SDSU CS 549 Spring 2024 Supervised Machine Learning
Lecture 3: Linear Regression

IRFAN KHAN

References

SDSU CS549 Lecture Notes by Prof Yang Xu, Spring 2023. Some updated slides used here

Coursera machine learning course by Dr Andrew Ng, Oct 2023

Introduction

Introduction

Model

$X^{(i)}$ → f(model parameters) → $y^{(i)}$

$(x_1^{(i)}, x_2^{(i)}, \ldots x_n^{(i)})$

n Input Features

Label/Ground Truth

SAT GPA Dataset

Source dataset: Prof. Yang Xu - SDSU

What is m and n for the dataset?

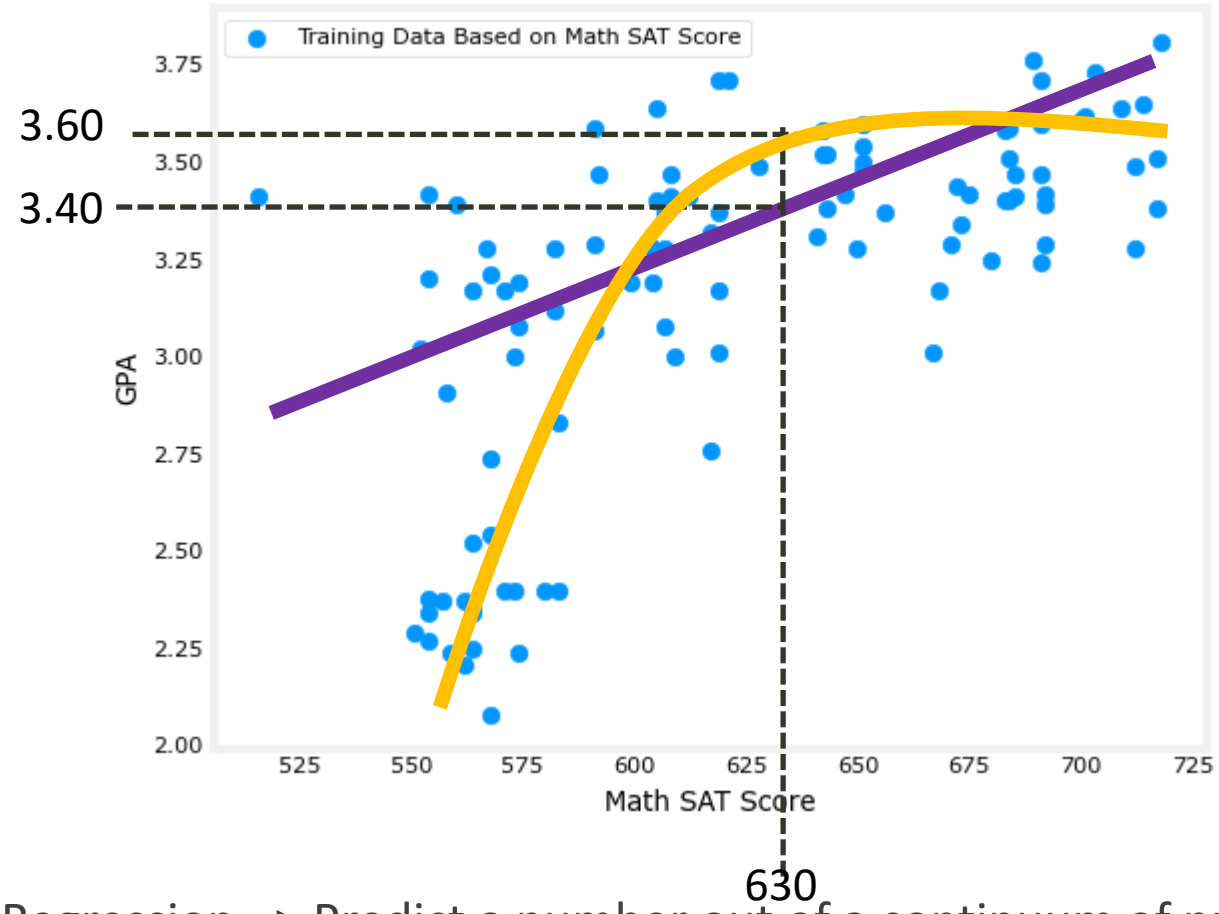
i=1,…m (m samples)

- Model learns/Trains from Features -> Label mapping to learn model parameters
- Trained Model, predicts output given a new set of input features

Examples

Input Features	Output	Application
Age, Gender, BMI, Children, Smoker?, region	Cost of Health Insurance	Health Insurance Premium Calculation
Math SAT Score, Verbal SAT Score	GPA	College Admission
Sq feet, Number of Rooms	Cost of House	Housing sales
Money spent on advertisement	Increase in revenue	Advertisement budgeting
Multiple sensor input	Position of other cars, Lane and Obstacle recognition	Self-Driving Cars

Regression: Predicting Future GPA from MATH SAT Scores



From SAT GPA dataset

The term "regression" was chosen by Galton (late 1800s) because, in his observations, extreme values (either exceptionally tall or short individuals) tended to move or "regress" toward the average or mean in subsequent generations.

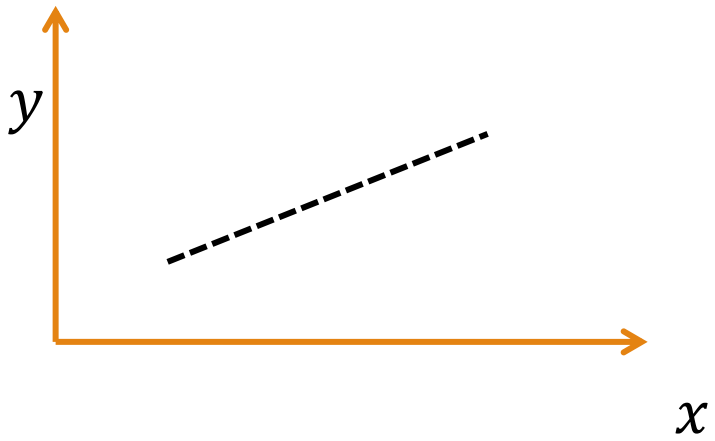
Regression => Predict a number out of a continuum of numbers

