SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Established in collaboration with MIT

Regression and Evaluation Metrics

Soujanya Poria

Objectives

- Understand linear, non-linear regression algorithms
- Understand evaluation algorithm for classification and regression

Regression

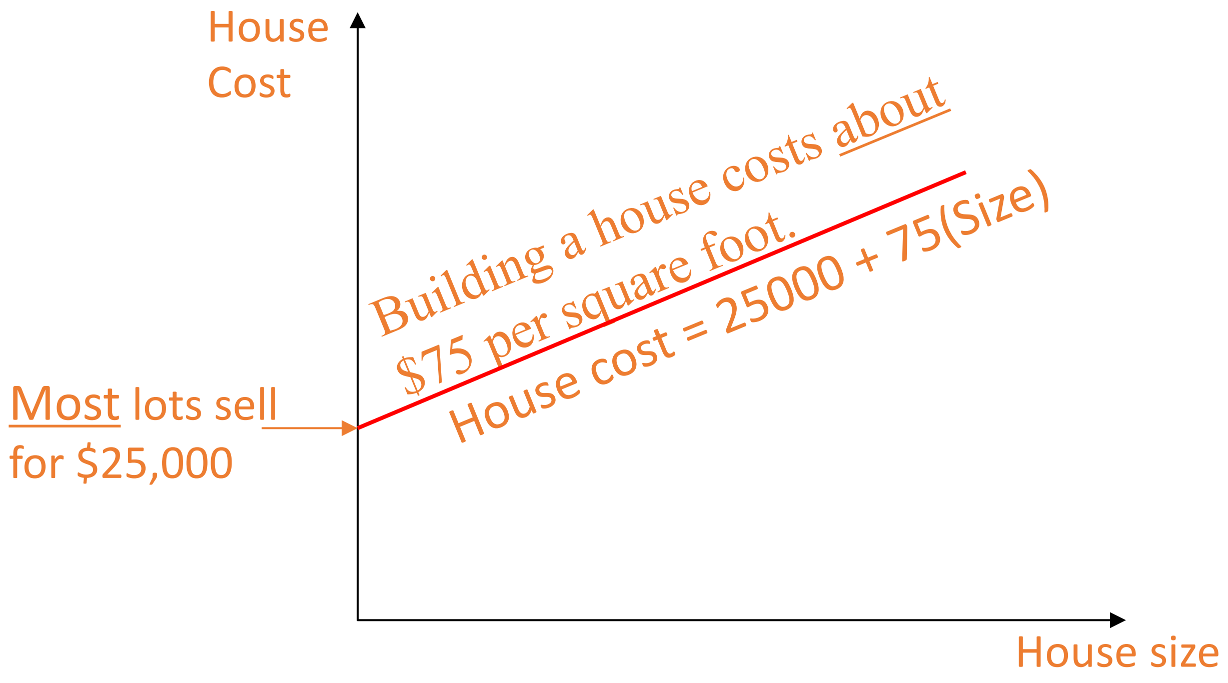
‘’Regression analysis is a form of predictive modelling technique which investigates the relationship between
a dependent (target) and independent variable (s) (predictor). This technique is used for forecasting, time series modelling and finding the causal effect relationship between the variables. For example, relationship between rash driving and number of road accidents by a driver is best studied through regression.’’

Introduction

- We will examine the relationship between quantitative variables x and y via a mathematical equation.
- The motivation for using the technique:
  - Forecast the value of a dependent variable (y) from the value of independent variables (x_1, x_2,…x_k.).
  - Analyze the specific relationships between the independent variables and the dependent variable.

The Model

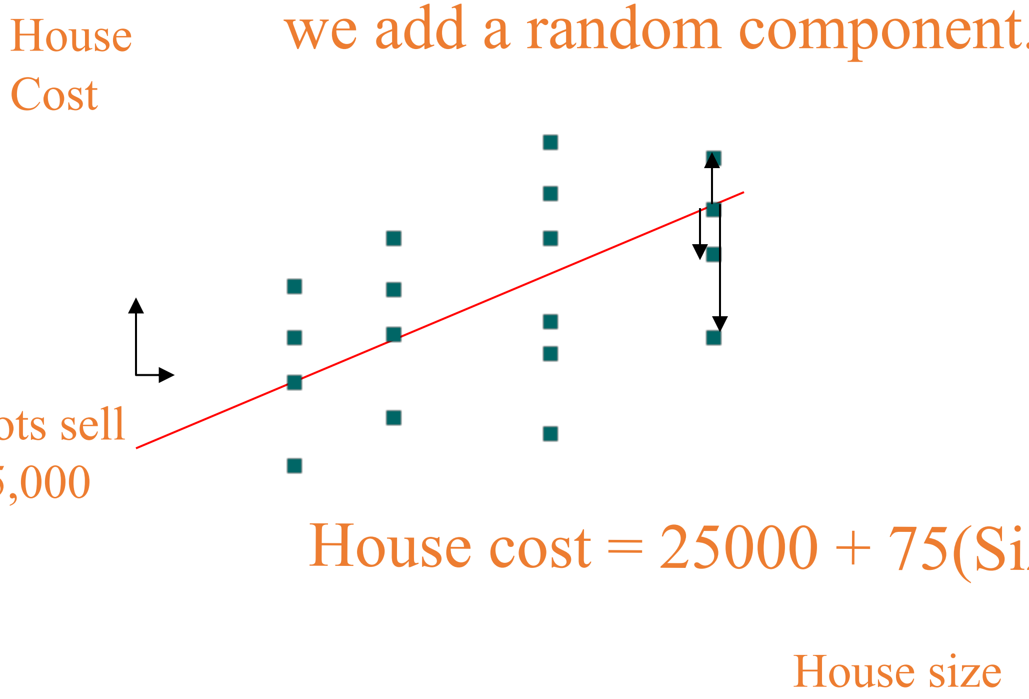
The model has a deterministic and a probabilistic components


House Cost
Building a house costs about $75 per square foot.
House cost = 25000 + 75(Size)
Most lots sell for $25,000
House size

The Model

However, house cost vary even among same size houses!

