From Data to Features: Techniques for Machine Learning

COSI 140 – Natural Language Annotation for Machine Learning

James Pustejovsky

March 29, 2016

Brandeis University

Slides from Tony Martinez

Statistical Significance and Performance Measures

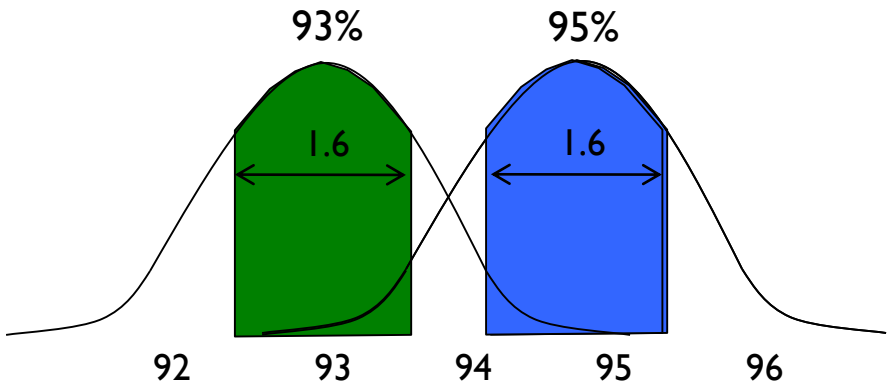
- Just a brief review of confidence intervals Assume you've seen *t*-tests, etc.
  - Confidence Intervals
  - Central Limit Theorem
- Permutation Testing
- Other Performance Measures
  - Precision
  - Recall
  - F-score
  - ROC

Statistical Significance

- How do we know that some measurement is statistically significant vs being just a random perturbation
  - How good a predictor of generalization accuracy is the sample accuracy on a test set?
  - Is a particular hypothesis really better than another one because its accuracy is higher on a validation set?
  - When can we say that one learning algorithm is better than another for a particular task or set of tasks?
- For example, if learning algorithm 1 gets 95% accuracy and learning algorithm 2 gets 93% on a task, can we say with some confidence that algorithm 1 is superior in general for that task?
- Question becomes: What is the likely difference between the sample error (estimator of the parameter) and the true error (true parameter value)?
- Key point – What is the probability the the differences in our results are just due to chance?

Confidence Intervals

- An *N*% confidence interval for a parameter p is an interval that is expected with probability *N*% to contain p
- The true mean (or whatever parameter we are estimating) will fall in the interval $\pm C_N\sigma$ of the sample mean with *N*% confidence, where σ is the deviation and C_N gives the width of the interval about the mean that includes *N*% of the total probability under the particular probability distribution. C_N is a distribution specific constant for different interval widths.
- Assume the filled in intervals are the 90% confidence intervals for our two algorithms. What does this mean?
  - The situation below says that these two algorithms are different with 90% confidence
  - Would if they overlapped?
  - How do you tighten the confidence intervals? – More data and tests



Central Limit Theorem

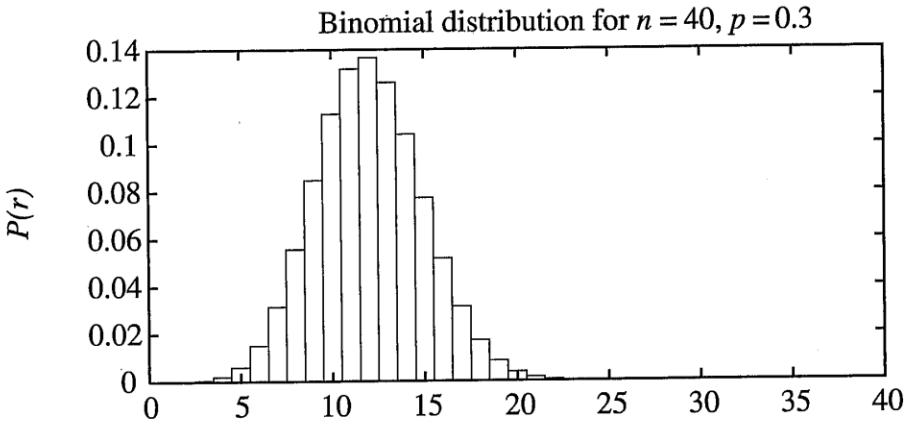
- Central Limit Theorem
  - If there are a sufficient number of samples, and
  - The samples are iid (independent, identically distributed) - drawn independently from the identical distribution
  - Then, the random variable can be represented by a Gaussian distribution with the sample mean and variance
- Thus, regardless of the underlying distribution (even when unknown), if we have enough data then we can assume that the estimator is Gaussian distributed
- And we can use the Gaussian interval tables to get intervals $\pm z_N \sigma$
- Note that while the test sets are independent in *n*-way CV, the training sets are not since they overlap (Still a decent approximation)

Binomial Distribution

- Given a coin with probability p of heads, the binomial distribution gives the probability of seeing exactly r heads in n flips.

$$P(r) = \frac{n!}{r!(n - r)!} p^r (1 - p)^{n - r}$$

- A random variable is a random event that has a specific outcome (X = number of times heads comes up in n flips)
  - For binomial, $\Pr(X = r)$ is $P(r)$
  - The mean (expected value) for the binomial is np
  - The variance for the binomial is $np(1 - p)$
- Same setup for classification where the outcome of an instance is either correct or in error and the sample error rate is r/n which is an estimator of the true error rate p



A *Binomial distribution* gives the probability of observing r heads in a sample of n independent coin tosses, when the probability of heads on a single coin toss is p. It is defined by the probability function

$$P(r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$

If the random variable X follows a Binomial distribution, then:

- The probability $\Pr(X = r)$ that X will take on the value r is given by $P(r)$
- The expected, or mean value of X, $E[X]$, is

$$E[X] = np$$

- The variance of X, $Var(X)$, is

$$Var(X) = np(1-p)$$

- The standard deviation of X, σ_X, is

$$\sigma_X = \sqrt{np(1-p)}$$

For sufficiently large values of n the Binomial distribution is closely approximated by a Normal distribution (see Table 5.4) with the same mean and variance. Most statisticians recommend using the Normal approximation only when $np(1-p) \geq 5$.

Binomial Estimators

- Usually want to figure out *p* (e.g. the true error rate)
- For the binomial the sample error *r*/*n* is an unbiased estimator of the true error *p*
  - An estimator *X* of parameter *y* is unbiased if

$$s_{err} = \frac{s_r}{n} = \sqrt{\frac{np(1-p)}{n^2}} = \sqrt{\frac{p(1-p)}{n}} \approx \sqrt{\frac{Err_{sample}(1-Err_{sample})}{n}}$$

- For the binomial the sample deviation is

Comparing two Algorithms - paired *t* test

- Do *k*-way CV for both algorithms on the same data set using the same splits for both algorithms (paired)
  - Best if $k > 30$ but that will increase variance for smaller data sets
- Calculate the accuracy difference δ_i between the algorithms for each split (paired) and average the *k* differences to get δ
- Real difference is with *N*% confidence in the interval

$$\delta \pm t_{N,k-1}\ \sigma$$

where σ is the standard deviation and $t_{N,k-1}$ is the *N*% *t* value for *k*-1 degrees of freedom. The *t* distribution is slightly flatter than the Gaussian and the *t* value converges to the Gaussian (z value) as *k* grows.

Paired *t* test - Continued

- σ for this case is defined as $$s = \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (d_i - \bar{d})^2}$$
- Assume a case with $\delta = 2$ and two algorithms M_1 and M_2 with an accuracy average of approximately 96% and 94% respectively and assume that $t_{90,29} \times \sigma = 1$. This says that with 90% confidence the true difference between the two algorithms is between 1 and 3 percent. This approximately implies that the extreme averages between the algorithm accuracies are 94.5/95.5 and 93.5/96.5. Thus we can say that with 90% confidence that M_1 is better than M_2 for this task. If $t_{90,29} \times \sigma$ is greater than δ then we could not say that M_1 is better than M_2 with 90% confidence for this task.
- Since the difference falls in the interval $\delta \pm t_{N,k-1}\sigma$ we can find the $t_{N,k-1}$ equal to δ/σ to obtain the best confidence value

Permutation Test

- With faster computing it is often reasonable to do a direct permutation test to get a more accurate confidence, especially with the common 10 fold cross validation (only 1000 permutations)

Menke, J., and Martinez, T. R., Using Permutations Instead of Student's *t* Distribution for *p*-values in Paired-Difference Algorithm Comparisons, Proceedings of the *IEEE International Joint Conference on Neural Networks IJCNN'04,* pp. 1331-1336, 2004.

- Even if two algorithms were really the same in accuracy you would expect some random difference in outcomes based on data splits, etc.
- How do you know that the measured difference between two situations is not just random variance?
- If it were just random, the average of many random permutations of results would give about the same difference (i.e. just the problem variance)

Permutation Test Details

- To compare the performance of models M_1 and M_2 using a permutation test:

1. Obtain a set of k estimates of accuracy $A = \{a_1, ..., a_k\}$ for M_1 and $B = \{b_1, ..., b_k\}$ for M_2 (e.g. each do k-fold CV on the same task, or accuracies on k different tasks, etc.)
2. Calculate the average accuracies, $\mu_A = (a_1 + ... + a_k)/k$ and $\mu_B = (b_1 + ... + b_k)/k$ (note they are not paired in this algorithm)
3. Calculate $d_{AB} = |\mu_A - \mu_B|$
4. let $p = 0$
5. Repeat n times (or just every permutation)
   a. let $S=\{a_1, ..., a_k, b_1, ..., b_k\}$
   b. randomly partition S into two equal sized sets, R and T (statistically best if partitions not repeated)
   c. Calculate the average accuracies, μ_R and μ_T
   d. Calculate $d_{RT} = |\mu_R - \mu_T|$
   e. if $d_{RT} \geq d_{AB}$ then $p = p+1$
6. p-value = p/n (Report p, n, and p-value)

   A low p-value implies that the algorithms really are different

	Alg 1	Alg 2	Diff
Test 1	92	90	2
Test 2	90	90	0
Test 3	91	92	-1
Test 4	93	90	3
Test 5	91	89	2
Ave	91.4	90.2	1.2

Statistical Significance Summary

- Required for publications
- No single accepted approach
- Many subtleties and approximations in each approach
  - Independence assumptions often violated
  - Degrees of freedom: Is LA_1 still better than LA_2 when
    - The size of the training sets are changed
    - Trained for different lengths of time
    - Different learning parameters are used
    - Different approaches to data normalization, features, etc.
    - Etc.
- Author's tuned parameters vs default parameters (grain of salt on results)
- Still can (and should) get higher confidence in your assertions with the use of statistical measures

Performance Measures

- Most common measure is accuracy
  - Summed squared error
  - Mean squared error
  - Classification accuracy

Issues with Accuracy

- Assumes equal cost for all errors
- Is 99% accuracy good; Is 30% accuracy bad?
  - Depends on baseline and problem complexity
  - Depends on cost of error (Heart attack diagnosis, etc.)
- Error reduction (1-accuracy)
  - Absolute vs relative
  - 99.90% accuracy to 99.99% accuracy is a 90% relative reduction in error, but absolute error is only reduced by .09%.
  - 50% accuracy to 75% accuracy is a 50% relative reduction in error and the absolute error reduction is 25%.
  - Which is better?

Binary Classification

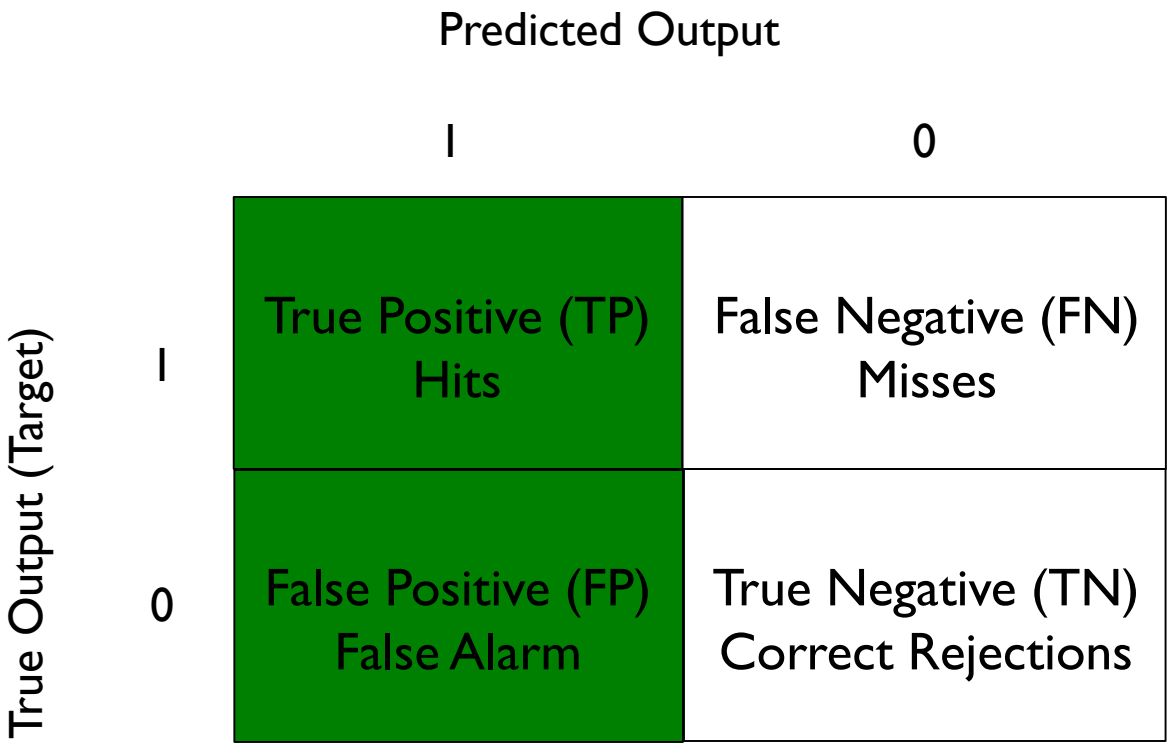
True Output (Target) \ Predicted Output	1	0
1	True Positive (TP) Hits	False Negative (FN) Misses
0	False Positive (FP) False Alarm	True Negative (TN) Correct Rejections

