SDSU CS 549 Spring 2024 Supervised Machine Learning
Lecture 4: Logistic 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

Outline

Linear Regression vs Logistic Regression

Logistic Regression Cost function

Gradient descent for logistic regression

Multinomial Regression

Linear Regression vs Logistic Regression

Linear Regression vs Logistic Regression Example Use Cases

- Linear Regression is suitable for predicting continuous outcomes, while Logistic Regression is commonly used for binary classification problems where the outcome is categorical with two classes.
- Linear Regression Use Case Examples:
- **House Price Forecasting**
  - **Use Case:** Predicting the price of a house based on features such as square footage, number of bedrooms, and location.
  - **Model:** Linear regression with the house price as the continuous target variable.
- **Sales Forecasting:**
  - **Use Case:** Predicting the sales of a product based on factors like advertising spending, seasonality, and historical sales data.
  - **Model:** Linear regression with sales as the continuous target variable.
- **Temperature Prediction:**
  - **Use Case:** Predicting the temperature based on features like time of day, season, and historical weather patterns.
  - **Model:** Linear regression with temperature as the continuous target variable.
- **GPA Prediction:**
  - **Use Case:** Predicting a student's GPA based on factors such as hours spent studying, attendance, and extracurricular activities.
  - **Model:** Linear regression with GPA as the continuous target variable.

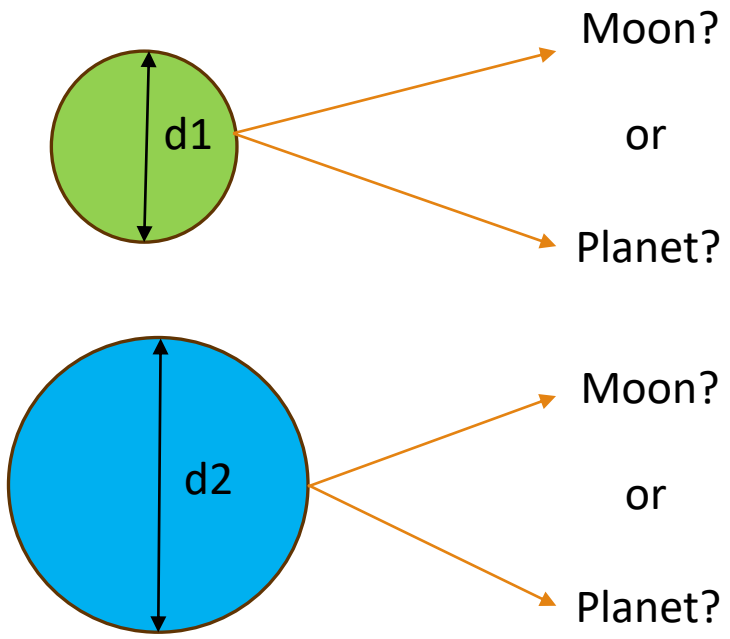
Logistic Regression – Example Use Cases

- **Customer Churn Prediction:**
  - **Use Case:** Predicting whether a customer will churn (leave) a subscription service based on customer behavior, usage patterns, and historical data.
  - **Model:** Logistic regression with binary outcome (churn or not churn).
- **Spam Email Classification:**
  - **Use Case:** Classifying emails as spam or non-spam based on features such as the presence of certain keywords, sender information, and email structure.
  - **Model:** Logistic regression with binary outcome (spam or not spam).
- **Credit Default Prediction:**
  - **Use Case:** Predicting whether a customer is likely to default on a loan based on credit history, income, and other financial indicators.
  - **Model:** Logistic regression with binary outcome (default or no default).
- **Medical Diagnosis:**
  - **Use Case:** Predicting whether a patient has a particular medical condition based on various diagnostic features.
  - **Model:** Logistic regression with binary outcome (presence or absence of the medical condition).
- **Employee Attrition Prediction:**
  - **Use Case:** Predicting whether an employee is likely to leave a company based on factors like job satisfaction, salary, and work-life balance.
  - **Model:** Logistic regression with binary outcome (attrition or no attrition).

Logistic Regression Formulation

Example Task: Celestial Body Classification

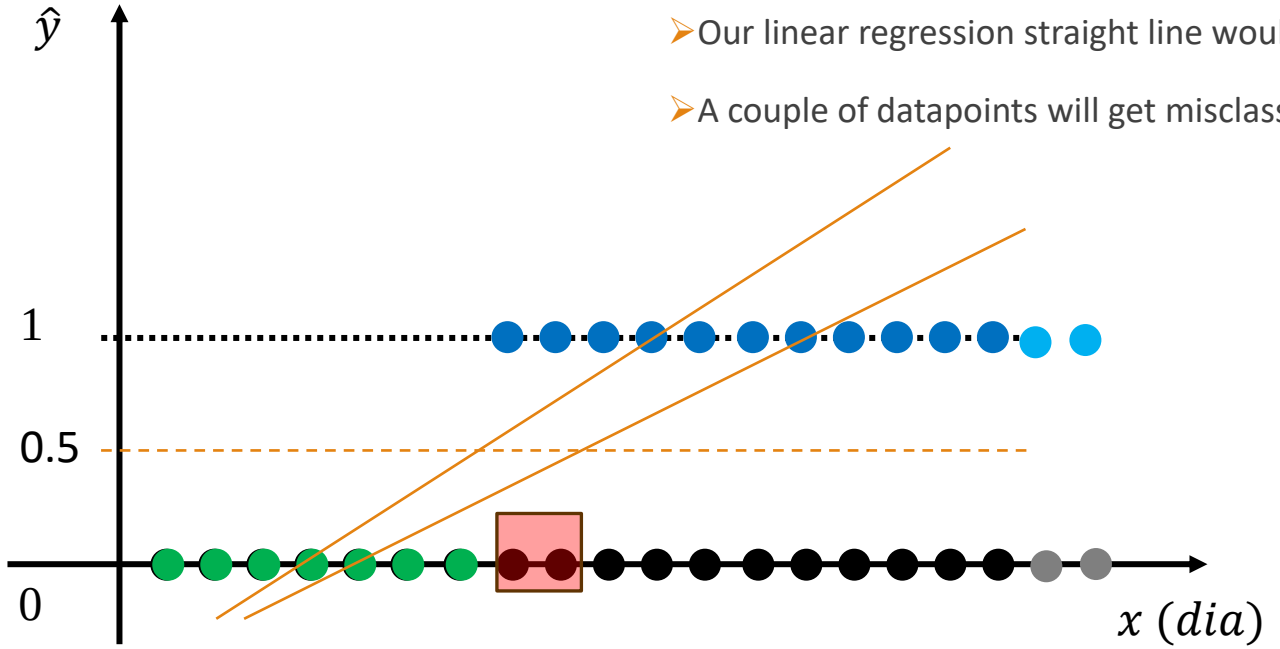
- Given a new celestial image diameter, can you help an astronomer predict if it is a moon or a planet? In general, this classification requires more input parameters (features) than just size. We start with only one feature – Diameter.