Since cost behave unpredictably, we add a random component.



House cost = 25000 + 75(Size) **+ e**

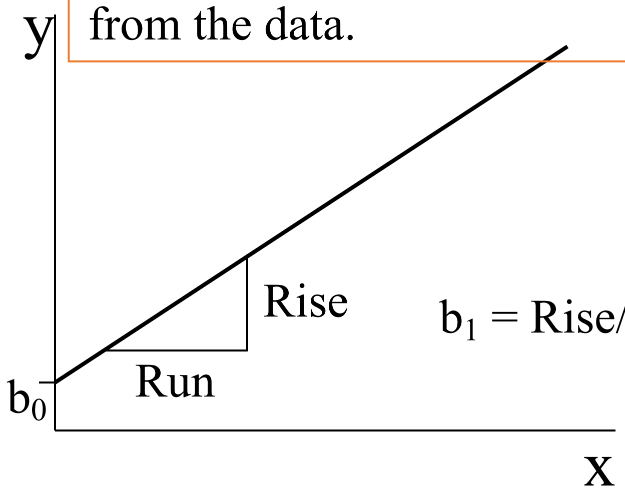
The Model

- The first order linear model

$$y = \beta_0 + \beta_1 x + \varepsilon$$

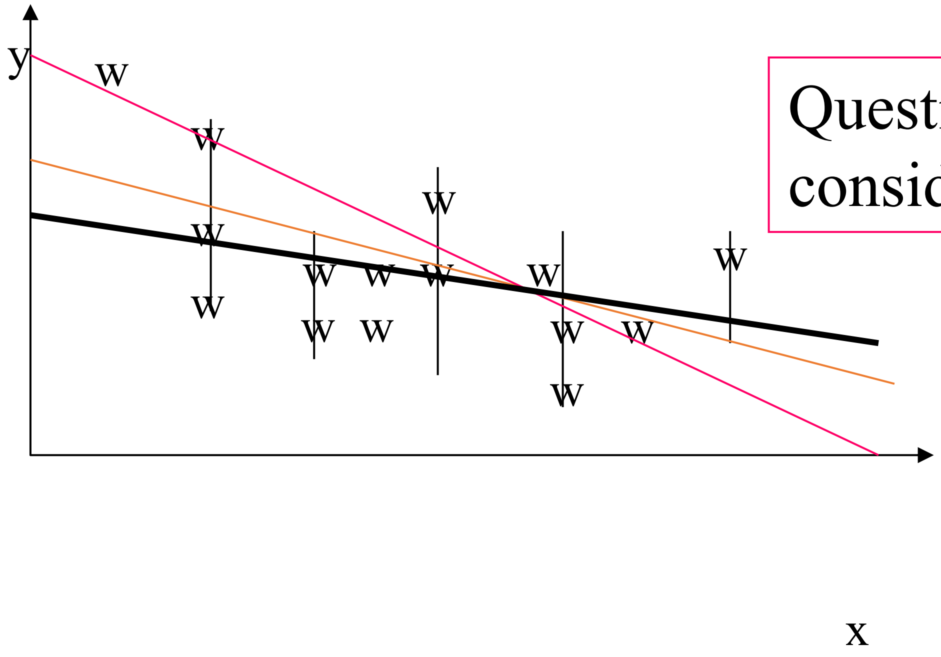
y = dependent variable
x = independent variable
b_0 = y-intercept
b_1 = slope of the line
e = error variable

b_0 and b_1 are unknown population parameters, therefore are estimated from the data.



Estimating the Coefficients

- The estimates are determined by
  - drawing a sample from the population of interest,
  - calculating sample statistics.
  - producing a straight line that cuts into the data.



Question: What should be considered a good line?

The Least Squares (Regression) Line

A good line is one that minimizes
the sum of squared differences between the points and the line.