Accuracy = (TP+TN)/(TP+TN+FP+FN)

Precision = TP/(TP+FP)

Recall = TP/(TP+FN)

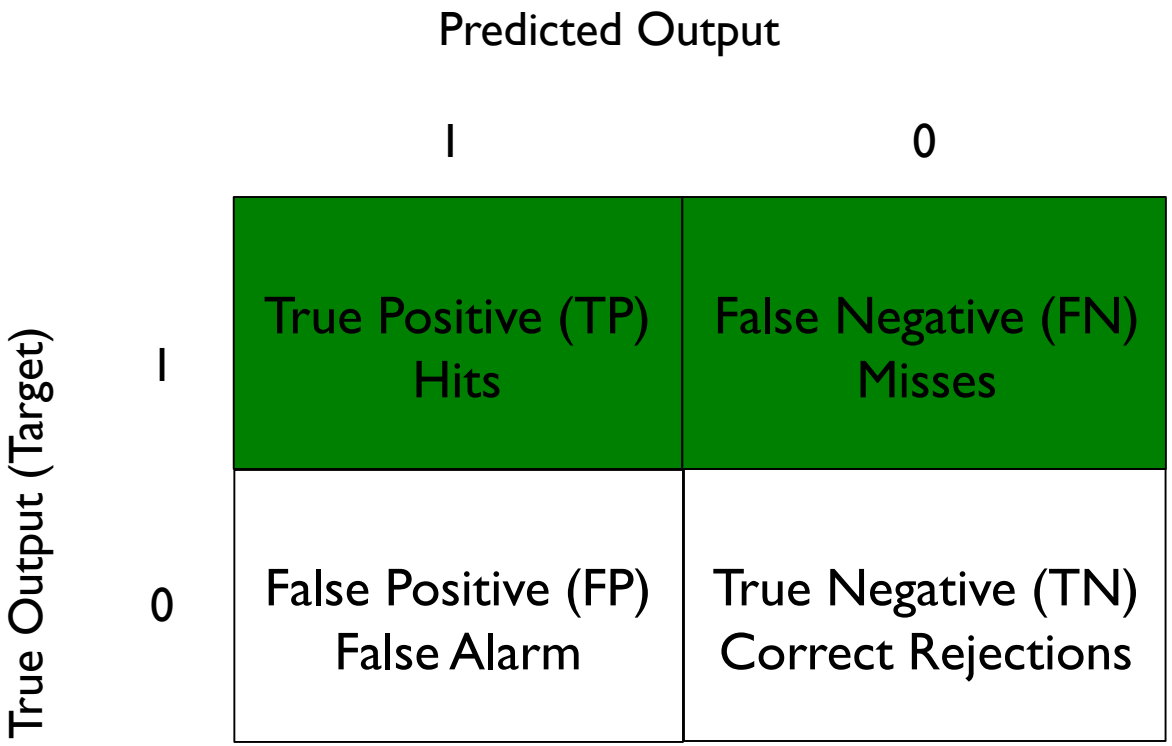
Precision



Precision = TP/(TP+FP)

The percentage of predicted true positives that are target true positives

Recall


Predicted Output
1
0
True Output (Target)
1
0
True Positive (TP)
Hits
False Negative (FN)
Misses
False Positive (FP)
False Alarm
True Negative (TN)
Correct Rejections

Other measures - Precision vs. Recall

- Considering precision and recall lets us choose a ML approach which maximizes what we are most interested in (precision or recall) and not just accuracy.
- Tradeoff - Can also adjust ML parameters to accomplish the goal of the application – Heart attack vs Google search
  - How would we do this with an MLP for example
- Break even point: precision = recall
- F_1 or F-score = $2 \times (\text{precision} \times \text{recall}) / (\text{precision} + \text{recall})$ - Harmonic average of precision and recall

Cost Ratio

- For binary classification (concepts) can have an adjustable threshold for deciding what is a True class vs a False class
  - For BP it could be what activation value is used to decide if a final output is true or false (default .5)
  - For ID3 it could be what percentage of the leaf elements need to be in a class for that class to be chosen (default is the most common class)
- Could slide that threshold depending on your preference for True vs False classes (Precision vs Recall)
- Radar detection of incoming missiles

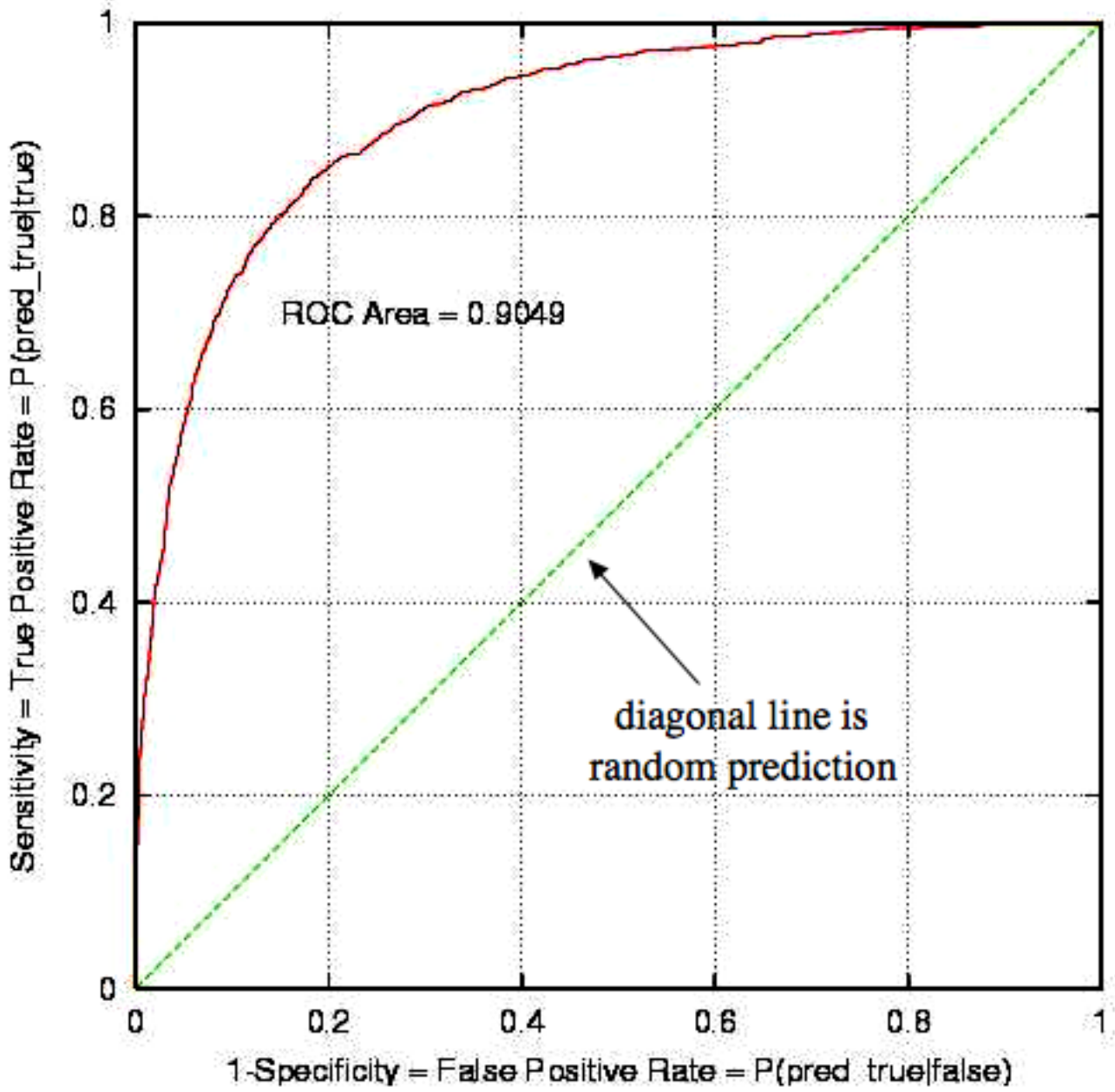
ROC Curves and ROC Area

- Receiver Operating Characteristic
- Developed in WWII to statistically model false positive and false negative detections of radar operators
- Standard measure in medicine and biology
- True positive rate (sensitivity) vs false positive rate (1- specificity)
- True positive rate (Probability of predicting true when it is true) P(Pred:T|T) = Sensitivity = recall = TP/P = TP/(TP+FN)
- False positive rate (Probability of predicting true when it is false)

  P(Pred:T|F) = FP/N = FP/(TN+FP) = 1 – specificity (true negative rate) = 1 – TN/N = 1 - TN/(TN+FP)
  - Want to maximize TPR and minimize FPR
  - How would you do each independently?

ROC Curves and ROC Area

- Neither extreme is acceptable
  - Want to find the right balance
  - But the right balance/threshold can differ for each task considered
- How do we know which algorithms are robust and accurate across many different thresholds? – ROC curve
- Each point on the ROC curve represents a different tradeoff (cost ratio) between true positive rate and false positive rate
- Standard measures just show accuracy for one setting of the cost/ratio threshold, whereas the ROC curve shows accuracy for all settings and thus allows us to compare how robust to different thresholds one algorithm is compared to another


ROC Area = 0.9049
diagonal line is
random prediction
Sensitivity = True Positive Rate = P(pred_true|true)
1-Specificity = False Positive Rate = P(pred_true|false)
0
0.2
0.4
0.6
0.8
1

- Assume Backprop threshold
- Threshold = 1 (0,0), then all outputs are 0

  P(T|T) = 0, P(T|F) = 0
- Threshold = 0, (1,1)

  P(T|T) = 1, P(T|F) = 1
- Threshold = .8 (.2,.2)

  P(T|T) = .38 P(T|F) = .02

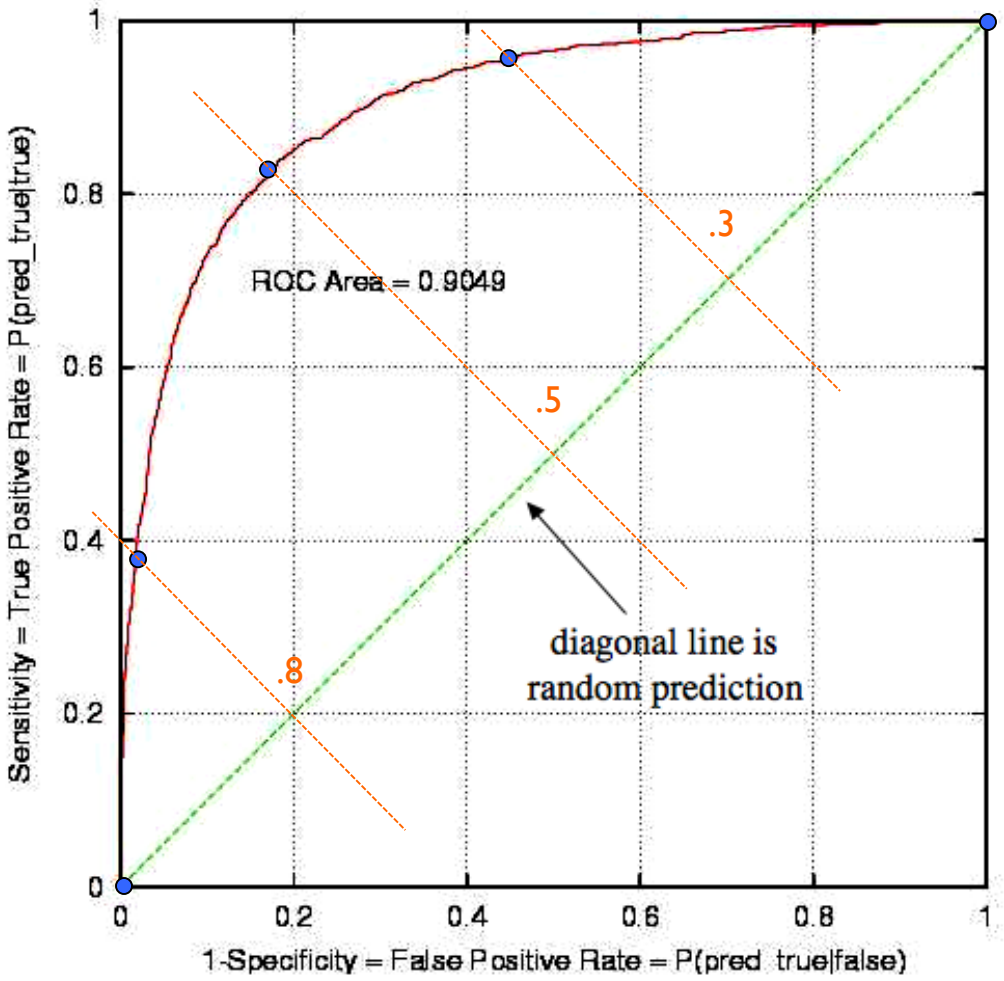
  - Better Precision
- Threshold = .5 (.5,.5)

  P(T|T) = .82 P(T|F) = .18

  - Better Accuracy
- Threshold = .3 (.7,.7)

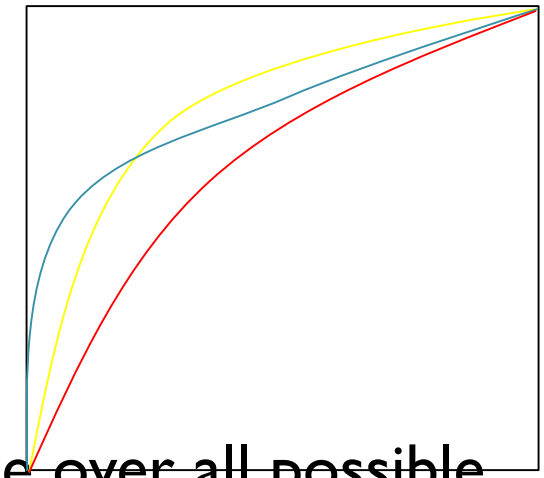
  P(T|T) = .95 P(T|F) = .43

  - Better Recall



Accuracy is maximized at point closest to the top left corner. Note that Sensitivity = Recall and the lower the false positive rate, the higher the precision.

