Machine Learning Techniques

Introduction

Dr. Ashish Tendulkar

IIT Madras

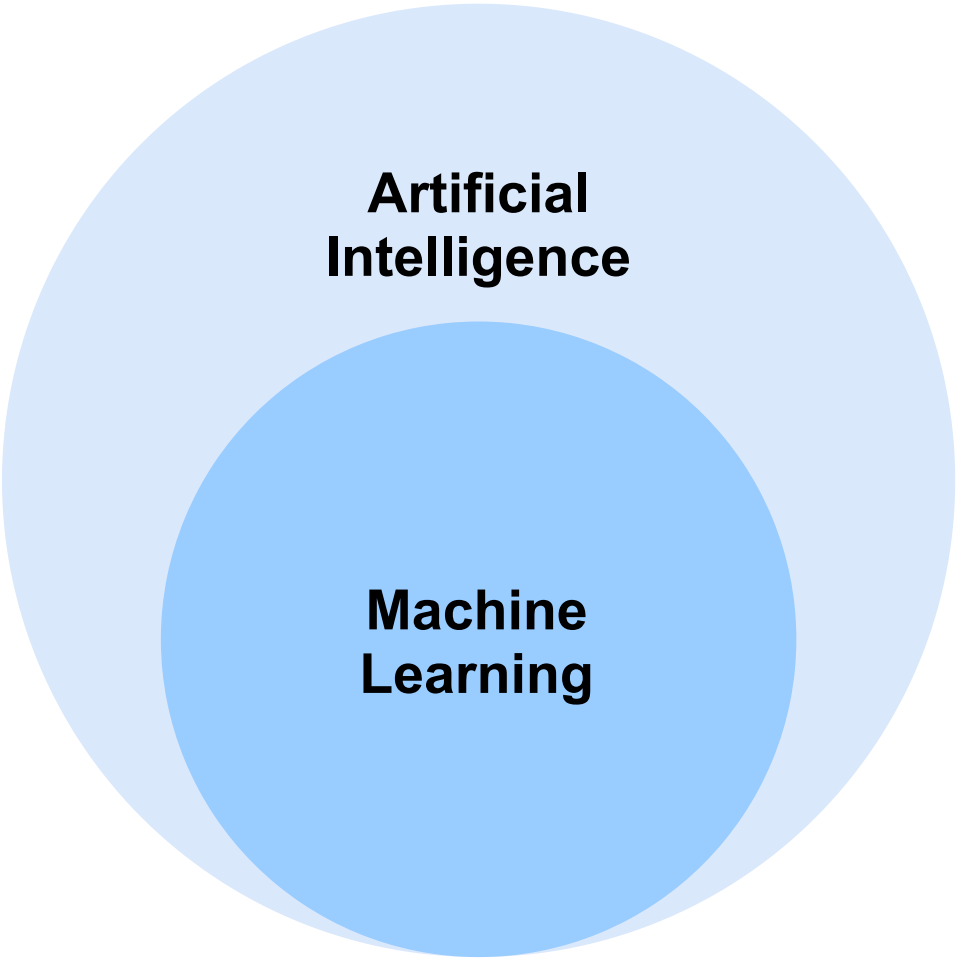
1. Introduction
2. ML Applications
3. Teaching Philosophy

Introduction

In this course:

- Machine learning algorithms from a theoretical perspective.
- Implement them from scratch.

ML and AI



Topics of the course

In this course, we will study many popular ML algorithms like

- Linear Regression
- Logistic Regression
- Decision Trees
- Support Vector Machines
- Neural Networks
- K-means clustering

1 Introduction

2 ML Applications

3 Teaching Philosophy

Application domains of ML

ML is widely used in different domains like:

- Communication and search technologies.
- E-Commerce
- Finance
- Fashion
- Manufacturing
- Health Science
- Wildlife conservation.
- Agriculture

ML in daily life

- Search Engines

Image Source: https://wildstonesolution.com/top-search-engines/

ML in daily life

- Product recommendation

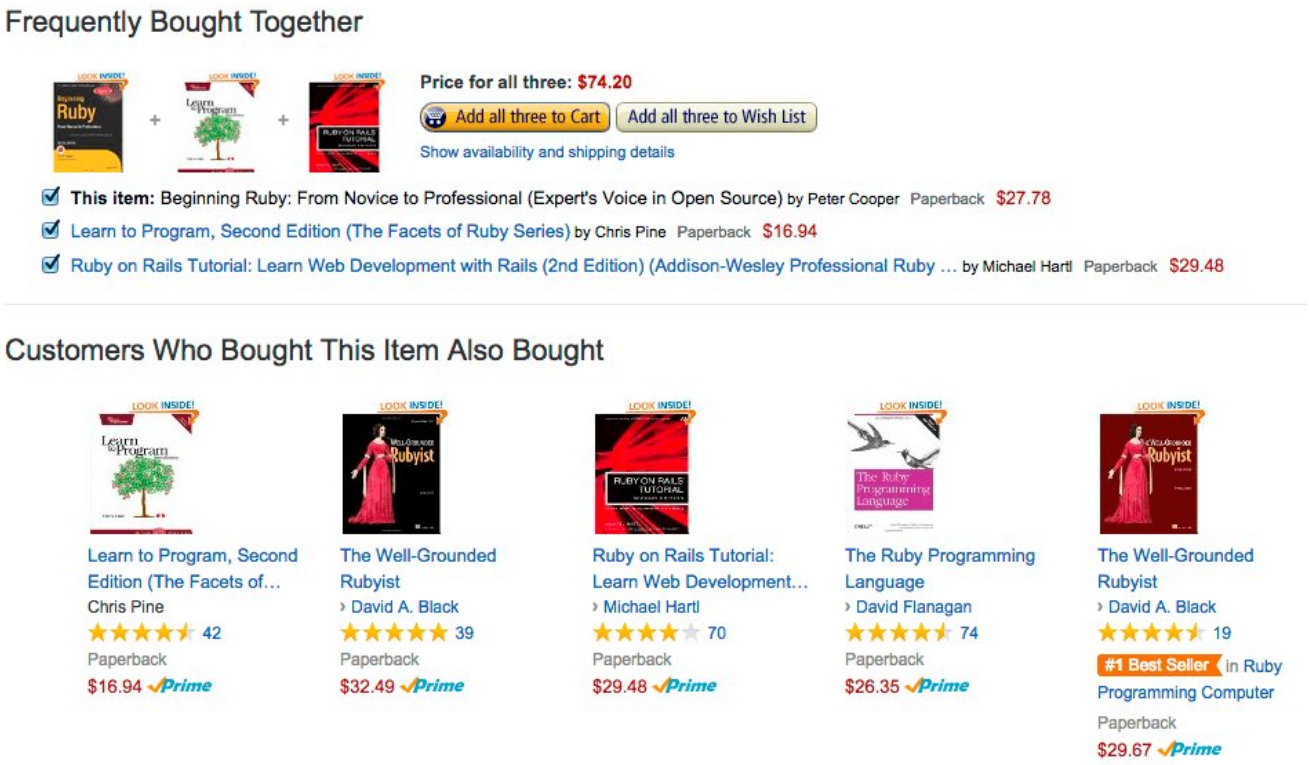


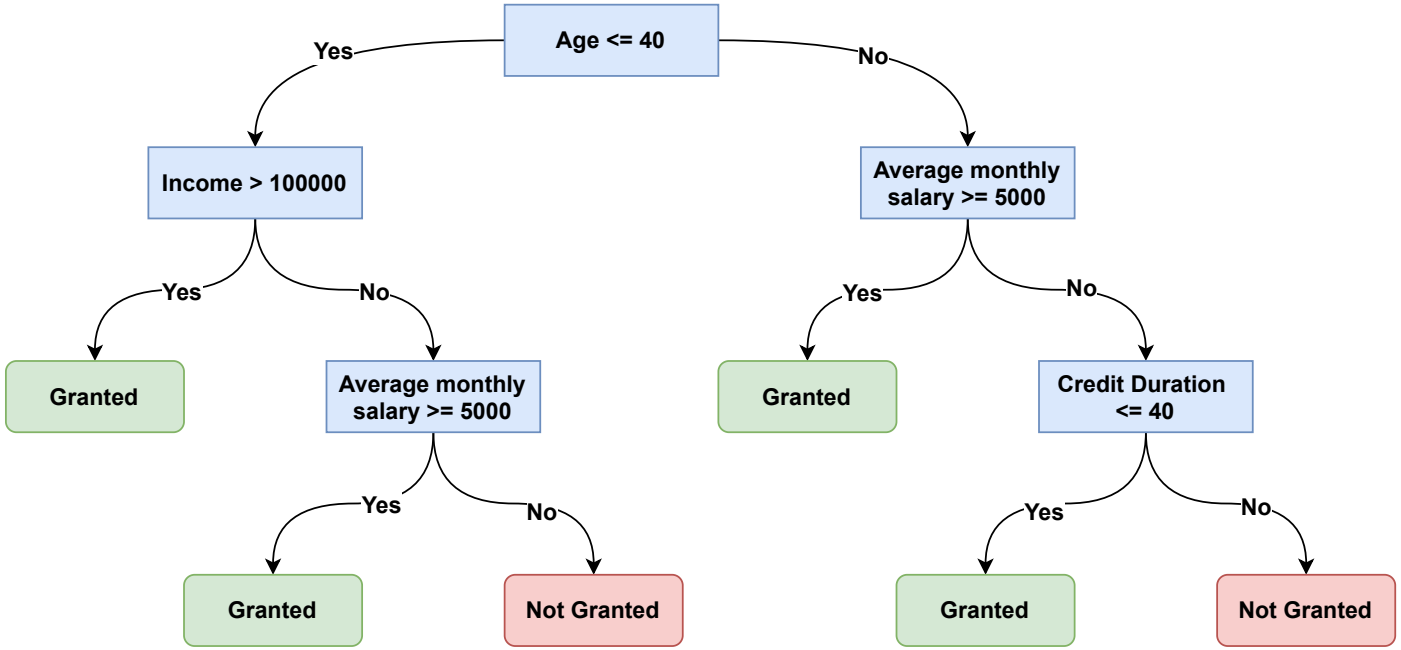
Image Source: Amazon.com

ML in daily life

- Social networks

Image Source:Whatmobile.net

- Loan approval



- Stock market prediction

Image Source: KDNuggets

- Fashion

Image Source: Tryon Jewellery

Image Source: Mirrar.com

- Manufacturing

Image Source: Forbes

- Protein folding

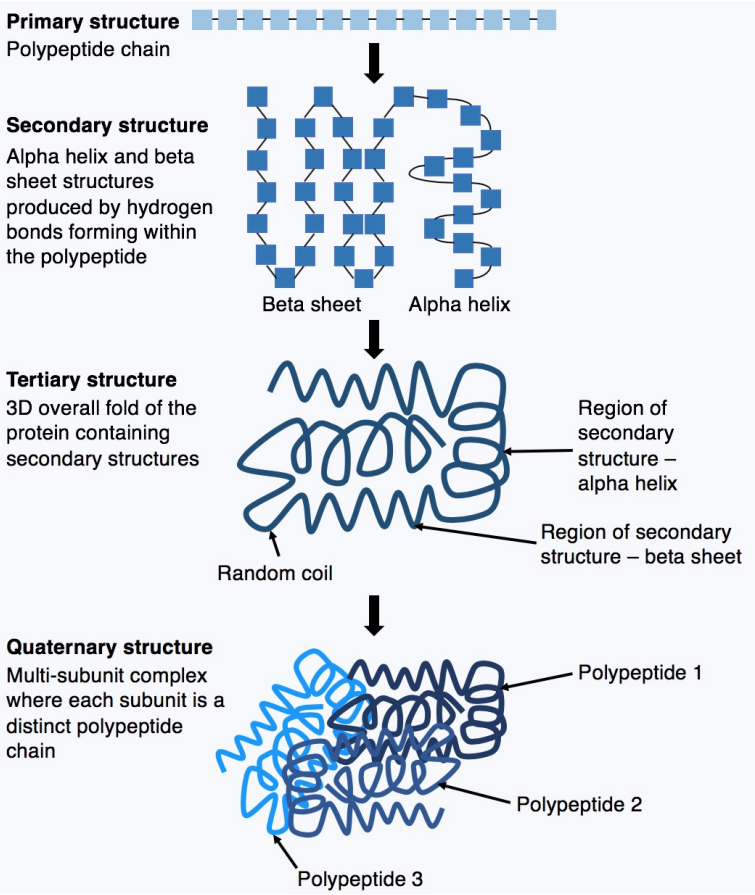
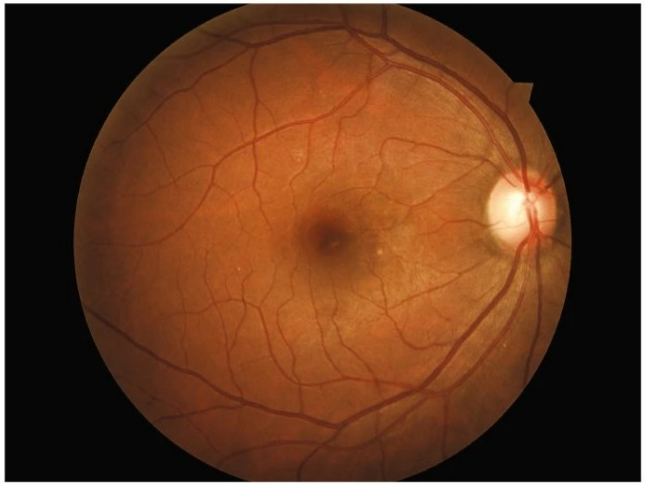


Image Source: Wikipedia Commons

- Treatment of diseases like retinopathy, diabetes etc.

