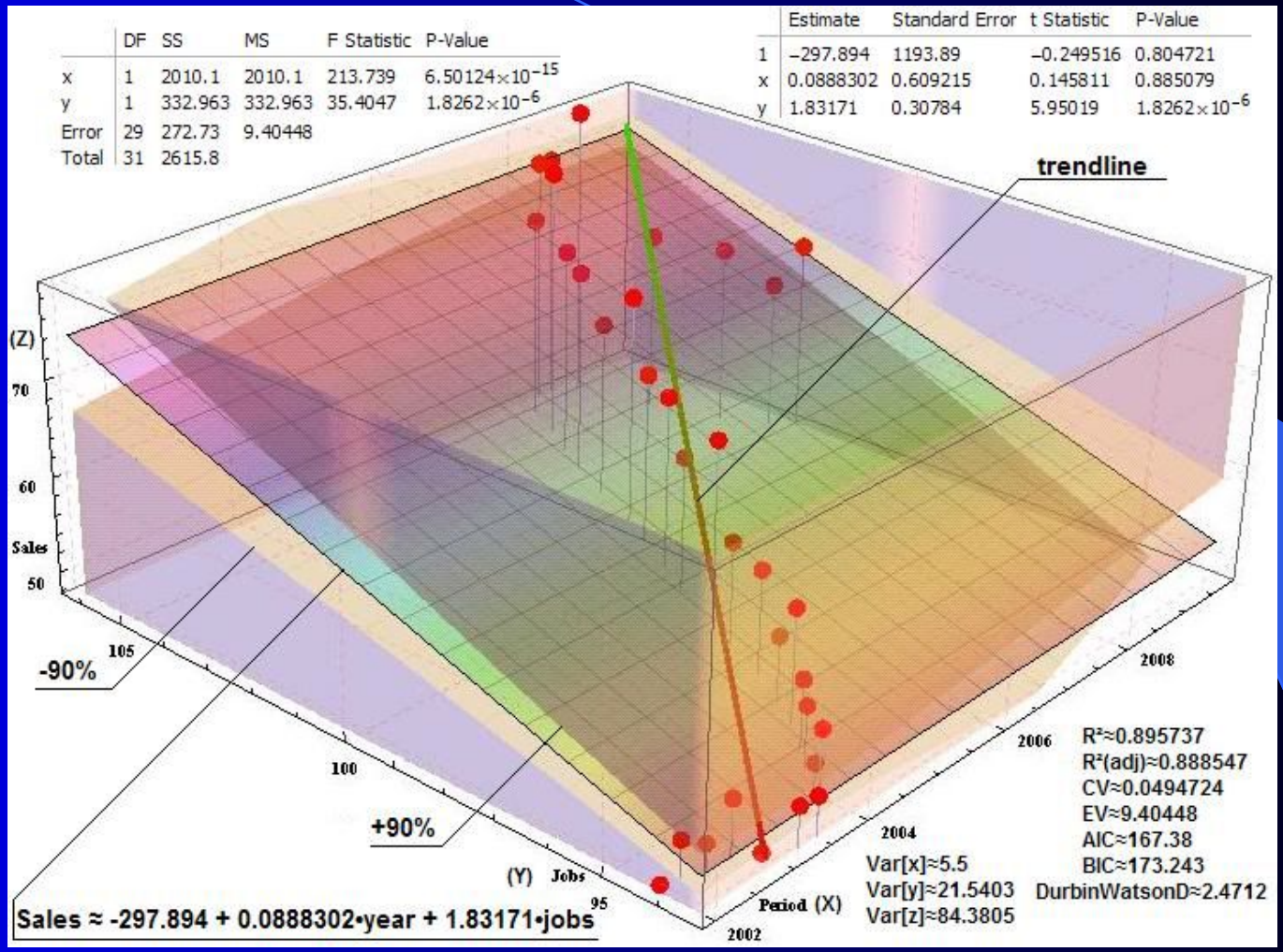



- Sigmoid fits many decision cases quite well! This is basically what logistic regression does.