ROC Properties



- Area Properties
  - 1.0 - Perfect prediction
  - .9 - Excellent
  - .7 - Mediocre
  - .5 - Random
- ROC area represents performance over all possible cost ratios
- If two ROC curves do not intersect then one method dominates over the other
- If they do intersect then one method is better for some cost ratios, and is worse for others
  - Blue alg better for precision, yellow alg for recall, red neither
- Can choose method and balance based on goals

Performance Measurement Summary

- Other measures (F-score, ROC) gaining popularity
- Can allow you to look at a range of thresholds
- However, they do not extend to multi-class situations which are very common
  - Could always cast problem as a set of two class problems but that can be inconvenient
- Accuracy handles multi-class outputs and is still the most common measure

Gathering a Data Set

- Consider a Task: Classifying the quality of pizza
- What features might we use?
- Data Types
  - Nominal (aka Categorical, Discrete)
  - Continuous (aka Real, Numeric)
  - Linear (aka Ordinal) – Is usually just treated as continuous, so that ordering info is maintained
- How to represent those features?
  - Will usually depend on the learning model we are using
- Classification assumes the output class is nominal. If output is continuous, then we are doing *regression*.

Fitting Data to the Model

- Continuous -> Nominal
  - Discretize into bins – more on this later
- Nominal -> Continuous (Perceptron expects continuous)
  a) One input node for each nominal value where one of the nodes is set to 1 and the other nodes are set to 0
     - Can also *explode* the variable into *n*-1 input nodes where the most common value is not explicitly represented (i.e. the all 0 case)
  b) Use 1 node but with a different continuous value representing each nominal value
  c) Distributed – $\log_b n$ nodes can uniquely represent *n* nominal values (e.g. 3 binary nodes could represent 8 values)
  d) If there is a very large number of nominal values, could cluster (discretize) them into a more manageable number of values and then use one of the techniques above

Data Normalization

- What would happen if you used two input features in an astronomical task as follows:
  - Weight of the planet in grams
  - Diameter of the planet in light-years

Performance Measures

- There are a number of ways to measure the performance of a learning algorithm:
  - Predictive accuracy of the induced model
  - Size of the induced model
  - Time to compute the induced model
  - etc.
- We will focus here on accuracy
- Fundamental Assumption:

  Future novel instances are drawn from the same/similar distribution as the training instances

Training/Testing Alternatives

- Four methods that we will consider:
  - Training set method: The model is evaluated on the same data set that was used for training
  - Static split test set method: Two distinct data sets are made available to the learning algorithm; one for training and one for testing
  - Random split test set method: A single data set is made available to the learning algorithm and the data set is split such that *x*% of the instances are randomly selected for training and the remainder are used for testing, where you supply the value of *x*.
  - *N*-fold cross-validation

Training Set Method

- Procedure
  - Build model from the dataset
  - Compute accuracy on the same dataset
- Simple but least reliable estimate of future performance on unseen data (a rote learner could score 100%!)
- Not used as a performance metric but it is often useful information in understanding how a machine learning model learns

Static Training/Test Set

- Static Split Approach
  - The data owner makes available to the machine learner two distinct datasets:
    - One is used for learning/training (i.e., inducing a model), and
    - One is used exclusively for testing
- Note that this gives you a way to do repeatable tests
- Can be used for challenges (e.g. to see how everyone does on one particular unseen set, etc.)
- Be careful not to overfit the Test Set ("Gold Standard")

Random Training/Test Set Approach

- Random Split Approach
  - The data owner makes available to the machine learner a single dataset
  - The machine learner splits the dataset into a training and a test set, such that:
    - Instances are randomly assigned to either set
    - The distribution of instances (with respect to the target class) is hopefully similar in both sets due to randomizing the data before the split (stratification is even better but not required here)
    - Typically 60% to 90% of instances are used for training and the remainder for testing – the more data there is the more that can be used for training and still get statistically significant test predictions
  - Useful quick estimate for computationally intensive learners
  - Not statistically optimal (high variance, unless lots of data)
  - Can avoid possible overfit of just one test set
  - Best to do multiple training runs with different splits. Train and test *m* times and then average the accuracy over the *m* runs to get a more statistically accurate prediction of generalization accuracy

N-fold Cross-validation

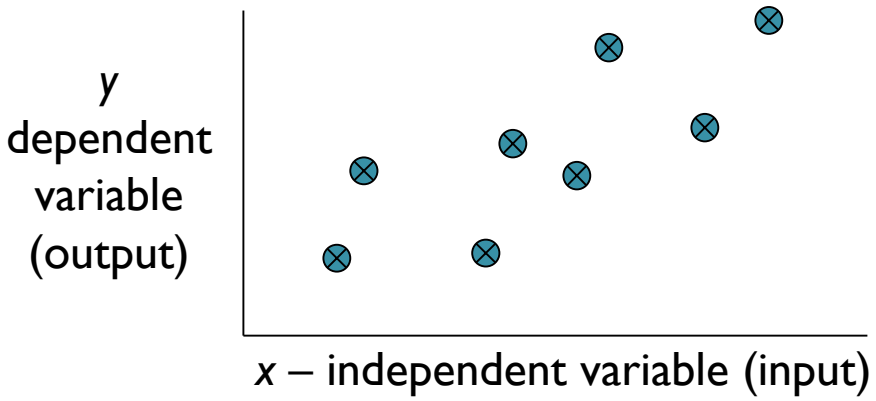
- Use all the data for both training and testing
  - Statistically more reliable
  - All data can be used which is good for small data sets
- Procedure
  - Partition the randomized dataset (call it *D*) into *N* equally-sized subsets $S_1, \ldots, S_N$
  - For k = 1 to N
    - Let M_k be the model induced from $D - S_k$
    - Let a_k be the accuracy of M_k on the instances of the test fold S_k
  - Return $(a_1+a_2+\ldots+a_N)/N$

N-fold Cross-validation (cont.)

- The larger *N* is, the smaller the variance in the final result
- The limit case where $N = |D|$ is known as *leave-one-out* and provides the most reliable estimate. However, it is typically only practical for small instance sets
- Generally, a value of $N=10$ is considered a reasonable compromise between time complexity and reliability
- Still must choose an actual model to use during execution - how?
  - Could select the one model that was best on its fold?
  - All data? With any of the above approaches
- Note that CV is just a better way to estimate how well we will do on novel data, rather than a way to do model selection

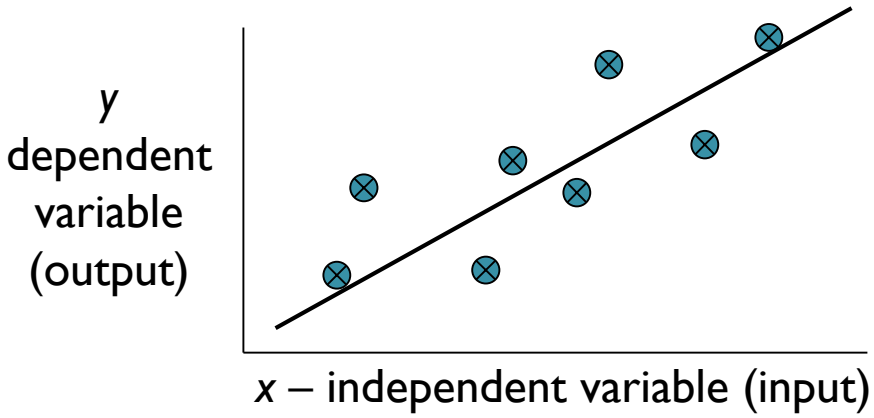
Regression

- 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



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 will "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)

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)

$$Y = b_0 + b_1 X$$

- 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.

$$b_1 = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - \left(\sum x\right)^2}$$

$$b_0 = \frac{\sum y - b_1 \sum x}{n}$$

Multiple Linear Regression

$$Y = b_0 + b_1 X_1 + b_2 X_2 + \square + b_n X_n$$

- 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, which is more natural for the delta rule equation

$$\Delta w_i = \eta \sum_{d \in D} (t_d - net_d) x_{id}$$

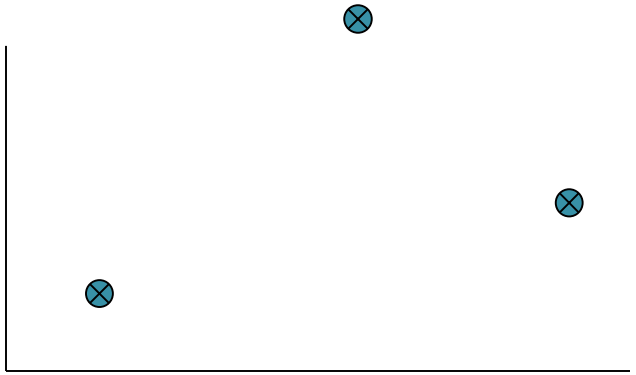
- Delta rule will update towards the objective of minimizing the SSE, thus solving multiple linear regression
- There are many other regression approaches that give different results by trying to better handle outliers and other statistical anomalies

Intelligibility

- One nice 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 1st order)
- Linear regression can be used to find best "indicators"
- However, be careful not to confuse correlation with causality

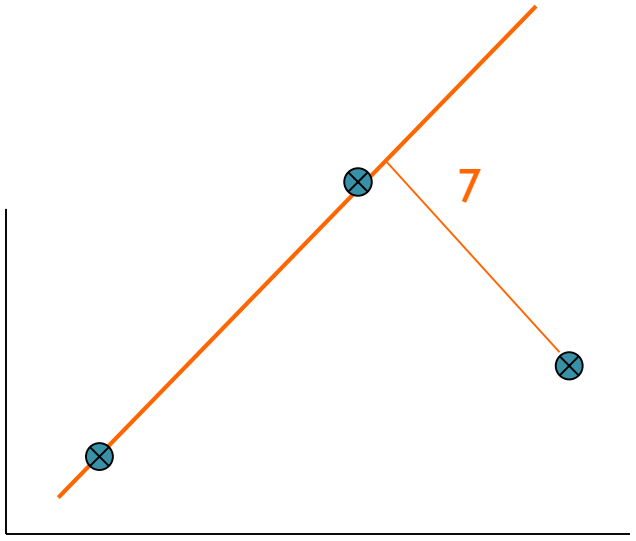
SSE and Linear Regression

- SSE chooses to square the difference of the predicted vs actual. Why square?
- Don't want residues to cancel each other
- Could use absolute or other distances to solve problem
  - $\Sigma\ |predicted_i - actual_i|$:
- 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



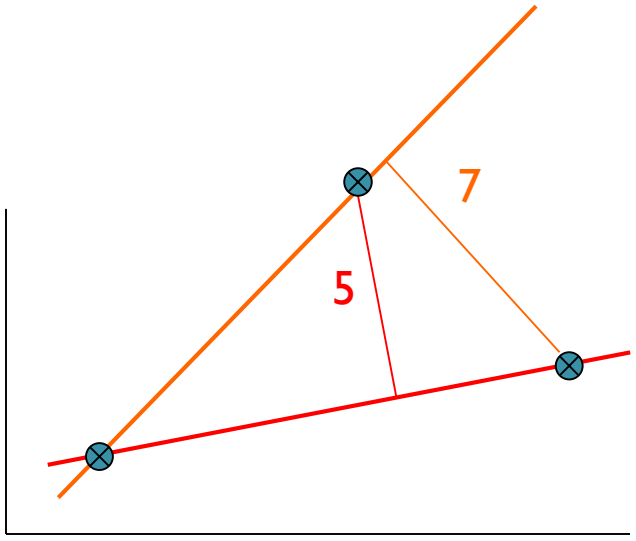
SSE and Linear Regression

- SSE chooses to square the difference of the predicted vs actual. Why square?
- Don't want residues to cancel each other
- Could use absolute or other distances to solve problem
  - $\Sigma\ |predicted_i - actual_i|$:
- 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