Patient without Diabetic Retinopathy



Patient with Diabetic Retinopathy

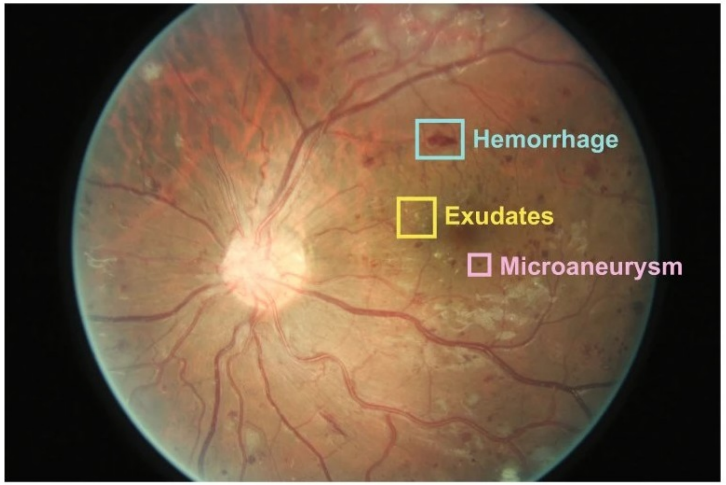


Image Source: Nature.com

- Detecting poachers and wild animals

Image Source: WildLabs

- Detecting illegal wood cutting from sound recording

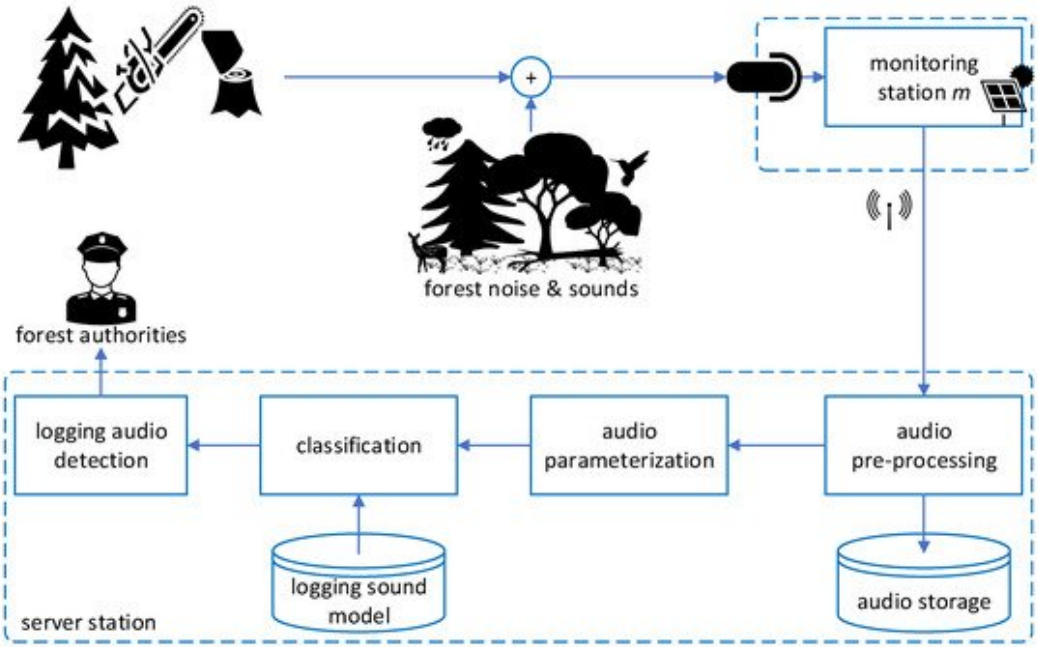


Image Source: MDPI

- Agriculture

Image Source: TheGeoSpatial.in

1 Introduction

2 ML Applications

3 Teaching Philosophy

ML Modular Approach

1. Training Data
2. Model
3. Loss Function
4. Optimization Technique
5. Evaluation

Learning each concept at three levels:

1. Intuition
2. Mathematical aspect
3. Implementation

- In the techniques course, we will be implementing ML algorithms from scratch.
- In the practice course, we will be implementing ML algorithms with tools and APIs provided in the scikit-learn library.
- Practice questions and programming exercises after each video or module.
- Questionnaire at the end.

Machine Learning Techniques

ML vs Traditional Programming

Dr. Ashish Tendulkar

IIT Madras

1. ML vs Traditional Programming
2. Machine Learning
3. Summary

ML vs Traditional Programming

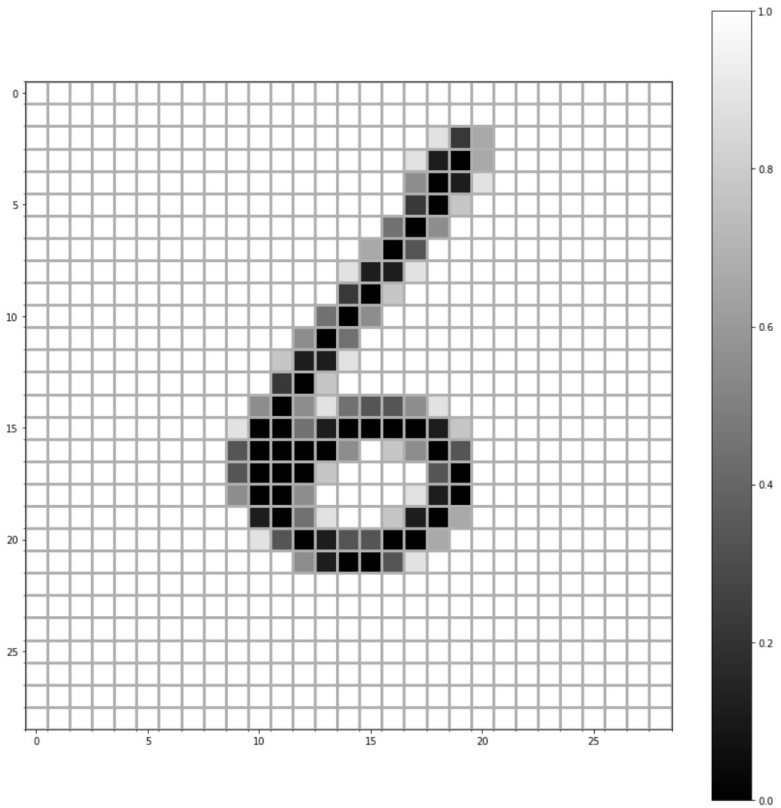
Let's take a couple of tasks and try to write programs to solve each of them:

- Task 1: Adding two numbers:

$$12 + 45 = ?$$

ML vs Traditional Programming

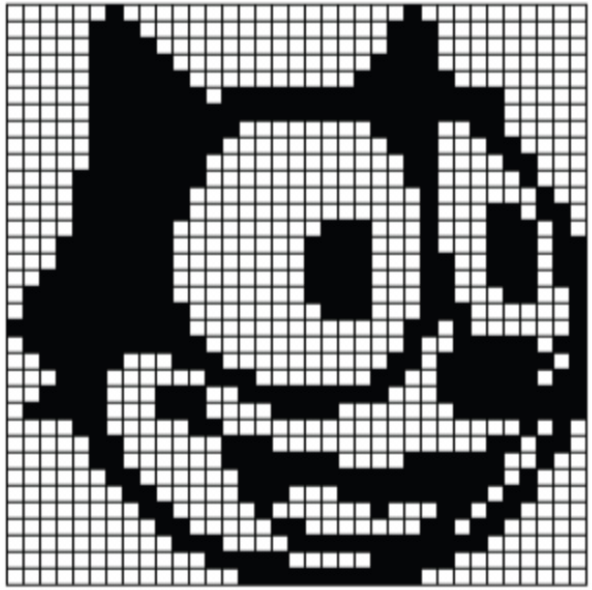
- Task 2: Recognize handwritten digit. An image is a matrix of numbers.



Which one is easier?

Image representation of a computer

How does a computer see an image?



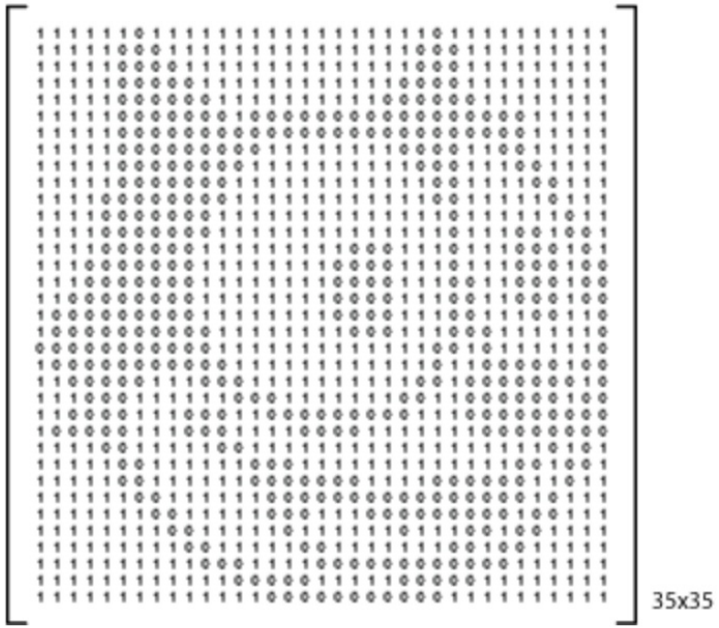


Image Source:uff.br

Adding Numbers

- Adding two numbers

$$12 + 45 = 57$$

- Trivial for any decent programmer.

Recognise a digit

Solving the second task:

- Design a bunch of rules for solving this task, which seems like a reasonable approach.
- However, such rules are brittle and one can readily come up with examples that will break such a rule-based system.

What is a more robust way of solving this task?

What makes the first task trivial and the second one tricky to program?

- Mathematical function for addition of two numbers, $c = a + b$, is ***already available!***
- In case of the second task of handwritten digit recognition, we ***do not have an exact mathematical function*** that takes an image as an input and outputs the recognized digit.
- With ML, we hope to uncover the mathematical form that maps images to the digits.

1 ML vs Traditional Programming

2 Machine Learning

3 Summary

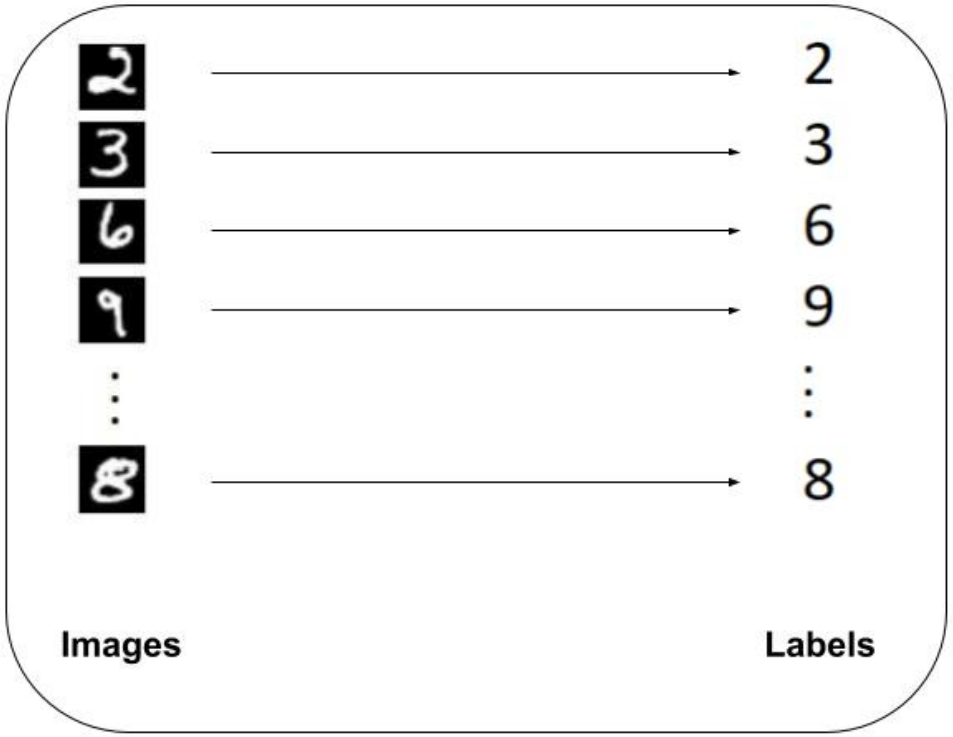
Tasks solvable with ML

- As a general rule, there are many tasks that humans do well, but are unable to elaborate the step by step process that they follow for solving those tasks.
- Such tasks are amenable to ML.
- We can at least attempt to address them through ML.

How can we do that?

- Think, how do you recognize a digit from the image?
- You learnt this in school and since then, we have looked at many digits and have become better at the task.
- **Can we replicate this process for computers?**
- **Can we train computers to recognize digits in this way?**
- **Maybe yes!**

Training Data



Training Data

Model: Mathematical relationship

- Our job is to take this training data and uncover the mathematical relationship between pixels in the image and the label.
- For example, a simple linear relationship between pixels and labels is as following:

$$w_0 + w_1 \cdot pixel_1 + w_2 \cdot pixel2 + \cdots + w_m \cdot pixel_m$$

- Weights parameters: $w_0, w_1, w_2, \ldots, w_m$
- Our job is to find these parameters. How can we do it?