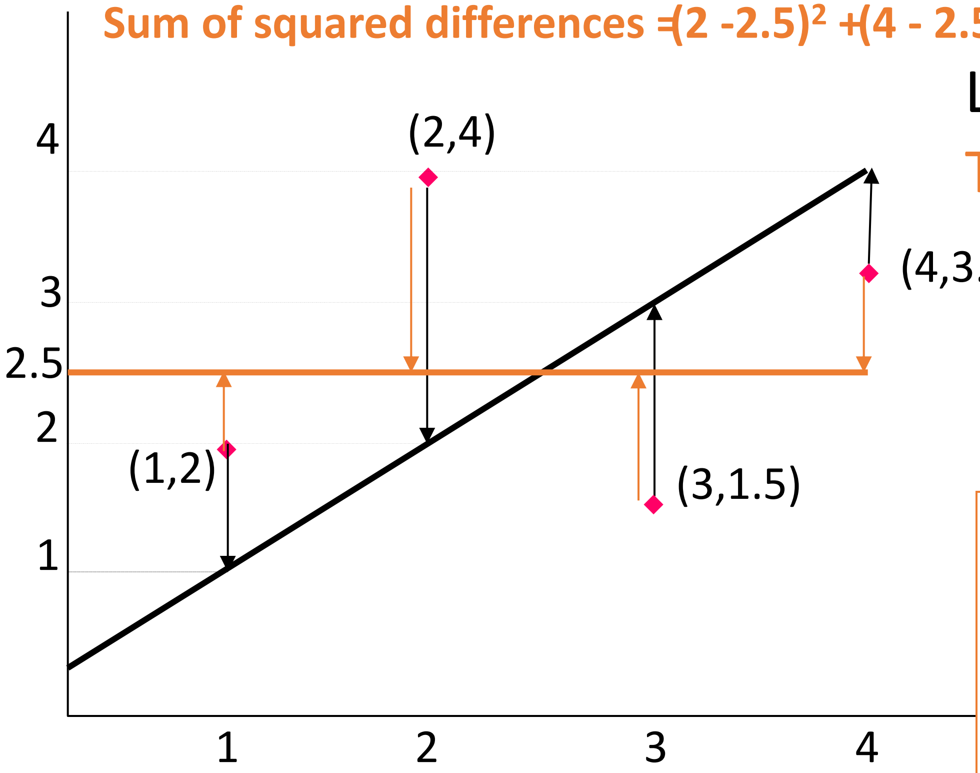
The Least Squares (Regression) Line

Sum of squared differences = $(2 - 1)^2 + (4 - 2)^2 + (1.5 - 3)^2 + (3.2 - 4)^2 = 6.89$

Sum of squared differences = $(2 - 2.5)^2 + (4 - 2.5)^2 + (1.5 - 2.5)^2 + (3.2 - 2.5)^2 = 3.99$

Let us compare two lines

The second line is horizontal



The smaller the sum of squared differences the better the fit of the line to the data.

1. Given N inputs and outputs…

$$(x_i, y_i)$$

2. We define the line of best fit line as…

$$\hat{Y}_i = a + Bx_i$$

3. Such that the best fit line looks to minimize the cost function we named S…

$$S = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

For our reference, we will input the line of best fit into our cost function distributing the subtraction, resulting in…

$$S = \sum_{i=1}^{n} (Y_i - a - Bx_i)^2$$

The idea is to differentiate the cost/loss function with respect to the weights (it is "*a*" in this case) and equate it to 0

Finding a

$$\frac{\partial S}{\partial a}[\sum_{i=1}^{n}(Y_i - a - Bx_i)^2]$$

$$0 = \sum_{i=1}^{n} -2(Y_i - a - Bx_i)$$

$$0 = \sum_{i=1}^{n} Y_i - \sum_{i=1}^{n} a - B \sum_{i=1}^{n} x_i$$

We notice that summation of a to n is simply…

$$\sum_{i=1}^{n} a = na$$

Substituting this back in and rearranging B, give us…

$$0 = \sum_{i=1}^{n} Y_i - na - B \sum_{i=1}^{n} x_i$$

$$a = \frac{\sum_{i=1}^{n} Y_i - B \sum_{i=1}^{n} x_i}{n}$$

$$a = \bar{Y} - B\bar{x}$$

To calculate the optimal value of the weight "**b**" when the loss is minimum, differentiate the loss/cost function with respect to "b" and equate it to 0.

$$\frac{\partial S}{\partial b}[\sum_{i=1}^{n}(Y_i - a - Bx_i)^2]$$

$$0 = \sum_{i=1}^{n} -2x_i(Y_i - a - Bx_i)$$

$$0 = \sum_{i=1}^{n} (x_i Y_i - a x_i - B x_i^2)$$

$$a = \bar{Y} - B\bar{x}$$

$$0 = \sum_{i=1}^{n} (x_i Y_i - (\bar{Y} - B\bar{x}) x_i - B x_i^2)$$

$$B = \frac{\sum_{i=1}^{n}(x_i Y_i - \bar{Y} x_i)}{\sum_{i=1}^{n}(x_i^2 - \bar{x} x_i)}$$

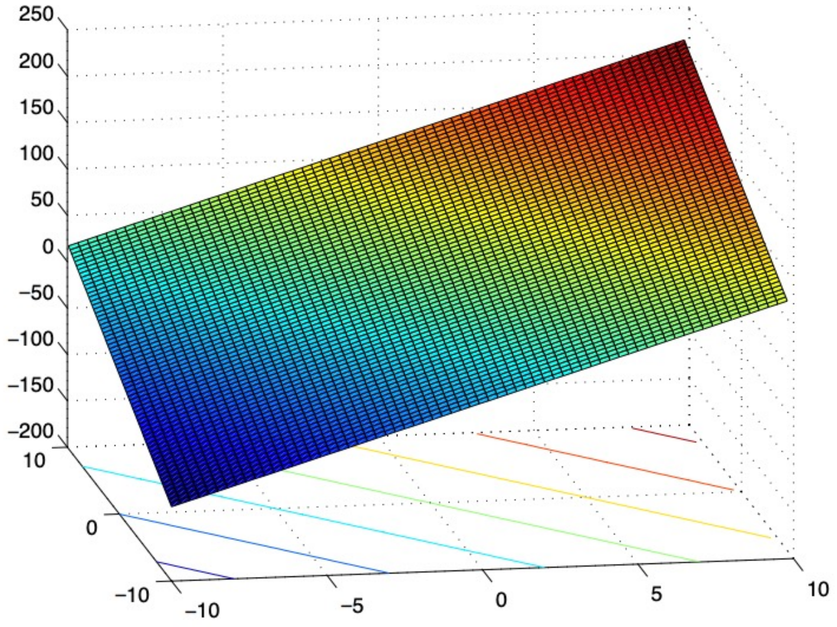
Multiple linear regression

The simplest form of this regression is

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

When we have more than two independent variables i.e., x_1 and x_2 in this case, we apply multiple linear regression

$$y = 50 + 10x_1 + 7x_2$$



Multiple linear regression

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1K} \\ x_{21} & x_{22} & \cdots & x_{2K} \\ \vdots & \ddots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{NK} \end{bmatrix} * \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_K \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{bmatrix}$$

it all comes down to minimzing $e'e$:

$$\epsilon'\epsilon = \begin{bmatrix} e_1 & e_2 & \cdots & e_N \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix} = \sum_{i=1}^{N} e_i^2$$