Linear Regression

Linear Regression

Goal: learn a linear model (function) that maps from x to y

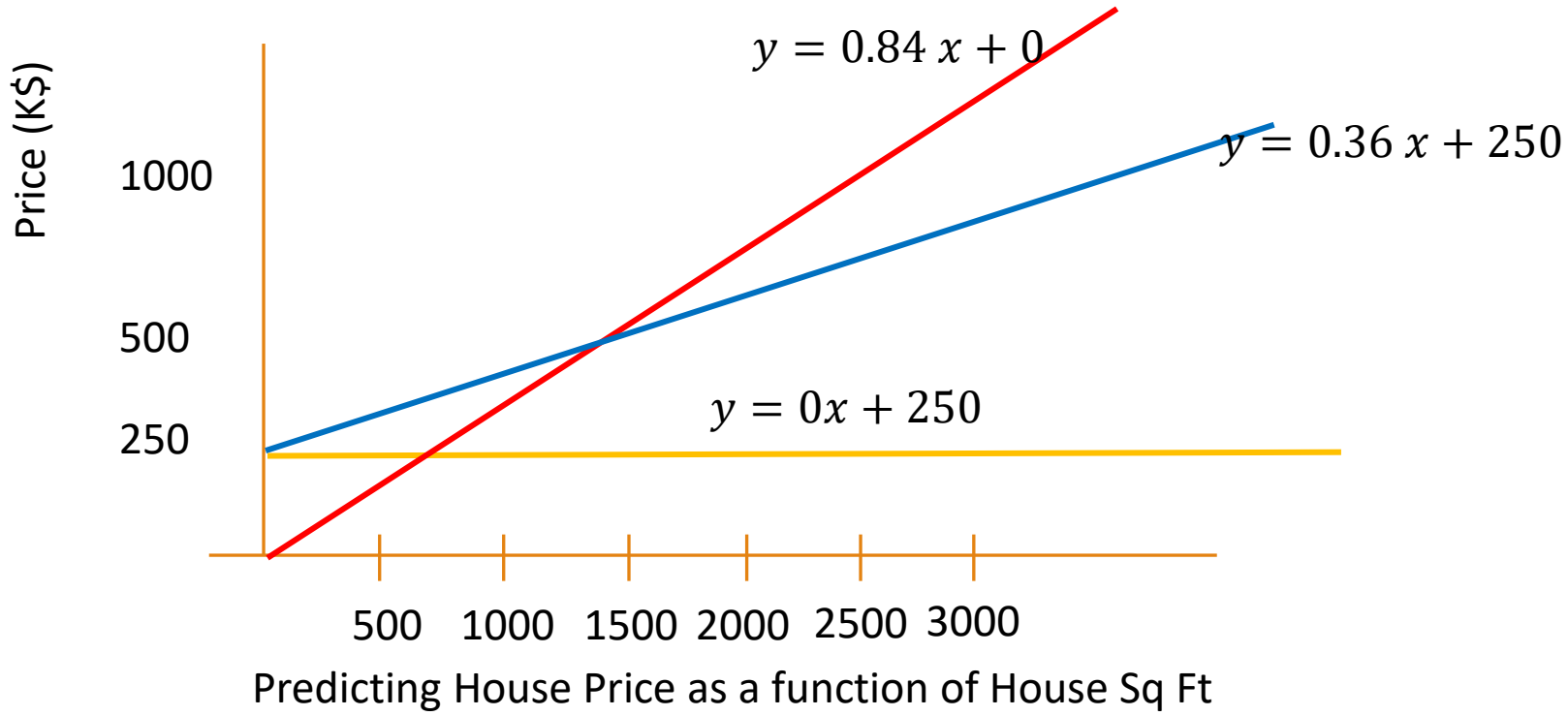
$$y = f(x) = wx + b$$



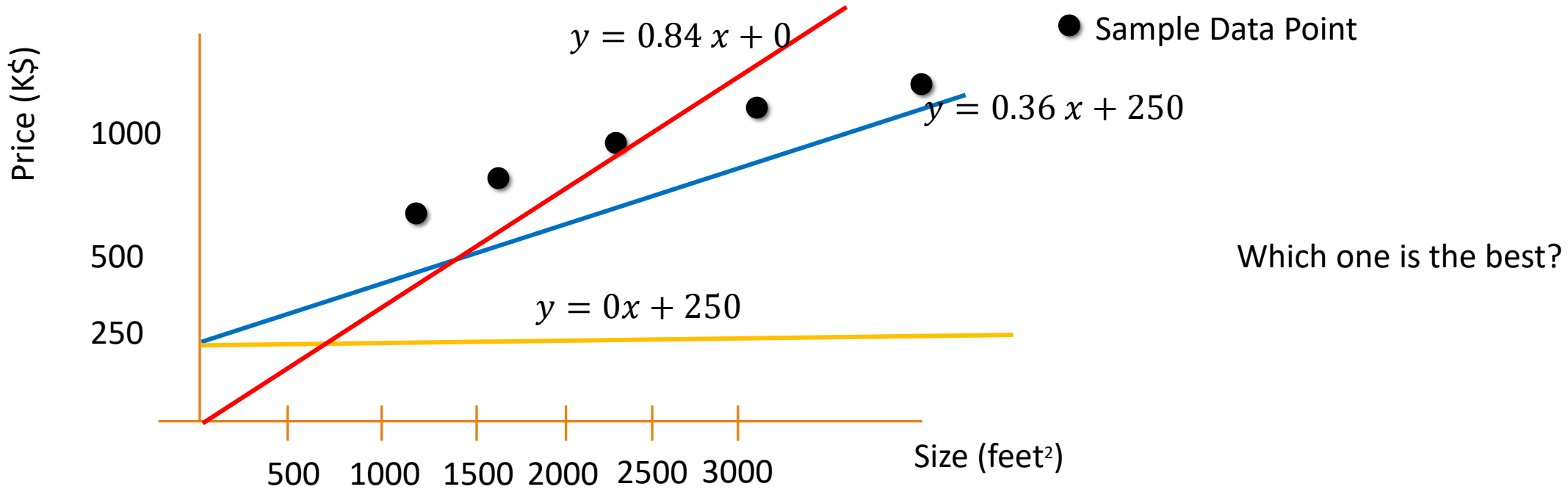
- *y=wx+b* is a linear equation.
- It represents a straight line when graphed on a coordinate system.
- In this equation:
  - *y* is the dependent variable.
  - *x* is the independent variable.
  - w is the coefficient of the *x* term, representing the slope of the line.
  - *b* is the y-intercept, representing the value of *y* when *x* is zero.

(w, b) are the model's parameters aka (weight and bias)

Meanings of parameters (w,b)

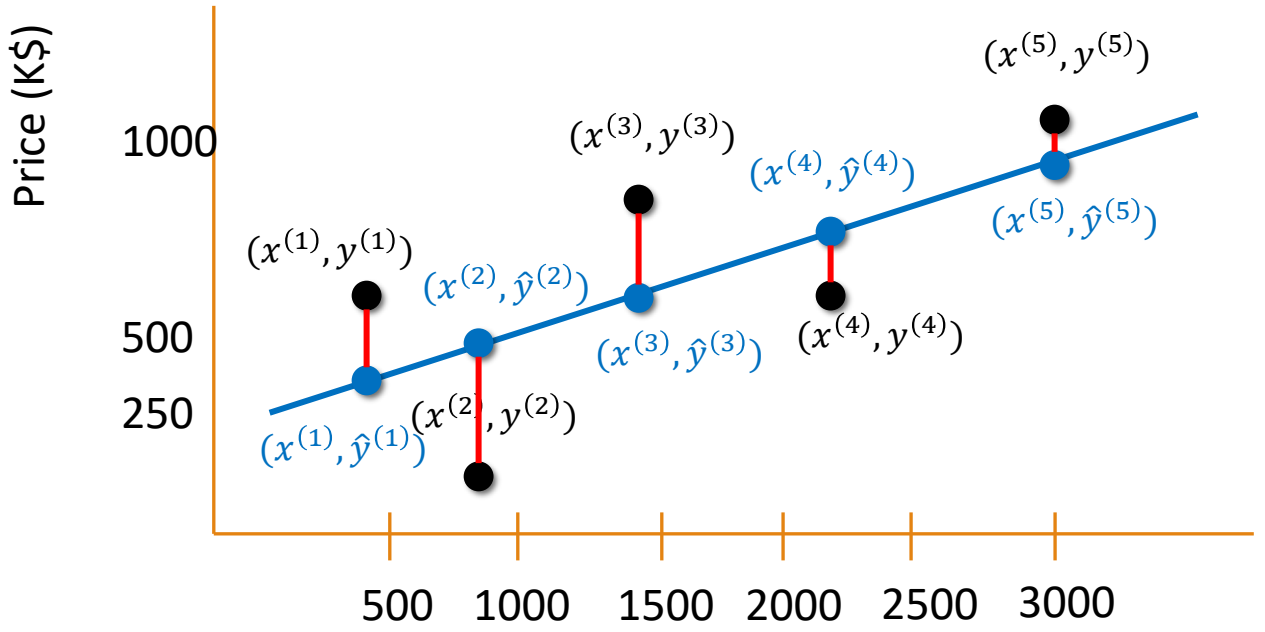


How to find the best (w, b)?



Goal: choose parameters (w, b) so that $f_{w,b}(x)$ is **close** to y for all training data points $(x^{(i)}, y^{(i)})$

How can“close” be Quantified?



Objective: minimize the *distances*

Predicted value $\hat{y}^{(i)} = f_{w,b}(x^{(i)}) = wx^{(i)} + b$

Learning objective

Minimize the difference between predicted value $\hat{y}^{(i)}$ and ground truth $y^{(i)}$, *(i)*: i^{th} sample

I.e., Minimize $(y^{(1)} - \hat{y}^{(1)})^2 + (y^{(2)} - \hat{y}^{(2)})^2 + \cdots (y^{(5)} - \hat{y}^{(5)})^2$

Cost function: measures how close the model is in predicting y with x. Smaller the better

$$J(w, b) = \frac{1}{2m}\sum_{i=1}^{m}\left(y^{(i)} - \hat{y}^{(i)}\right)^2 = \frac{1}{2m}\sum_{i}\left(y^{(i)} - f_{w,b}(x^{(i)})\right)^2$$

Mean Square Error (MSE)

J is a function of variables (w, b)

$\{x^{(i)}, y^{(i)}\}$ $(i = 1, \cdots, m)$ are known, *m*=no of samples

The best w, b need to be found

How to Minimize the Cost Function *J*?

$$J(w,b) = \frac{1}{2m}\sum_i \left(y^{(i)} - f_{w,b}(x^{(i)})\right)^2$$

$$= \frac{1}{2m}\sum_i \left(y^{(i)} - wx^{(i)} - b\right)^2$$

$$= \frac{1}{2m}\sum_i \left(y^{(i)2} + w^2x^{(i)2} + b^2 - 2x^{(i)}y^{(i)}w - 2y^{(i)}b + 2x^{(i)}wb\right)$$