Finding parameters: Loss function

- Can these parameters be assigned some random numbers to get a model? Would that model be good enough? How do we measure it?
- **Loss function**: It measures disagreement between the actual outcome and the predicted outcome by applying the model to the input.
- The ideal set of parameters result in the minimum loss.
- Optimization procedure: search over the parameter space and find the optimal parameters that minimize the loss function.

Evaluation

- Once the model is trained, it is evaluated with appropriate metrics like accuracy.
- If this metric is satisfactory, we have a model that can be used for making predictions.
- In case, it is not satisfactory, the process is repeated by adding more features or more data, depending on the metric.

Fundamental Assumption

Why does this method of learning parameters from training data work? What are some of the fundamental assumptions that we make in ML?

The training and test data would come from the same distribution.

Fundamental Assumption:Example of History test

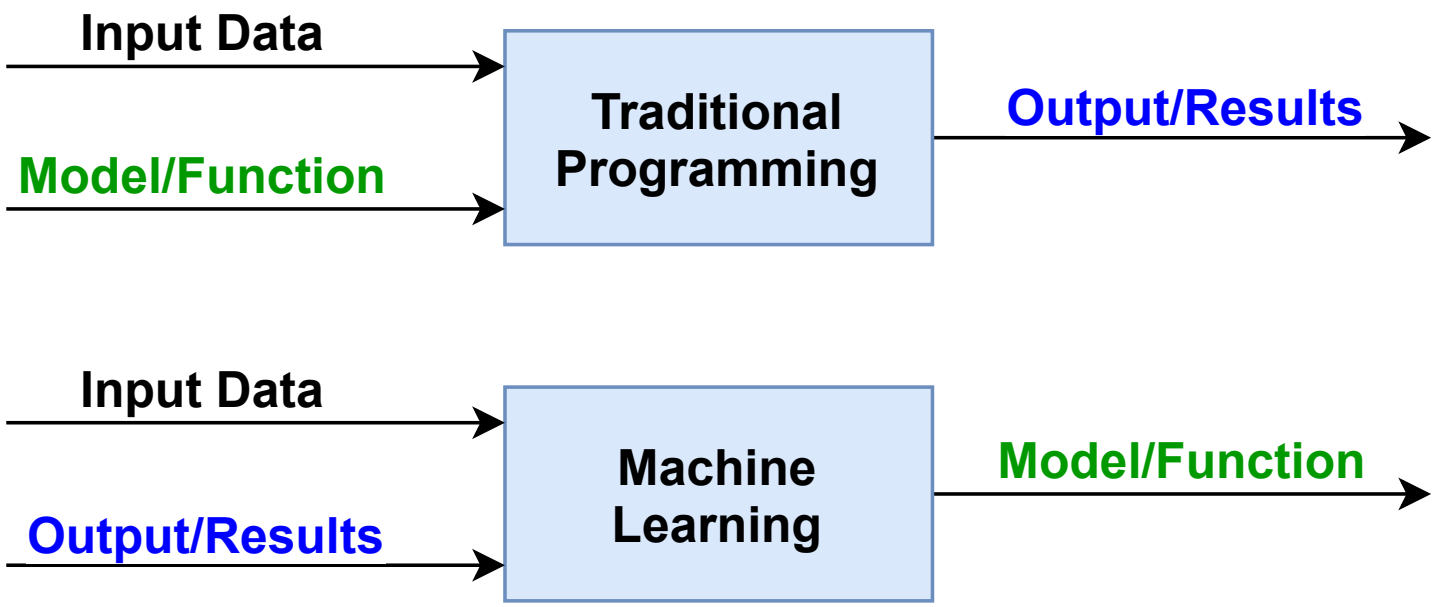
- Students study history syllabus.
- Training data and test data are from the same distribution, if the teacher asks questions from this syllabus.
- If the questions are not from the syllabus - then training and test data do not come from the same distribution.
- Students perform better if the exam paper is based on the syllabus that they study or train on.
- Students do not perform well if questions are NOT from the syllabus.

1 ML vs Traditional Programming

2 Machine Learning

3 Summary

Summary



- Once the model or rule is learnt, traditional programming can be used to get output for a given input label.

Machine Learning Techniques

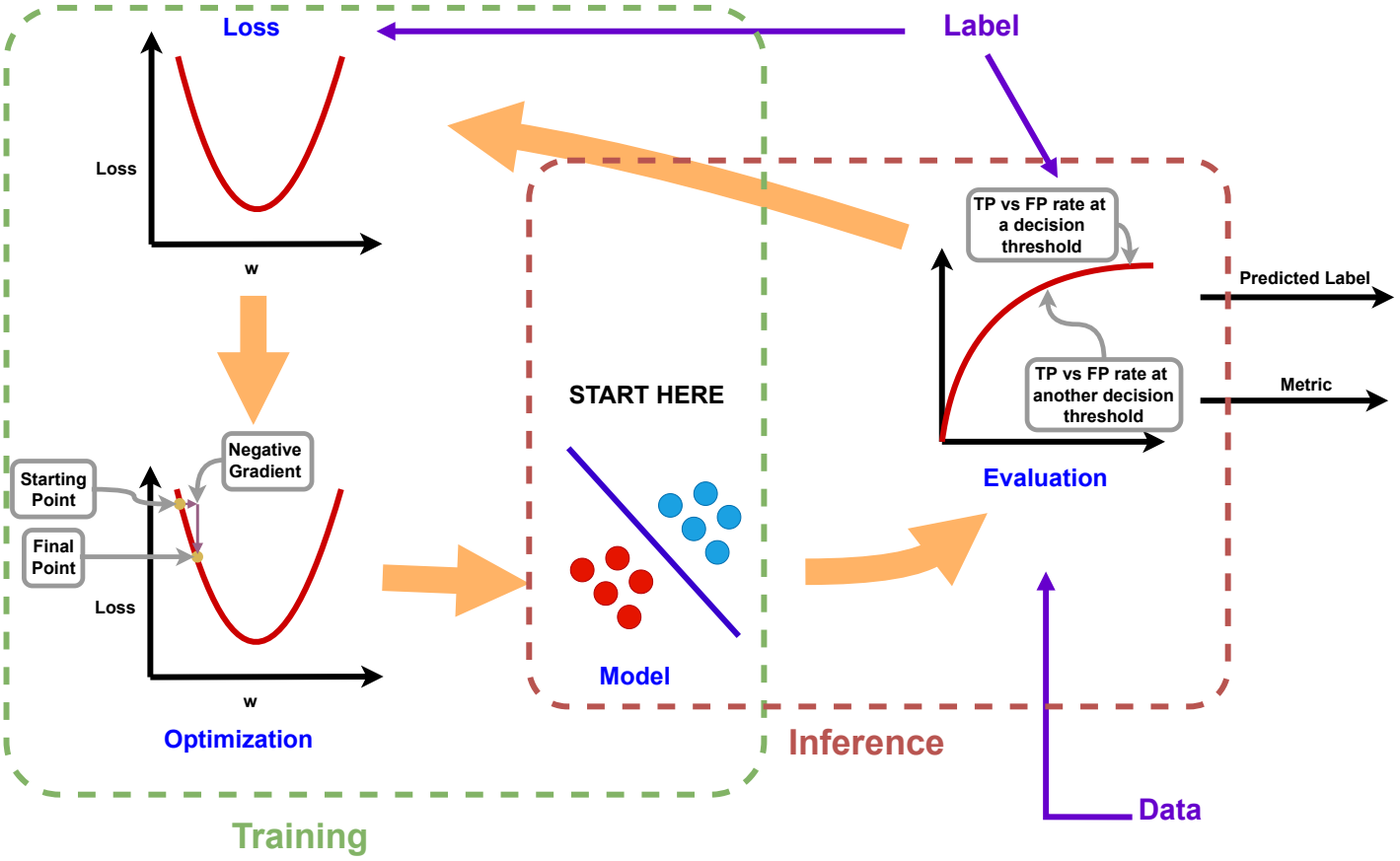
ML Component Framework

Dr. Ashish Tendulkar

IIT Madras

1. ML Component Framework
2. Training Data
3. Model
4. Loss Function
5. Optimization
6. Evaluation
7. Summary

Machine Learning Process



ML Component Framework

- Training data
- Model
- Loss function
- Optimization procedure
- Evaluation criteria

1 ML Component Framework

2 Training Data

3 Model

4 Loss Function

5 Optimization

6 Evaluation

7 Summary

Training Data

- No data, no ML.
- In supervised learning, training data consists of input and output pairs.
- Each input is represented by a bunch of numbers called features or attributes.
- Apply certain transformations to convert input into a bunch of numbers.

Training Data

For example,

- Input may be an image or a piece of text.
- Certain pre-processing steps or transformations have to be applied to obtain feature representation for such input data.

Training Data

- **Features**: are provided by the domain experts.
- For example, while predicting price of a house:
  - The expert would tell us which features are most important in determining the price.
  - The expert would also provide us with a dataset of houses with their features and prices (which is what we are interested in predicting.)
- **Output**: a real number or a discrete value from a predefined set.

Training Data

Example of a real valued label:

- The housing price prediction problem.
- Output label is price, a real number.

Training Data

Examples of a discrete valued label:

- The handwritten digit prediction problem. 10 different values as class labels - digit 0 to 9.
- Loan sanctioning. Binary labels - yes or no corresponding to application being sanctioned or rejected.

Training Data

- Once we have obtained training data, we get an idea about **input** and **output**, which helps us in defining suitable ML problems and choosing appropriate components like **model**, **loss** and **optimization procedure** for training.

1 ML Component Framework

2 Training Data

3 Model

4 Loss Function

5 Optimization

6 Evaluation

7 Summary

Model

- Model provides a mathematical form of mapping between input and output.
- The input is represented with a bunch of features and output can be a real number or a discrete value from some finite set.

Model

Example of a linear model :

- $Output = weight_0 + weight_1 \cdot feature_1 + weight_2 \cdot feature_2 + \cdots + weight_m \cdot feature_m$
- The key problem here is to estimate values of weights.
- All weights together form an entity called weight vector.
- We estimate the weight vector by training the model on the training data.
- The ideal weights are the ones that when used in the model, produce output that is close to the actual output for all training data points.

Model

- Depending on the nature of the output, we choose our model.
- When the **output is a real number**, we choose models of regression - which are capable of producing a real valued output.
- When the **output is a discrete value**, we choose models of classification - which produce a discrete quantity as an output.

1 ML Component Framework

2 Training Data

3 Model

4 Loss Function

5 Optimization

6 Evaluation

7 Summary

Loss Function

- The key objective of training a model is to estimate the weight vector and we need a principled way of doing that.
- We need a suitable method for measuring the difference between predicted and actual output.
- Loss function provides that measure.
- Loss function is a function of weight vector - as we change the weight vector, we obtain a new model, which will have different a loss.

Loss Function

- Denote loss with letter J.
- $J : W \longrightarrow R$
- $J(W)$ = Difference between actual and predicted output for all training samples.
- In the following example model:
  $Output = weight_0 + weight_1 \cdot feature_1 + weight_2 \cdot feature_2 + \cdots + weight_m \cdot feature_m$
- Everything except weight vector is fixed on the right side of the equation- features are specified as part of the training data, which is fixed and weights are variables.
- The weights can be changed to obtain a different result or output.

Loss Function

- An example of loss function is squared loss function: It is calculated as a sum of square of differences between the actual and predicted values.

$$J(W) = \sum_{i=1}^{n} [predicted^{(i)} - actual^{(i)}]^2$$

- Above equation can be simplified to:
$J(W) = \sum_{i=1}^{n} [\{weight_0 + weight_1 \cdot feature_1{}^{(i)} + weight_2 \cdot feature_2{}^{(i)} + \cdots + weight_m \cdot feature_m{}^{(i)}\} - actual^{(i)}]^2$

Loss Function

- Our job is to find the weight vector that results in the lowest loss as per the defined loss function.
- How to estimate such a weight vector?
- Brute force search for the optimal weight vector, but that is not at all efficient.
- What is the best way to estimate it?
- Well that's what is the purpose of our next component: optimization procedure.

1 ML Component Framework

2 Training Data

3 Model

4 Loss Function

5 Optimization

6 Evaluation

7 Summary

Finding Weight Vector

- Obtain a weight vector that minimizes the loss function.
- Formally,

$$W = argmin_W J(W)$$

- Applications of derivatives are typically in 12th calculus in the context of such problems.
- Take a derivative of loss function w.r.t. the weight vector and set it to 0.

$$\frac{d}{dW} J(W) = 0$$

- Solve this equation directly or with analytical methods to obtain the optimal weight vector.
- The optimization procedure is a cornerstone of model training.

1 ML Component Framework

2 Training Data

3 Model

4 Loss Function

5 Optimization

6 Evaluation

7 Summary

Evaluation

After optimization:

- How to evaluate the model on unseen data?
- What metric to use?
- The evaluation metric changes as per the problem - different metric for regression and classification problems.