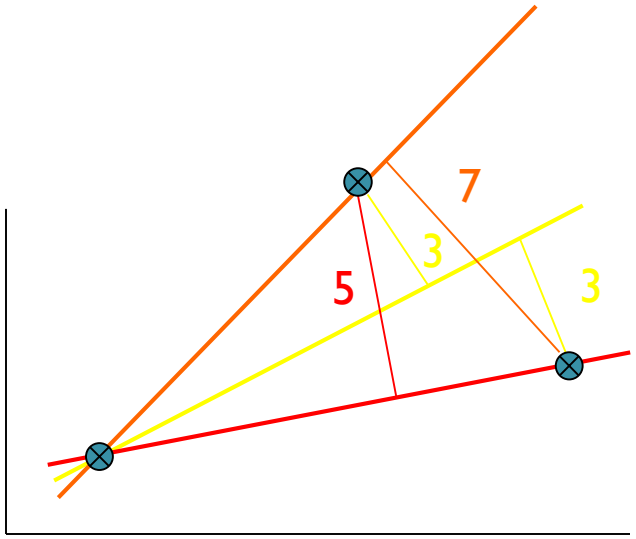
SSE and Linear Regression

- SSE chooses to square the difference of the predicted vs actual. Why square?
- Don't want residues to cancel each other
- Could use absolute or other distances to solve problem
  - $\Sigma\ |predicted_i - actual_i|$:
- 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



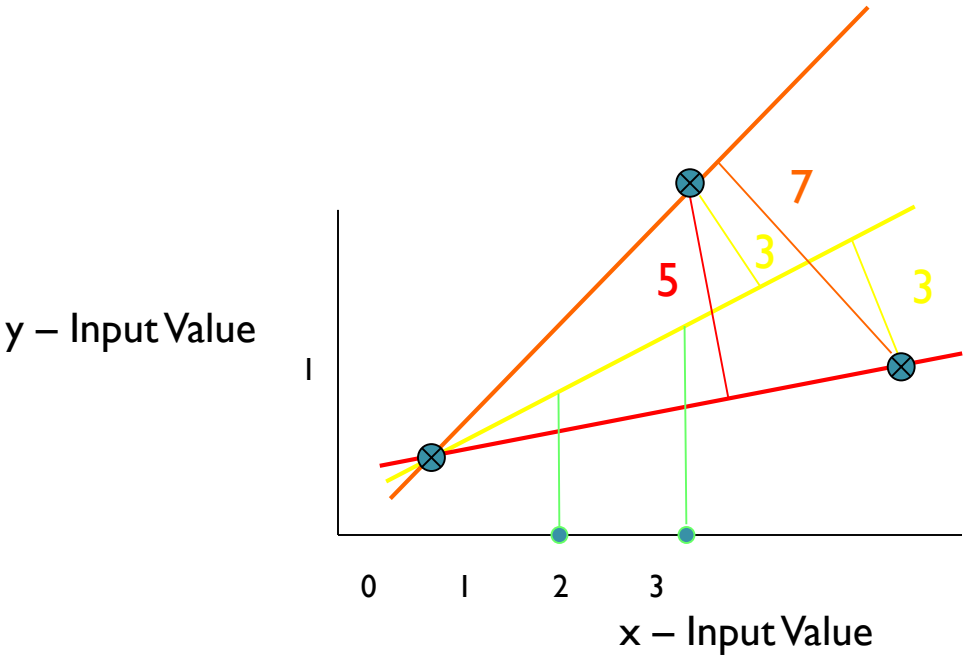
SSE and Linear Regression

- SSE chooses to square the difference of the predicted vs actual. Why square?
- Don't want residues to cancel each other
- Could use absolute or other distances to solve problem
  - $\Sigma\ |predicted_i - actual_i|$:
- 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

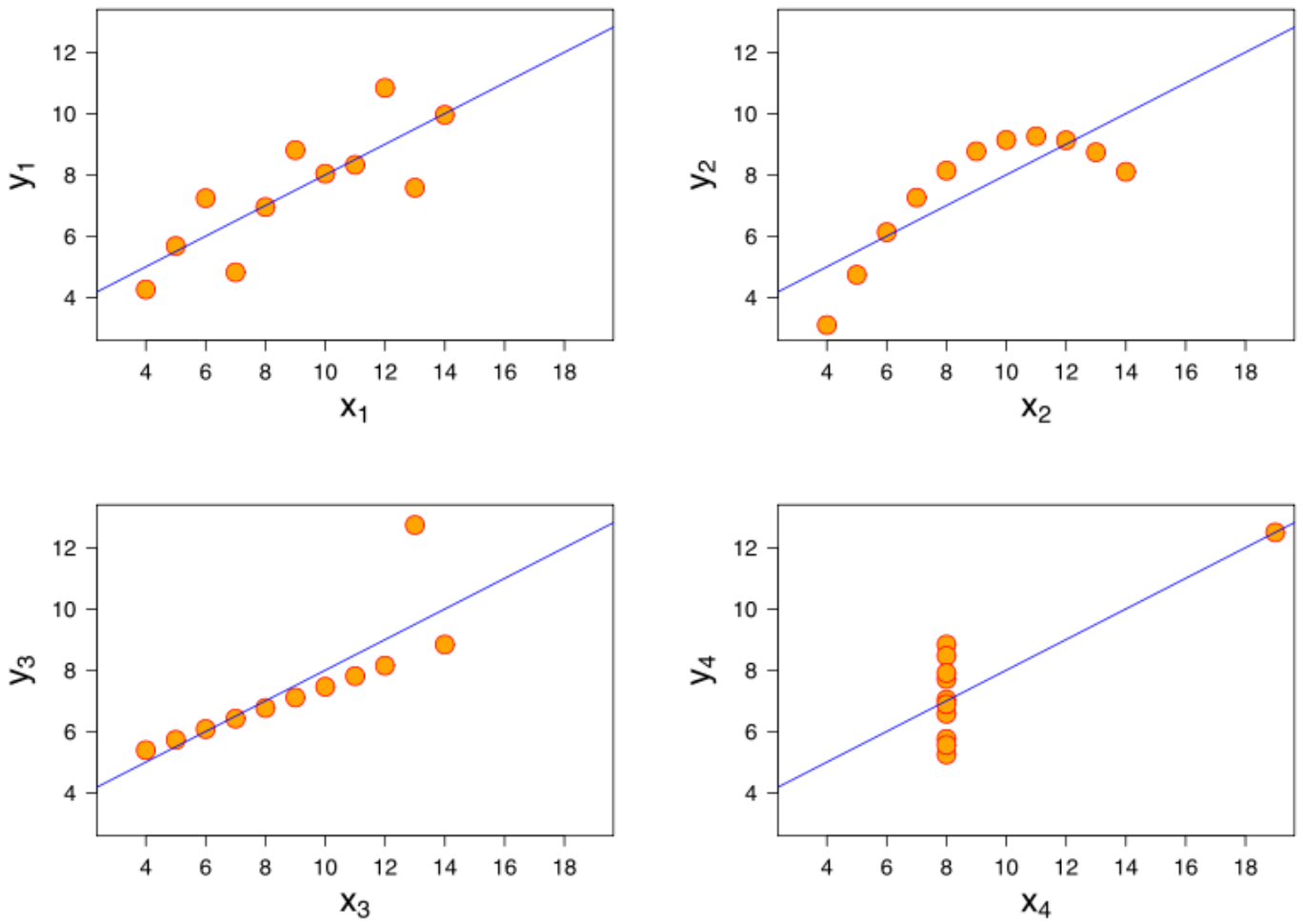


SSE and Linear Regression Generalization

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



Anscombe's Quartet



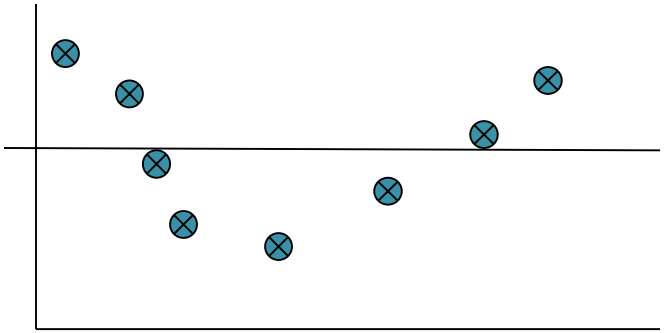
What lines "really" best fit each case? – different approaches

Non-Linear Tasks

- Linear Regression will not generalize well to the task below
- Needs a non-linear surface
- Could do a feature pre-process as with the quadric machine
  - For example, we could use an arbitrary polynomial in *x*
  - Thus it is still linear in the coefficients, and can be solved with delta rule, etc.

$$Y = b_0 + b_1 X + b_2 X^2 + \square + b_n X^n$$

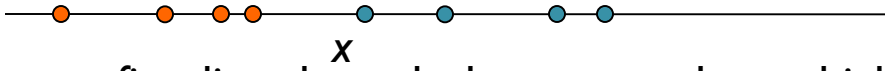
  - What order polynomial should we use? – Overfit issues occur as we'll discuss later



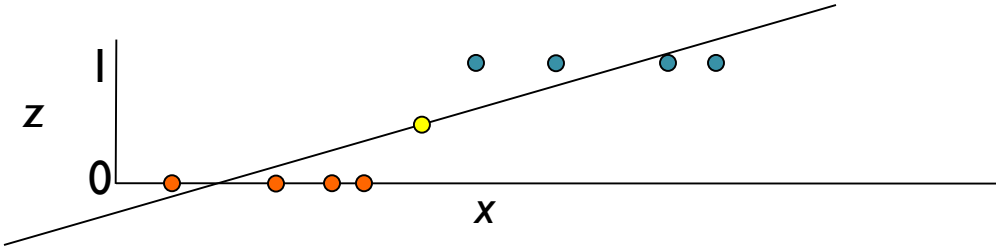
Delta Rule for Classification

$$\Delta w_i = \eta \sum_{d \in D} (t_d - net_d) x_{id}$$

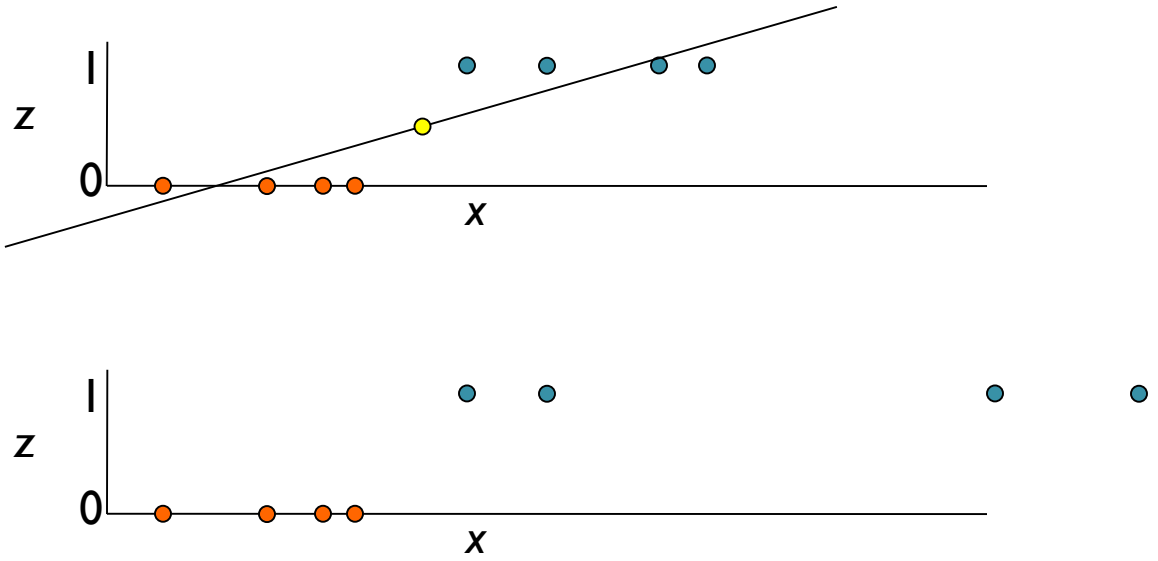
- First consider the one dimensional case
- The decision surface for the perceptron would be any (first) point that divides instances



- Delta rule will try to fit a line through the target values which minimizes SSE and the decision point will be 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