Observation: Consider the 2 input perceptron case without a bias weight. Note that the output $z$ is a function of 2 input variables for the 2 input case ($x_1$, $x_2$), and thus we really have a 3-$d$ decision surface (i.e. a plane accounting for the two input variables and the 3rd dimension for the output), yet the decision boundary is still a line in the 2-$d$ input space when we represent the outputs with different colors, symbols, etc. The Delta rule would fit a regression plane to these points with the decision line being that line where the plane went through .5. What would logistic regression do?

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$



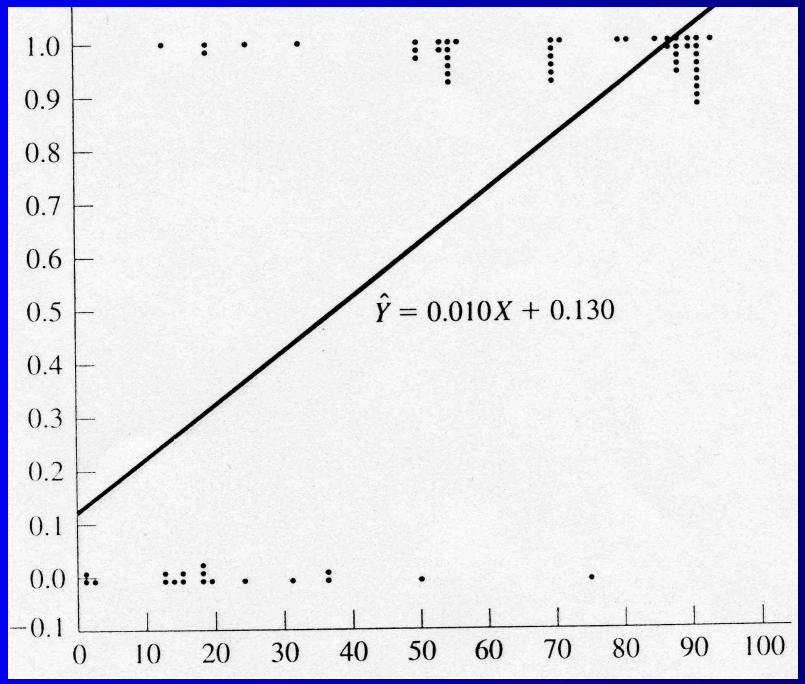

DF SS MS F Statistic P-Value
x 1 2010.1 2010.1 213.739 6.50124×10^-15
y 1 332.963 332.963 35.4047 1.8262×10^-6
Error 29 272.73 9.40448
Total 31 2615.8
Estimate Standard Error t Statistic P-Value
1 −297.894 1193.89 −0.249516 0.804721
x 0.0888302 0.609215 0.145811 0.885079
y 1.83171 0.30784 5.95019 1.8262×10^-6
trendline
(Z)
70
60
Sales
50
-90%
105
100
+90%
(Y) Jobs
95
2002
2004
2006
2008
Period (X)
R²≈0.895737
R²(adj)≈0.888547
CV≈0.0494724
EV≈9.40448
AIC≈167.38
BIC≈173.243
DurbinWatsonD≈2.4712
Var[x]≈5.5
Var[y]≈21.5403
Var[z]≈84.3805
Sales ≈ -297.894 + 0.0888302•year + 1.83171•jobs