Evaluation

- What if the model does not perform as per the expectation as found via evaluation metric?
- Reiterate the ML pipeline.
- Try to add superior
  - features
  - models
  - loss function
  - optimization procedure

1 ML Component Framework

2 Training Data

3 Model

4 Loss Function

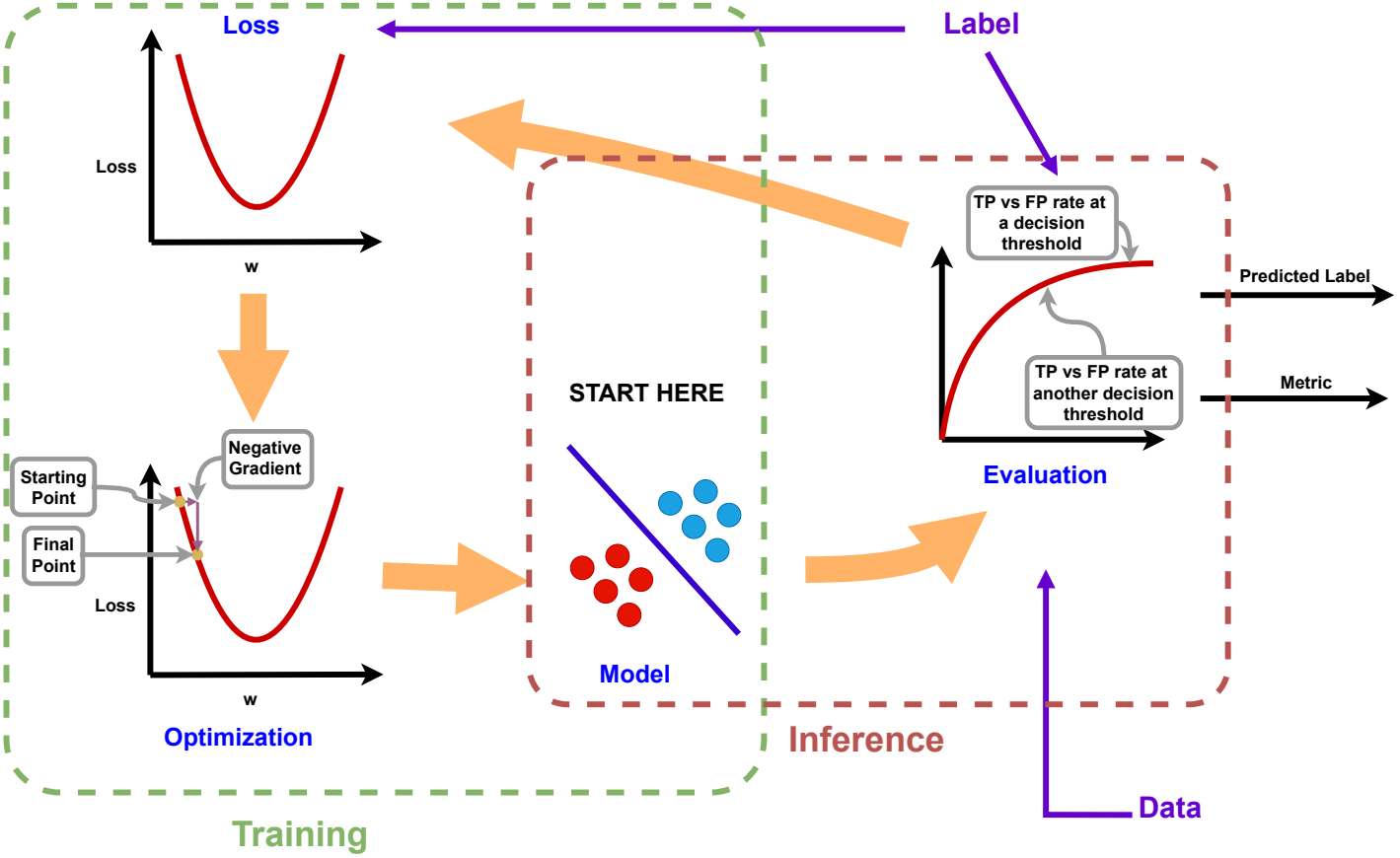
5 Optimization

6 Evaluation

7 Summary

Summary

Machine Learning Process



Machine Learning Techniques

A demo of ML Component Framework with Tensorflow Playground

Dr. Ashish Tendulkar

IIT Madras

1. Overview of TensorFlow Playground
2. Data
3. Model
4. Loss Function
5. Optimization Procedure
6. Evaluation

A demo of Component Framework

- A short demonstration of the component framework.
- TensorFlow playground
  Link: https://playground.tensorflow.org/
- It is built in Javascript.

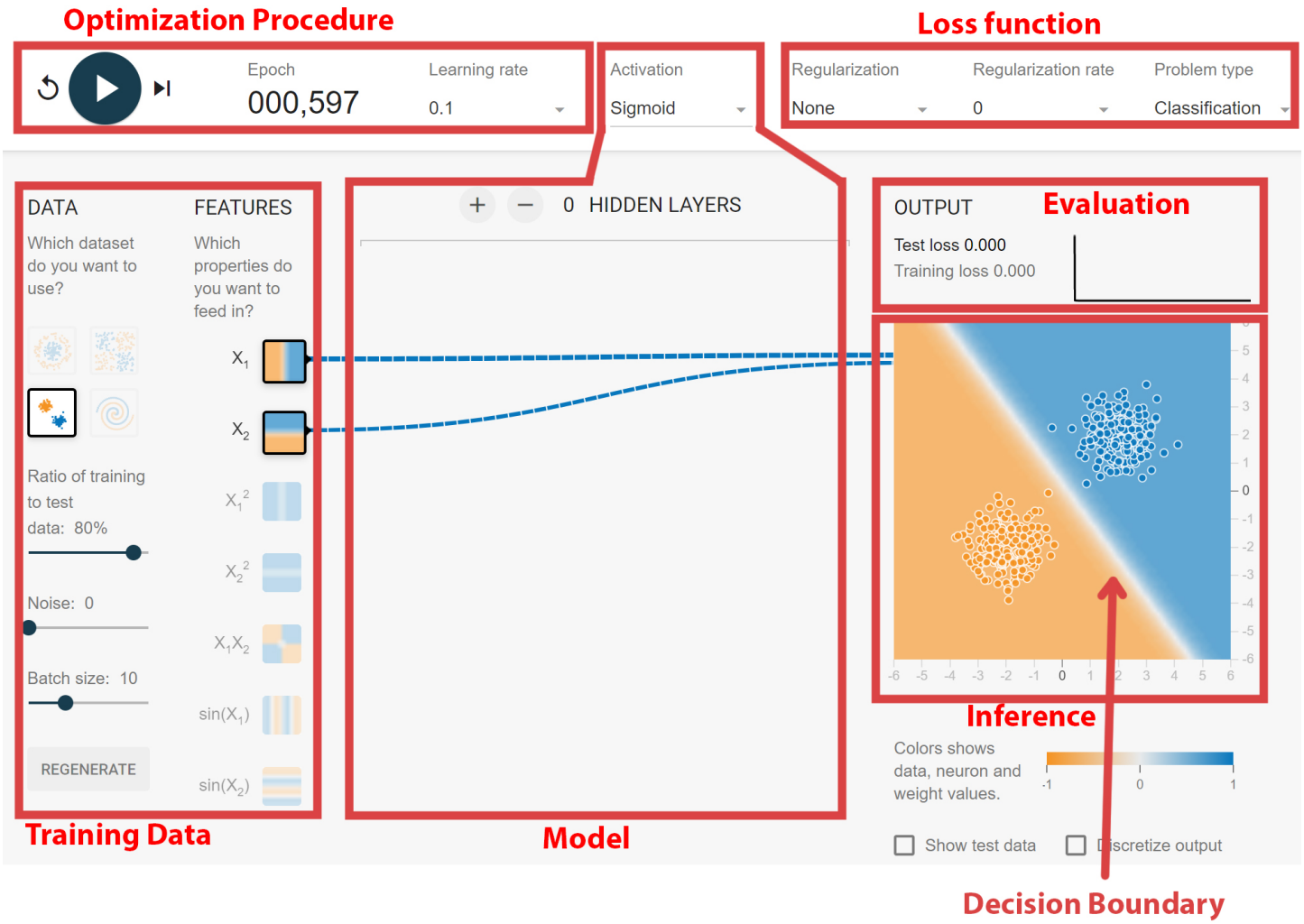
TensorFlow Playground

- It provides a way to visualize ML training.
- It allows changing different components like training data, model, loss function (to some extent) and parameters of the optimization algorithm.
- Evaluation protocol is fixed.

A tour of TensorFlow Playground

- The playground is mainly meant for experimenting with neural networks, but we will repurpose it for understanding various important concepts in ML.
- It will be used to highlight components of the ML framework and visually demonstrate how ML works.
- Play with it freely without fear of breaking anything down!

Overview: Tensorflow Playground



1 Overview of TensorFlow Playground

2 Data

3 Model

4 Loss Function

5 Optimization Procedure

6 Evaluation

Training Data

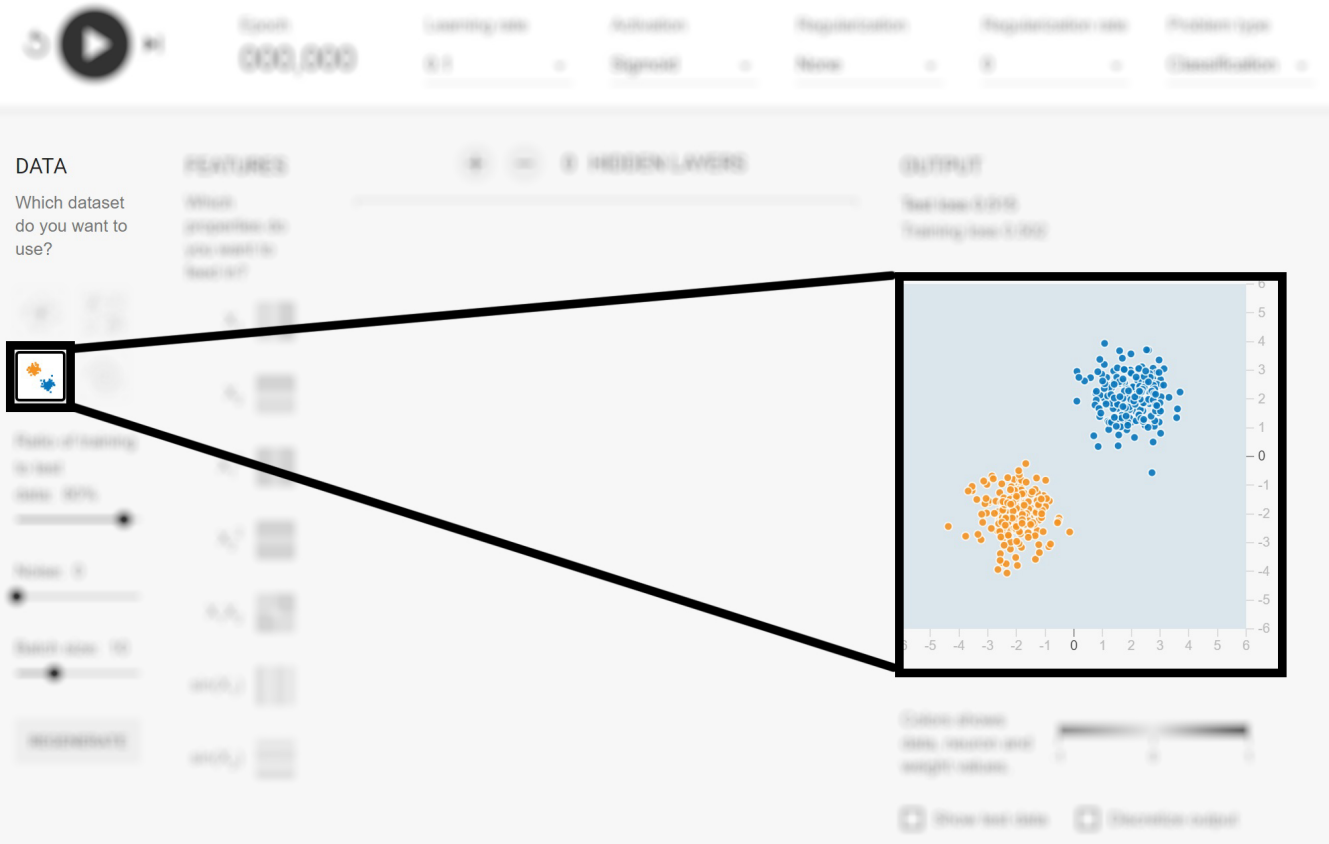
- The most important ML component: data.
- No data, no ML.
- In TensorFlow playground, we have data with two classes: positive and negative classes.
- The examples from the positive class are represented with blue color and the ones from the negative class are represented with orange color.

Data

The playground provides different permutations of training data.

Data

Linearly separable classes.