Multiple linear regression

So minimizing $e'e'$ gives us:

$$min_b \, e'e = (y - Xb)'(y - Xb)$$

$$min_b \, e'e = y'y - 2b'X'y + b'X'Xb$$

$$\frac{\partial(e'e)}{\partial b} = -2X'y + 2X'Xb \overset{!}{=} 0$$

$$X'Xb = X'y$$

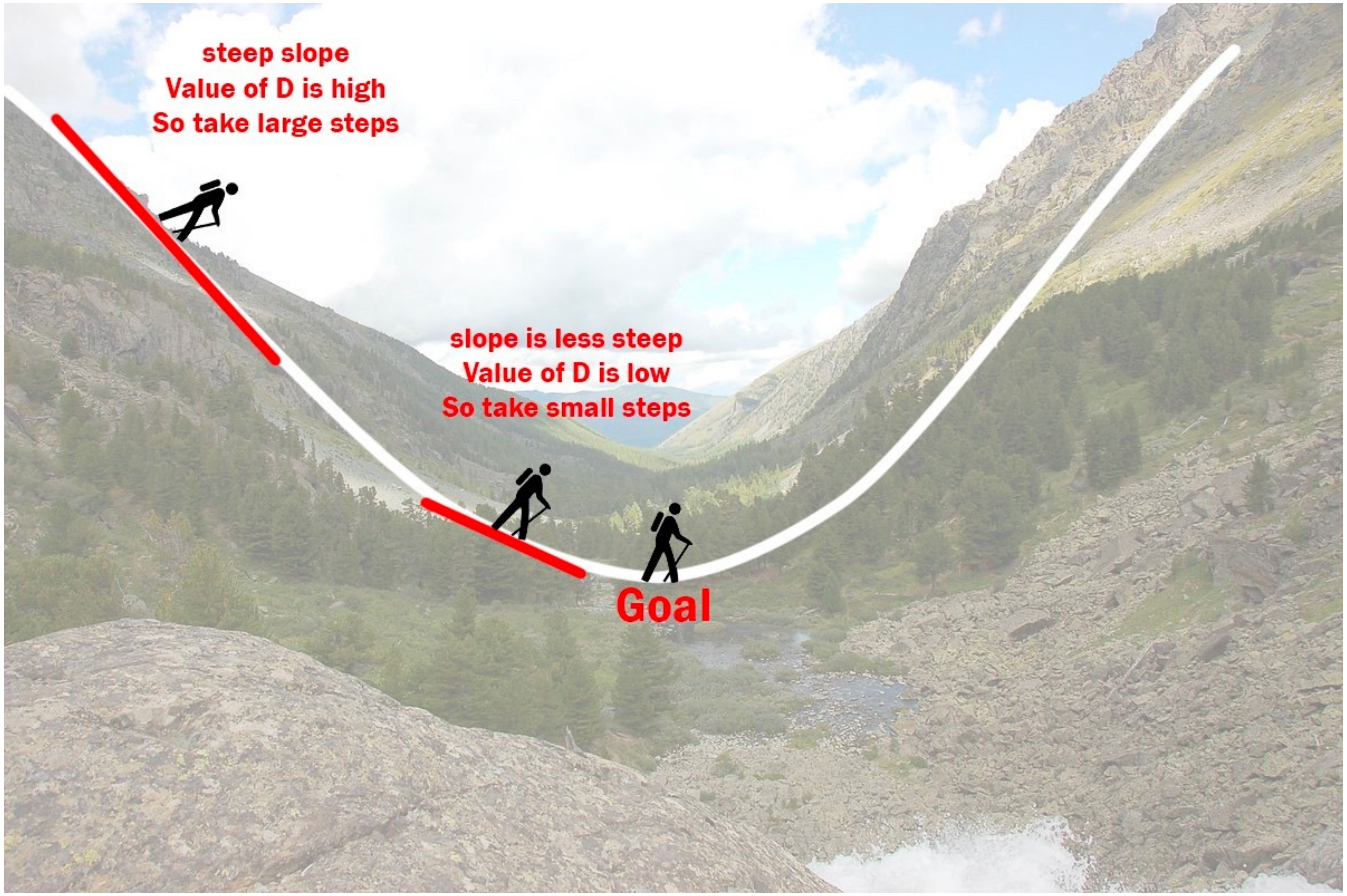
$$b = (X'X)^{-1}X'y$$

This operation is computationally VERY expensive when we have large dataset.