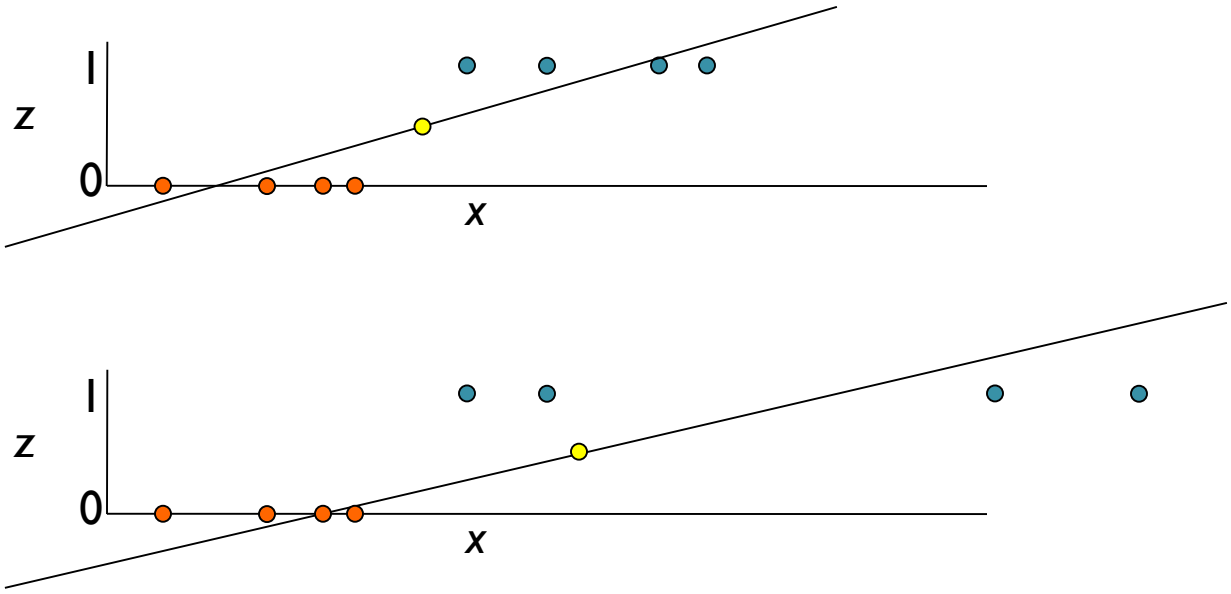


Delta Rule for Classification



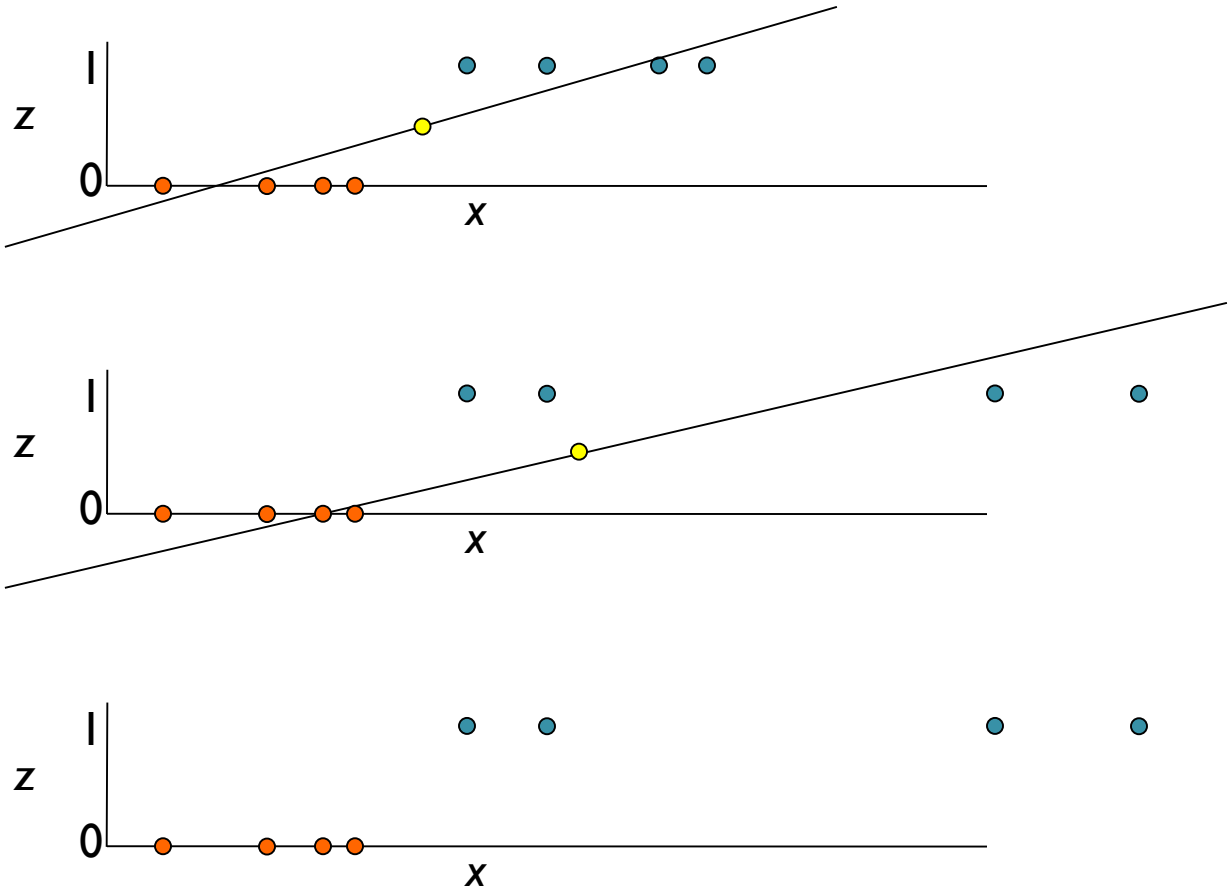
- What would happen in this adjusted case for perceptron and delta rule and where would the decision point (i.e. .5 crossing) be?

Delta Rule for Classification



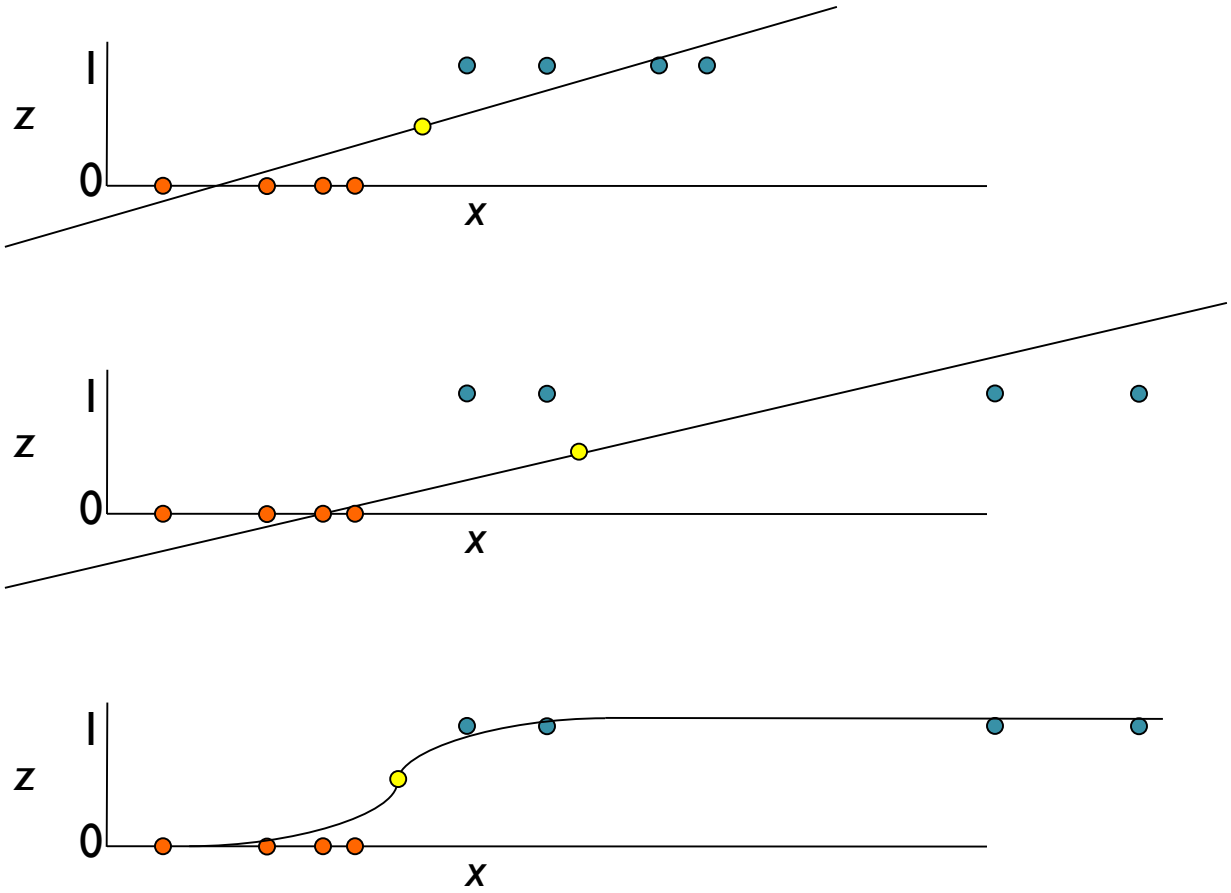
- Leads to misclassifications even though the data is linearly separable
- For Delta rule the objective function is to minimize the regression line SSE, not maximize classification

Delta Rule for Classification



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

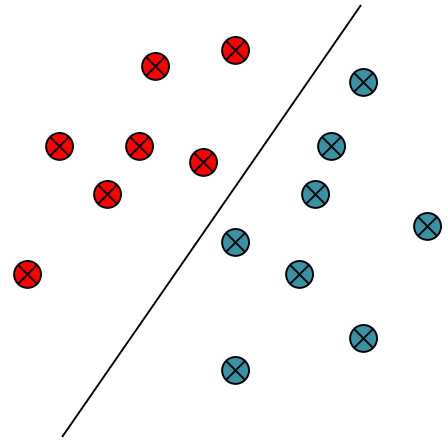
Delta Rule for Classification

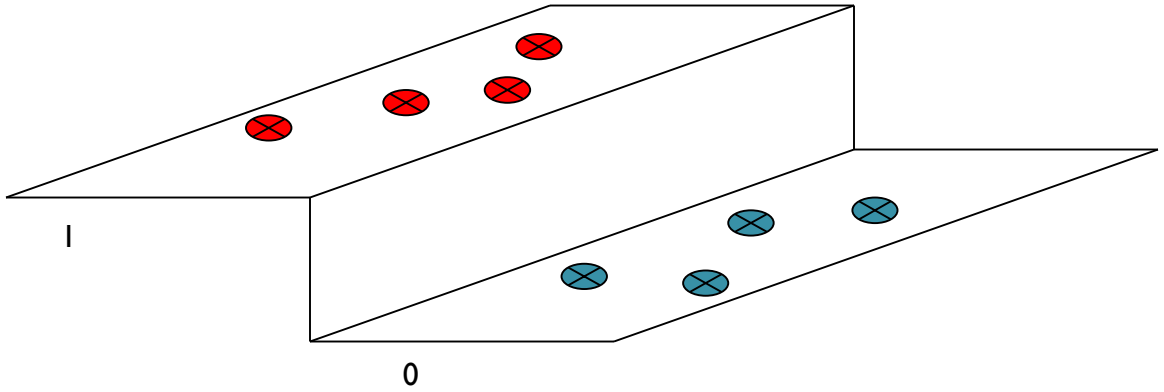


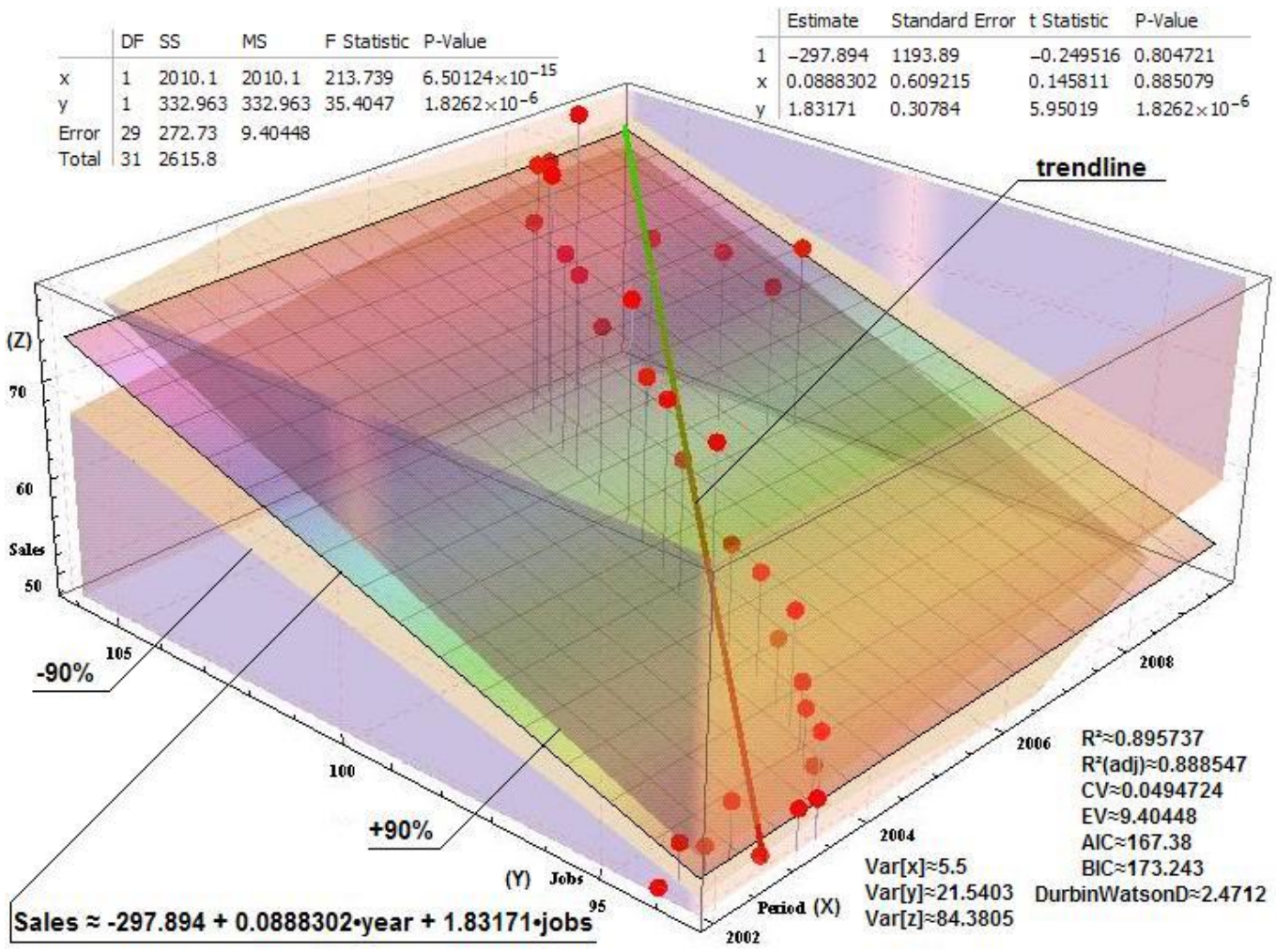
- Sigmoid fits many decision cases quite well! This is basically what logistic regression does.