A binary classification problem

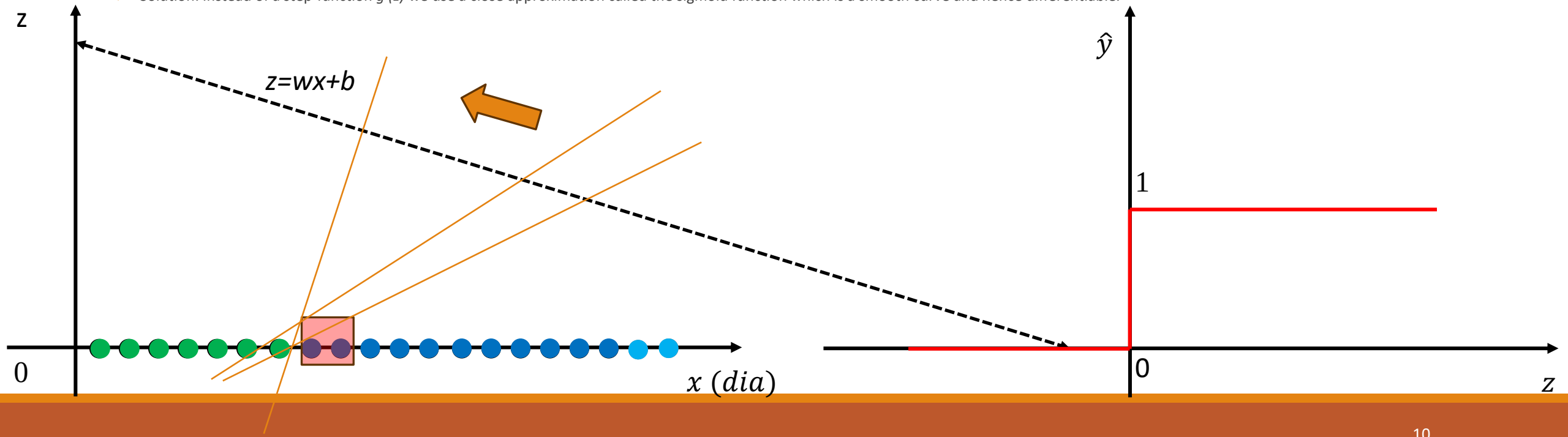
Let us Start with Linear Regression

- $\hat{y} = fwb(x) = b + wx$
- We could then say that if $\hat{y} \leq 0.5$ classify datapoint as *0* and if $\hat{y} > 0.5$ classify datapoint as 1.
- What if we obtain a couple more data points?
- Our linear regression straight line would shift
- A couple of datapoints will get misclassified as 0 while they should be 1

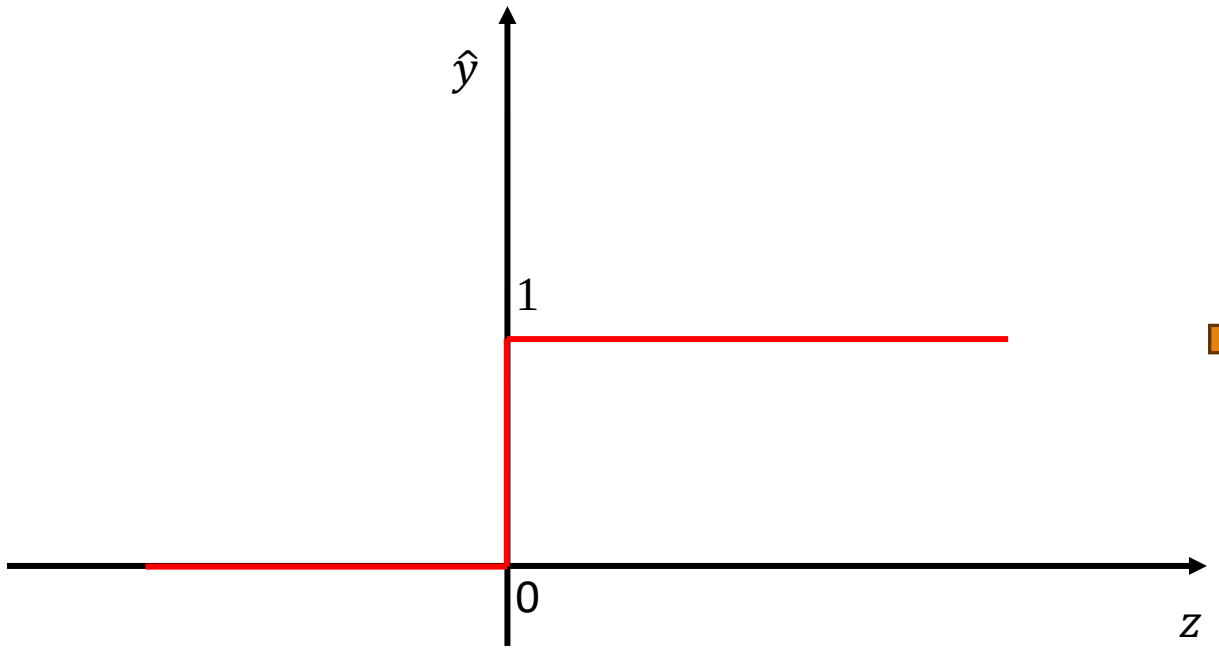


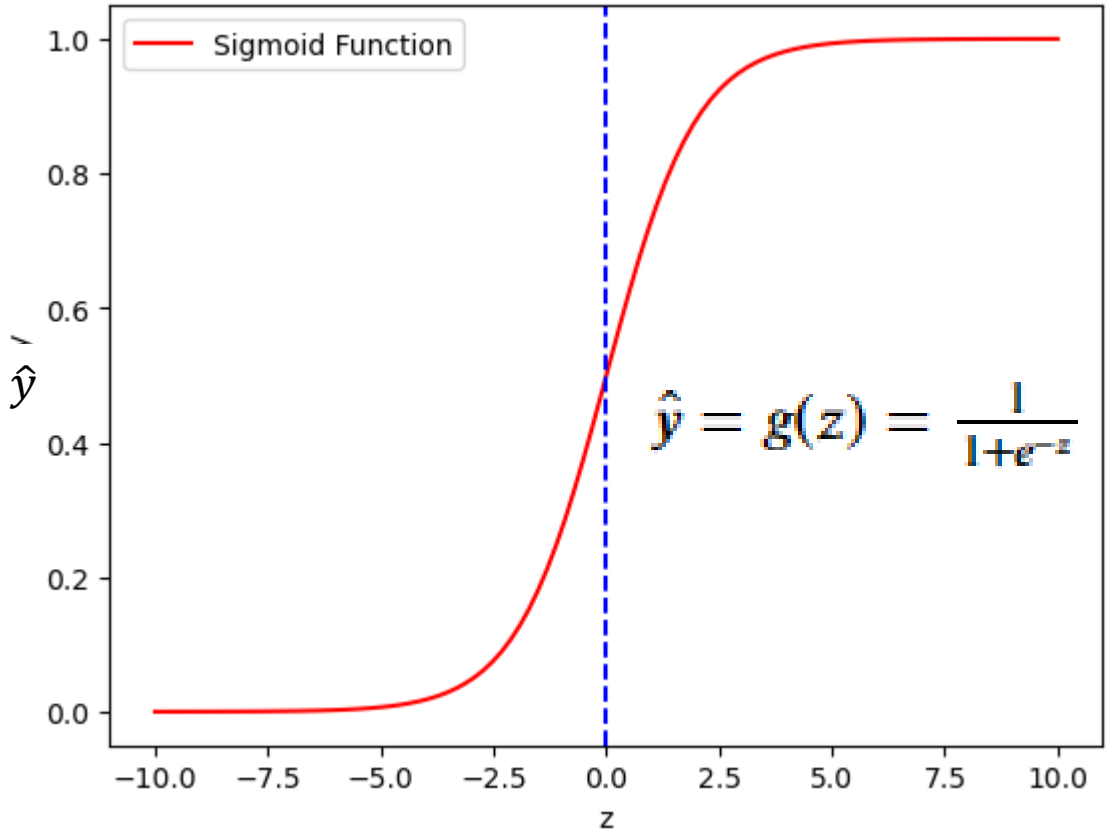
Need to Learn *w,b* differently

- Let us introduce a new variable *z = wx+b* with parameters *(w,b)* to be learned, and a new function *g(z)* that gives the predicted $\hat{y}$ (0 or 1), $\hat{y} = g(z)$
  - If $z \leq 0$ classify datapoint as *0* and if $z > 0$ classify datapoint as 1.
- Learning will happen differently than in Linear Regression. Essentially for this model, z becomes the threshold value against which x is compared
- Could g(z) be a step-function?
  - This illustrates the concept but we cannot use a step-function because it is not-differentiable and later we optimize the cost function by obtaining its gradient
  - Solution: Instead of a step-function *g (z)* we use a close approximation called the sigmoid function which is a smooth curve and hence differentiable.