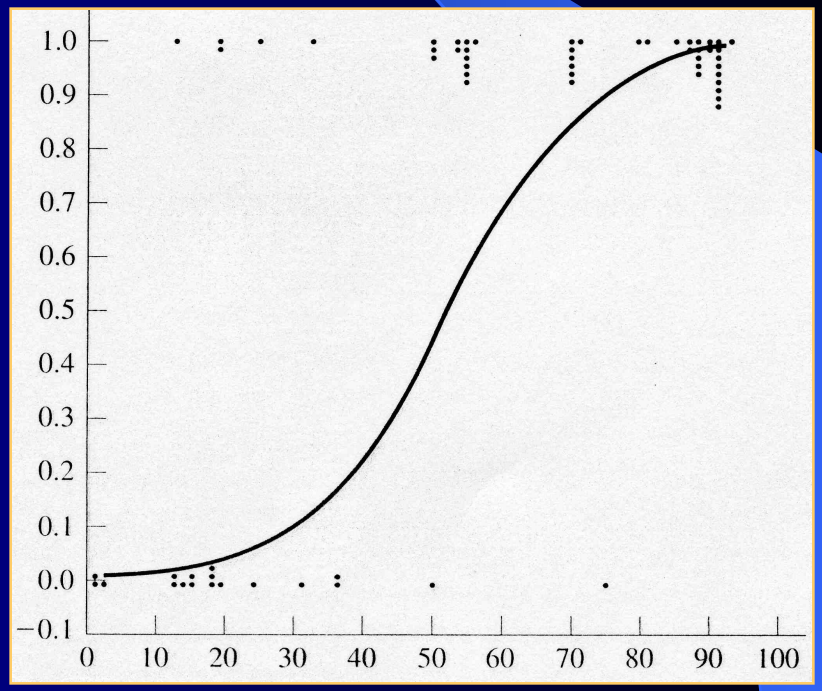
# Logistic Regression

- One commonly used algorithm is Logistic Regression
- Assumes that the dependent (output) variable is binary which is often the case in medical and other studies. (Does person have disease or not, survive or not, accepted or not, etc.)
- Like Quadric, Logistic Regression does a particular non-linear transform on the data after which it just does linear regression on the transformed data
- Logistic regression fits the data with a sigmoidal/logistic curve rather than a line and outputs an approximation of the probability of the output given the input

# Logistic Regression Example

- Age (X axis, input variable) – Data is fictional
- Heart Failure (Y axis, 1 or 0, output variable)
- If use value of regression line as a probability approximation
  - Extrapolates outside 0-1 and not as good empirically
- Sigmoidal curve to the right gives empirically good probability approximation and is bounded between 0 and 1



# Logistic Regression Approach

## Learning

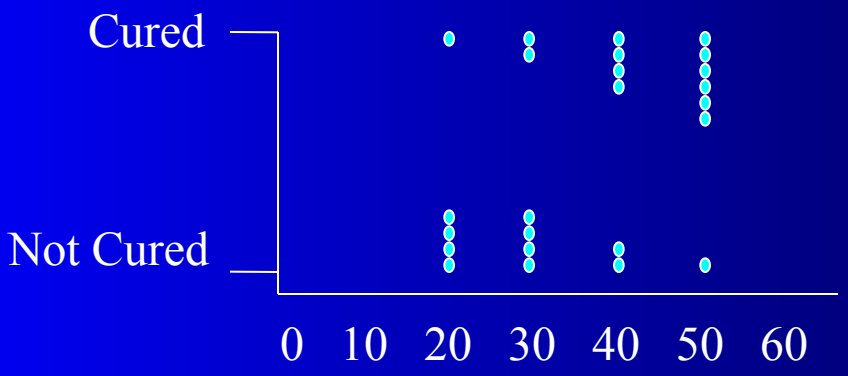
1. Transform initial input probabilities into log odds (logit)
2. Do a standard linear regression using the logit values
   - This effectively fits a logistic curve to the data, while still just doing a linear regression with the transformed input (ala quadric machine, etc.)