Observation: Consider the 2 input perceptron case. 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?

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





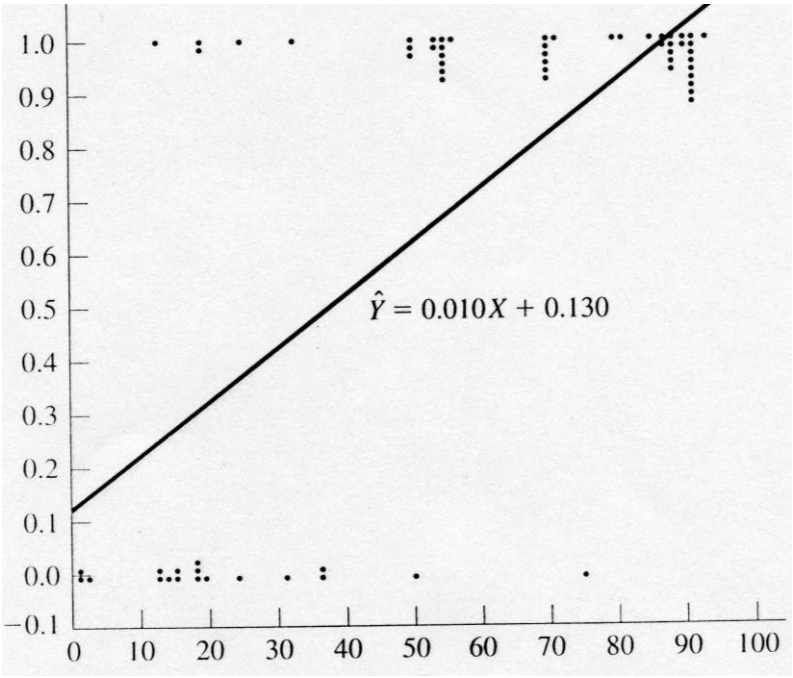

DF SS MS F Statistic P-Value
x 1 2010.1 2010.1 213.739 6.50124×10^-15
y 1 332.963 332.963 35.4047 1.8262×10^-6
Error 29 272.73 9.40448
Total 31 2615.8
Estimate Standard Error t Statistic P-Value
1 −297.894 1193.89 −0.249516 0.804721
x 0.0888302 0.609215 0.145811 0.885079
y 1.83171 0.30784 5.95019 1.8262×10^-6
trendline
(Z)
70
60
Sales
50
105
100
95
(Y) Jobs
-90%
+90%
2002
2004
2006
2008
Period (X)
Sales ≈ -297.894 + 0.0888302•year + 1.83171•jobs
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

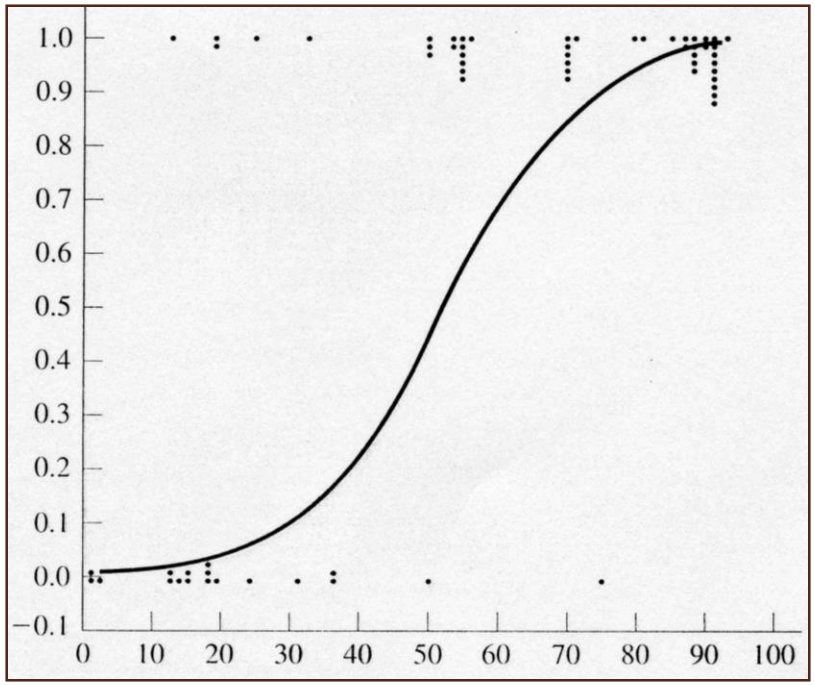
Logistic Regression

- 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 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)
- Could 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

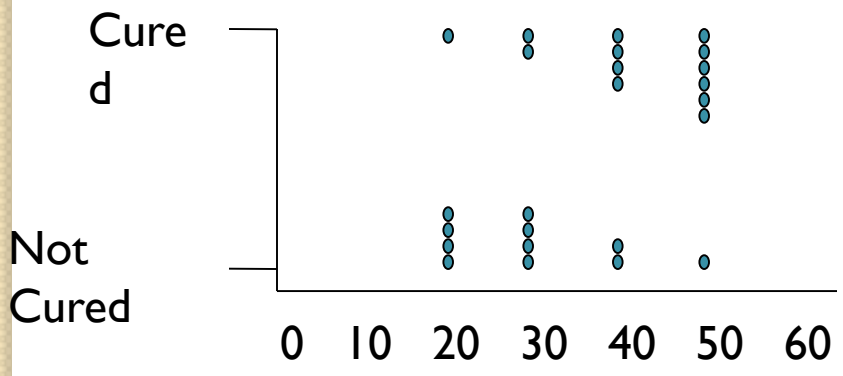
1. Transform initial input probabilities into log odds (logit)
2. Do a standard linear regression on the logit values
   - This effectively fits a logistic curve to the data, while still just doing a linear regression with the transformed input (ala quadric machine, etc.)

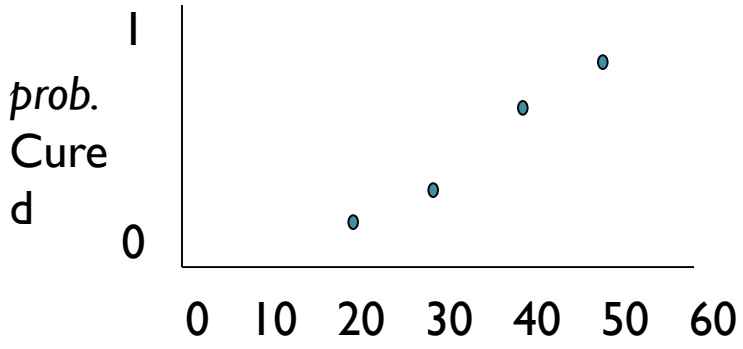
Generalization

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)

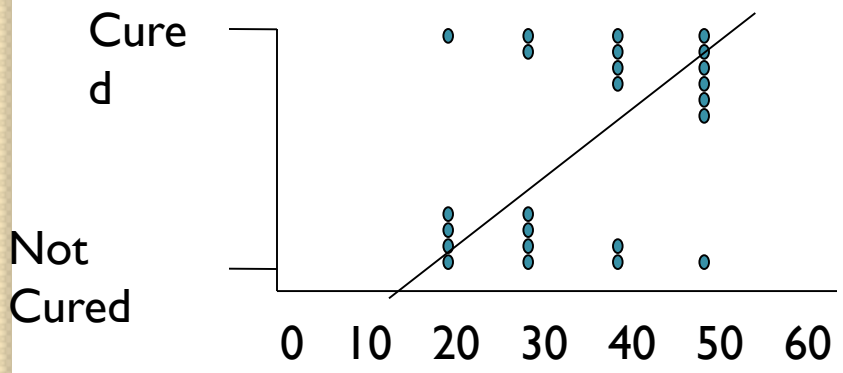
Medication Dosage	# Cured	Total Patients	Probability: # Cured/Total Patients
20	1	5	.20
30	2	6	.33
40	4	6	.67
50	6	7	.86

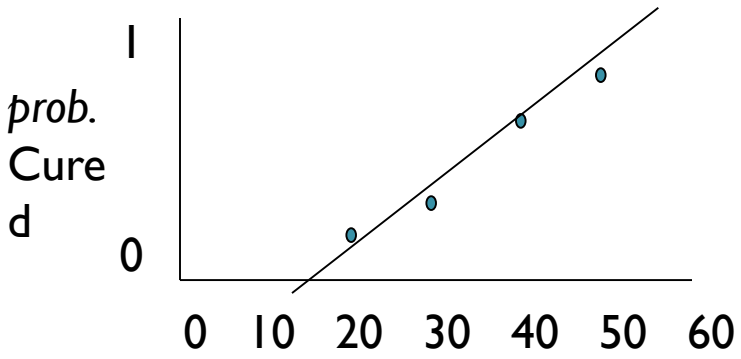




Non-Linear Pre-Process to Logit (Log Odds)

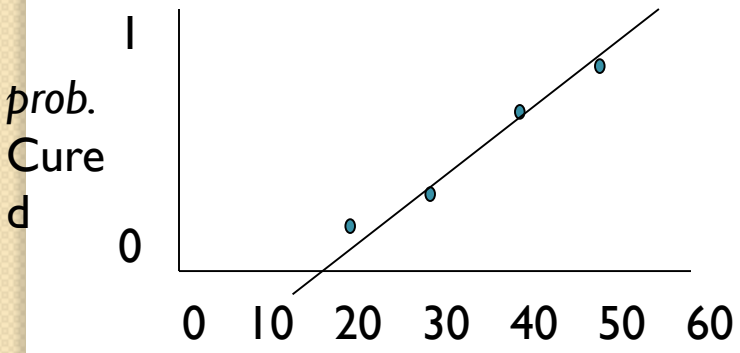
Medication Dosage	# Cured	Total Patients	Probability: # Cured/Total Patients
20	1	5	.20
30	2	6	.33
40	4	6	.67
50	6	7	.86

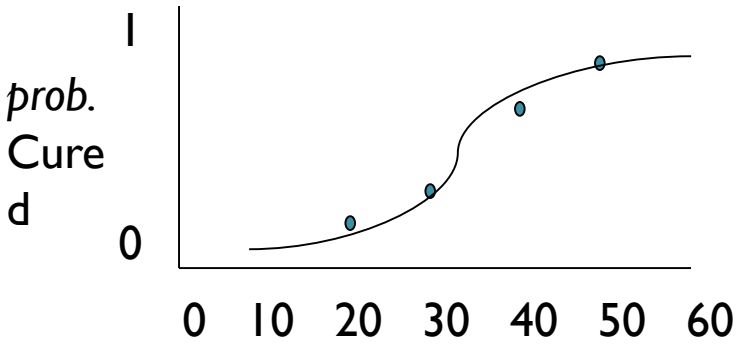




Logistic Regression Approach

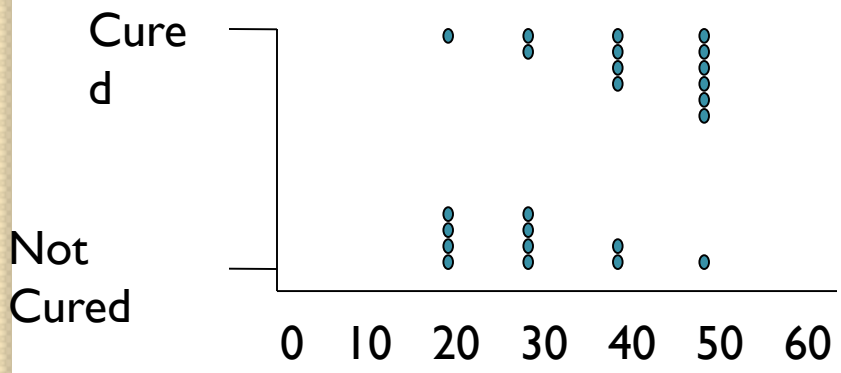
- Could use linear regression with the probability points, but that would not extrapolate well
- Logistic version is better but how do we get it?
- Similar to Quadric we do a non-linear pre-process of the input and then do linear regression on the transformed values – do a linear regression on the log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

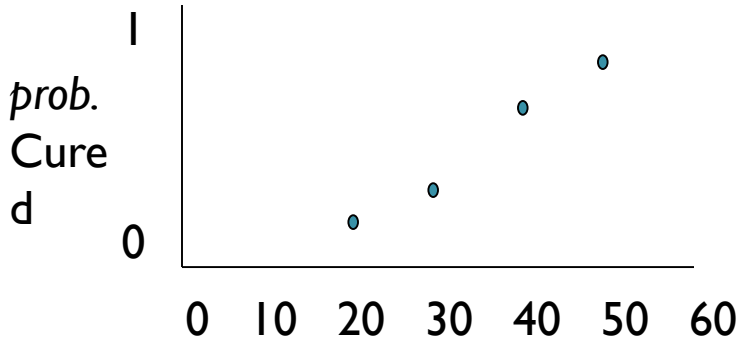




Non-Linear Pre-Process to Logit (Log Odds)