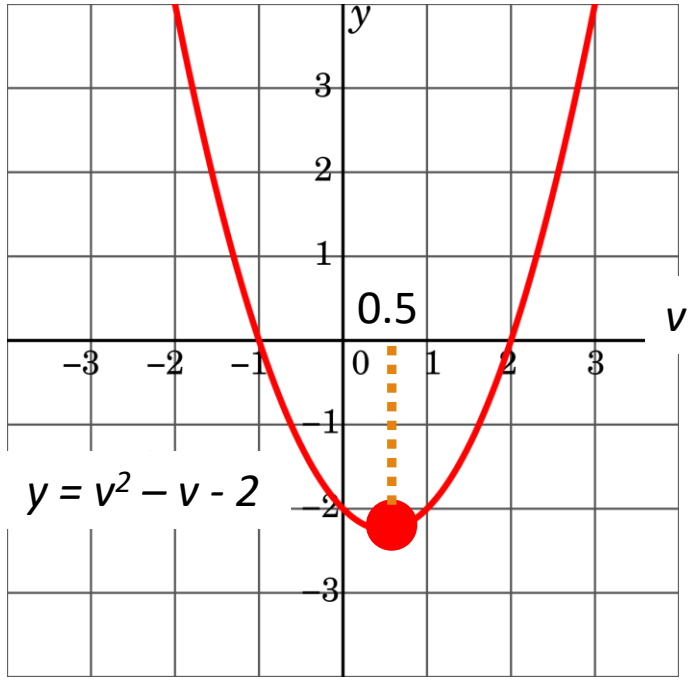
$$J(w,b) = c_1b^2 + c_2b + c_3w^2 + c_4w + c_4\,wb + \mathrm{c}_5$$

C_i: coefficients

Cost function is a quadratic function in *w,b*

Building Intuition – Shape of a Quadratic Function (variable *v*, coefficients d_i)

$$y = f(v) = d_1 v^2 + d_2 v + d_3$$



Min *y* of -2.25 at *v* = 0.5

$f(v)$ has minimal value when $\frac{df}{dv} = 0$

$$\frac{df}{dv} = 2d_1 v + d_2 = 0 \Rightarrow \mathrm{v} = -\frac{d_2}{2d_1}$$

So, a closed-form solution exists for this case!

Derivatives of Cost Function *J*

$J(w, b\ )$ is a function of *w,b*

It reaches minimum when $\frac{\partial J}{\partial w} = \frac{\partial J}{\partial b} = 0$

b can be viewed as constant when computing $\frac{\partial J}{\partial w}$ and vice versa.

Derivatives of Cost Function *J* (cont.)

- $J(w,b) = \frac{1}{2m}\sum_i \left(y^{(i)} - b - wx^{(i)}\right)^2$

- $\frac{\partial J}{\partial b} = -\frac{1}{m}\sum_i (y^{(i)} - b - wx^{(i)}) = 0$
- $\frac{\partial J}{\partial w} = -\frac{1}{m}\sum_i (y^{(i)} - b - wx^{(i)})x^{(i)} = 0$

- $-\bar{y} + b \quad + \bar{x}w = 0$
- $-\frac{1}{m}\sum_i y^{(i)}x^{(i)} + \bar{x}b + \frac{1}{m}\sum_i x^{(i)2}w = 0$

} *n+1* linear equations in *n+1* variables.
=>Solution can be found

- where $\bar{x} = \frac{1}{m}\sum_i x^{(i)}$ and $\bar{y} = \frac{1}{m}\sum_i y^{(i)}$

Normal equation solutions

$$w = \frac{\sum_i (y^{(i)} - \bar{y})(x^{(i)} - \bar{x})}{\sum_i (x^{(i)} - \bar{x})^2}$$

$$b = \bar{y} - w\bar{x}$$

- b and w are the model parameters
  - E.g., Price= *b + w* * Size
  - Interpret w: how much Price changes when Size changes for one unit (sq. ft.)

Given a new data point x^{new} , the predicted value $y^{pred} = b + w * x^{new}$

Multiple Feature Linear Regression

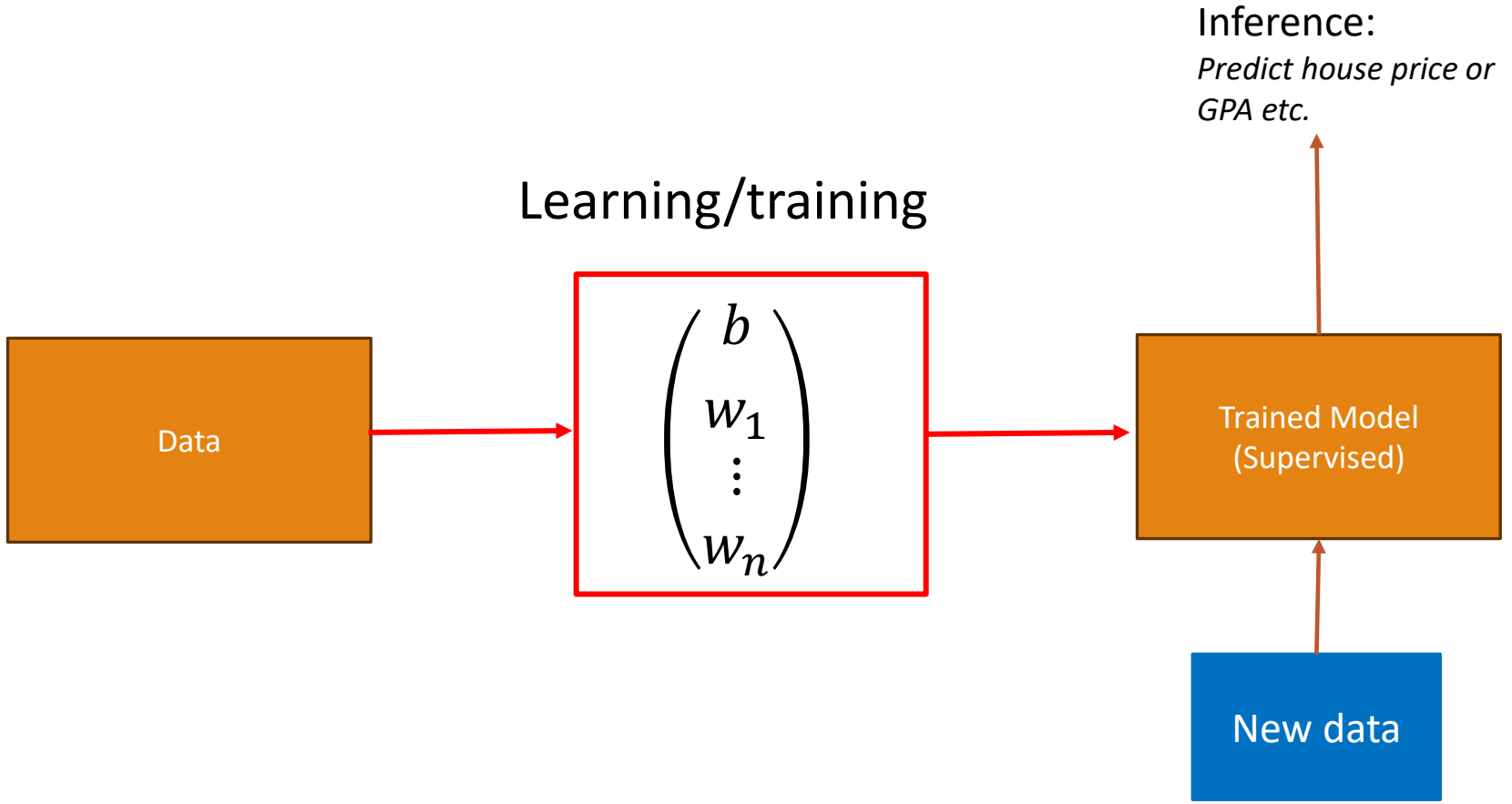
Multiple Features

- Often more than one feature is required to improve predictions
- For instance, both verbal and math SAT scores could better predict future GPA instead of just one score
- House price depends on multiple factors in addition to area. Features like year built, number of rooms, etc.

		Label (e.g. predicted GPA)
Feature 1 (e.g. Math SAT	Feature 2 (e.g. Verbal SAT)	
$x_1^{(1)}$	$x_2^{(1)}$	$y^{(1)}$
$x_1^{(2)}$	$x_2^{(2)}$	$y^{(2)}$
$x_1^{(3)}$	$x_2^{(3)}$	$y^{(3)}$
…	…	…

- In general, the LR model parameters for an *n* feature problem are: $b, w_1, w_2, \ldots w_n$
- Predicted $\hat{y} = b + w_1x_1 + w_2x_2 + \ldots\ldots w_nx_n$

LR Model Extended to multiple features



Multiple feature linear regression with $n + 1$ model parameters

Multiple feature linear regression

Goal: Learn parameters of a linear function that maps $\vec{x}$ to y.

$$y = f_{w,b}(\vec{x}) = b + w_1 x_1 + \cdots + w_n x_n$$

Predicted value $$\hat{y}^{(i)} = f_{w,b}\left(x^{(i)}\right) = b + w_1 x_1^{(i)} + \cdots + w_n x_n^{(i)}$$