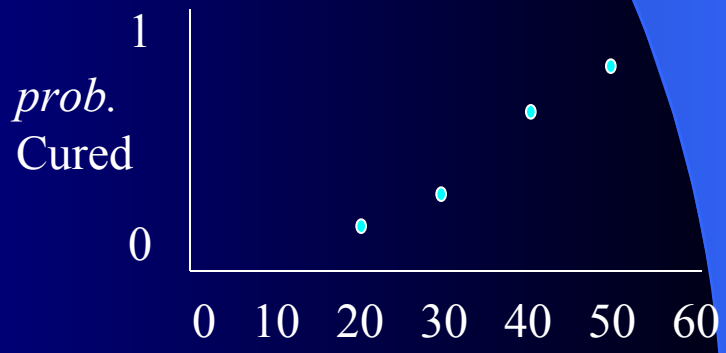
## Generalization

1. Find the value for the new input on the logit line
2. Transform that logit value back into a probability

# Non-Linear Pre-Process to Logit (Log Odds)

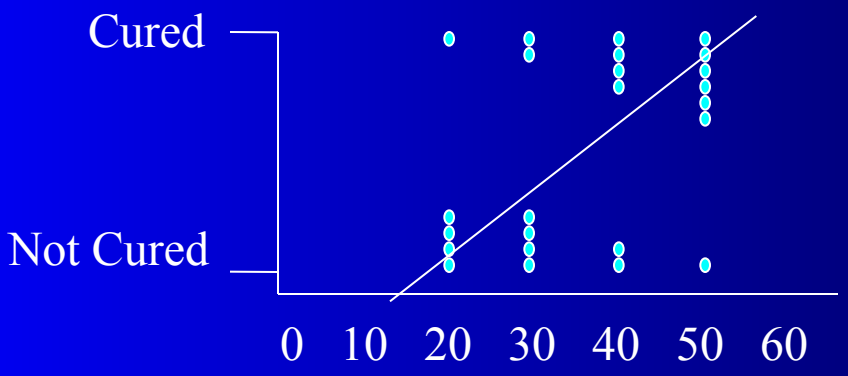
| Medication Dosage | # Cured | Total Patients | Probability: # Cured/Total Patients |
|---|---|---|---|
| 20 | 1 | 5 | .20 |
| 30 | 2 | 6 | .33 |
| 40 | 4 | 6 | .67 |
| 50 | 6 | 7 | .86 |

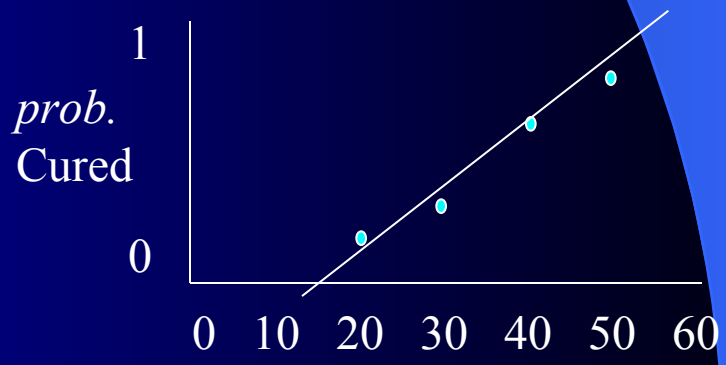


# Non-Linear Pre-Process to Logit (Log Odds)

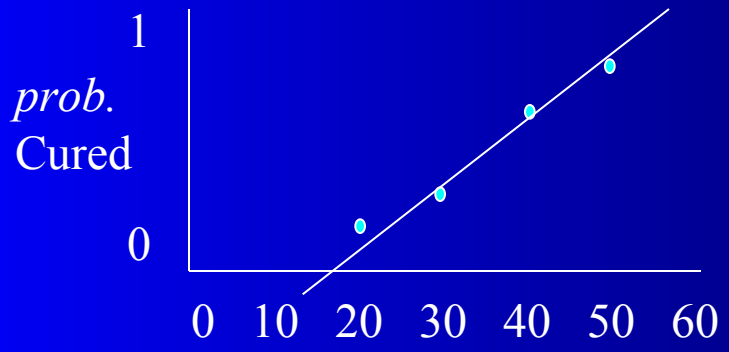
| Medication Dosage | # Cured | Total Patients | Probability: # Cured/Total Patients |
|---|---|---|---|
| 20 | 1 | 5 | .20 |
| 30 | 2 | 6 | .33 |
| 40 | 4 | 6 | .67 |
| 50 | 6 | 7 | .86 |