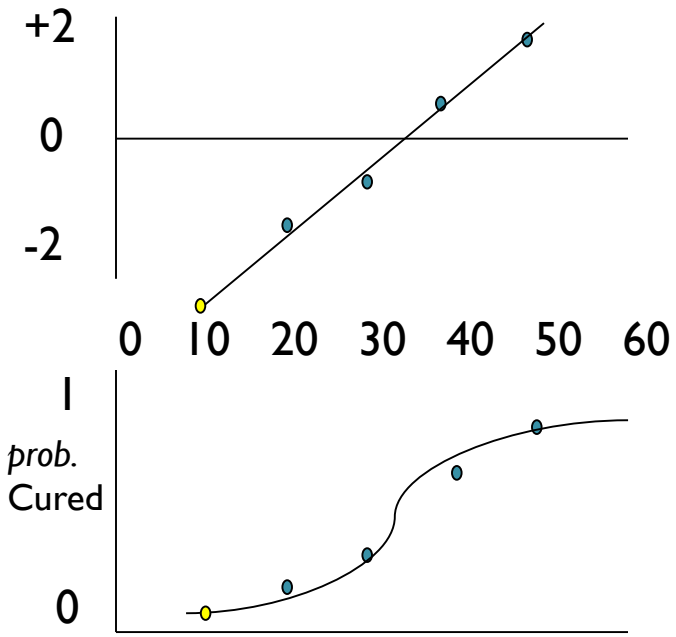
Medication Dosage	# Cured	Total Patients	Probability: # Cured/Total Patients	Odds: $p/(1-p)$ = # cured/ # not cured	Logit Log Odds: ln(Odds)
20	1	5	.20	.25	-1.39
30	2	6	.33	.50	-0.69
40	4	6	.67	2.0	0.69
50	6	7	.86	6.0	1.79





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 interpolate 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)

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.

Feature Selection, Preparation, and Reduction

- Learning accuracy depends on the data!
  - *Is the data representative of future novel cases* - critical
  - Relevance
  - Amount
  - Quality
    - Noise
    - Missing Data
    - Skew
  - Proper Representation
  - How much of the data is labeled (output target) vs. unlabeled
  - Is the number of features/dimensions reasonable?
    - Reduction

Gathering Data

- Consider the task – What kinds of features could help
- Data availability
  - Significant diversity in cost of gathering different features
  - More the better (in terms of number of instances, not necessarily in terms of number of dimensions/features)
    - The more features you have the more data you need
  - Jitter – Increased data can help with overfit – handle with care!
- Labeled data is best
- If not labeled
  - Could set up studies/experts to obtain labeled data
  - Use unsupervised and semi-supervised techniques
    - Clustering
    - Active Learning, Bootstrapping, Oracle Learning, etc.

Feature Selection - Examples

- Invariant Data
  - For character recognition: Size, Rotation, Translation Invariance
    - Especially important for visual tasks
  - Chess board features
    - Is vector of board state invariant?
- Character Recognition Class Assignment Example
  - Assume we want to draw a character with an electronic pen and have the system output which character it is
  - Assume an MLP approach with backpropagation learning
  - What features should we use and how would we train/test the system?

Data Representation

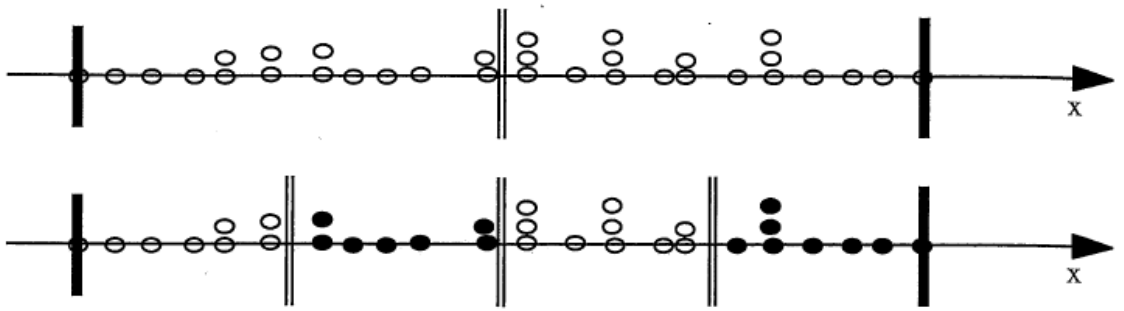
- Data Types
  - Continuous
  - Categorical/Symbolic
    - Nominal – No natural ordering
    - Ordered/Ordinal
    - Special cases: Time/Date, Addresses, Names, IDs, etc.
    - Already discussed how to transform categorical to continuous data for models (e.g. perceptrons) which want continuous inputs
- Normalization for continuous values (0-1 common)
  - What if data has skew, outliers, etc.
  - Standardization (z-score) – Transform the data by subtracting the average and then dividing by the standard deviation – allows more information on spread/outliers
  - Look at the data to make these and other decisions!

Transforming Continuous to Ordered Data

- Some models are better equipped to handle nominal/ordered data
- Basic approach is to discretize/bin the continuous data
  - How many bins – what are tradeoffs? – seek balance
  - Equal-Width Binning
    - Bins of fixed ranges
    - Does not handle skew/outliers well
  - Equal-Height Binning
    - Bins with equal number of instances
    - Uniform distribution, can help for skew and outliers
    - More likely to have breaks in high data concentrations
  - Clustering
    - More accurate, though more complex
  - Bin borders are always an issue

Supervised Binning

- The previous binning approaches do not consider the classification of each instance and thus they are unsupervised (Class-aware vs. Class-blind)
- Could use a supervised approach which attempts to bin such that learning algorithms may more easily classify
- Supervised approaches can find bins while also maximizing correlation between output classes and values in each bin
  - Often rely on information theoretic techniques



Relevant Data

- Typically do not use features where
  - Almost all instance have the same value (no information)
    - If there is a significant, though small, percentage of other values, then might still be useful
  - Almost all instances have unique values (SSN, phone-numbers)
    - Might be able to use a variation of the feature (such as area code)
  - The feature is highly correlated with another feature
    - In this case the feature may be redundant and only one is needed
  - Careful if feature is too highly correlated with the target
    - Check this case as the feature may just be a synonym with the target and will thus lead to overfitting (e.g. the output target was bundled with another product so they always occur together)

Missing Data

- Need to consider approach for learning and execution (could differ)
- Throw out data with missing attributes
  - Could lose a significant amount of training set
  - Missing attribute may contain important information, (didn't vote can mean something about congressperson, extreme measurements aren't captured, etc.).
  - Doesn't work during execution
- Set (impute/imputation) attribute to its mode/mean (based on rest of data set)
  - too big an assumption?
- Set attribute to its mode/mean given the output class (only works for training)
- Use a learning scheme (NN, DT, etc) to impute missing values
  - Train imputing models with a training set which has the missing attribute as the target and the rest of the attributes (including the original target) as input features. Better accuracy, though more time consuming - multiple missing values?
- Impute based on the most similar complete instance(s) in the data set
- Train multiple reduced input models to handle common cases of missing data
- Let unknown be just another attribute value – Can work well in many cases
  - Natural for nominal data
  - With continuous data, can use an indicator node, or a value which does not occur in the normal data (-1, outside range, etc.), however, in the latter case, the model will treat this as an extreme ordered feature value and may cause difficulties

Dirty Data and Data Cleaning

- Dealing with bad data, inconsistencies, and outliers
- Many ways errors are introduced
  - Measurement Noise/Outliers
  - Poor Data Entry
  - User lack of interest
    - Most common birthday when B-day mandatory: November 11, 1911
    - Data collectors don't want blanks in data warehousing so they may fill in (impute) arbitrary values
- Data Cleaning
  - Data analysis to discover inconsistencies
  - Noise/Outlier removal – Requires care to know when it is noise and how to deal with this during execution – Our experiments show outlier removal during training increases subsequent accuracy.
  - Clustering/Binning can sometimes help

Labeled and Unlabeled Data

- Accurately labeled data is always best
- Often there is lots of cheaply available unlabeled data which is expensive/difficult to label – internet data, etc.
- Semi-Supervised Learning – Can sometimes augment a small set of labeled data with lots of unlabeled data to gain improvements
- Active Learning – Out of a large collection of unlabeled data, interactively select the next most informative instance to label
- Bootstrapping: Iteratively use current labeled data to train model, use the trained model to label the unlabeled data, then train again including most confident newly labeled data, and re-label, etc., until some convergence
- Combinations of above and other techniques being proposed

Feature Selection and Feature Reduction

- Given *n* original features, it is often advantageous to reduce this to a smaller set of features for actual training
  - Can improve/maintain accuracy if we can preserve the most relevant information while discarding the most irrelevant information
  - and/or Can make the learning process more computationally and algorithmically manageable by working with less features
  - Curse of dimensionality requires an exponential increase in data set size in relation to the number of features to learn without overfit – thus decreasing features can be critical
- *Feature Selection* seeks a subset of the *n* original features which retains most of the relevant information
  - Filters, Wrappers
- *Feature Reduction* combines the *n* original features into a new smaller set of features which hopefully retains most of the relevant information from all features - Data fusion (e.g. LDA, PCA, etc.)

Feature Selection - Filters

- Given *n* original features, how do you select size of subset
  - User can preselect a size *m* (< *n*)
  - Can find the smallest size where adding more features does not yield improvement
- Filters work independent of any particular learning algorithm
- Filters seek a subset of features which maximize some type of between class separability – or other merit score
- Can score each feature independently and keep best subset
  - e.g. 1st order correlation with output, fast, less optimal
- Can score subsets of features together
  - Exponential number of subsets requires a more efficient, sub-optimal search approach
  - How to score features independent of the ML model to be trained on is an important research area
  - Decision Tree or other ML model pre-process

Feature Selection - Wrappers

- Optimizes for a specific learning algorithm
- The feature subset selection algorithm is a "wrapper" around the learning algorithm
  1. Pick a feature subset and pass it in to learning algorithm
  2. Create training/test set based on the feature subset
  3. Train the learning algorithm with the training set
  4. Find accuracy (objective) with test set
  5. Repeat for all feature subsets and pick the feature subset which led to the highest predictive accuracy (or other objective)
- Basic approach is simple
- Variations are based on how to select the feature subsets, since there are an exponential number of subsets

Feature Selection - Wrappers

- Exhaustive Search - Exhausting
- Forward Search – O(n^2 · learning/testing time) - Greedy
  1. Score each feature by itself and add the best feature to the initially empty set FS (FS will be our final Feature Set)
  2. Try each subset consisting of the current FS plus one remaining feature and add the best feature to FS
  3. Continue until either hit goal of *m*, or stop getting significant improvement
- Backward Search – O(n^2 · learning/testing time) - Greedy
  1. Score the initial complete set FS (FS will be our final Feature Set)
  2. Try each subset consisting of the current FS minus one feature in FS and drop the feature from FS causing least decrease in accuracy
  3. Continue until either hit goal of *m*, or begin to get significant decreases in accuracy
- Branch and Bound and other heuristic approaches available

PCA – Principal Components Analysis

- PCA is one of the most common feature reduction techniques
- A linear method for dimensionality reduction
- Allows us to combine much of the information contained in *n* features into *m* features where $m < n$
- PCA is *unsupervised* in that it does not consider the output class/value of an instance – Are other algorithms which do (e.g. Linear Discriminant Analysis)
- PCA works well in many cases where data has mostly linear correlations
- Non-linear dimensionality reduction is also a relatively new and successful area and can give much better results for data with significant non-linearities

PCA Overview

- Seek new set of bases which correspond to the highest variance in the data
- Transform *n*-dimensional data to a new *n*-dimensional basis
  - The new dimension with the most variance is the first principal component
  - The next is the second principal component, etc.
  - Note z_1 fuses significant information from both x_1 and x_2
- Drop those dimensions for which there is little variance

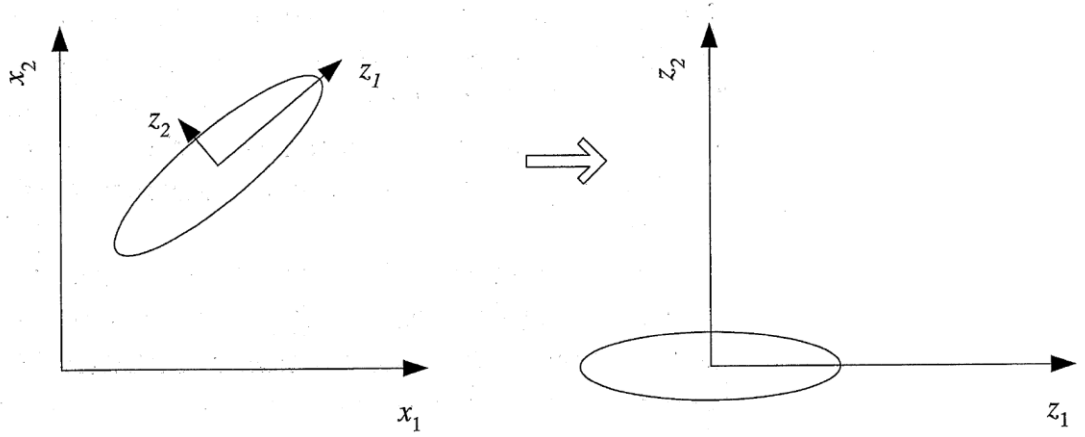


Figure 6.1 Principal components analysis centers the sample and then rotates the axes to line up with the directions of highest variance. If the variance on z_2 is too small, it can be ignored and we have dimensionality reduction from two to one.