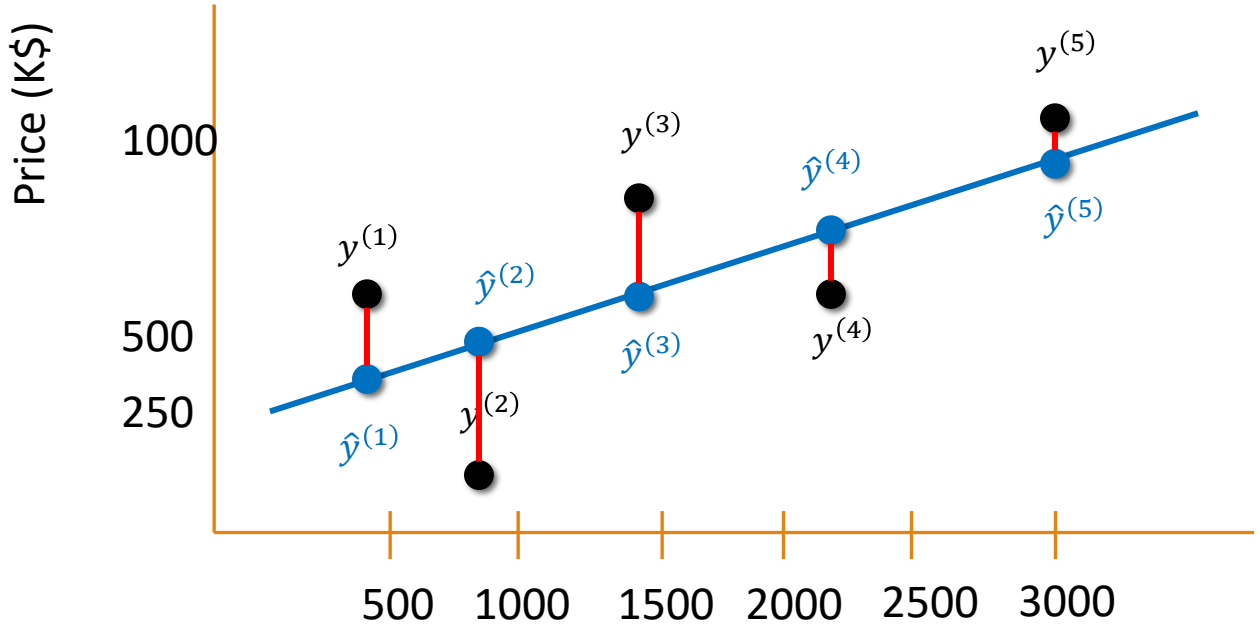
Cost function $$J(b, w_1, \cdots w_n) = \frac{1}{2m}\sum_i \left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

$$= \frac{1}{2m}\sum_i \left(y^{(i)} - b - w_1 x_1^{(i)} - \cdots - w_n x_n^{(i)}\right)^2$$

How Good is Our Model?

Residuals: The differences between $y^{(i)}$ and $\hat{y}^{(i)}$

$$e^{(i)} = y^{(i)} - \hat{y}^{(i)}$$



$\sum_i e^{(i)}$ is always 0. Why?

So, use residual sum of squares (RSS):

$$RSS = \sum_i (y^{(i)} - \hat{y}^{(i)})^2$$

Matrix representations

Sample data *X (m x n), y(m x 1)*

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \cdots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

Model parameters *b and* w_i*. Total: (n x 1)*

$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

Matrix representations (cont.)

Predicted value $\hat{\boldsymbol{y}} = b + Xw$

$$\hat{y} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{bmatrix} = b + \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots x_n^{(2)} \\ \vdots & \vdots & \cdots \vdots \\ x_1^{(m)} & x_2^{(m)} & \cdots x_n^{(m)} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \cdots x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \cdots x_n^{(2)} \\ \vdots & \vdots & \cdots \vdots & \\ 1 & x_1^{(m)} & x_2^{(m)} & \cdots x_n^{(m)} \end{bmatrix} \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

m x 1 *m x n* *n x 1* *m x (n+1)* *(n+1) x 1*

=X'θ

where $\theta = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$

(n+1) x 1

and *X'* is the augmented matrix *X*

Matrix representations (cont.)

Cost function $J(\theta) = \frac{1}{2m}(y - \hat{y})^T(y - \hat{y})$

$$\boldsymbol{J} = \frac{1}{2m}(\boldsymbol{y} - \boldsymbol{X'\theta})^T(\boldsymbol{y} - \boldsymbol{X'\theta})$$

$$= \frac{1}{2m}(\boldsymbol{y}^T - (\boldsymbol{X'\theta})^T)(\boldsymbol{y} - \boldsymbol{X'\theta})$$

$$= \frac{1}{2m}(\boldsymbol{y}^T\boldsymbol{y} - (\boldsymbol{X'\theta})^T\boldsymbol{y} - \boldsymbol{y}^T(\boldsymbol{X'\theta}) + (\boldsymbol{X'\theta})^T\boldsymbol{X'\theta})$$

$$= \frac{1}{2m}(\boldsymbol{y}^T\boldsymbol{y} - 2(\boldsymbol{X'\theta})^T\boldsymbol{y} + (\boldsymbol{X'\theta})^T\boldsymbol{X'\theta})$$

$$\boldsymbol{J(\theta)} = \frac{1}{2m}((\boldsymbol{X'\theta})^T\boldsymbol{X'\theta} - 2(\boldsymbol{X'\theta})^T\boldsymbol{y} + \boldsymbol{y}^T\boldsymbol{y})$$

Analogous to the quadratic cost function for one-feature linear regression cost function

$$J(w, b) = c_1b^2 + c_2b + c_3w^2 + c_4w + c_4\,wb + \mathrm{c}_5$$

Derivative of $J(\Theta)$

Like in the scalar case, compute the derivative of $J(\theta)$ w.r.t. θ and set it to 0

$$J(\theta) = \frac{1}{2m}((X'\theta)^T X'\theta - 2(X'\theta)^T y + y^T y)$$

$$\frac{\partial J(\theta)}{\partial \theta} = 0$$

$$\theta = (X'^T X')^{-1} X'^T y$$

Closed Form Solution
(Normal Equation)

Why not Use Normal Equation All the Time?

Very slow to compute the inverse of a matrix when n is large

$$(X'^T X')^{-1}$$

What are the dimensions?

Alternative: Use Gradient Descent Method

Recap Linear Regression and Normal Equation Method

- Given n features and m sample data points
- Prediction $\hat{y} = X'\theta, where\ X' is\ the\ augmented\ X\ matrix$ and θ is the vector of model parameters, *b* and w_i, *i=1,2,…n*
- Cost function is given by the quadratic function:

$$J(\theta) = \frac{1}{2m}((X'\theta)^T X'\theta - 2(X'\theta)^T y + y^T y)$$

- Which can be minimized by setting its gradient to zero

$$\frac{\partial J(\theta)}{\partial \theta} = 0$$

- Which can be solved for *Θ* in a closed form Normal Equation

$$\theta = (X'^T X')^{-1} X'^T y$$

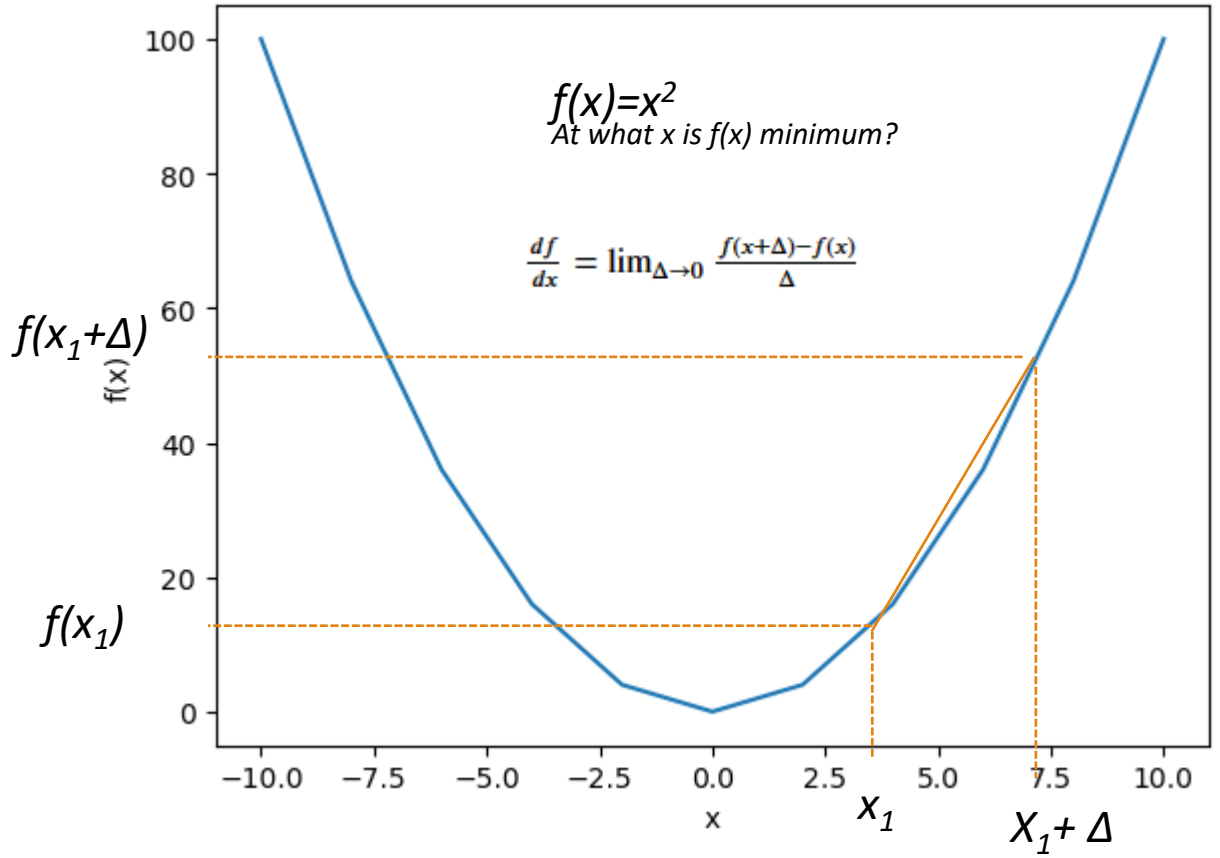
- Using normal equation for large *n* not practical for the matrix inversion above