Step Function -> Sigmoid function





Sigmoid function (σ) aka Logistic function – hence the name Logistic Regression for Classification machine learning

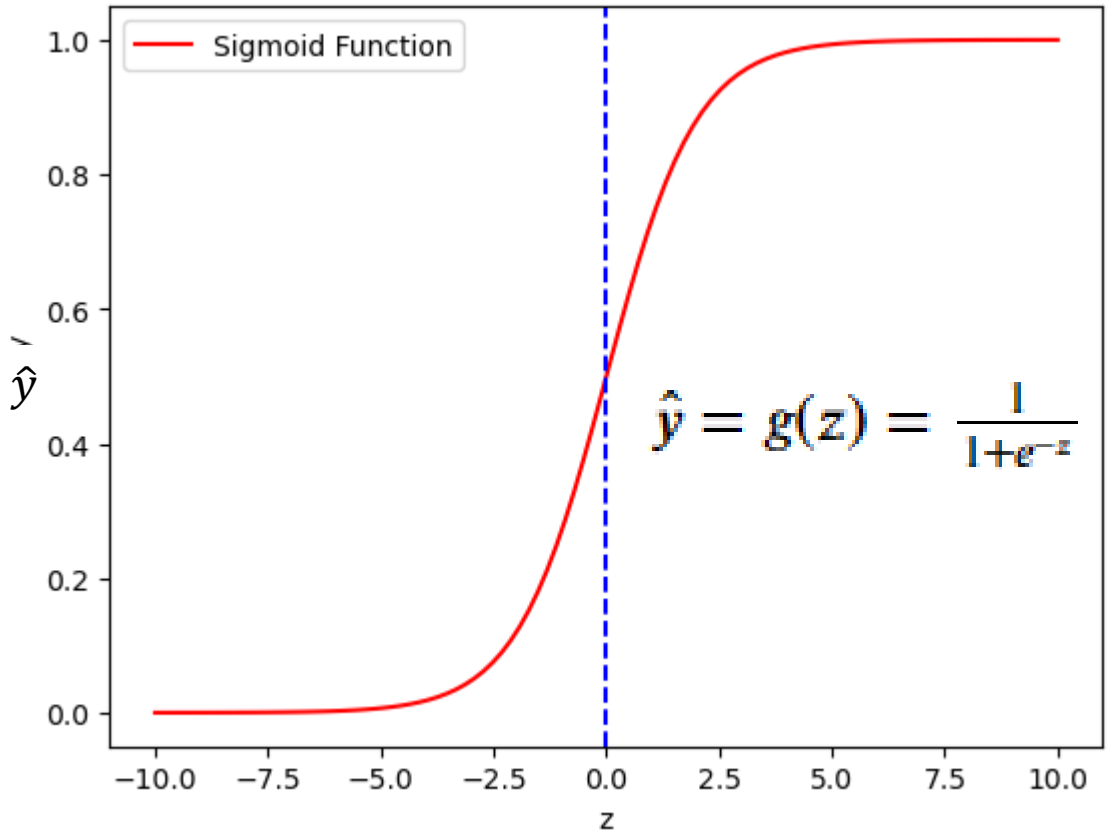
Probabilistic Interpretation of $g_{w,b}(x)$

$\hat{y} = g_{w,b}(x)$ estimates the probability of y = 0 *or* 1

$$\hat{y} = g_{w,b}(x) = \frac{1}{1+e^{-(wx+b)}}$$

$$\hat{y} = P(y = 1|x)$$

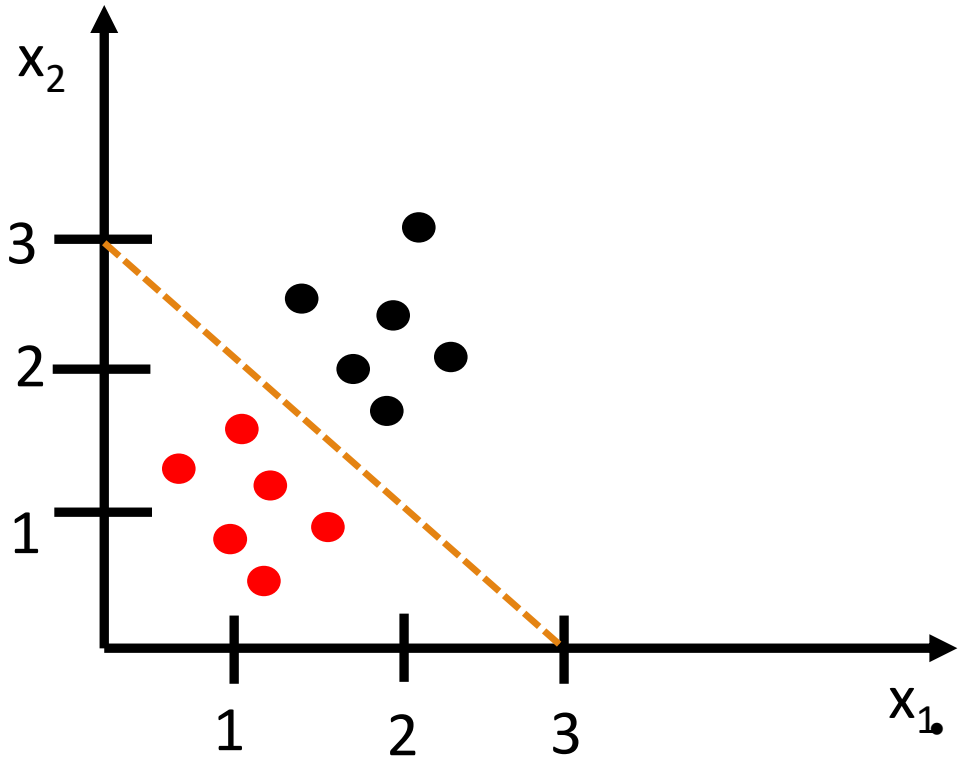
$$1 - \hat{y} = P(y = 0|x)$$



If $\hat{y} \geq 0.5$, i.e., wx + b > 0, the output is classified as 1

If $\hat{y} \leq 0.5$, i.e., wx + b ≤ 0, the output is classified as 0

A Two-Feature example for Logistic Regression



- $f(x) = \mathrm{b} + w_1 x_1 + w_2 x_2$
- Pass it through a Sigmoid Function (to get 0/1 output)
  - $g_{w,b}(x) = \frac{1}{1+e^{-f(x)}} = \frac{1}{1+e^{-(b+w_1 x_1 + w_2 x_2)}}$
- What is the decision boundary?
  - $f(x) = b + w_1 x_1 + w_2 x_2 = 0$
- For the example shown on the left, the following straight line seems to be a good decision boundary
  - $x_1 + x_2 = 3$
  - $b = -3, w_1 = 1, w_2 = 1$

- In general, for *n* features, how do we optimize for $b, w_1, w_2, \ldots, wn$?
- Let us define a cost function J, like we did for Linear Regression and find b and w_i which will minimize J

Generalizing to *n* features

- Sample data for *n* features is represented by a vector $\boldsymbol{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \cdots x_n^{(i)})$ for the i^{th} sample
- As usual such a model has *n+1* parameters, *b*, and w_i, *i=1,…n*
- $g_{w,b}(\boldsymbol{x}^{(i)}) = \frac{1}{1+e^{-(b+\sum_j w_j x_j^{(i)})}}$
- Notation: bold ***x*** or ***w*** => vector