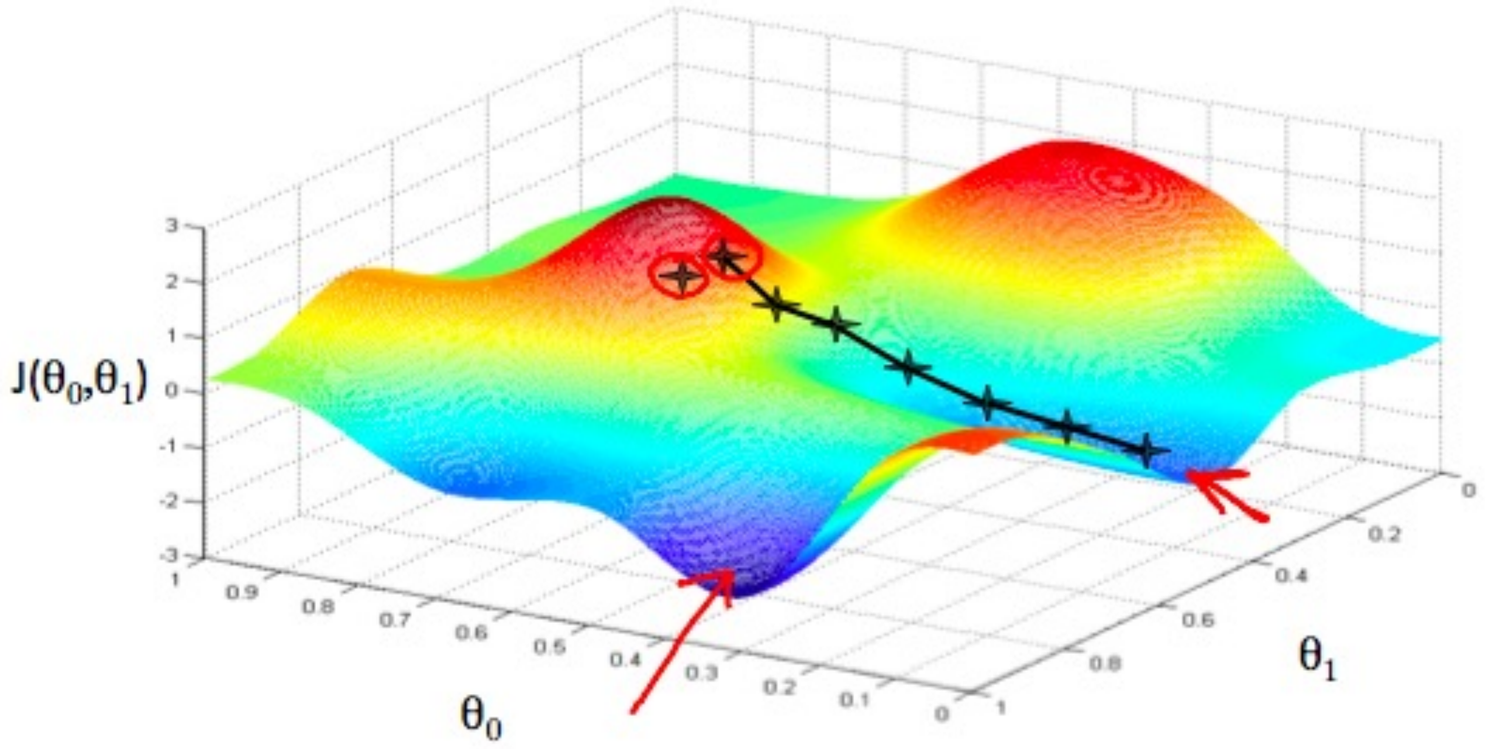
Solution???

- Apply gradient descent
  - A greedy approach to find the optimum solution
  - At each iteration we find the gradient of the loss with respect to the weights that we want to learn.
  - Update the weights using the computed gradient until the training converges.

Gradient Descent



Gradient Descent



Gradient Descent

α is learning rate and a hyperparameter. Setting it too high can skip the optimal solution. Setting it too low can make the training very slow.

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1)$$

}

Gradient of the loss with respect to the weight

Updated theta

Correct: Simultaneous update

$$\begin{aligned} \text{temp0} &:= \theta_0 - \alpha \tfrac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\ \text{temp1} &:= \theta_1 - \alpha \tfrac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\ \theta_0 &:= \text{temp0} \\ \theta_1 &:= \text{temp1} \end{aligned}$$

Gradient Descent

α is learning rate

Incorrect:

$$
\begin{aligned}
&\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\
&\theta_0 := \text{temp0} \\
&\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\
&\theta_1 := \text{temp1}
\end{aligned}
$$

Linear regression using gradient descent

$$Y = mX + c$$

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)^2$$

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - (mx_i + c))^2$$

Linear regression using gradient descent

$$D_m = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$

$$D_m = \frac{-2}{n}\sum_{i=0}^{n} x_i(y_i - \bar{y}_i)$$

Update m at each iteration

L is learning rate

$$m = m - L \times D_m$$

Update c at each iteration

$$c = c - L \times D_c$$

Evaluation metrics

- Evaluation metrics for classification
- Evaluation metrics for regression

Precision and Recall

When evaluating a search tool or a classifier, we are interested in at least two performance measures:

Precision: Within a given set of positively-labeled results, the fraction that were true positives = tp/(tp + fp)

Recall: Given a set of positively-labeled results, the fraction of all positives that were retrieved = tp/(tp + fn)

Positively-labeled means judged “relevant” by the search engine or labeled as in the class by a classifier. tp = true positive, fp = false positive etc.

Be careful of “Accuracy”

The simplest measure of performance would be the fraction of items that are correctly classified, or the “accuracy” which is:

$$\frac{tp + tn}{tp + tn + fp + fn}$$

But this measure is dominated by the larger set (of positives or negatives) and favors trivial classifiers.

e.g. if 5% of items are truly positive, then a classifier that always says “negative” is 95% accurate.

Weighted loss

We can instead try to minimize a weight sum:

$$w_1 \text{ fn} + \ w_2 \text{ fp}$$

And typically $w_1 \gg \ w_2$, since positives are often much rarer (clicks or purchases or viewing a movie).