				Label (e.g. predicted GPA)
Feature 1 (e.g. Math SAT	Feature 2 (e.g. Verbal SAT)	….	Feature *n*	
$x_1^{(1)}$	$x_2^{(1)}$	….	$x_n^{(1)}$	$y^{(1)}$
$x_1^{(2)}$	$x_2^{(2)}$	….	$x_n^{(2)}$	$y^{(2)}$
$x_1^{(3)}$	$x_2^{(3)}$	…	$x_n^{(3)}$	$y^{(3)}$
…	…	….	….	…
$x_1^{(m)}$	$x_2^{(m)}$	….	$x_n^{(m)}$	$y^{(m)}$

Gradient Descent Method

Scalar function, Derivative

- A scalar function, *f*, is a mathematical function that takes one or more input variables and returns a single scalar value.
  - *f*: $\mathbf{R}^n$->**R**, $\mathbf{R}^n$ is n-dimensional real vector and **R** represents a scalar
  - Linear regression model is a scalar function, *n* represents the number of features in the sample dataset
- The derivative of a scalar function in one variable, denoted as $\frac{df}{dx}$ represents the rate of change of *y* with respect to *x*.
- For a convex function, derivative points in the direction of fastest change of y
- If we are looking for a minimum, *we will traverse the curve (y)* in a direction opposite to the direction with fastest increase of y



x used here is not to be mixed with our usual representation of a sample data point. This slide is to build intuition about a derivative

Partial Derivative and Gradient

- If $f(x_1,x_2)$ is a scalar function of two variables, x_1, and x_2, then the partial derivative of f with respect to x_1, defined as:

$$\frac{\partial f}{\partial x_1} = \lim_{\Delta \to 0} \frac{f(x_1+\Delta,x_2)-f(x_1,x_2)}{\Delta}$$

- The above notation indicates that we are finding the derivative of f with respect to x_1 while holding x_2 as a constant

- Similarly, the partial derivative of f with respect to x_1, defined as:

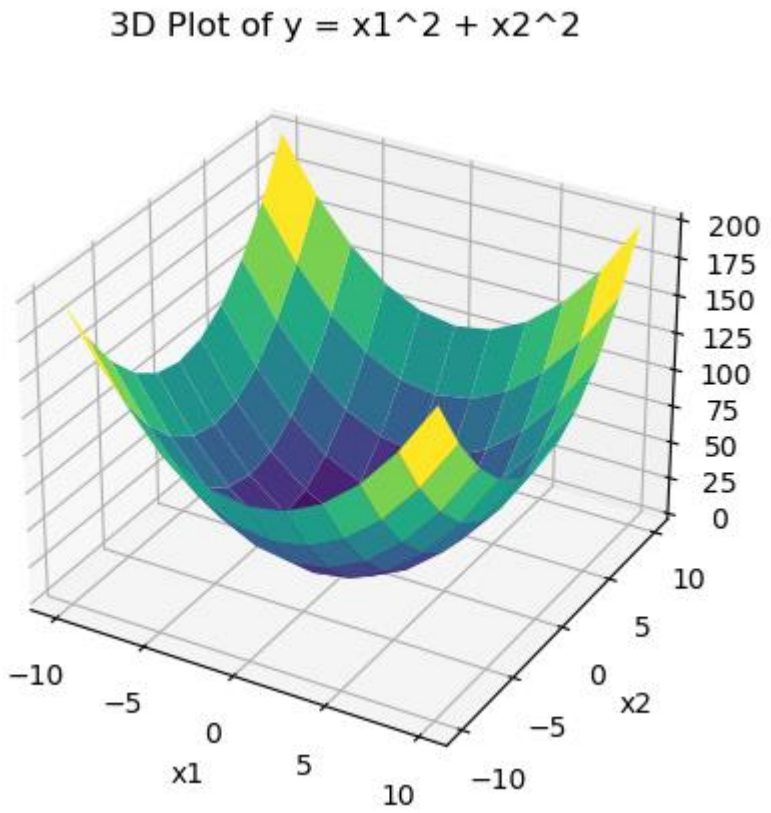
$$\frac{\partial f}{\partial x_2} = \lim_{\Delta \to 0} \frac{f(x_1,x_2+\Delta)-f(x_1,x_2)}{\Delta}$$

- The above notation indicates that we are finding the derivative of f with respect to x_2 while holding x_1 as a constant

- For a scalar function f of n variables $x_1,x_2,\ldots x_n$, the gradient, ∇f is a vector given by: $\nabla f = (\frac{d}{b},)$

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$$

  - Analogous to the derivative, Gradient points in the direction of fastest increase of y
  - If we are looking for a minimum, we will traverse y in a direction opposite to the gradient

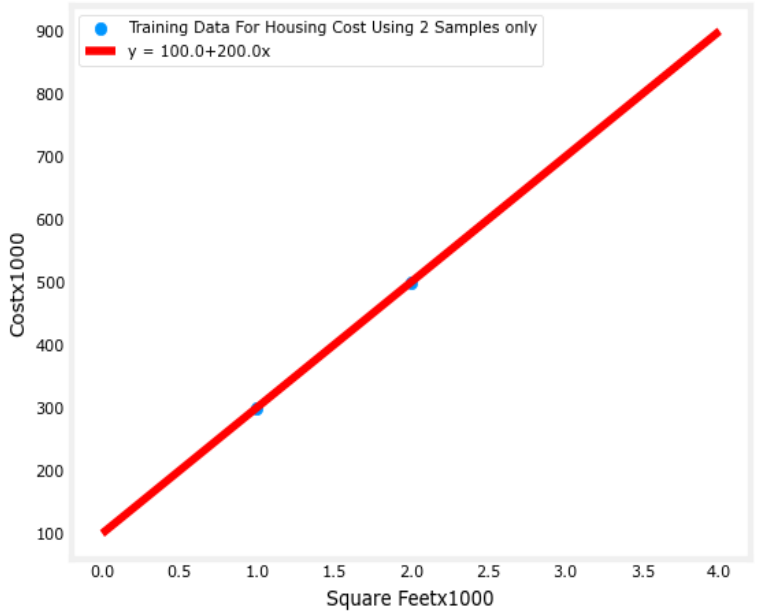


- At a fixed x2, f(x1,x2) will look like the parabola in the previous slide
- Similarly at a fixed x1, f(x1,x2) will look like the parabola in the previous slide

$x_1,x_2,\ldots x_n$ used here are not to be mixed with our usual representation of sample data points. This slide is to build intuition about derivatives and gradient

Building Intuition – Recap Normal Equation

Sq Ft	House Prices ($)
1000	300000
2000	500000



$$y = f(x) = b + wx$$

$$w = \frac{\sum_i (y^{(i)} - \bar{y})(x^{(i)} - \bar{x})}{\sum_i (x^{(i)} - \bar{x})^2}$$

$$b = \bar{y} - w\bar{x}$$

$$\bar{x} = 1500$$

$$\bar{y} = 400000$$

$$w = \frac{(300000 - 400000)(1000 - 1500) + (500000 - 400000)(2000 - 1500)}{(1000 - 1500)^2 + (2000 - 1500)^2}$$