Variance and Covariance

- Variance is a measure of data spread in one dimension (feature)
- Covariance measures how two dimensions (features) vary with respect to each other

$$\text{var}(X) = \frac{\sum_{i=1}^{n} (X_i - \overline{X})(X_i - \overline{X})}{(n - 1)}$$

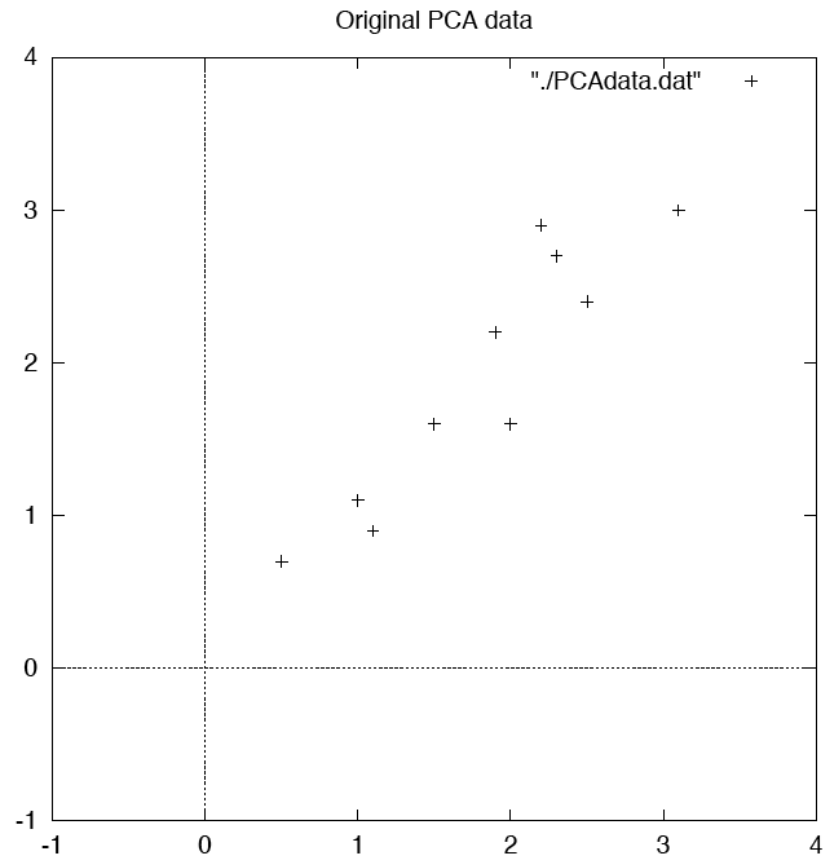
$$\text{cov}(X,Y) = \frac{\sum_{i=1}^{n} (X_i - \overline{X})(Y_i - \overline{Y})}{(n - 1)}$$

Covariance and the Covariance Matrix

- Considering the sign (rather than exact value) of covariance:
  - Positive value means that as one feature increases or decreases the other does also (positively correlated)
  - Negative value means that as one feature increases the other decreases and vice versa (negatively correlated)
  - A value close to zero means the features are independent
  - If highly covariant, are both features necessary?
- Covariance matrix is an $n \times n$ matrix containing the covariance values for all pairs of features in a data set with n features (dimensions)
- The diagonal contains the covariance of a feature with itself which is the variance (which is the square of the standard deviation)
- The matrix is symmetric

PCA Example

- First step is to center the original data around 0 by subtracting the mean in each dimension



X	Y				$X¢$	$Y¢$
2.5	2.4				0.69	0.49
0.5	0.7				- 1.31	- 1.21
2.2	2.9				0.39	0.99
1.9	2.2				0.09	0.29
3.1	3.0	Þ	$\overline{X}=1.81$ $\overline{Y}=1.91$	Þ	1.29	1.09
2.3	2.7				0.49	0.79
2.0	1.6				0.19	- 0.31
1.0	1.1				- 0.81	- 0.81
1.5	1.6				- 0.31	- 0.31
1.2	0.9				- 0.71	- 1.01

PCA Example

- Second: Calculate the covariance matrix of the centered data
- Only 2 × 2 for this case

X	Y			$X¢$	$Y¢$
2.5	2.4			0.69	0.49
0.5	0.7			- 1.31	- 1.21
2.2	2.9			0.39	0.99
1.9	2.2			0.09	0.29
3.1	3.0	Þ	$\overline{X}=1.81$ $\overline{Y}=1.91$	Þ 1.29	1.09
2.3	2.7			0.49	0.79
2.0	1.6			0.19	- 0.31
1.0	1.1			- 0.81	- 0.81
1.5	1.6			- 0.31	- 0.31
1.2	0.9			- 0.71	- 1.01

$$\text{cov}(X,Y) = \frac{\sum_{i=1}^{n} (X_i - \overline{X})(Y_i - \overline{Y})}{(n - 1)}$$

$$\text{cov} = \begin{bmatrix} 0.616555556 & 0.615444444 \\ 0.615444444 & 0.716555556 \end{bmatrix}$$

PCA Example

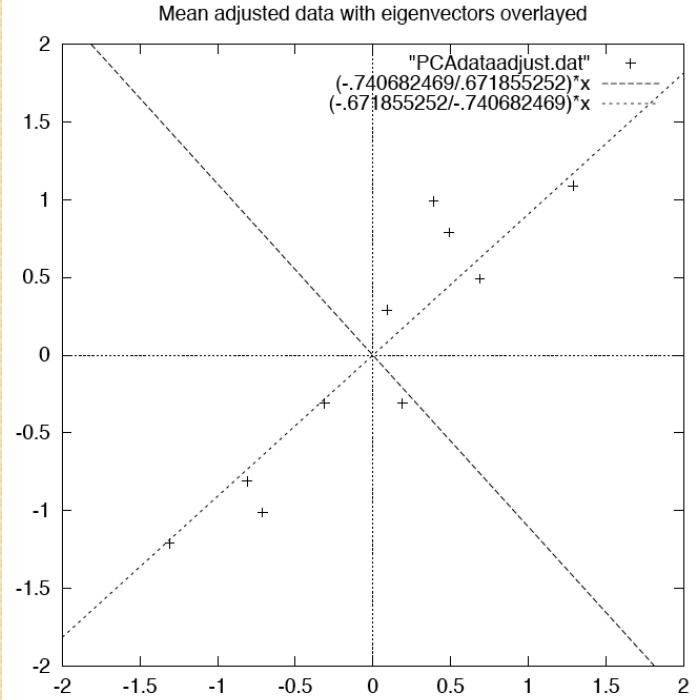
- Third: Calculate the unit eigenvectors and eigenvalues of the covariance matrix (remember linear algebra)
  - Covariance matrix is always square $n \times n$ and positive semi-definite, thus n non-negative eigenvalues will exist
  - All eigenvectors (principal components/dimensions) are orthogonal to each other and will make the new set of bases/dimensions for the data
  - The magnitude of each eigenvalue corresponds to the variance along that new dimension – Just what we wanted!
  - We can sort the principal components according to their eigenvalues
  - Just keep those dimensions with the largest eigenvalues

$$eigenvalues = \begin{bmatrix} 0.490833989 \\ 1.28402771 \end{bmatrix}$$

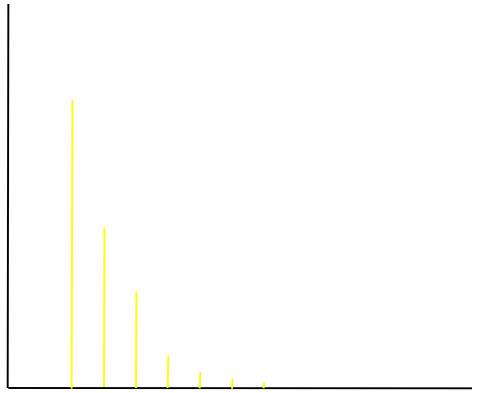
$$eigenvectors = \begin{bmatrix} -0.735178656 & -0.677873399 \\ 0.677873399 & -0.735178656 \end{bmatrix}$$

PCA Example

- Below are the two eigenvectors overlaying the centered data
- Which eigenvector has the largest eigenvalue?
- Fourth Step: Just keep the *p* eigenvectors with the largest eigenvalues
  - Do lose some information, but if we just drop dimensions with small eigenvalues then we lose only a little information
  - We can then have *p* input features rather than *n*
  - The *p* features contain the most pertinent combined information from all *n* original features
  - How many dimensions *p* should we keep?



Eigenvalue



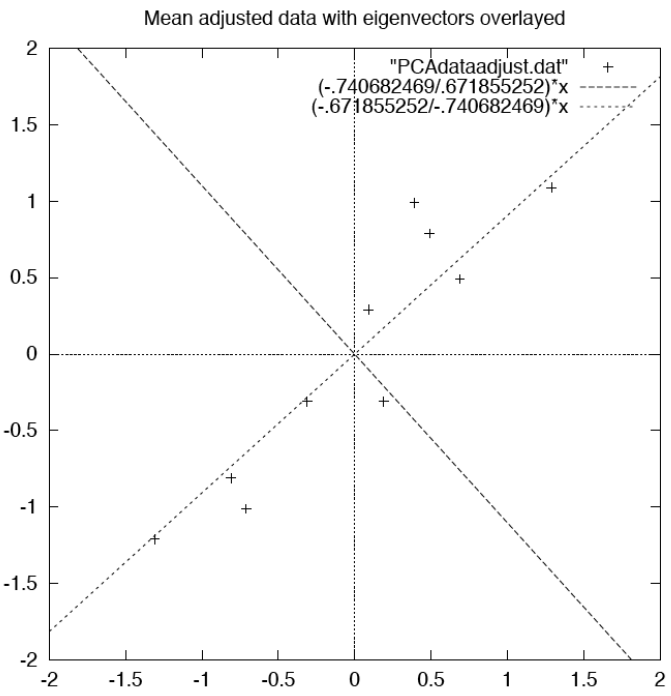
1 2 3 4 5 6 7 … *n*

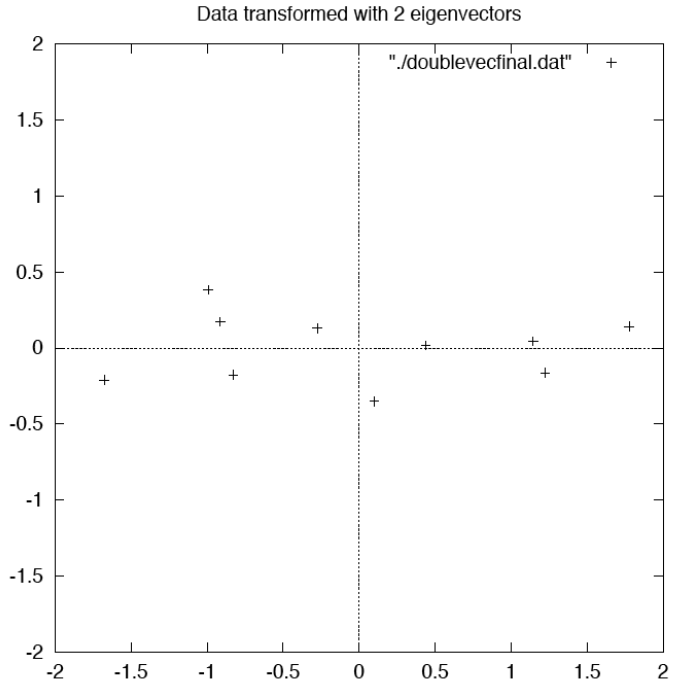
Proportion of Variance

$$\frac{\sum_{i=1}^{p} \lambda_i}{\sum_{i=1}^{n} \lambda_i} = \frac{\lambda_1 + \lambda_2 + \cdots + \lambda_p}{\lambda_1 + \lambda_2 + \cdots + \lambda_p + \cdots + \lambda_n}$$

PCA Example

- Last Step: Transform the features to the p chosen bases (Eigenvectors)
- Transformed data (m instances) is a matrix multiply $T = A \times B$
  - A is a $p \times n$ matrix with the p principal components in the rows, component one on top
  - B is a $n \times m$ matrix containing the transposed centered original data set
  - T^T is a $m \times p$ matrix containing the transformed data set
- Now we have the new transformed data set with dimensionality p
- Keep matrix A to transform future 0-centered data instances
- Below shows transform of both dimensions, would if we just kept the 1st component





PCA Algorithm Summary

1. Center the *m* TS features around 0 (subtract *m* means)
2. Calculate the covariance matrix of the centered TS
3. Calculate the unit eigenvectors and eigenvalues of the covariance matrix
4. Keep the *p* (< *m*) eigenvectors with the largest eigenvalues
5. Matrix multiply the *p* eigenvectors with the centered TS to get a new TS with only *p* features

- Given a novel instance during execution
  1. Center instance around 0
  2. Do the matrix multiply (step 5 above) to change the new instance from *m* to *p* features