Data

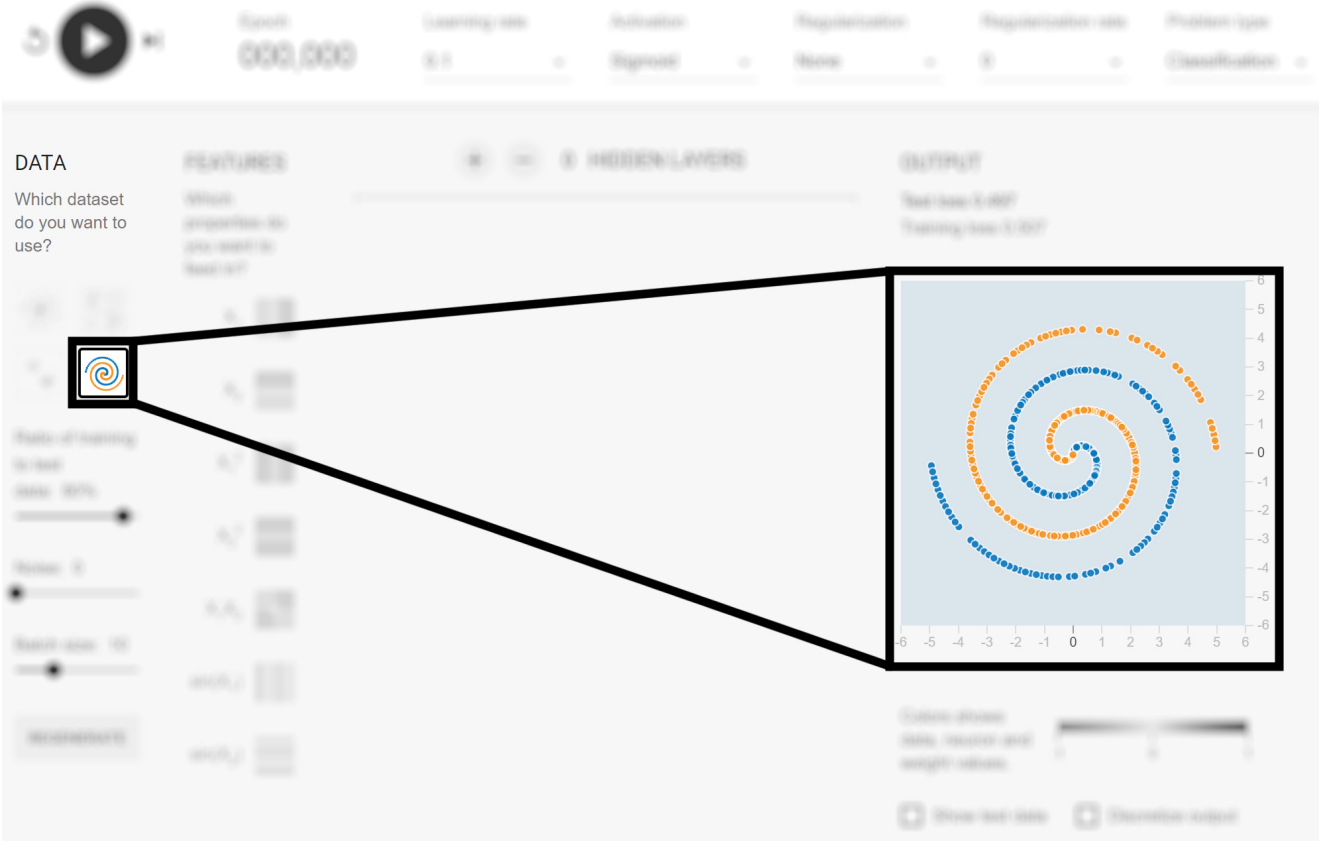
Classes separable by a circle.

Data

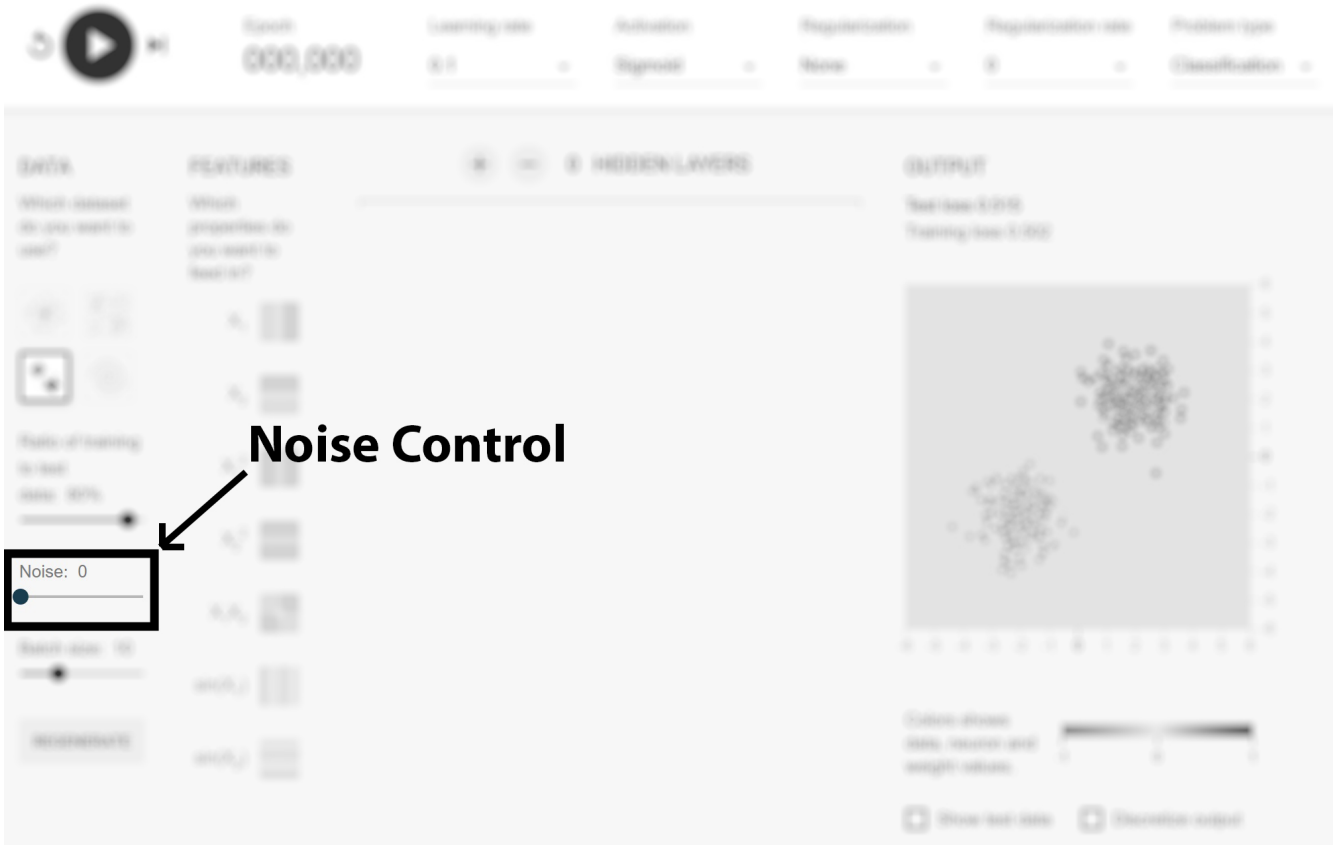
Classes arranged in an oblique manner - like x-or.

Data

Classes separable along a curvilinear path



Noise controller



Data

- Each data point has two features (x_1, x_2)
- Training set: 80%
- Test set: 20%

1 Overview of TensorFlow Playground

2 Data

3 Model

4 Loss Function

5 Optimization Procedure

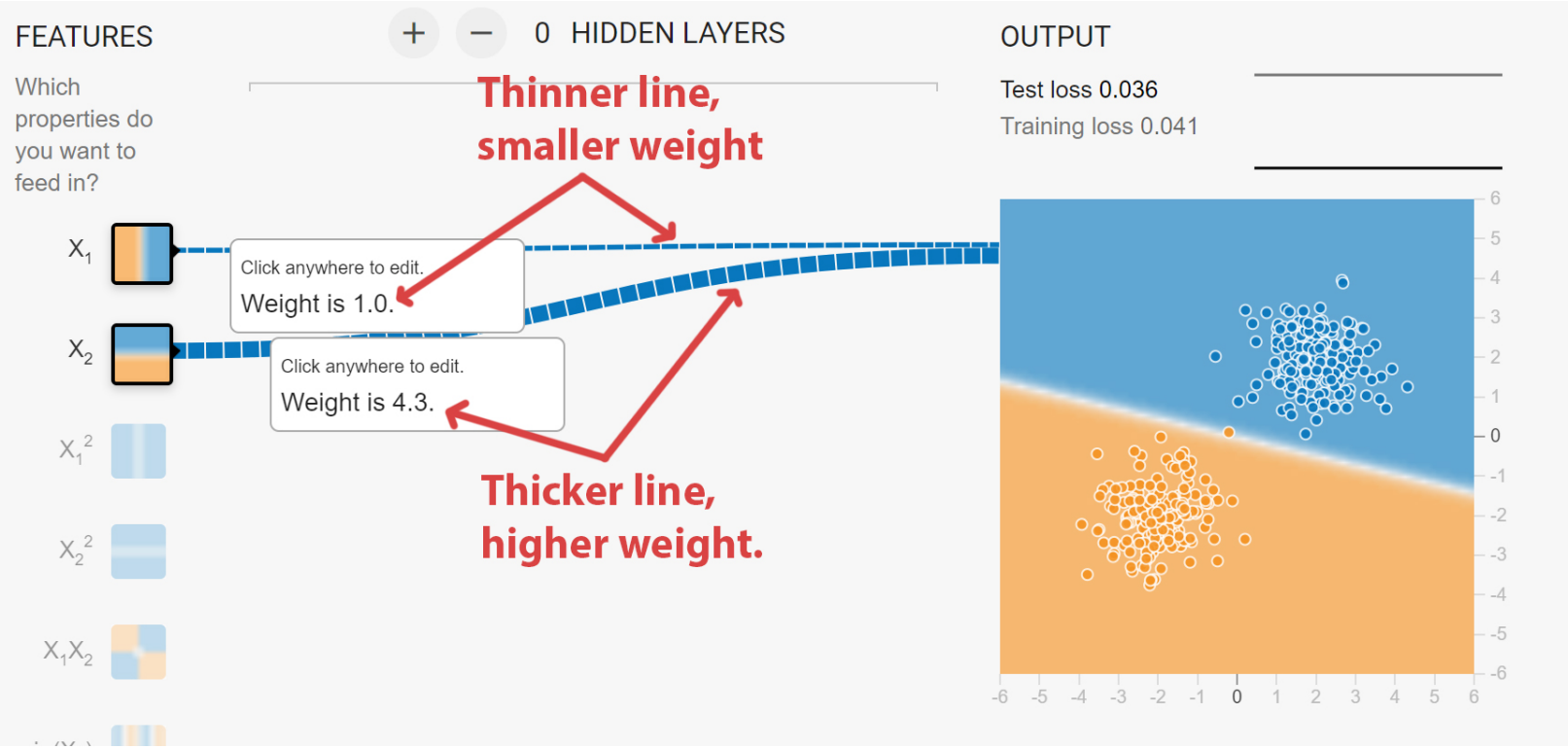
6 Evaluation

Model

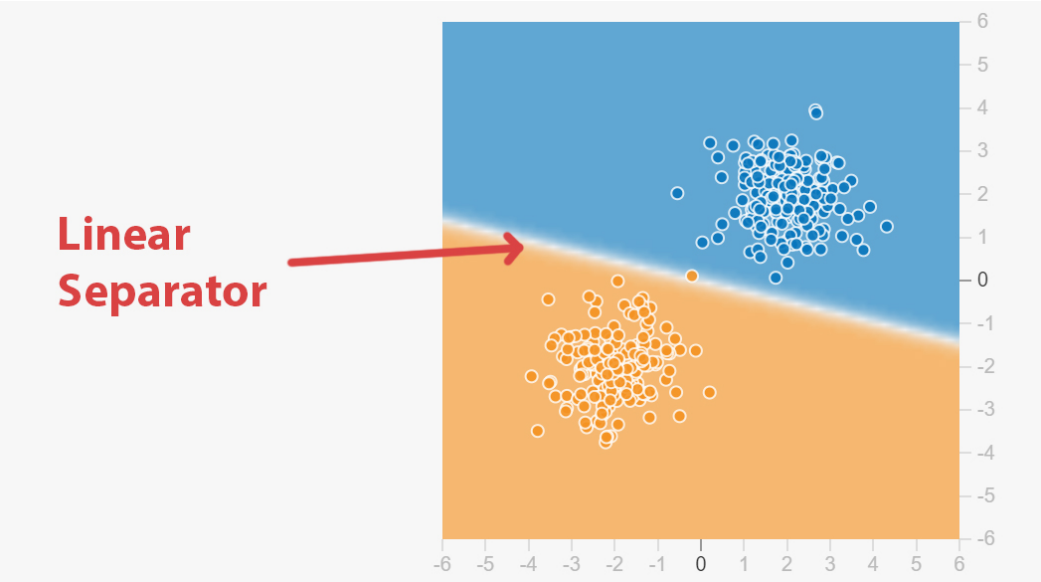
- Logistic regression
- Train model to map features to classes.
- Learn weights of the features.