= 200

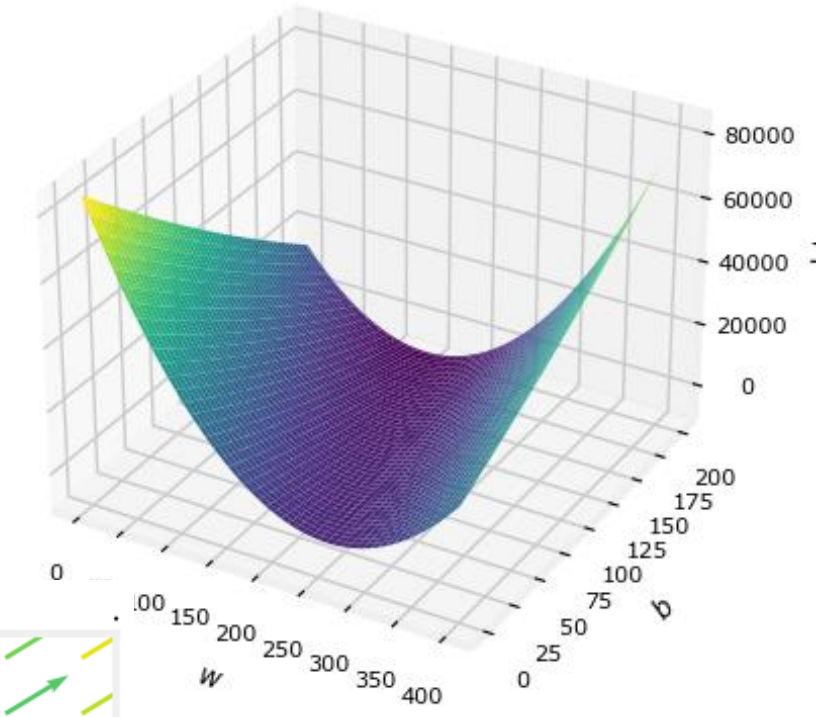
*b = 400000 – 200 * 1500 = 100000*

Building Intuition – Gradient Descent

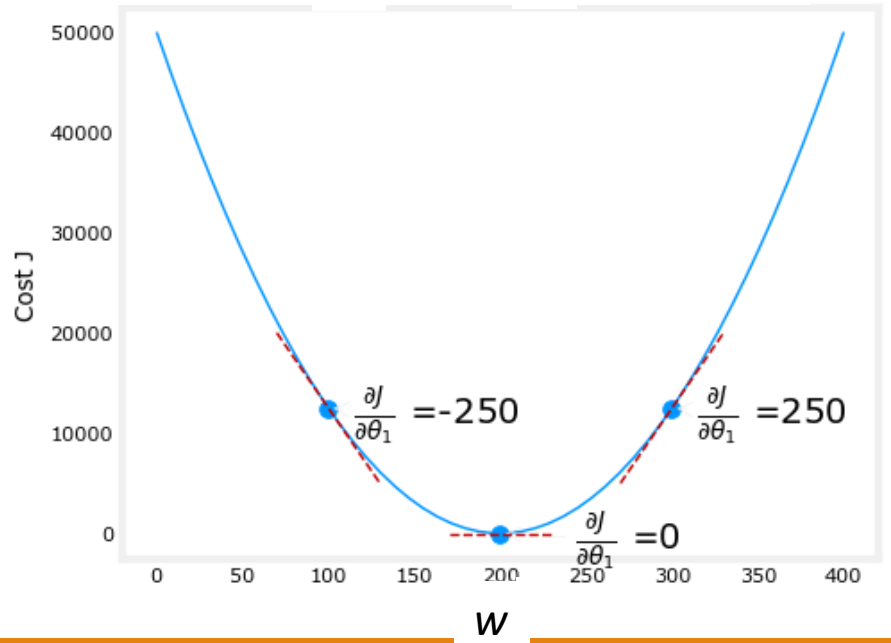
$$J(w,b) = c_1 b^2 + c_2 b + c_3 w^2 + c_4 w + c_4\, wb + c_5$$

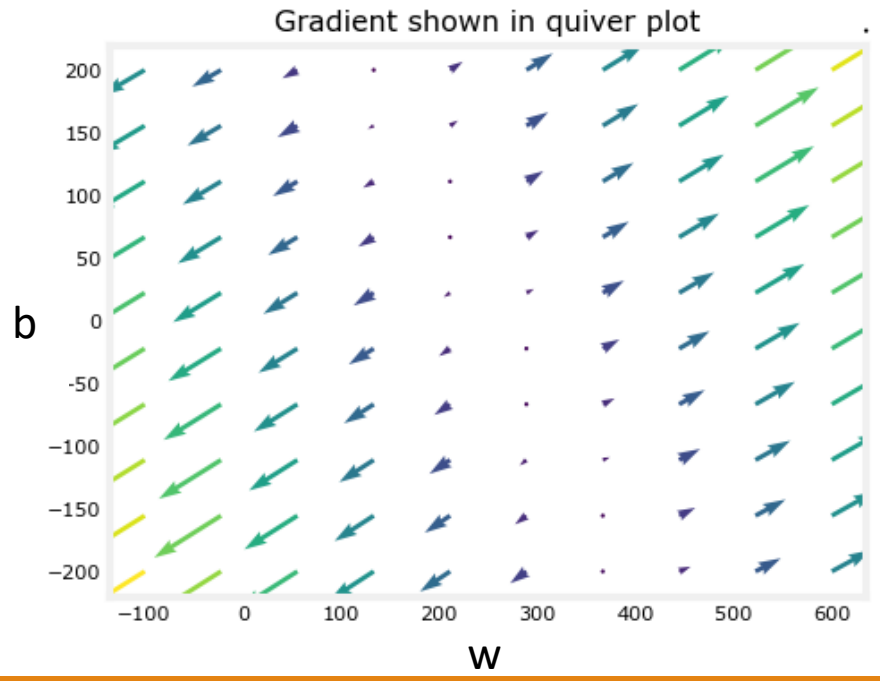
$$c_i : coefficients$$

Cost function is a quadratic function in *w,b*



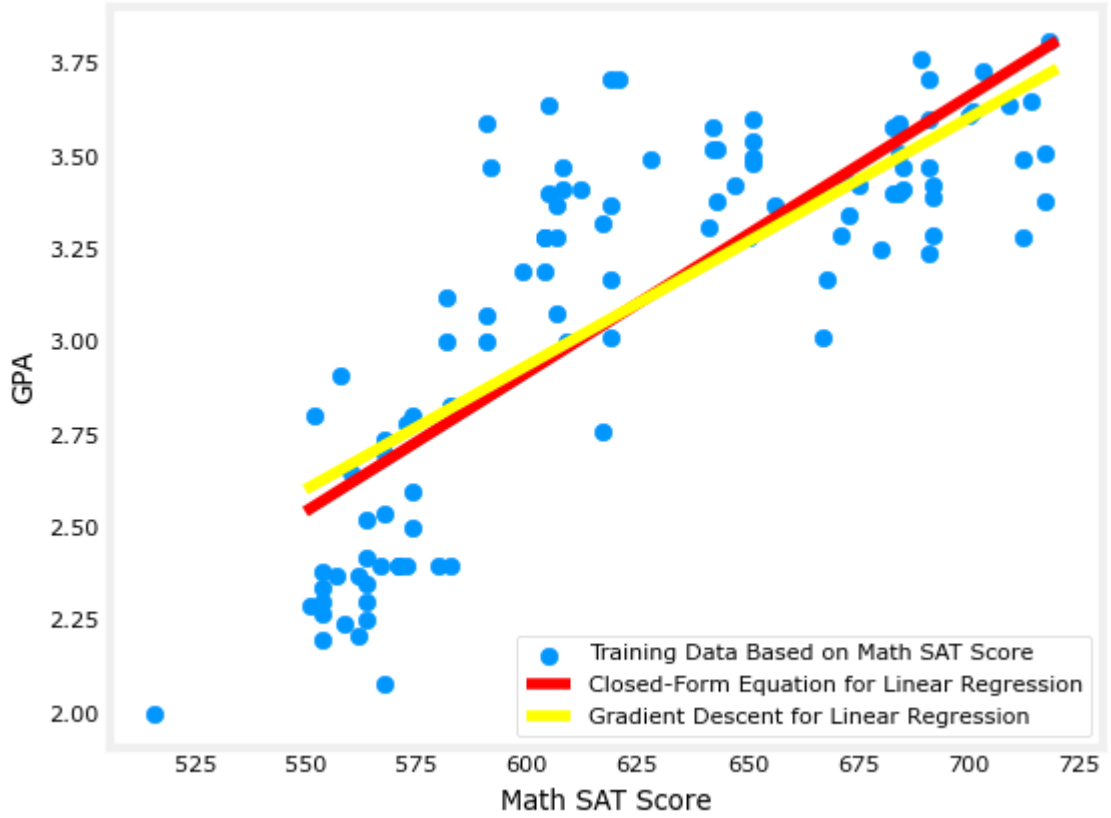
Cost *J* vs *w* with *b* set to 100





The cost function "surface" can be visualized in a 3D plot like above only for one feature case to develop intuition which will help us generalize the concepts for multiple features

If designed “properly”, gradient descent solution will converge with the solution obtained using normal method