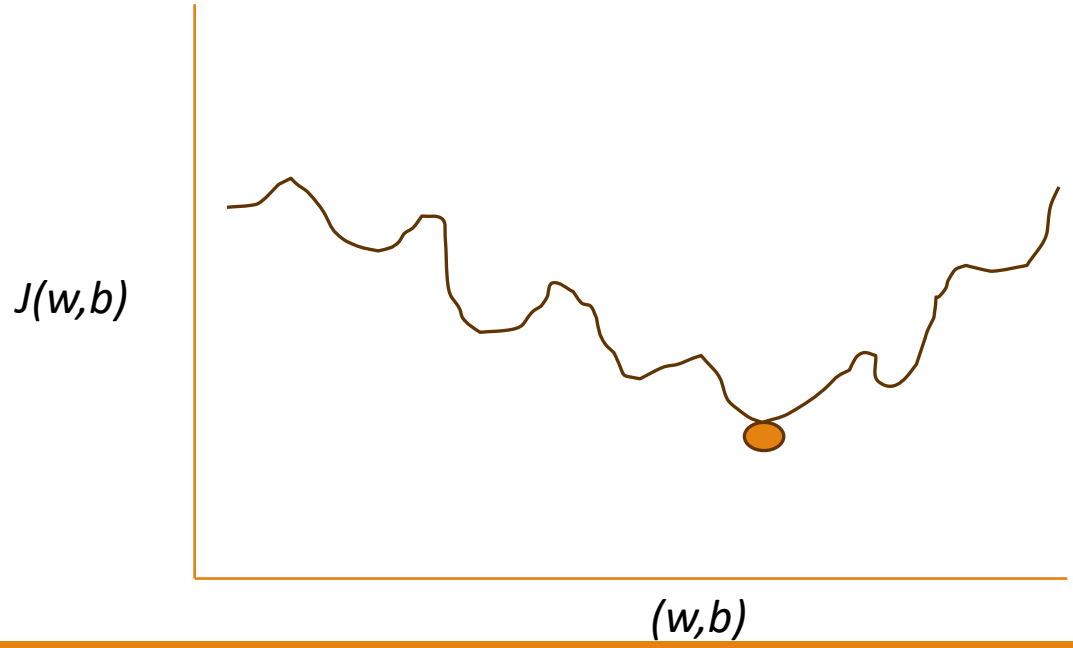
Logistic Regression Cost Function

Cost Function for Logistic Regression

We cannot use Mean Square Error (MSE) for logistic regression

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

- Cost function will become non-convex with local minima and getting to the global minimum is not guaranteed



Cost Function for Logistic Regression (Contd.)

$J(w,b) = \sum \frac{1}{m} \sum_i L_{w,b}(g_{w,b}(\boldsymbol{x}^{(i)}), \mathrm{y}^{(\mathrm{i})})$

$= \frac{1}{m} \sum_i L_{w,b}(g_{w,b}(\boldsymbol{x}^{(i)}), y^{(i)})$

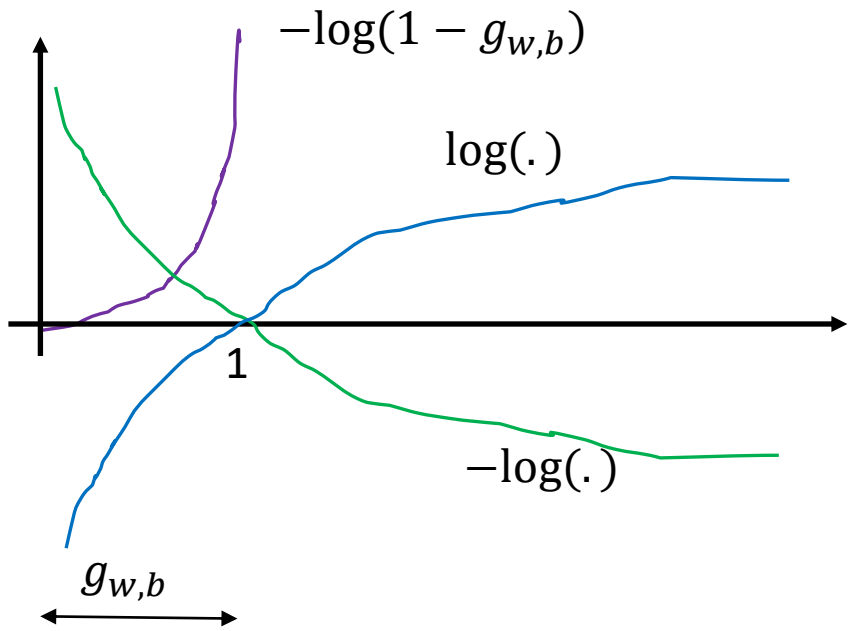
- L *is termed the Loss Function*

Logistic Loss Function is defined as:

- $\mathrm{L}_{w,b}(g_{w,b}(\boldsymbol{x}^{(i)}), \mathrm{y}^{(\mathrm{i})}) = -\log\left(\mathrm{g}_{w,b}(\boldsymbol{x}^{(i)})\right), \text{if } \mathrm{y}^{(\mathrm{i})} = 1$
- $\mathrm{L}_{w,b}(g_{w,b}(\boldsymbol{x}^{(i)}), \mathrm{y}^{(\mathrm{i})}) = -\log((1 - \mathrm{g}_{w,b}(\boldsymbol{x}^{(i)}))), \text{if } \mathrm{y}^{(\mathrm{i})} = 0$

Why is this a good loss function?

- If $y^i = 1, and\ we\ predict\ a\ 1, L_{w,b} = 0$
- If $y^i = 1, and\ we\ predict\ a\ 0, L_{w,b} = \infty$
- If $y^i = 0, and\ we\ predict\ a\ 1, L_{w,b} = \infty$
- If $y^i = 0, and\ we\ predict\ a\ 0, L_{w,b} = 0$
- We penalize wrong predictions heavily!



Turns out that the above cost function is convex => we can use gradient descent to find optimal *w,b*

Logistic Loss Function and Cost Function

Logistic Loss Function is defined as:

- $\mathrm{L}_{w,b}\left(g_{w,b}\left(\boldsymbol{x}^{(i)}\right), \mathrm{y}^{(\mathrm{i})}\right) = -\log\left(\mathrm{g}_{w,b}\left(\boldsymbol{x}^{(i)}\right)\right), \text{if } \mathrm{y}^{(\mathrm{i})} = 1$
- $\mathrm{L}_{w,b}\left(g_{w,b}\left(\boldsymbol{x}^{(i)}\right), \mathrm{y}^{(\mathrm{i})}\right) = -\log\left((1 - \mathrm{g}_{w,b}\left(\boldsymbol{x}^{(i)}\right)\right)), \text{if } \mathrm{y}^{(\mathrm{i})} = 0$

This can be written in one equation as:

- $\mathrm{L}_{w,b}\left(g_{w,b}\left(\boldsymbol{x}^{(i)}\right), \mathrm{y}^{(\mathrm{i})}\right) = -\mathrm{y}^{(\mathrm{i})}\log\left(\mathrm{g}_{w,b}\left(\boldsymbol{x}^{(i)}\right)\right) - \left(1 - \mathrm{y}^{(\mathrm{i})}\right)\log(1 - \mathrm{g}_{w,b}\left(\boldsymbol{x}^{(i)}\right))$

Previously we defined the Cost Function *J(w,b)* as:

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

$$= -\frac{1}{m}\sum_i [y^{(i)}\log\left(\mathrm{g}_{w,b}\left(\boldsymbol{x}^{(i)}\right)\right) + \left(1 - \mathrm{y}^{(\mathrm{i})}\right)\log(1 - \mathrm{g}_{w,b}\left(\boldsymbol{x}^{(i)}\right))]$$

The above cost function is convex => we can use gradient descent to find optimal *w,b*

Probability Interpretation of Cost Function

- $P(y = 1|\boldsymbol{x}; \boldsymbol{w}, b) = g_{\boldsymbol{w}b}(\boldsymbol{x})$
- $P(y = 0|\boldsymbol{x}; \boldsymbol{w}, b) = 1 - g_{\boldsymbol{w}b}(\boldsymbol{x})$
- $P(y|\boldsymbol{x}; \boldsymbol{w}, b) = g_{\boldsymbol{w}b(\boldsymbol{x})}{}^{y}\ (1 - g_{\boldsymbol{w}b(\boldsymbol{x})})^{1-y}$
- We want to find *w, b* that maximize the "likelihood": the measure of how well the model represents the observed data
- So $Likelihood\ (\boldsymbol{w}, \boldsymbol{b}) = \boldsymbol{\pi_i}\, g_{\boldsymbol{w}b}(\boldsymbol{x}^{(i)})^{\boldsymbol{y}^{(i)}}(1 - g_{\boldsymbol{w}b}(\boldsymbol{x}^{(i)})^{1-\boldsymbol{y}^{(i)}}$
- Instead of maximizing likelihood, we can maximize log likelihood
- Log($Likelihood\ (\boldsymbol{w}, \boldsymbol{b})$)= $\sum_i [y^{(i)} \log\left(\mathrm{g}_{\boldsymbol{w},b}\left(\boldsymbol{x}^{(i)}\right)\right) + \left(1 - \mathrm{y}^{(\mathrm{i})}\right)\log(1 - \ \mathrm{g}_{\boldsymbol{w},b}\left(\boldsymbol{x}^{(i)}\right))]$
- Recall the logistic regression cost function that we are minimizing: $-\frac{1}{m}\sum_i [y^{(i)} \log\left(\mathrm{g}_{\boldsymbol{w},b}\left(\boldsymbol{x}^{(i)}\right)\right) + \left(1 - \mathrm{y}^{(\mathrm{i})}\right)\log(1 - \ \mathrm{g}_{\boldsymbol{w},b}\left(\boldsymbol{x}^{(i)}\right))]$
- So the two are equivalent. Minimizing the cost function for logistic regression is equivalent to maximizing the log-likelihood that the model represents the data accurately