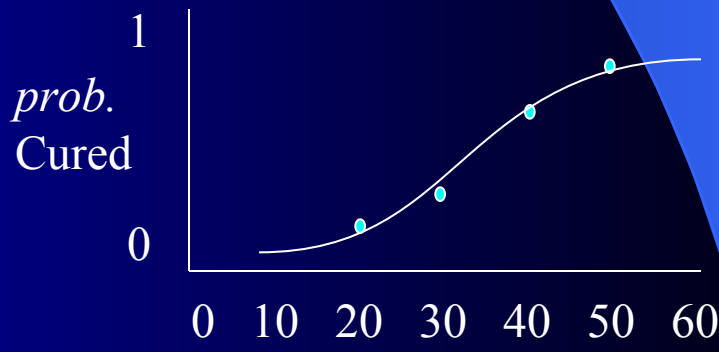


# Logistic Regression Approach

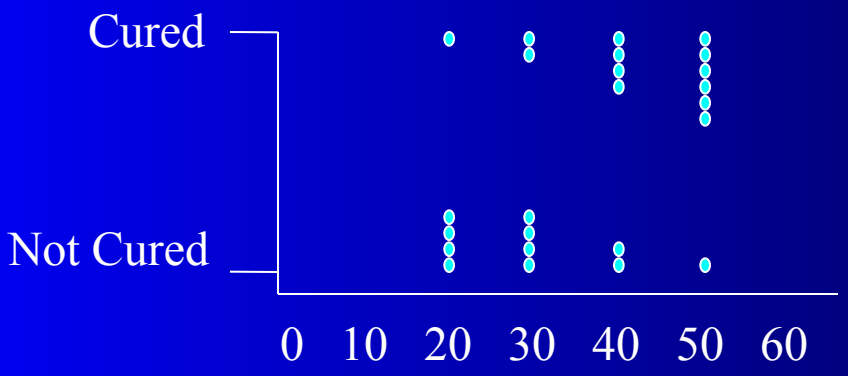
- Could use linear regression with the probability points, but that would not extrapolate well
- Logistic version is better but how do we get it?
- Similar to Quadric we do a non-linear pre-process of the input and then do linear regression on the transformed values – do a linear regression on the log odds - Logit