Model

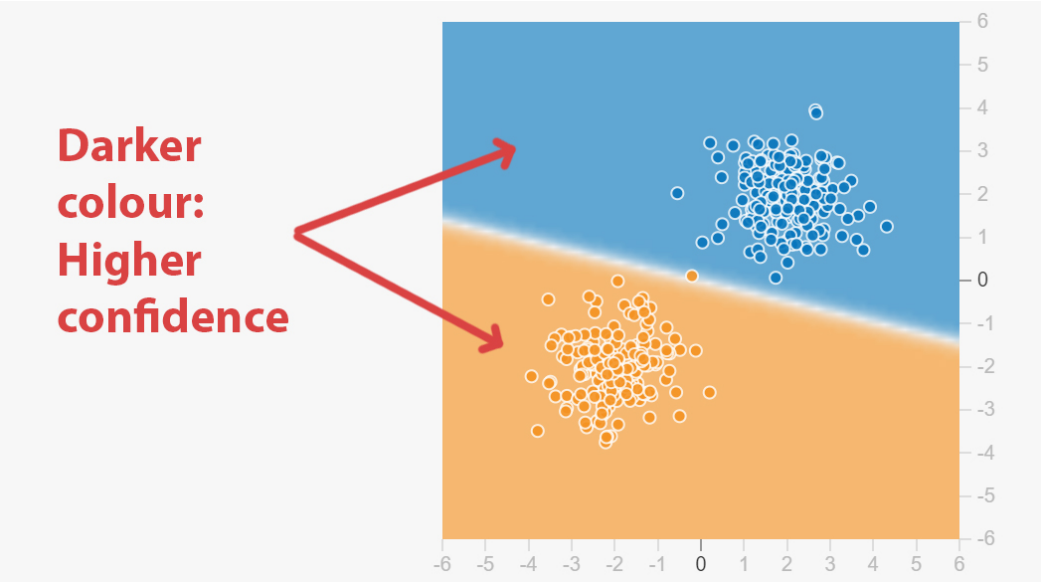
Hover over the line to see weights.



Model



Model


Darker colour: Higher confidence
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6

1 Overview of TensorFlow Playground

2 Data

3 Model

4 Loss Function

5 Optimization Procedure

6 Evaluation

Loss function

- Can't explicitly specify loss function in Tensorflow Playground.
- It changes with the type of problem.
- For classification 'binary cross entropy' loss function is used.
- Regularization: its type and rate can be adjusted.

1 Overview of TensorFlow Playground

2 Data

3 Model

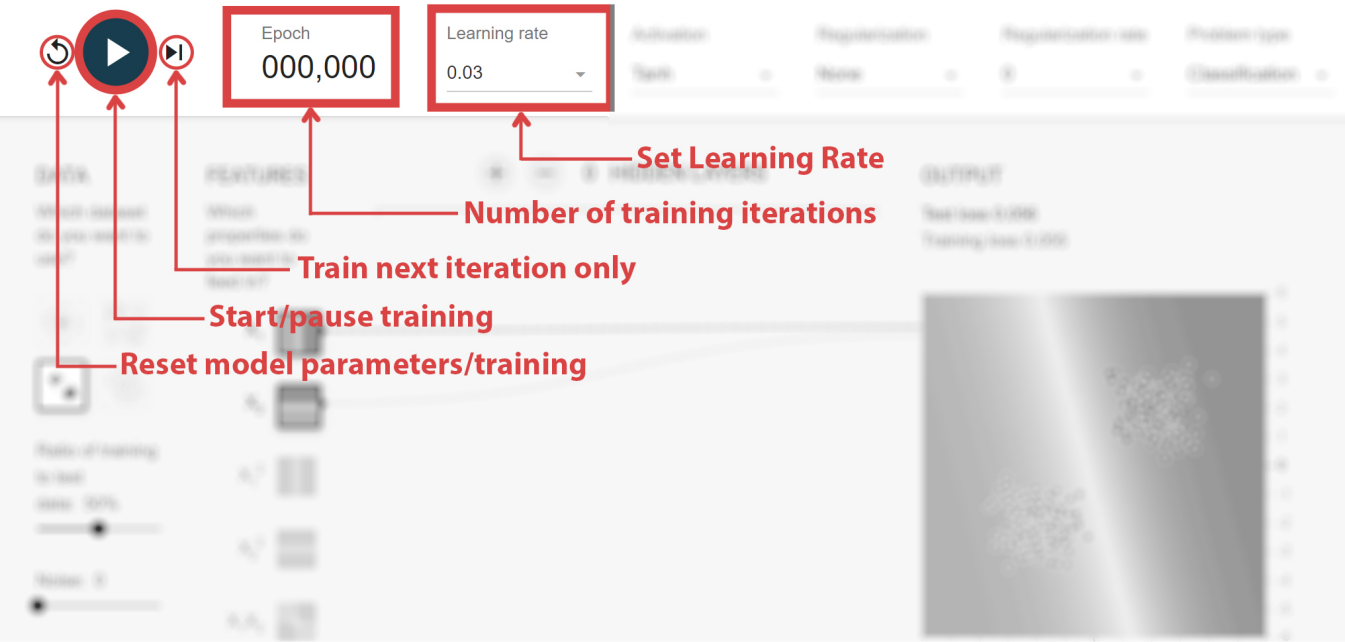
4 Loss Function

5 Optimization Procedure

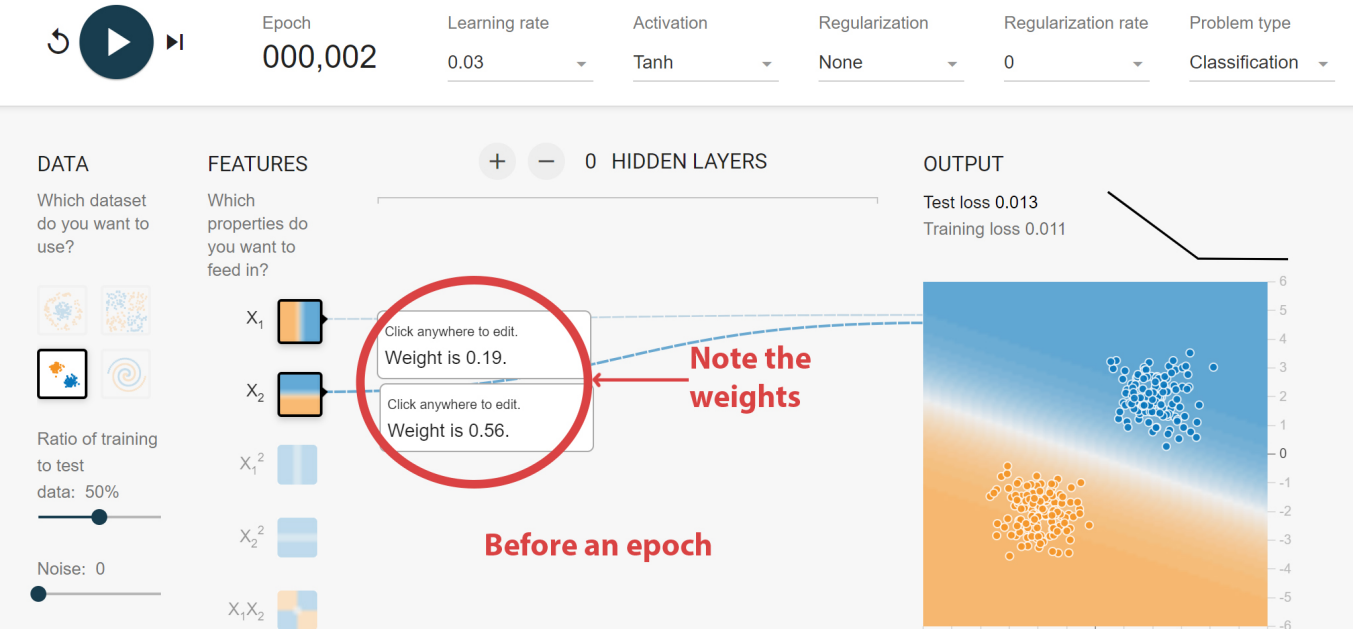
6 Evaluation

Optimization Procedure

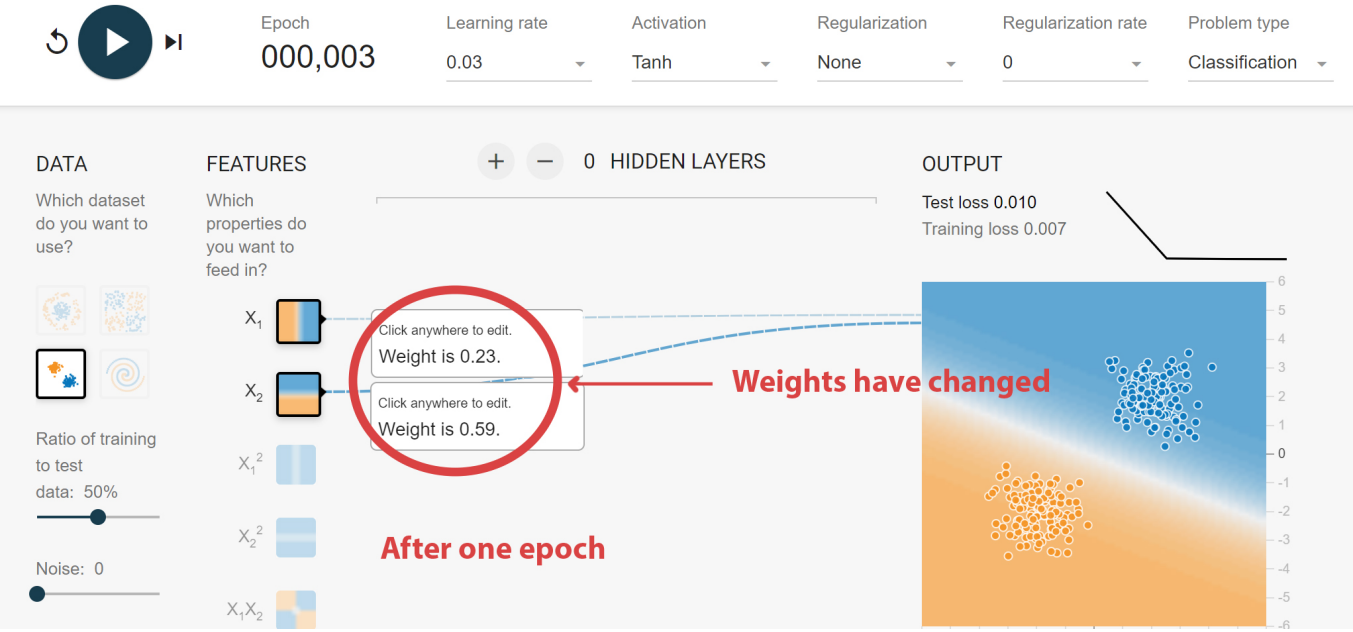
- Batch-gradient descent for optimization.



New Model Instance per epoch



New Model Instance per epoch



1 Overview of TensorFlow Playground

2 Data

3 Model

4 Loss Function

5 Optimization Procedure

6 Evaluation

Evaluation Procedure

- Evaluate the goodness of the model with an evaluation protocol.
- The playground uses only training and test loss for evaluation. There is no scope for any customization or selection.
- As the training proceeds, we get to see changes in the test and training loss.

Evaluation Procedure

When the model is getting trained:

- Both the losses should gradually decrease.
- The rate of decrease is proportional to the learning rate.

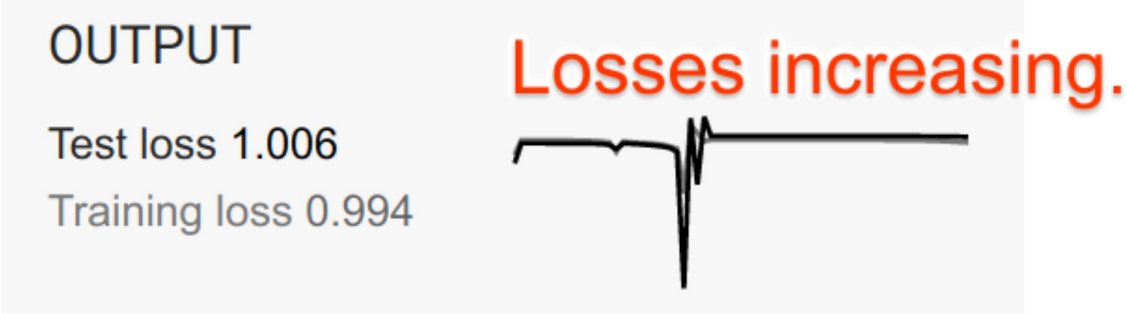
Evaluation Procedure

When the model is getting trained:

- Both the losses should gradually decrease.

- The rate of decrease is proportional to the learning rate.

Evaluation Procedure



If the losses are not going down, that means the model is not training, possibly because:

- The learning rate is too high, or
- There are no informative features describing the training examples.

Evaluation Procedure

If the model is getting trained properly:

- Both the training and test losses go down.
- Both of them should be close to 0.

A tour

Let's take a tour of TensorFlow Playground now.