Gradient Descent

Gradient Descent Algorithm

- Similar to Gradient Descent for Linear Regression
- Initialize the starting point on the cost function surface, $\boldsymbol{w}_{int}$ 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 $\boldsymbol{w}_{opt}$ and b_{opt} at the point of convergence are the optimal values at which *J* is minimum
- Given $\boldsymbol{w}_{opt}$ and b_{opt}, we can calculate the predicted value $\hat{y}$ given a new $\boldsymbol{x}$ value, $\boldsymbol{x}_{new}$ as:
  - $\hat{y} = g_{\boldsymbol{w}b}(\boldsymbol{x}_{\boldsymbol{new}}) \Rightarrow \boldsymbol{If}\ b_{opt}+\boldsymbol{w}_{\text{opt}}.\boldsymbol{x}_{new} > 0$ *=> Classify as "1" and If* $b_{opt}+\boldsymbol{w}_{\text{opt}}.\boldsymbol{x}_{new} \leq 0$ *=> Classify as "0"*
- We discussed the probabilistic interpretation of $g_{\boldsymbol{w}b}(\boldsymbol{x})$
  - $P(y = 1|\boldsymbol{x}; \boldsymbol{w}, b) = g_{\boldsymbol{w}b}(\boldsymbol{x})$
  - $P(y = 0|\boldsymbol{x}; \boldsymbol{w}, b) = 1 - g_{\boldsymbol{w}b}(\boldsymbol{x})$
- The logit function, also known as the "log-odds" function, is the inverse of the sigmoid function. It transforms probability back to a real number. It is defined as: $\log(\frac{g_{\boldsymbol{w}b}(\boldsymbol{x})}{1-g_{\boldsymbol{w}b}(\boldsymbol{x})})$

What are the Gradients of *J*?

$$J(w,b) = -\frac{1}{m}\sum_i \left(y^{(i)} \log g_{w,b}\left(\boldsymbol{x}^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-g_{w,b}\left(\boldsymbol{x}^{(i)}\right)\right)\right)$$

$$\frac{\partial J}{\partial b} = -\frac{1}{m}\sum_i \frac{\partial\left[y^{(i)} \log g_{w,b}\left(\boldsymbol{x}^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-g_{w,b}\left(\boldsymbol{x}^{(i)}\right)\right)\right]}{\partial b}$$

$$\frac{\partial J}{\partial w_k} = -\frac{1}{m}\sum_i \frac{\partial\left[-y^{(i)} \log g_{w,b}\left(\boldsymbol{x}^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-g_{w,b}\left(\boldsymbol{x}^{(i)}\right)\right)\right]}{\partial w_k}$$

$$\frac{\partial J}{\partial b} = -\frac{1}{m}\sum_i \frac{y^{(i)}}{\hat{y}^{(i)}}\frac{\partial g_{w,b}\left(\boldsymbol{x}^{(i)}\right)}{\partial b} + \frac{\left(1-y^{(i)}\right)}{\left(1-\hat{y}^{(i)}\right)} - \frac{\partial g_{w,b}\left(\boldsymbol{x}^{(i)}\right)}{\partial b}$$

$$\frac{\partial J}{\partial w_k} = -\frac{1}{m}\sum_i \frac{y^{(i)}}{\hat{y}^{(i)}}\frac{\partial g_{w,b}\left(\boldsymbol{x}^{(i)}\right)}{\partial w_k} + \frac{\left(1-y^{(i)}\right)}{\left(1-\hat{y}^{(i)}\right)} - \frac{\partial g_{w,b}\left(\boldsymbol{x}^{(i)}\right)}{\partial w_k}$$

Partial Derivative of $g_{wb}(x^{(i)})$ wrt b and w_j

$$g_{\mathbf{w},b}(\mathbf{x}^{(i)}) = \frac{1}{e^{-(b+\sum_{j=1}^{n} w_j x_j^{(i)})}}$$

$$\frac{\partial g_{\mathbf{w},b}(\mathbf{x}^{(i)})}{\partial b} = \frac{e^{-(b+\sum_{j=1}^{n} w_j x_j^{(i)})}}{(1+e^{-(b+\sum_{j=1}^{n} w_j x_j^{(i)})})^2} \frac{\partial (b+\sum_{j=1}^{n} w_j x_j^{(i)})}{\partial b}$$

$$= \frac{1}{1+e^{-(b+\sum_{j=1}^{n} w_j x_j^{(i)})}} \frac{e^{-(b+\sum_{j=1}^{n} w_j x_j^{(i)})}}{1+e^{-(b+\sum_{j=1}^{n} w_j x_j^{(i)})}}$$

$$= \frac{1}{1+e^{-(b+\sum_{j=1}^{n} w_j x_j^{(i)})}} (1 - \frac{1}{1+e^{-(b+\sum_{j=1}^{n} w_j x_j^{(i)})}})$$

$$= \hat{y}^{(i)}(1 - \hat{y}^{(i)})$$

$$\frac{\partial g_{\mathbf{w},b}(\mathbf{x}^{(i)})}{\partial w_k} = \frac{e^{-(b+\sum_{j=1}^{n} w_j x_j^{(i)})}}{(1+e^{-(b+\sum_{j=1}^{n} w_j x_j^{(i)})})^2} \frac{\partial (b+\sum_{j=1}^{n} w_j x_j^{(i)})}{\partial w_k}$$

$$= \frac{1}{1+e^{-(b+\sum_{j=1}^{n} w_j x_j^{(i)})}} \frac{e^{-(b+\sum_{j=1}^{n} w_j x_j^{(i)})}}{1+e^{-(b+\sum_{j=1}^{n} w_j x_j^{(i)})}} x_k^i$$

$$= \frac{1}{1+e^{-(b+\sum_{j=1}^{n} w_j x_j^{(i)})}} (1 - \frac{1}{1+e^{-(b+\sum_{j=1}^{n} w_j x_j^{(i)})}}) x_k^i$$

$$= \hat{y}^{(i)}(1 - \hat{y}^{(i)}) x_k^i$$

Substituting in the Gradient Equations for *J*

$$\frac{\partial J}{\partial b} = -\frac{1}{m}\sum_i \frac{y^{(i)}}{\hat{y}^{(i)}}\frac{\partial g_{w,b}(\boldsymbol{x}^{(i)})}{\partial b} - \frac{(1-y^{(i)})}{(1-\hat{y}^{(i)})}\frac{\partial g_{w,b}(\boldsymbol{x}^{(i)})}{\partial b}$$

$$\frac{\partial J}{\partial w_k} = -\frac{1}{m}\sum_i \frac{y^{(i)}}{\hat{y}^{(i)}}\frac{\partial g_{w,b}(\boldsymbol{x}^{(i)})}{\partial w_k} - \frac{(1-y^{(i)})}{(1-\hat{y}^{(i)})}\frac{\partial g_{w,b}(\boldsymbol{x}^{(i)})}{\partial w_k}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum_i (\hat{y}^{(i)} - y^{(i)})$$

$$\frac{\partial J}{\partial w_k} = \frac{1}{m}\sum_i (\hat{y}^{(i)} - y^{(i)})\, \mathrm{x_k}^{(i)} \text{ , } k=1,2,\ldots n$$