The weighted “F” measure

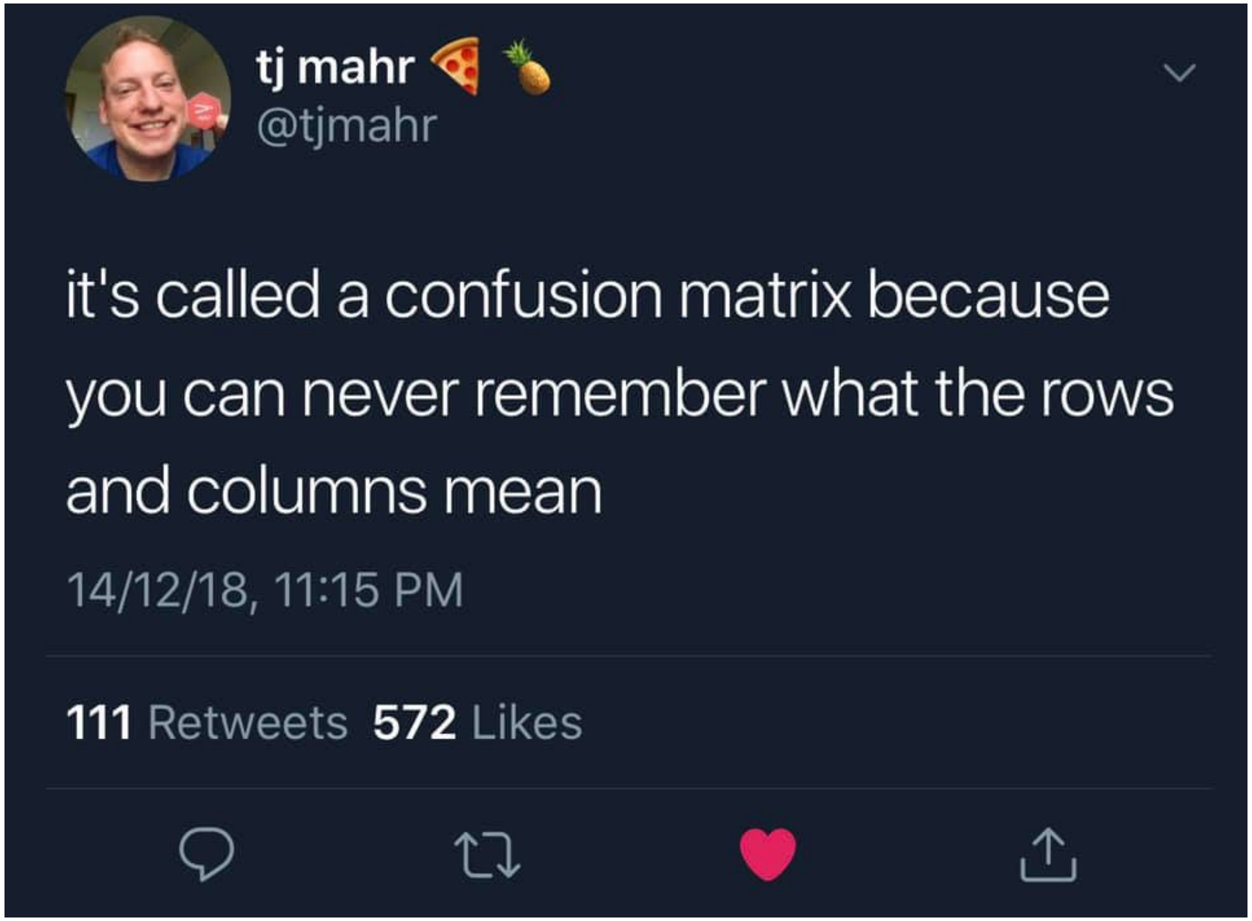
A measure that naturally combines precision and recall is the β-weighted F-measure:

$$F = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

Which is the weighted harmonic mean of precision and recall. Setting β = 1 gives us the F_1 – measure. It can also be computed as:

$$F_{\beta=1} = \frac{2PR}{P+R}$$

Confusion matrix


tj mahr 🍕 🍍
@tjmahr

it's called a confusion matrix because you can never remember what the rows and columns mean

14/12/18, 11:15 PM

111 Retweets 572 Likes

Confusion matrix

	Predict as 1	Predict as 0
Actual 1	8	2
Actual 0	20	970

RMSE (Root Mean Square Error)

Mathematically, RMSE is calculated using this formula:

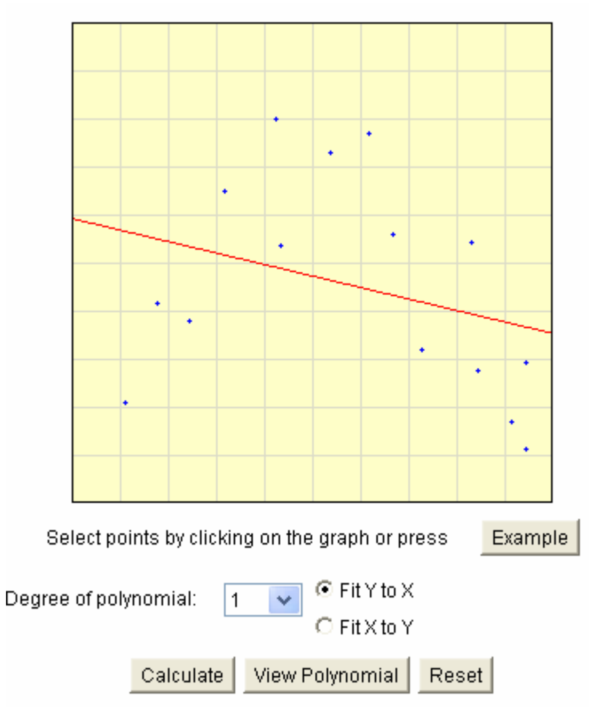
$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Mean absolute error (MAE)

MAE is the average of the absolute difference between the predicted values and observed value. The MAE is a linear score which means that all the individual differences are weighted equally in the average.