Machine Learning Techniques

Taxonomy of ML Techniques

Dr. Ashish Tendulkar

IIT Madras

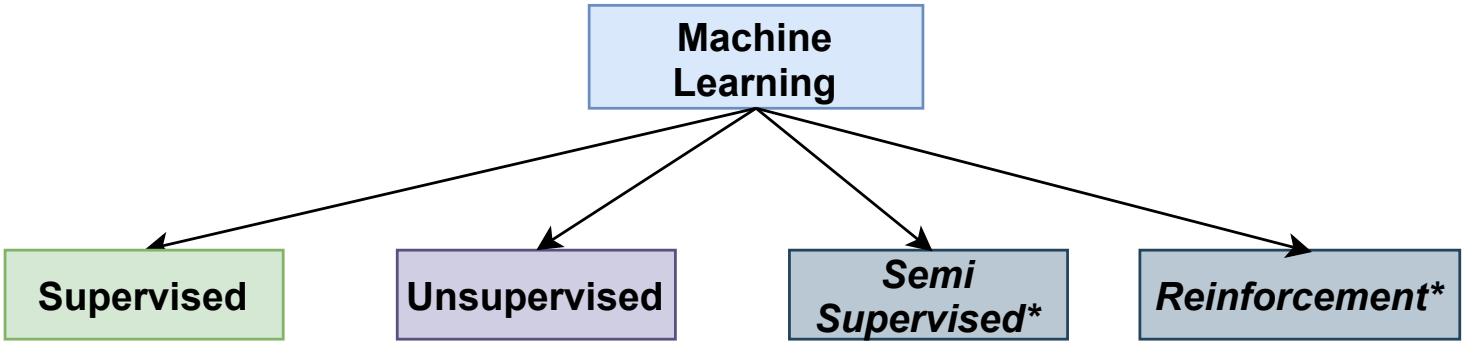
1. Taxonomy of ML Techniques
2. Supervised Learning
3. Unsupervised Learning

The ML techniques can be categorized based on the following criteria:

- Labels
- Models
- Learning styles

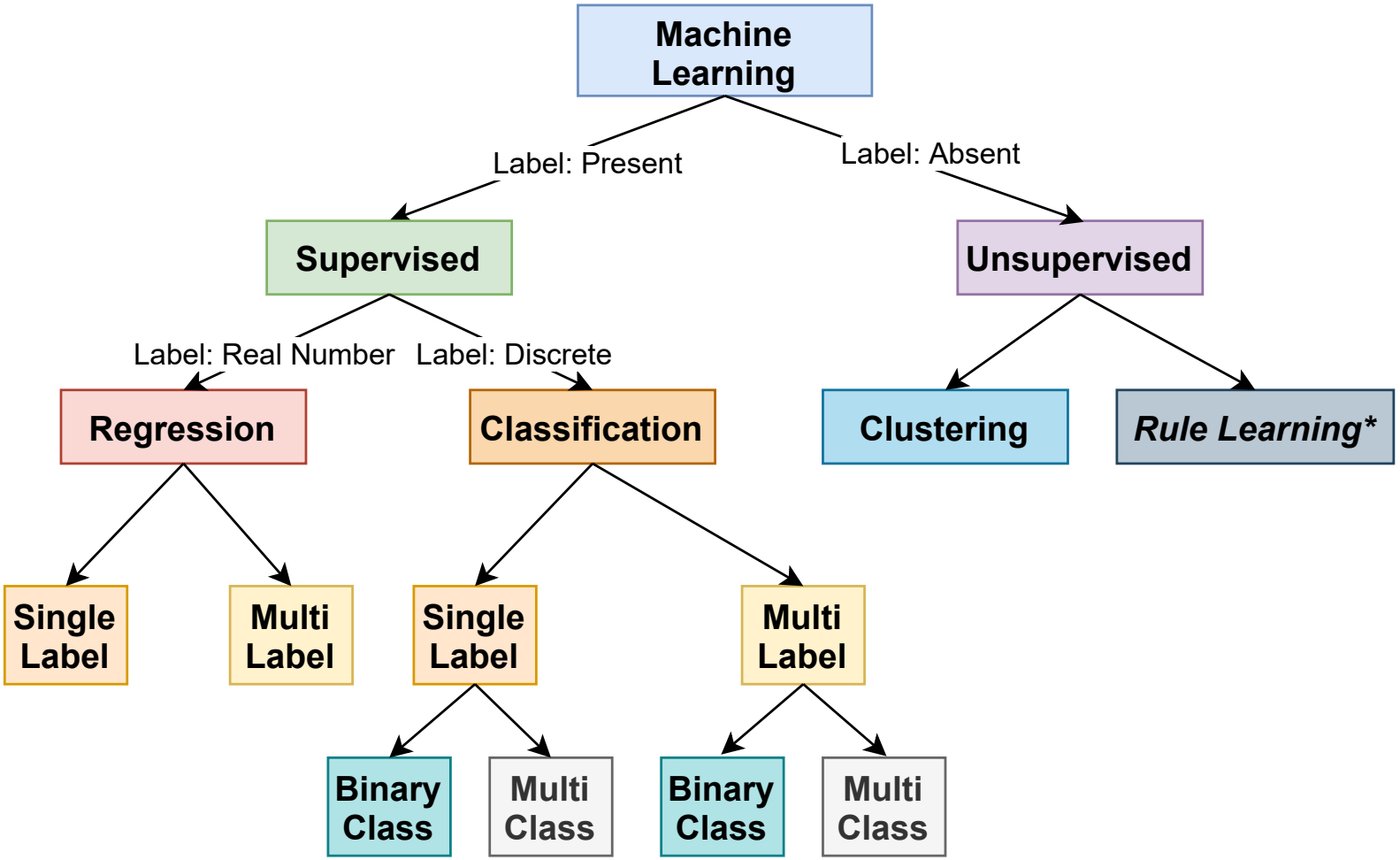
ML Techniques in a picture

Broad categories



* Not covered in this course

ML Techniques in a picture



* Not covered in this course

Is Label present?

Based on the presence or absence of label, ML techniques are divided into two types:

- **Supervised**: labels are present.
- **Unsupervised**: labels are not present.

1 Taxonomy of ML Techniques

2 Supervised Learning

3 Unsupervised Learning

Types of Supervised Learning

The supervised techniques are again divided into two types:

- **Regression**: the label is a real number.
- **Classification**: the label is a discrete quantity from some finite set.

Supervised Learning

Based on the number of labels, we get further division:

- **Single label**
- **Multiple labels** - multi-label classification/regression

Supervised Learning

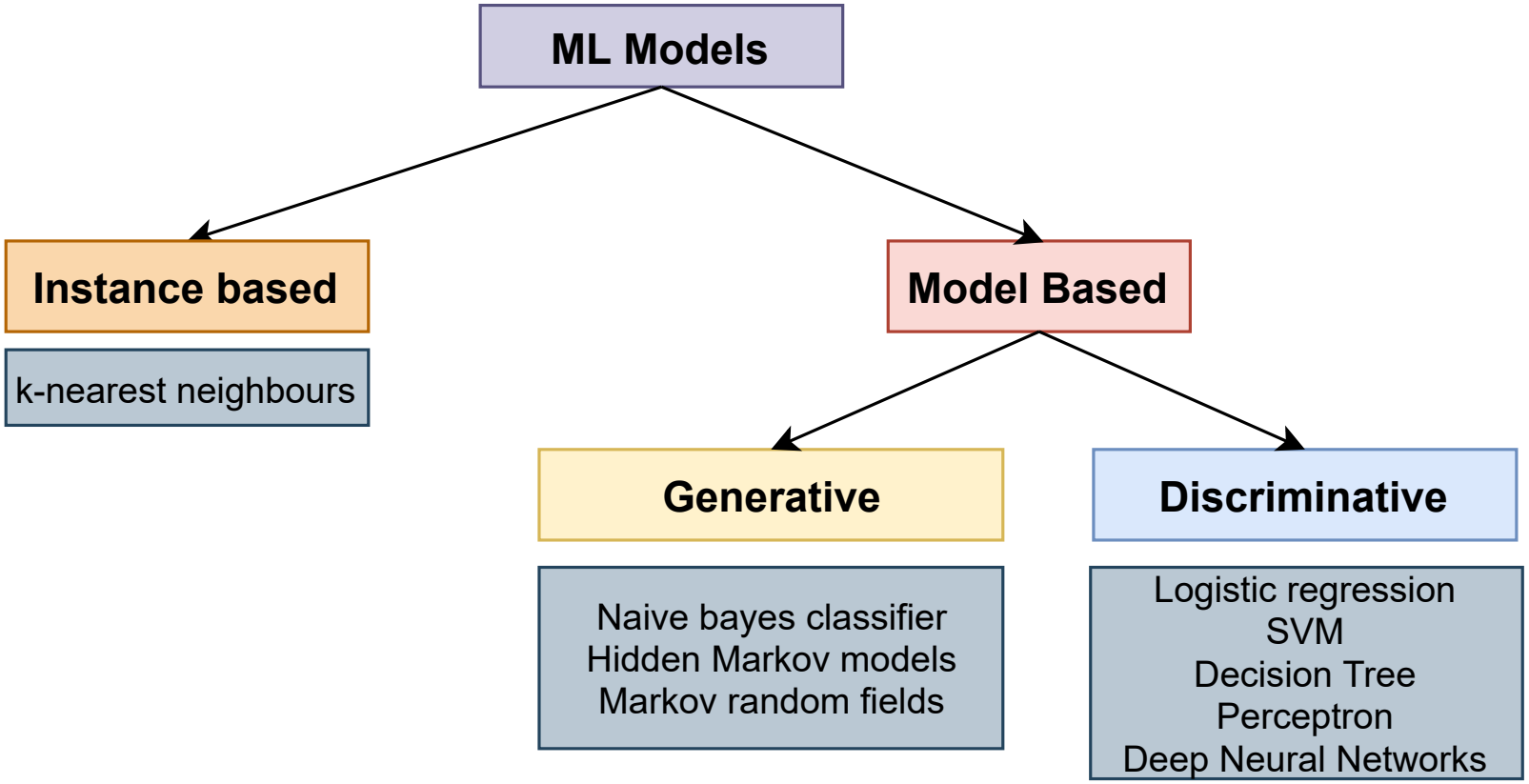
Based on the number of labels in the label class:

- **Binary class**: the label is either positive or negative.
- **Multi-class**: there are more than two possibilities of labels.

Examples

Type	Example	Label Characteristics
Single label regression	Prediction of price of a house	Single real number
Multi-label regression	Predict stock price for next 5 days	Multiple real numbers
Single label classification	Predict if the online transaction is fraudulent transaction	Single binary label- Fraudulent transaction: 1 Genuine transaction: 0
Multi-label classification	Label all fruits in the image	Multiple possible fruits, which are labels

Model Based Taxonomy



Model based techniques

Model based techniques: They learn a function mapping between input and output with a concrete mathematical function.

They are further divided as:

- Generative models.
- Discriminative models

Generative models

Generative models: They use probability distributions for modeling the underlying data generation process and learn the parameters of distribution based on training examples.

Examples:

- Naive Bayes classifier
- Hidden Markov Model
- Markov random fields

Discriminative models

Discriminative models: they learn separators between classes, instead of learning the underlying data distributions, based on the training examples.

Examples:

- Logistic regression.
- SVM, Decision Tree.
- Neural network.

Instance-based techniques

Instance-based techniques: They store all the examples and rely on the notion of similarity between test and training examples for label assignment.

Examples:

- k-nearest neighbor classification/regression

Learning Style based taxonomy

ML Techniques can be categorized into following based on learning style:

- Batch learning
- Online learning

This will be covered in detail in this course later.

1 Taxonomy of ML Techniques

2 Supervised Learning

3 Unsupervised Learning

Unsupervised Learning

Following are two types of unsupervised learning:

- Clustering
- Rule Learning

Machine Learning Techniques

ML Component Framework

Dr. Ashish Tendulkar

IIT Madras

1. Training Data
2. Model
3. Loss Function
4. Optimization Procedure
5. Evaluation

Data

- Training data in form of features and labels (if present).
- Training data is represented by $\boldsymbol{D}$.
- Each example has a feature vector associated with it and is denoted by $x^{(i)}$. Since it is a feature vector, it is represented with a bold face:

$$\boldsymbol{x} = [feature_1, feature_2, \ldots, feature_m]$$

- A label (for single label problem) - $\boldsymbol{y}$ or a list of labels for a multi-label problem, denoted by a vector $\boldsymbol{Y}$.
- $\boldsymbol{D}$ is the set of n training examples: a set of ordered pairs **(features, labels)**: $\boldsymbol{D} = \{(x^{(i)}, y^{(i)})\}$.

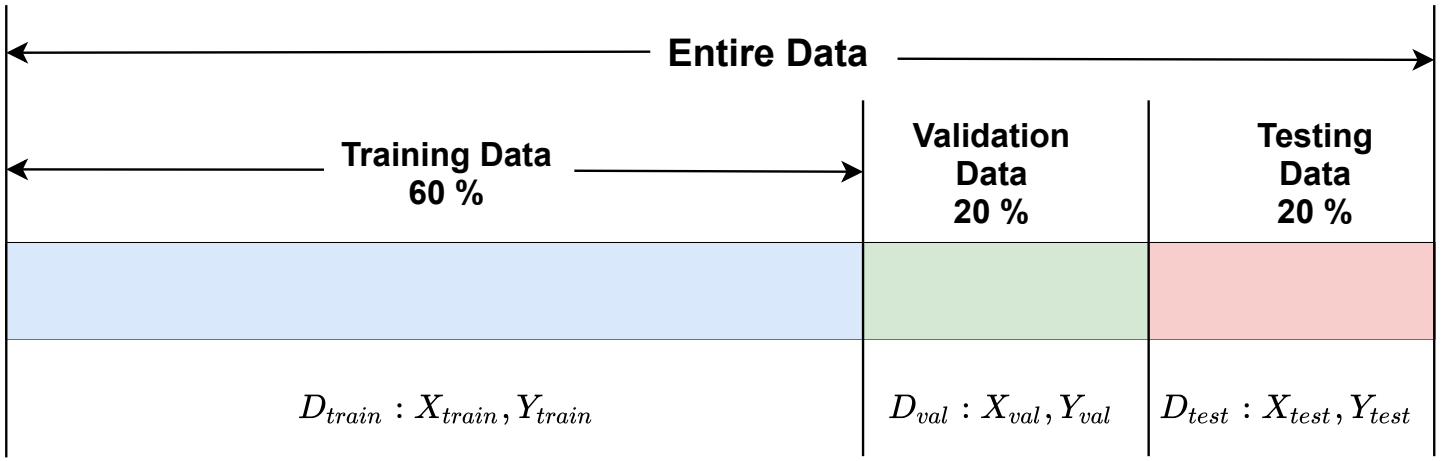
Data

- Vectorized implementation of algorithms, the features from all examples are represented as a feature matrix: $\mathbf{X}$,

$$\mathbf{X} = [feature^{(1)^T}, feature^{(2)^T}, \ldots, feature^{(n)^T}]$$

- The labels are represented either as a label vector (for single label problems) or a label matrix (for multi-label problems) - label(s) for i^{th} example is on i^{th} row.