- Similar to Linear Regression
- Only difference is how $\hat{y}^{(i)}$ is computed

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)
  - $\frac{\partial J}{\partial b} = \frac{1}{m}\sum_i (\hat{y}^{(i)} - y^{(i)})$
  - $\frac{\partial J}{\partial wj} = \frac{1}{m}\sum_i (\hat{y}^{(i)} - y^{(i)})\, \mathrm{x_j}^{(i)}$, *k=1,2,…n*
- Move along cost function "surface" in a direction opposite to the gradient by a small amount ($\propto$)
  - $b = b - \alpha(\frac{\partial J}{\partial b})$
  - $w_j = wj - \alpha(\frac{\partial J}{\partial wj})$
- Keep repeating until "convergence"

Multi-Class Classification

Multi-Class Classification Use Case Examples

Multi-class classification is a type of machine learning task where the goal is to classify instances into more than two classes or categories. Here are several real-world use cases for multi-class classification:

- Recognizing handwritten digits (0-9) is a classic example of multi-class classification. This application is common in postal services, where automated systems need to read zip codes or recognize digits on checks.
- Classifying spoken words or phrases into predefined categories is a multi-class classification task. This is used in virtual assistants, voice-activated devices, and transcription services.
- Determining the sentiment of a piece of text (positive, negative, or neutral) is often formulated as a multi-class classification problem. Applications include analyzing customer reviews, social media sentiment, and feedback.
- Diagnosing medical conditions based on various features such as symptoms, test results, and patient history involves multi-class classification. Examples include classifying diseases, predicting patient outcomes, and identifying medical conditions from imaging data.
- Identifying objects in images where there are multiple classes is a multi-class classification problem. This is widely used in applications like autonomous vehicles, surveillance systems, and image-based search engines.
- Categorizing documents into topics or types is a common use case for multi-class classification. It is employed in areas such as spam detection, news categorization, and document management.
- Classifying species based on features like DNA sequences, physical characteristics, or ecological traits involves multi-class classification. This is common in biodiversity studies and conservation efforts.
- Identifying faults or anomalies in industrial systems, such as machinery or manufacturing processes, is often approached as a multi-class classification problem. It helps in preventive maintenance and minimizing downtime.
- Predicting whether a customer will churn or stay with a service involves multi-class classification. Different classes could represent categories like "churned," "at risk of churning," and "loyal."
- Classifying human activities, such as walking, running, or sitting, based on sensor data from wearable devices or smartphones is a multi-class classification task. This is used in fitness tracking and healthcare applications.
- Determining the language of a given text is a multi-class classification problem. It is useful in applications like language translation, content localization, and text processing.

How to build a Model for Multiclass Classification?

- Generalization of Logistic Regression
- Known as Softmax or multinomial Logistic Regression
- As in Logistic Regression, let us start with sample data
- Sample data for *n* features is represented by a vector $\boldsymbol{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \cdots x_n^{(i)})$ for the i^{th} sample
- Instead of a scalar label, *y(i)*, now we have a vector label $\boldsymbol{y}(i) = (y_1^{(i)}, y_2^{(i)}, \cdots y_K^{(i)})$, for *K* classes

Cost Function

- Cost function for Logistic Regression: $J(w,b) = -\frac{1}{m}\sum_i \left(y^{(i)} \log g_{w,b}\left(\boldsymbol{x}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - g_{w,b}\left(\boldsymbol{x}^{(i)}\right)\right)\right)$
  - Where $g_{w,b}\left(\boldsymbol{x}^{(i)}\right) = \frac{1}{1+e^{-(b+\sum_j . w_j x_j{}^{(i)})}}$ is the sigmoid function and the prob that $y^{(i)} = 1$
- Analogous cost function for multi-class classification can be represented as:
  - $J(w,b) = -\frac{1}{m}\sum_i \sum_k v_k{}^{(i)} . \log(p_{ki})$
  - Where $v_k{}^{(i)}$ is a one-hot encoded vector representing the true class labels, basically it is *1* if the i^{th} sample is class *k*
  - p_{ki} is the predicted probability that the i_{th} sample belongs to class *k*
  - $p_{ki} = \frac{e^{(b_k+\sum_j . w_{jk} x_j{}^{(i)})}}{\sum_l e^{(b_l+\sum_j . w_{jl} x_j{}^{(i)})}}$, *l=1,2,…K, also known as the softmax activation function*
  - The model parameters are b_l and w_{jl}, *j = 1,2,…n and l=1,2,..K*

Gradient Descent

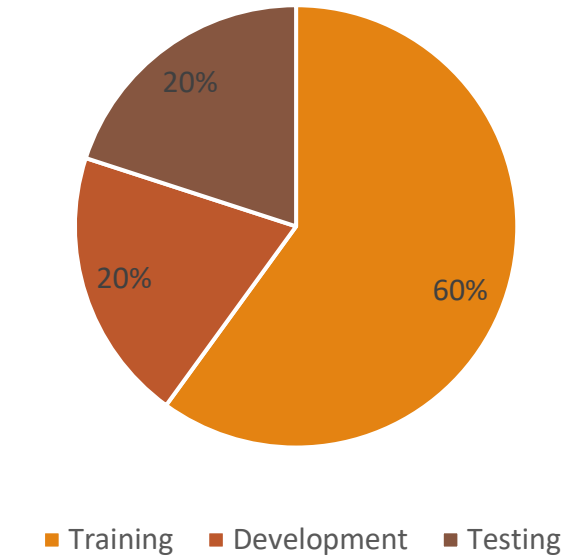
- Logistic Regression Gradients are:
  - $\frac{\partial J}{\partial b} = \frac{1}{m}\sum_i(\hat{y}^{(i)} - y^{(i)})$
  - $\frac{\partial J}{\partial wj} = \frac{1}{m}\sum_i(\hat{y}^{(i)} - y^{(i)})\, x_j^{(i)}$, *j=1,2,…n*
- Analogous Multi-class classification Gradients are:
  - $\frac{\partial J}{\partial b_l} = \frac{1}{m}\sum_i(\hat{y}^{(i)} - v_l^{(i)})$, *l=1,2,…K*
  - $\frac{\partial J}{\partial w_{jl}} = \frac{1}{m}\sum_i(\hat{y}^{(i)} - v_l^{(i)})\, x_j^{(i)}$, *j=1,2,…n*, l = 1,2, … K
- We apply Gradient Descent algorithm as usual

Practical Considerations and Performance Evaluation with Logistic Regression Model

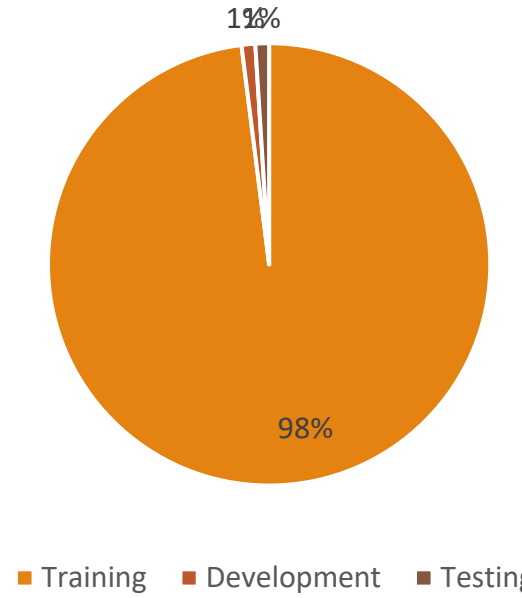
Training/Development(Validation)/test sets (Applies to all machine learning models)

- 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)

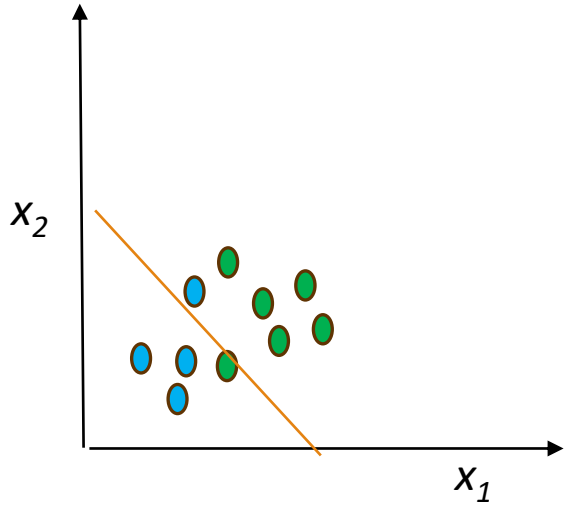


Performance Metrics

Use Confusion Matrix

- Also used in statistics for hypothesis testing performance metrics

	Predicted Class 0	Predicted Class 1
Actual Class 1	FN	TP
Actual Class 0	TN	FP



- **TP (True Positive):** The model correctly predicted instances of Class 1.
- **FP (False Positive):** The model incorrectly predicted instances as Class 1 when they were actually Class 0 (Type I error).
- **FN (False Negative):** The model incorrectly predicted instances as Class 0 when they were actually Class 1 (Type II error).
- **TN (True Negative):** The model correctly predicted instances of Class 0.