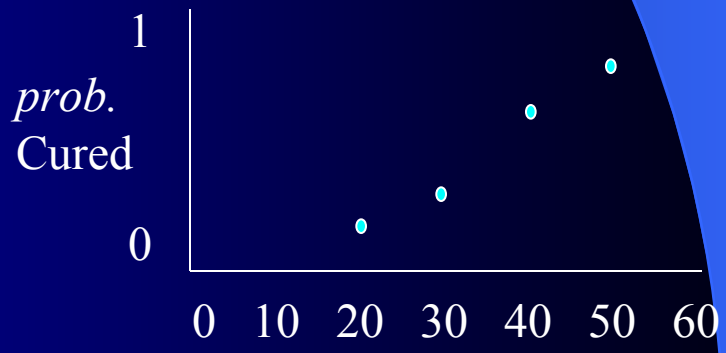


# Non-Linear Pre-Process to Logit (Log Odds)

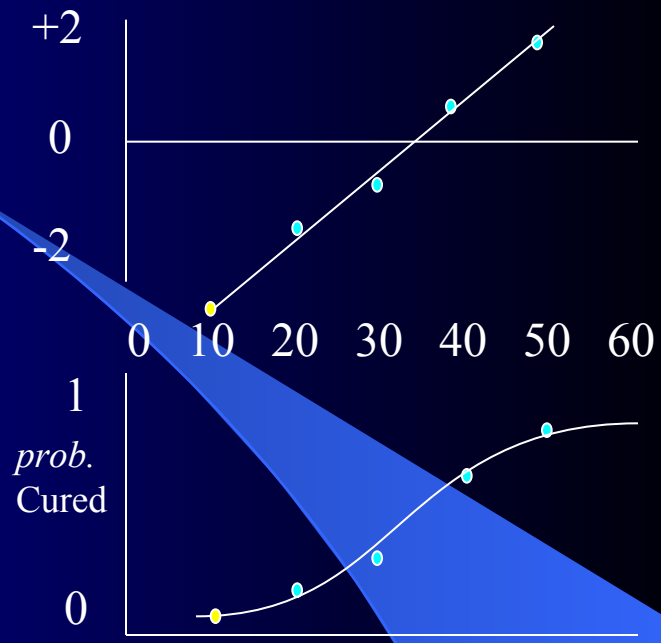
| Medication Dosage | # Cured | Total Patients | Probability: # Cured/Total Patients | Odds: $p/(1-p)$ = # cured/ # not cured | Logit Log Odds: ln(Odds) |
|---|---|---|---|---|---|
| 20 | 1 | 5 | .20 | .25 | -1.39 |
| 30 | 2 | 6 | .33 | .50 | -0.69 |
| 40 | 4 | 6 | .67 | 2.0 | 0.69 |
| 50 | 6 | 7 | .86 | 6.0 | 1.79 |



# Regression of Log Odds

| Medication Dosage | # Cured | Total Patients | Probability: # Cured/Total Patients | Odds: $p/(1-p)$ = # cured/ # not cured | Log Odds: ln(Odds) |
|---|---|---|---|---|---|
| 20 | 1 | 5 | .20 | .25 | -1.39 |
| 30 | 2 | 6 | .33 | .50 | -0.69 |
| 40 | 4 | 6 | .67 | 2.0 | 0.69 |
| 50 | 6 | 7 | .86 | 6.0 | 1.79 |




- $y = .11x - 3.8$ - Logit regression equation
- Now we have a regression line for log odds (logit)
- To generalize, we use the log odds value for the new data point
- Then we transform that log odds point to a probability: $p = e^{\text{logit}(x)}/(1+e^{\text{logit}(x)})$
- For example assume we want $p$ for dosage = 10

$$\text{Logit}(10) = .11(10) - 3.8 = -2.7$$

$p(10) = e^{-2.7}/(1+e^{-2.7}) = .06$ [note that we just work backwards from logit to $p$]

- These $p$ values make up the sigmoidal regression curve (which we never have to actually plot)

# Logistic Regression Homework
## No longer a required homework

- You don't actually have to come up with the weights for this one, though you could do so quickly by using the closed form linear regression approach
- Sketch each step you would need to learn the weights for the following data set using logistic regression
- Sketch how you would generalize the probability of a heart attack given a new input heart rate of 60

| Heart Rate | Heart Attack |
|---|---|
| 50 | Y |
| 50 | N |
| 50 | N |
| 50 | N |
| 70 | N |
| 70 | Y |
| 90 | Y |
| 90 | Y |
| 90 | N |
| 90 | Y |
| 90 | Y |

# Summary

- Linear Regression and Logistic Regression are nice tools for many simple situations
  - But both force us to fit the data with one shape (line or sigmoid) which will often underfit
- Intelligible results
- When problem includes more arbitrary non-linearity then we need more powerful models which we will introduce
  - Though non-linear data transformation can help in these cases while still using a linear model for learning
- These models are commonly used in data mining applications and also as a "first attempt" at understanding data trends, indicators, etc.

# Non-Linear Regression

- Note that linear regression is to regression what the perceptron is to classification
  - Simple, useful models which will often underfit
- All of the more powerful classification models which we will be discussing later in class can also be used for non-linear regression, though we will mostly discuss them using classification
  - MLP with Backpropagation, Decision Trees, Nearest Neighbor, etc.
- They can learn functions with arbitrarily complex high dimensional shapes