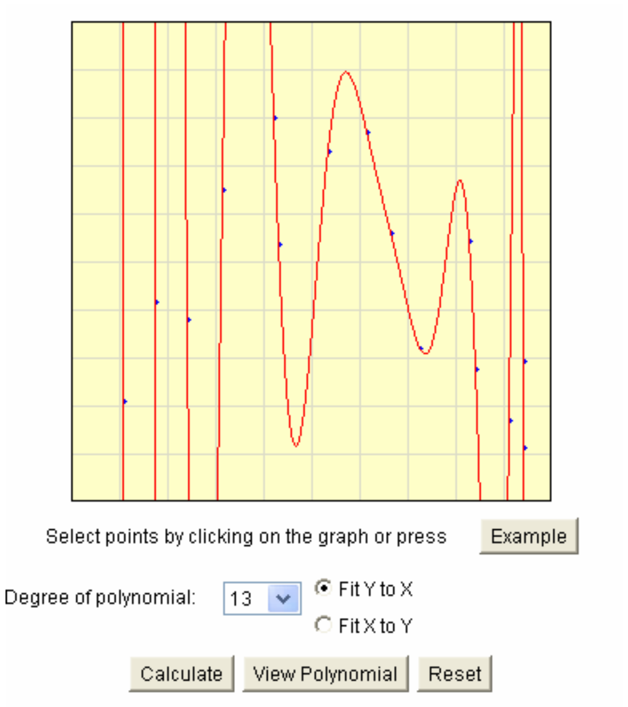
$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n} |y_j - \hat{y}_j|$$

Training and test set error as a function of model complexity



Select points by clicking on the graph or press Example

Degree of polynomial: 1 [✓ Fit Y to X] Fit X to Y



Select points by clicking on the graph or press Example

Degree of polynomial: 13 [✓ Fit Y to X] Fit X to Y

Simple greedy model selection algorithm

- Pick a dictionary of features
  - e.g., polynomials for linear regression
- Greedy heuristic:
  - Start from empty (or simple) set of features $F_0 = \emptyset$
  - Run learning algorithm for current set of features F_t
    - Obtain h_t
  - Select **next feature $\mathbf{X_i}^*$**
    - e.g., X_j is some polynomial transformation of X
  - $F_{t+1} \leftarrow F_t \cup \{X_i^*\}$
  - Recurse

Greedy model selection

- Applicable in many settings:
  - Linear regression: Selecting basis functions
  - Naïve Bayes: Selecting (independent) features $P(X_i|Y)$
  - Logistic regression: Selecting features (basis functions)
  - Decision trees: Selecting leaves to expand
- Only a heuristic!
  - But, sometimes you can prove something cool about it

Simple greedy model selection algorithm

- Greedy heuristic:
  - …
  - Select **next best feature $\mathbf{X_i}^*$**
    - e.g., X_j that results in lowest training error learner when learning with $F_t \cup \{X_j\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_i^*\}$
  - Recurse

When do you stop???

- When training error is low enough?
- When test set error is low enough?

Validation set

- Thus far: Given a dataset, **randomly** split it into two parts:
  - Training data – $\{\mathbf{x}_1,\ldots,\mathbf{x}_{\mathrm{Ntrain}}\}$
  - Test data – $\{\mathbf{x}_1,\ldots,\mathbf{x}_{\mathrm{Ntest}}\}$
- But **Test data must always remain independent**!
  - Never ever ever ever learn on test data, including for model selection
- Given a dataset, **randomly** split it into three parts:
  - Training data – $\{\mathbf{x}_1,\ldots,\mathbf{x}_{\mathrm{Ntrain}}\}$
  - Validation data – $\{\mathbf{x}_1,\ldots,\mathbf{x}_{\mathrm{Nvalid}}\}$
  - Test data – $\{\mathbf{x}_1,\ldots,\mathbf{x}_{\mathrm{Ntest}}\}$
- Use validation data for tuning learning algorithm, e.g., model selection
  - Save test data for very final evaluation

Simple greedy model selection algorithm

- Greedy heuristic:
  - …
  - Select **next best feature $\mathbf{X_i^*}$**
    - e.g., X_j that results in lowest training error learner when learning with $F_t \cup \{X_j\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_i^*\}$
  - Recurse

When do you stop???

- When training error is low enough?
- When test set error is low enough?
- When validation set error is low enough?

Sometimes, but there is an even better option …

Validating a learner, not a hypothesis (intuition only, not proof)

- With a validation set, get to estimate error of 1 hypothesis on 1 dataset
  - - e.g. Should I use a polynomial of degree 3 or 4

- Need to estimate error of learner over multiple datasets to

select parameters $E_{\{x, y\}}[h_t]$

Expected error over all datasets

(LOO) Leave-one-out cross validation

- Consider a **validation set with 1 example**:
  - *D* – training data
  - *D*\i – training data with *i* th data point moved to validation set
- **Learn classifier $h_{D\backslash i}$ with the *D\i* dataset**
- **Estimate true error** as:
  - 0 if $h_{D\backslash i}$ classifies *i* th data point correctly
  - 1 if $h_{D\backslash i}$ is wrong about *i* th data point
  - Seems really bad estimator, but wait!
- **LOO cross validation**: Average over all data points *i*:
  - **For each data point you leave out, learn a new classifier $h_{D\backslash i}$**
  - **Estimate error** as:

$$error_{LOO} = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}\left(h_{\mathcal{D}\backslash i}(\mathbf{x}^i) \neq y^i\right)$$

LOO cross validation is (almost) unbiased estimate of true error!