Performance Metrics (Contd.)

	Actual Class 0	Actual Class 1
Predicted Class 1	TP	FP
Predicted Class 0	FN	TN

- **Accuracy:** The proportion of correctly classified instances $\frac{TP+TN}{TP+FP+TN+FN}$
- **Precision:** The proportion of true positive predictions among all positive predictions $\frac{TP}{TP+FP}$
- **Recall (Sensitivity):** The proportion of true positive predictions among all actual positives $\frac{TP}{TP+FN}$
- **F1 Score:** The harmonic mean of precision and recall $2 * \frac{Precision*Recall}{Precision+ecall}$, max value 1 (perfect precision and recall), min value 0 (no precision or recall)

Performance Metrics (Contd.)

- Tradeoff between Precision and Recall
  - **Precision:** $\frac{TP}{TP+FP}$
  - **Recall (Sensitivity):** $\frac{TP}{TP+FN}$

- **High Precision, Low Recall:**
  - If precision is prioritized, the model tends to make positive predictions only when it is very confident, leading to fewer false positives. However, this may result in missing some actual positive instances, leading to lower recall.

- **High Recall, Low Precision:**
  - If recall is prioritized, the model aims to capture as many positive instances as possible, even if it means making more false positive predictions. This can lead to a lower precision, as the model may be less selective in its positive predictions.

- The choice between precision and recall depends on the specific goals and consequences of false positives and false negatives in the given application. For example:
  - In medical diagnosis, where false negatives may be more critical, recall might be prioritized.
  - In spam email detection, where false positives are undesirable, precision might be prioritized.

- It's essential to consider the application's requirements and the potential impact of false positives and false negatives when deciding on the right balance between precision and recall.

Perf Metrics for Multi-Classification Model

Confusion Matrix for a Multi-Classification Model				
	Actual Class 1	Actual Class 2	……..	Actual Class k
Predicted Class 1	P11	P12	…..	P1k
Predicted Class 2	P21	P22	……	P2k
….	….	….	….	….
Predicted Class k	Pk1	Pk2		Pkk

- **Pij:** The model predicted instances of class *j* as class *i*
- The diagonal elements (from top-left to bottom-right) represent the true positives for each class, and the off-diagonal elements represent misclassifications.

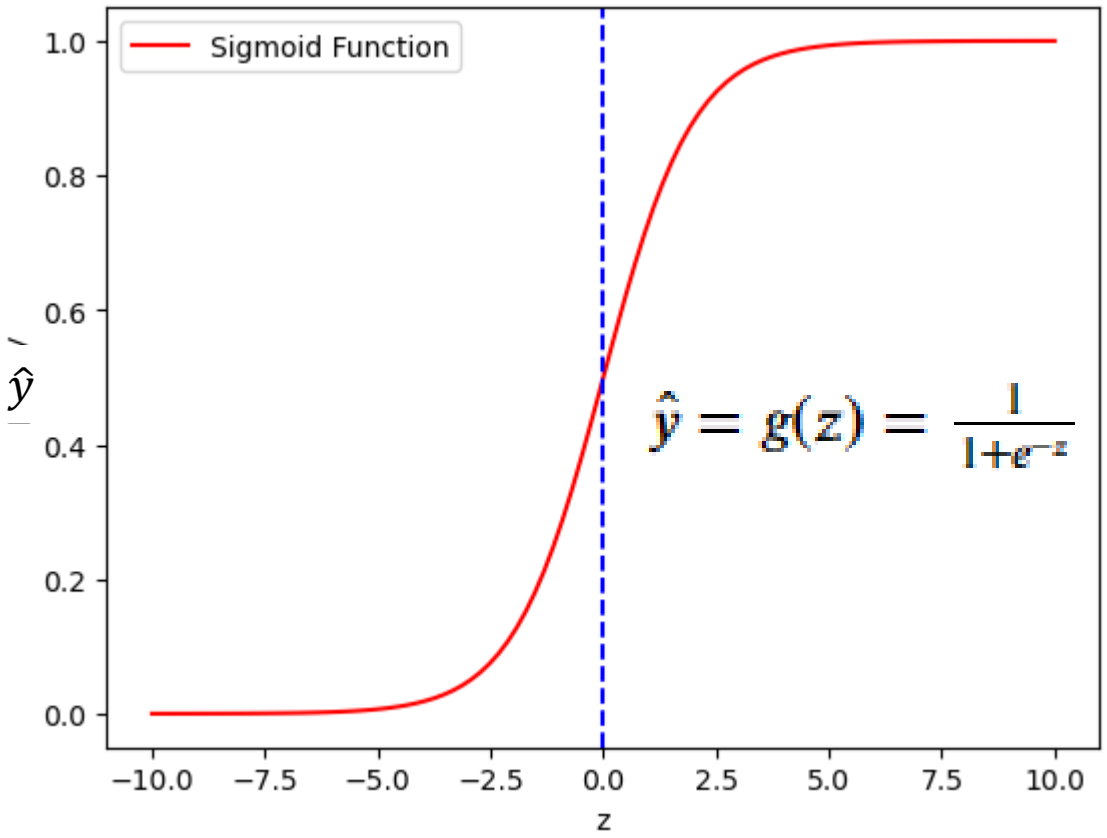
Perf Metrics for Multi-Classification Model (Contd.)

Confusion Matrix for a Multi-Classification Model				
	Actual Class 1	Actual Class 2	……..	Actual Class k
Predicted Class 1	P11	P12	…..	P1k
Predicted Class 2	P21	P22	……	P2k
….	….	….	….	….
Predicted Class k	Pk1	Pk2		Pkk

- **Precision for Class (i)=** $\frac{Pii}{Pi1+Pi2+\cdots Pik}$
- **Recall (Sensitivity) for Class (i) =** $\frac{Pii}{P1i+P2i+\cdots Pki}$
- **F1 Score for Class (i):** $2 * \frac{\boldsymbol{Precision_i * Recalli}}{\boldsymbol{Precision_i + Recalli}}$

How to Tradeoff among Various Metrics

- **Precision:** The proportion of true positive predictions among all positive predictions $\frac{TP}{TP+FP}$
- **Recall (Sensitivity):** The proportion of true positive predictions among all actual positives $\frac{TP}{TP+FN}$
- The default classification threshold for logistic regression is usually set at 0.5.
  - By adjusting the threshold, we can influence the trade-off between precision and recall.
- **Lowering the Threshold:**
  - Increases sensitivity/recall.
  - Decreases precision.
  - More instances are predicted as positive, which may result in more false positives.
- **Raising the Threshold:**
  - Increases precision.
  - Decreases sensitivity/recall.
  - Fewer instances are predicted as positive, which may result in more false negatives.
- **Choosing the Threshold:**
  - It depends on the specific requirements of the problem.
  - In medical diagnoses, we might prefer a higher recall.
  - In fraud detection, we might prefer higher precision.