Gradient Descent Algorithm

- Initialize the starting point on the cost function surface, w_{int} and b_{int}
- Compute the gradient of the cost function J
- Move along cost function "surface" in a direction opposite to the gradient by a small amount ($\propto$)
  - $\propto$ is called the learning rate
- Repeat the process until convergence
- The w_{opt} and b_{opt} at the point of convergence are the optimal values at which J is minimum
- Given w_{opt} and b_{opt}, we can calculate the predicted value $\hat{y}$ given a new x value, x_{new} as:
- $\hat{y} = b_{opt} + w_{opt} x_{new}$

Gradient Descent Algorithm - Equations

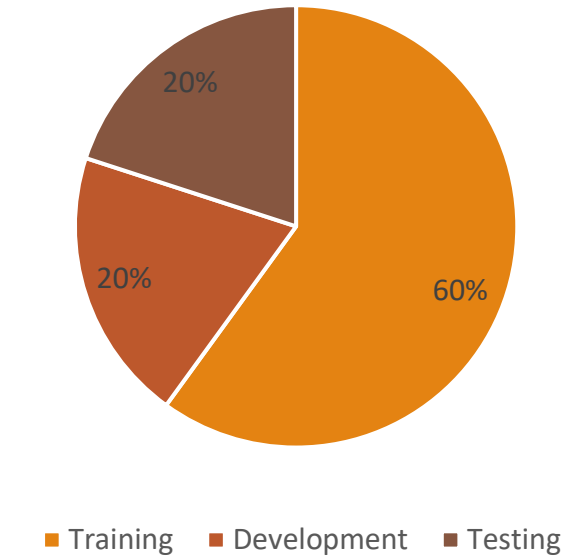
- Initialize *w,b*
- Compute gradient of the cost function *J*:
- Earlier we derived the derivative of *J* for the one feature case
  - $\frac{\partial J}{\partial b} = -\frac{1}{m}\sum_i \left(y^{(i)} - b - wx^{(i)}\right) = \frac{1}{m}\sum_i \left(b + wx^{(i)} - y^{(i)}\right)$
  - $\frac{\partial J}{\partial w} = -\frac{1}{m}\sum_i \left(y^{(i)} - b - wx^{(i)}\right)x^{(i)} = \frac{1}{m}\sum_i (b + wx^{(i)} - y^{(i)})x^{(i)}$
- These can be generalized for multiple features (*n* features)
  - $\frac{\partial J}{\partial b} = \frac{1}{m}\sum_i \left(b + \sum_j w_j {x_j}^{(i)} - y^{(i)}\right)$
  - $\frac{\partial J}{\partial w_k} = \frac{1}{m}\sum_i \left(b + \sum_j w_j {x_j}^{(i)} - y^{(i)}\right){x_k}^{(i)}, k = 1,2,..n$
- Move along cost function "surface" in a direction opposite to the gradient by a small amount ($\propto$)
  - $b = b - \alpha(\frac{\partial J}{\partial b})$
  - $w_k = wk - \alpha(\frac{\partial J}{\partial wk})$
- Keep repeating until "convergence"

Practical Considerations and Performance Evaluation with Linear Regression Model

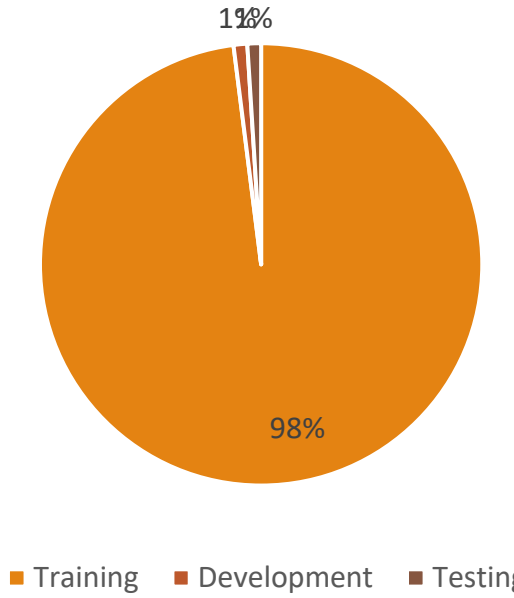
Training/Development(Validation)/test sets (Applies to all machine learning models)

- Don’t use all the sample data for training, use a large %age for it
- Use some sample data for development/validation to evaluate different hyper-parameter values, no of iterations etc.
- Use remaining sample data to test the model to make sure it is a good and a stable model

Small Data Set (say, m<10K)



Large Data Set (say, m>10K)



Performance metrics

Sample Data Set $\{({x_1}^{(i)}, {x_2}^{(i)}, \ldots\ldots {x_n}^{(i)})\}, y^{(i)}$

Test set $\{({x_1}^{(j)}, {x_2}^{(j)}, \ldots\ldots {x_n}^{(j)})\}, y^{(j)} (j = 1, \ldots, m_{test})$

Performance on test set

- $RSS = \sum_j (y^{(j)} - \hat{y}^{(j)})^2$ aka L2 error
- L1 error: $\sum_j |y^{(j)} - \hat{y}^{(j)}|$ could also be used
- Standard deviation of L1,L2 error could also be used

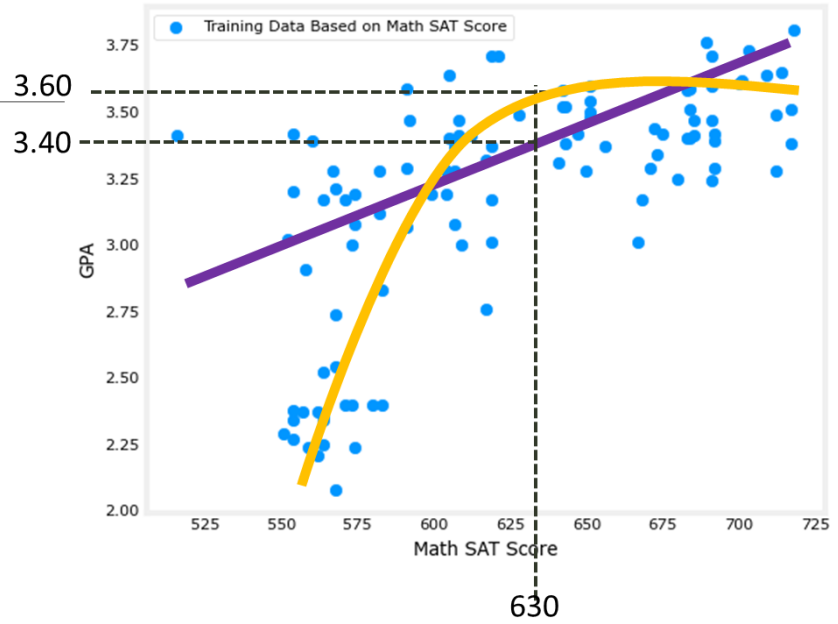
How many samples (*m*) and how many features (*n*) to use?

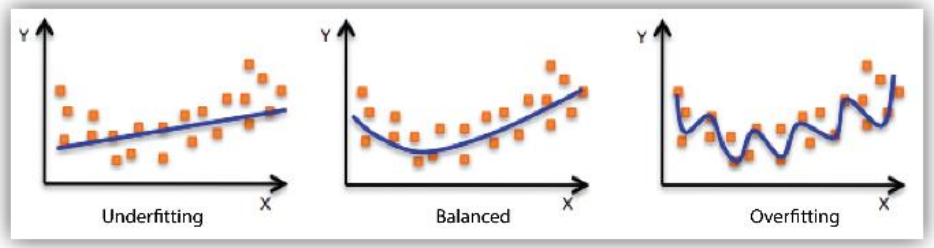
- **Number of Samples (Data Points):**
  - In general, having more samples is beneficial for training a more robust and accurate linear regression model.
  - The "rule of thumb" is that the number of samples should be significantly larger than the number of features to avoid overfitting.
  - The exact number considered "good" can vary depending on the complexity of the problem. However, having at least several hundred to a few thousand samples is often recommended for meaningful results.

1. **Number of Features**
   - The number of features should be carefully chosen based on the relevance and importance of each feature to the target variable.
   - If the number of features is too high relative to the number of samples, it may lead to under-fitting.
   - Feature selection or dimensionality reduction techniques (like PCA) can be considered if there are a large number of features.

Under-Fitting and Over-Fitting the Data

- Under-fitting:
  - Simpler model than data: The model lacks complexity to capture the underlying patterns in the data, resulting in high bias.
  - Poor performance on both training and testing data: The model fails to learn generalizable patterns, leading to inaccurate predictions across all data points.
  - Examples: Simple linear regression might underfit complex data with non-linear relationships.
- Over-fitting:
  - Complex model captures too much detail: The model memorizes noise and specific training data points, resulting in high variance.
  - Good performance on training data but poor on testing data: The model doesn't generalize well to unseen data, leading to unreliable predictions.
  - Examples: A high-degree polynomial model might overfit training data with random noise, failing to predict accurately on new data points.
- In essence:
  - Under-fitting: Underestimates the data's complexity, leading to inaccurate predictions overall.
  - Over-fitting: Overestimates the data's complexity, leading to inaccurate predictions for unseen data.