- When computing **LOOCV error, we only use *m-1* data points**
  - So it's not estimate of true error of learning with *m* data points!
  - Usually pessimistic, though – learning with less data typically gives worse answer

- **LOO is almost unbiased**!
  - Let $error_{true,m-1}$ be true error of learner when you only get *m-1* data points
  - LOO is unbiased estimate of $error_{true,m-1}$:

$$E_{\mathcal{D}}[error_{LOO}] = error_{true,m-1}$$

- **Great news!**
  - **Use LOO error for model selection!!!**

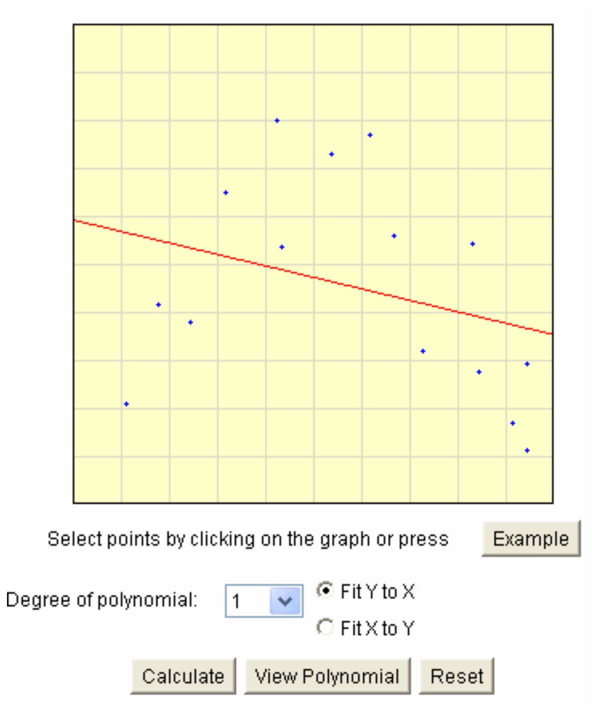
Simple greedy model selection algorithm

- Greedy heuristic:
  - …
  - Select **next best feature X_i^***
    - e.g., X_j that results in lowest training error learner when learning with $F_t \cup \{X_j\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_i^*\}$
  - Recurse

When do you stop???

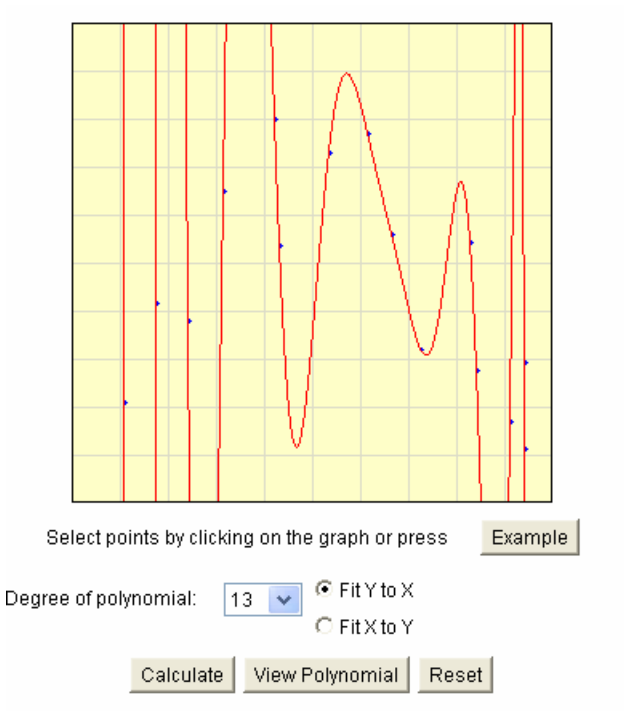
- When training error is low enough?
- When test set error is low enough?
- When validation set error is low enough?
- **STOP WHEN error$_{LOO}$ IS LOW!!!**

LOO cross validation error


Select points by clicking on the graph or press Example

Degree of polynomial: 1 [✓ Fit Y to X] Fit X to Y

Calculate View Polynomial Reset


Select points by clicking on the graph or press Example

Degree of polynomial: 13 [✓ Fit Y to X] Fit X to Y

Calculate View Polynomial Reset

Computational cost of LOO

- Suppose you have 100,000 data points
- You implemented a great version of your learning algorithm
  - Learns in only 1 second
- Computing LOO will take about 1 day!!!
  - If you have to do for each choice of basis functions, it will take forever!

Solution: Use *k*-fold cross validation

- Randomly **divide training data into *k* equal parts**
  - $D_1,\ldots,D_k$
- For each *i*
  - **Learn classifier $h_{D\backslash Di}$ using data point not in D_i**
  - **Estimate error of $h_{D\backslash Di}$ on validation set D_i**:

$$error_{\mathcal{D}_i} = \frac{k}{m} \sum_{(\mathbf{x}^j,y^j)\in\mathcal{D}_i} \mathbb{1}\left(h_{\mathcal{D}\backslash\mathcal{D}_i}(\mathbf{x}^j) \neq y^j\right)$$

- ***k*-fold cross validation error is average** over data splits:

$$error_{k-fold} = \frac{1}{k}\sum_{i=1}^{k} error_{\mathcal{D}_i}$$

- *k*-fold cross validation properties:
  - **Much faster to compute** than LOO
  - **More (pessimistically) biased** – using much less data, only *m(k-1)/k*

Summary

- Discussed the four types of features and their properties
- Discussed possible issues with datasets
- Discussed various data pre-processing steps