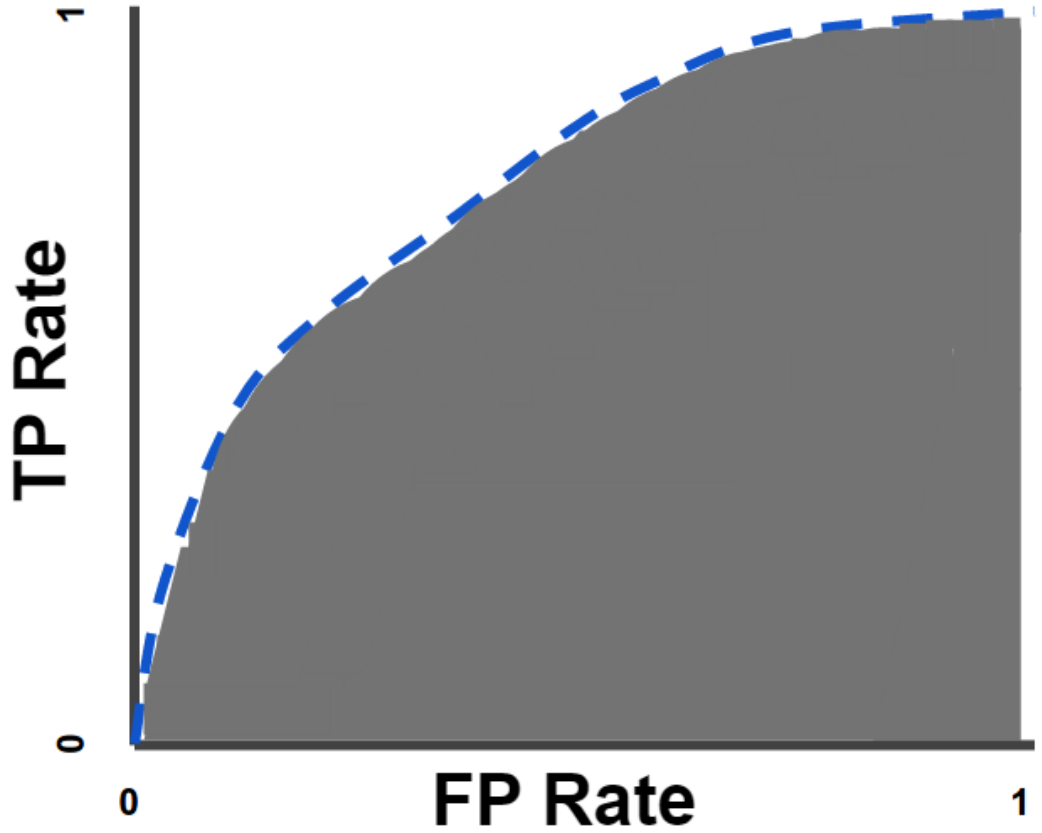


Receiver Operating Characteristic (ROC) Curve

- **True Positive Rate** (**TPR**) is a synonym for recall and is therefore defined as : $\frac{TP}{TP+FN}$
- **False Positive Rate** (**FPR**) is defined as: $\frac{FP}{FP+TN}$
- An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.
- Area under ROC (AUC) provides an aggregate measure of performance across all possible classification thresholds

AUC is desirable for the following two reasons:

- AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.
- AUC is **classification-threshold-invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.



From: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc#:~:text=An%20ROC%20curve%20(receiver%20operating,False%20Positive%20Rate

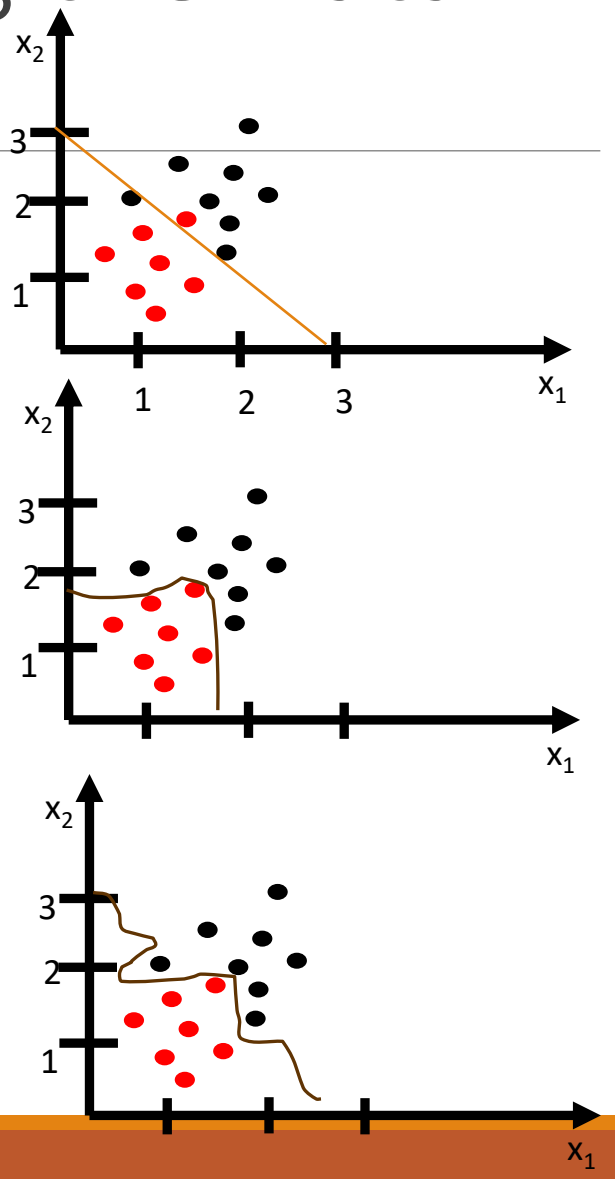
How many samples (*m*) and how many features (*n*) to use?

- **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.

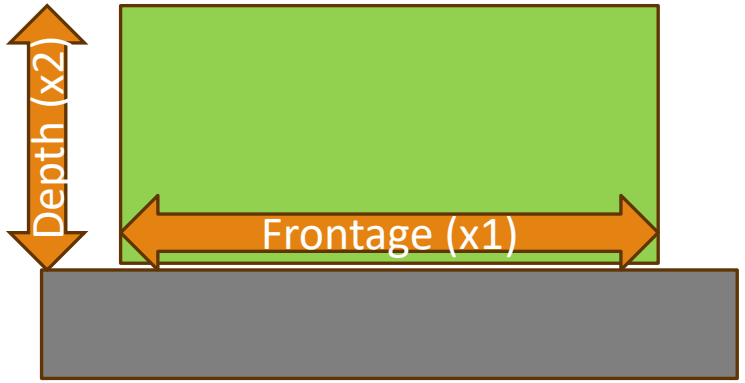
Under-Fitting and Over-Fitting the Data

- 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.



Avoiding Under-Fitting

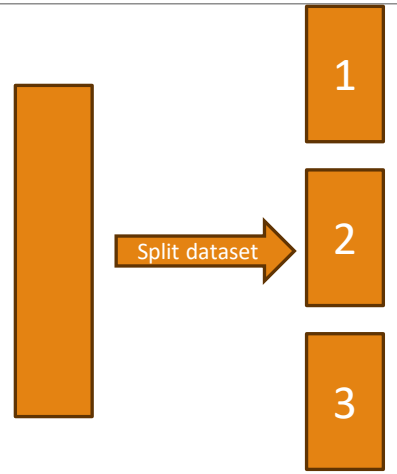
- Underfitting occurs when a machine learning model is too simple to capture the underlying patterns in the training data, resulting in poor performance on both the training and unseen data. To avoid underfitting, consider the following strategies:
  - **Use a More Complex Model:** If your current model 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 attractiveness predictor $\hat{y} = \sigma(b + w_1x_1 + w_2x_2)$ and we are under-fitting sample data
- Consider creating a third variable $x_3 = x_1 * x_2$ and use the following predictor: $\hat{y} = \sigma(b + w_1x_1 + w_2x_2 + w_3x_3)$

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 Logistic Regression

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

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

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

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

- $\alpha\frac{\lambda}{m}$ will always be *<1*

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.