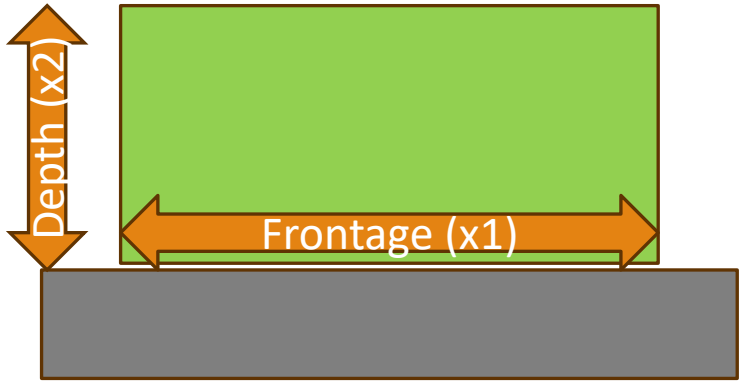




From: https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

Avoiding Under-Fitting

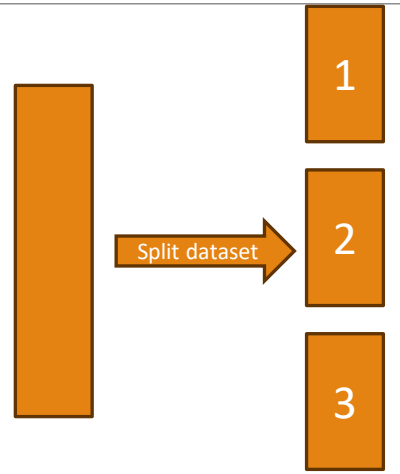
- Underfitting occurs when a machine learning model is too simple to capture the underlying patterns in the training data, resulting in poor performance on both the training and unseen data. To avoid underfitting, consider the following strategies:
  - **Use a More Complex Model:** If your current model is too simple, try using a more complex model with more parameters or a higher degree of flexibility. For example, if using a linear model, consider using a polynomial model or a more complex algorithm.
  - **Add More Features:** Ensure that your dataset has sufficient features to represent the underlying complexity of the problem. If the dataset is too simple, the model may struggle to generalize.
  - **Feature Engineering:** Create new features that might help the model understand the underlying patterns better. Feature engineering involves transforming or combining existing features to provide more relevant information to the model.
  - **Reduce Regularization:** If your model uses regularization (e.g., L1 or L2 regularization), consider reducing the regularization strength. Regularization can prevent overfitting, but too much regularization may lead to underfitting.
  - **Increase Training Time:** In some cases, underfitting might be due to insufficient training time. Training the model for more epochs or longer periods may allow it to learn more complex patterns.
  - **Adjust Model Hyperparameters:** Experiment with different hyperparameter settings. For example, adjusting the learning rate, batch size can impact performance.
  - **Cross-Validation:** Use cross-validation to assess your model's performance on multiple subsets of the data. This can help you identify if the model is consistently underfitting or if the issue is specific to a particular subset.
  - **Evaluate Training Loss and Validation Loss:** Monitor both the training loss and the validation loss during training. If the training loss is not decreasing, and the validation loss is not improving, it indicates underfitting.



- Say we start with land-cost predictor $\hat{y} = b + w_1x_1 + w_2x_2$ and we are under-fitting sample data
- Consider creating a third variable $x_3 = x_1 * x_2$ and use the following predictor: $\hat{y} = b + w_1x_1 + w_2x_2 + w_3x_3$

Avoiding Over-Fitting

- Overfitting occurs when a machine learning model learns the training data too well, capturing noise and patterns that do not generalize to new, unseen data. To avoid overfitting, consider the following strategies:
- Use cross-validation techniques (e.g., k-fold cross-validation) to assess how well your model generalizes to different subsets of the data. This helps detect overfitting by evaluating performance on multiple validation sets.
- Split your dataset into training and validation sets. Train the model on the training set and evaluate its performance on the validation set. This allows you to check for overfitting by comparing training and validation performance.
- Simplify the model architecture by reducing the number of features.
- Apply regularization techniques, such as L2 regularization, to penalize overly complex models. Regularization helps prevent the model from fitting noise in the training data.
- Increase the size of your training dataset by using data augmentation techniques. This involves applying random transformations to the existing data, such as rotation, scaling, or flipping, to provide the model with more diverse examples.
- Monitor the model's performance on a validation set during training. If the validation performance stops improving or starts to degrade, halt the training process early to prevent overfitting.
- Experiment with different hyperparameter settings, such as learning rates, batch sizes, or regularization strengths. Fine-tune these hyperparameters to find the right balance between underfitting and overfitting.
- Increasing the size of your training dataset can help the model generalize better. More data allows the model to learn a broader range of patterns and reduces the risk of overfitting.



- Round 1: Train on 2+3, Validate on 1
- Round 2: Train on 1+3, Validate on 2
- Round 3: Train on 1+2, Validate on 3
- In general training on k-1 subsets, validating on the remaining subset
- K-fold approach also useful if sample data size is small by averaging over k rounds

Regularization

- Instead of minimize just the cost, we also minimize over-emphasis on the training dataset

- Over-emphasis on the training dataset can be quantified by L2 regularization quantity

- L2 regularization quantity = $w_1{}^2 + w_2{}^2$+……. $w_n{}^2$

L2 Regularization for Linear Regression

- Cost function $J(w,b) = \left(\frac{1}{2m}\sum_i \left(y^{(i)} - b - w_1 x_1^{(i)} - \cdots - w_n x_n^{(i)}\right)^2\right) + \frac{\lambda}{2m}(w_1{}^2 + w_2{}^2 + \cdots . w_n{}^2)$, in which hyper-parameter λ is the ***strength*** of regularization . Larger λ => larger damping of w

- $\frac{\partial J}{\partial wj} = \frac{1}{m}\sum_i \left(\hat{y}^{(i)} - y^{(i)}\right) x^{(i)} + \frac{\lambda}{m} w_j$
- $\frac{\partial J}{\partial b} = \frac{1}{m}\sum_i \left(b + \sum_j w_j x_j{}^{(i)} - y^{(i)}\right)$

- Gradient descent iteration:
  - Update b as before:
  - $b = b - \alpha(\frac{1}{m}\sum_i \left(b + \sum_j w_j x_j{}^{(i)} - y^{(i)}\right))$
  - Update w_j as: $w_j = w_j - \alpha\frac{1}{m}\sum_i \left(\hat{y}^{(i)} - y^{(i)}\right) x_j{}^{(i)} - \alpha\frac{\lambda}{m} w_j$

$$= \left(1 - \alpha\frac{\lambda}{m}\right) w_j - \alpha\frac{1}{m}\sum_i \left(\hat{y}^{(i)} - y^{(i)}\right) x_j{}^{(i)}$$

How to choose α

- **α is called a hyper-parameter of the algorithm**
- **α Tradeoff**
  - Small α => Algorithm will take long to converge
  - Large α => Algorithm may not converge and become unstable
- **Other Tips for α:**
  - **Grid Search:** Perform a grid search over a range of learning rates. Start with a small learning rate and gradually increase it. Observe the training performance and choose the learning rate that converges well without oscillations or divergence.
  - **Learning Rate Schedules:** Implement learning rate schedules that reduce the learning rate over time. For example, you can start with a relatively large learning rate and decay it during training.
  - **Use Learning Rate Finder:** Some advanced techniques involve using a learning rate finder, which systematically increases the learning rate during training and monitors the loss. The point where the loss starts to diverge is an indicator of a suitable learning rate.

How to choose Number of Iterations

There is no fixed number of iterations that universally applies to all scenarios, and it often requires experimentation. Here are some considerations:

- **Convergence Criteria:**
  - Monitor the change in the cost (or RSS) function over iterations. If the cost is decreasing and stabilizes, it suggests that the optimization process is converging. You can set a threshold for the change in the cost and stop the iterations when it falls below that threshold.
- **Batch Size:**
  - If you are using batch gradient descent, where each iteration involves the entire dataset, convergence may take longer. Mini-batch or stochastic gradient descent, where only a subset of the data is used in each iteration, can sometimes converge faster. Experiment with different batch sizes.
- **Dataset Size:**
  - Larger datasets may require more iterations for convergence. Conversely, smaller datasets may converge quickly but are prone to overfitting. Regularization techniques (e.g., L1 or L2 regularization) can be useful in preventing overfitting.
- **Early Stopping:**
  - Implement early stopping to halt the training process if the model performance on a validation set stops improving. This helps prevent overfitting and saves computational resources.

How to choose Initial Values for w_k and b?

- For linear regression, the choice of initial values for weights and biases is less critical compared to more complex models like neural networks. Here are a couple of common approaches:
  - Initialize the weights and bias to zero. This is a simple and often effective choice for linear regression.
  - Initialize the weights and bias with small random values.
  - Initialize the bias to the mean of the target variable and the weights to small random values. This will set the initial prediction close to the average target value.