Dataset division



Dataset division

A dataset can be divided into following:

- **Training data** is used for training the model
- **Validation data** is used for tuning any hyper parameters, e.g. regularization rates.
- **Test data** is never used for training the model and is only used for evaluation of model performance.

Cross validation

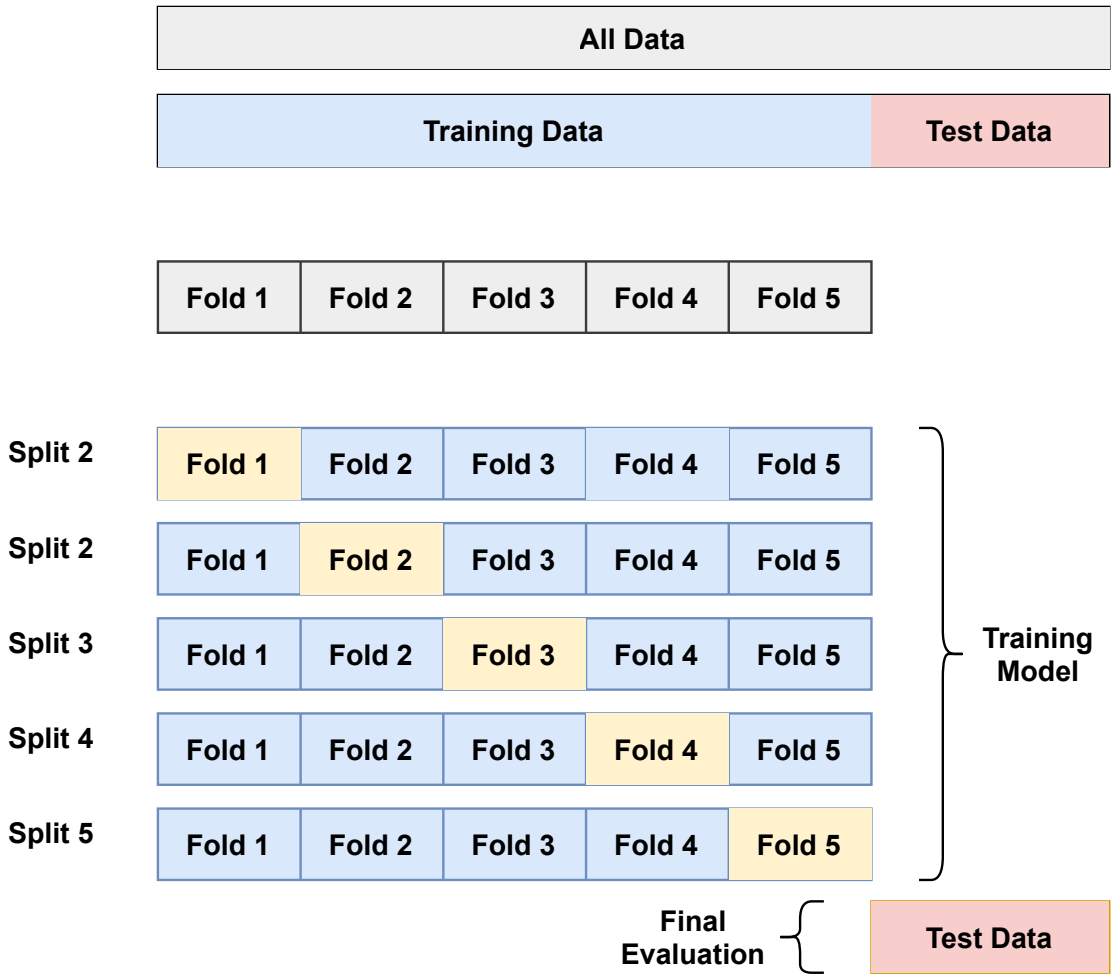
Cross-validation:

- Instead of using just a fixed set of examples for training and validation - different sets are used for training and validation.

K-fold Cross Validation (CV)

1. Partition the training set into k equal partitions.
2. Use $k-1$ partitions for training and the remaining one for validation.
3. The extreme form of this is $n-1$ fold CV, where each example goes into a separate partition, thus creating n partitions for training set with n examples.
4. Use $n-1$ examples for training and one example for validation.
5. Train the model k times and hence evaluate the metric also k times on the validation.
6. Compute evaluation metrics by averaging across k training/validation runs.
7. Select the best out of these models and report their metric on the test set.

5 Fold CV


All Data
Training Data
Test Data
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Split 2
Split 2
Split 3
Split 4
Split 5
Training Model
Final Evaluation
Test Data

Preprocessing

- The continuous or real valued features need to be normalized. This leads to faster convergence.
- The discrete attributes are converted to numbers by:
  - One-hot-encoding
  - Hashing
  - Embeddings

Relationship between features and labels

To examine the relationship between features and labels:

- Use data visualization techniques such as histograms, pair plots, check if the features are correlated to the labels:
  - If the features contain some information to predict the outcome or the label.
  - If effective features are not present, a useful model cannot be learnt.
- Statistical tests like the chi-square test.

Handling missing values

If dataset has missing values:

- Remove examples containing missing values.
  - Some examples are lost.
- Compute average of the feature vector and fill the missing values.

Class imbalance problem

Class imbalance problem:

- Too many training examples from one class and too few examples from the other class.

Solution:

- Oversample examples from a minority class, the one which has too few examples and make the training set balanced in terms of examples from different classes.

1 Training Data

2 Model

3 Loss Function

4 Optimization Procedure

5 Evaluation

Model

- We have a training set in terms of feature matrix $\boldsymbol{X}$ and a label vector $\boldsymbol{y}$ for single label and $\boldsymbol{Y}$ - a label matrix for multi-label problems.
- h_{w} : the model
  w: a weight vector of the model.
- The model maps the input feature vector to the output label:

$$h_w : \boldsymbol{X} \rightarrow \boldsymbol{y}$$

1 Training Data

2 Model

3 Loss Function

4 Optimization Procedure

5 Evaluation

Loss Function

- Loss function: Measurement of difference between the actual and the predicted labels.
- The loss function is represented by $\boldsymbol{J}$ and is a function of weight vector.

$$J(w) = f(y, \hat{y})$$

$\hat{y}$: predictions
y : actual labels.

- The predicted label depends on the model parametrized by the weight vector w and hence the loss function is a function of the weight vector.
- Every time the weight vector is changed, we get a new instance of the model, which leads to different predictions and hence causes a change in the loss.

Loss Function

- The loss function can be convex or non-convex depending on the model function class.
- Linear/logistic regressions, Support Vector Machines have convex loss functions, while the loss functions for neural networks are non-convex in nature.
- Convex loss functions are easier to optimize than the non-convex ones.

1 Training Data

2 Model

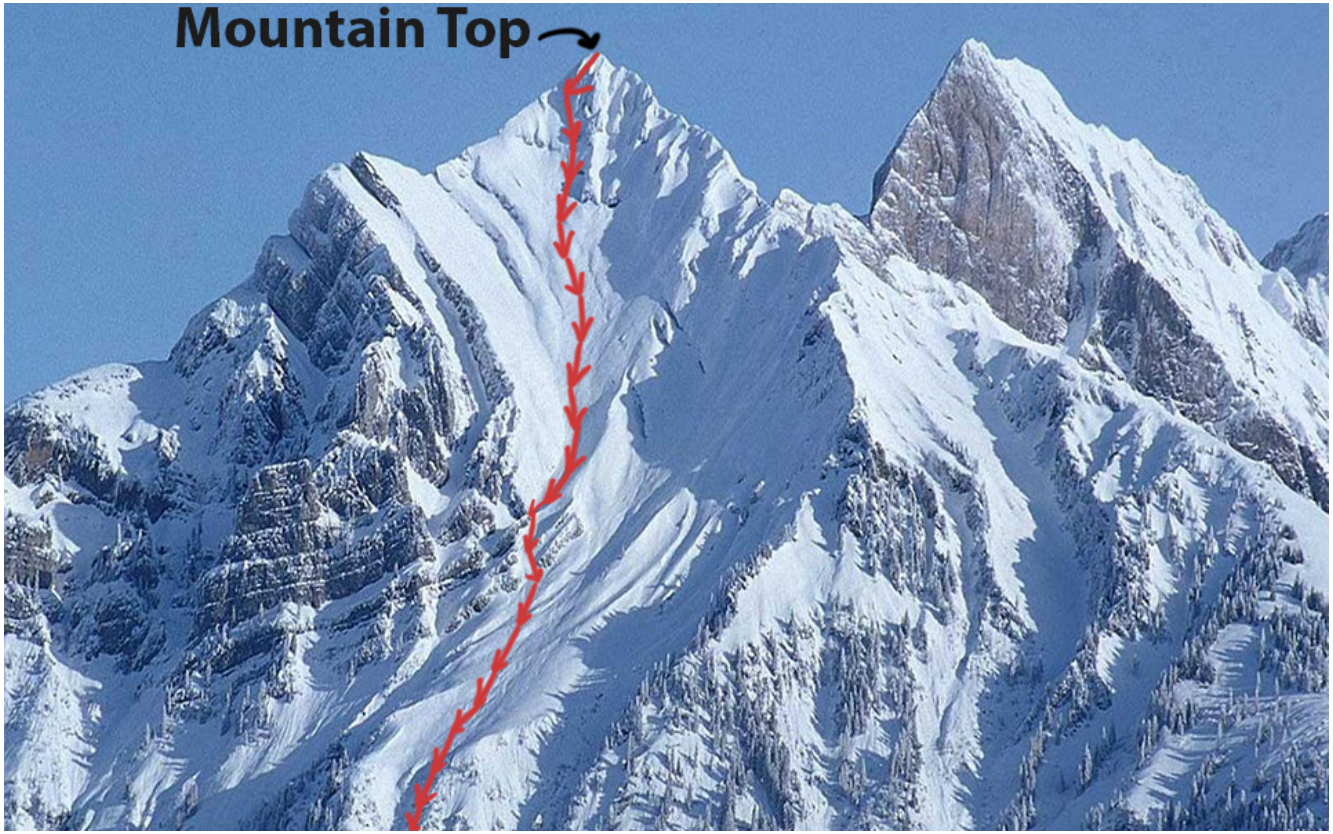
3 Loss Function

4 Optimization Procedure
- Gradient Descent
- Mini Batch Gradient

5 Evaluation

Optimization Procedure

Use gradient descent for optimizing the loss function w.r.t. values of weight vector.



Gradient Descent: Mathematical Framework

- Derivatives are used to calculate the direction of steepest descent.
- Derivative is calculated w.r.t. weight vector - specifically partial derivatives of loss function w.r.t. each weight and use it to make a transition to the new point.

Gradient Descent: Mathematical Framework

i^{th} weight *after* update

i^{th} weight *before* update

Derivative of loss function w.r.t i^{th} weight

$$w_i[t+1] \longleftarrow w_i[t] - \alpha \cdot \frac{\partial J(w)}{\partial w_i[t]}$$

Learning Rate

Gradient Descent: Algorithm

1. Randomly initialize the weight vector.
2. Repeat:
    1. Look around and get the direction of steepest descent.
    2. The next step deals with updating the weight - again this is similar to moving to the next point.
    3. Take steps proportional to the derivative value - use a special constant called learning rate that controls the step size. We do not want to blindly follow this path forever and learning rate helps to control that tendency.
    4. Note that we calculate the next position, but do not update it instantly. We first calculate these positions for all weights.
    5. At the end, all the weights are updated to their new values.

Gradient Descent can be time consuming

- The partial derivative calculation is performed based on the loss incurred at each training example due to choice of the specific weights.
- This can be the most time consuming step for training sets containing millions of examples.
- **Can the model be trained faster?**

1 Training Data

2 Model

3 Loss Function

4 Optimization Procedure
- Gradient Descent
- Mini Batch Gradient

5 Evaluation

Mini Batch Gradient Descent

- Can we calculate the partial derivatives on a small number of examples and use it to update the weights?
- **Mini-batch gradient descent** - a small number of examples are used for calculating the weight updates.
- The original version we studied is called **batch gradient descent**, where weight update is calculated based on partial derivative calculation over the entire example set.

Mini Batch Gradient Descent

- In mini-batch gradient descent, the training set is divided into multiple batches of a fixed size and at a time one batch is used to calculate the weight updates.
- Once we complete all the batches, we complete what is called **a single epoch** or **a full iteration** of the training set.
- A single epoch contains multiple iterations of weight updates in a mini-batch gradient descent.
- The extreme form of this is when we use exactly one example to carry out weight updates and that is called **stochastic gradient descent** or **SGD**.

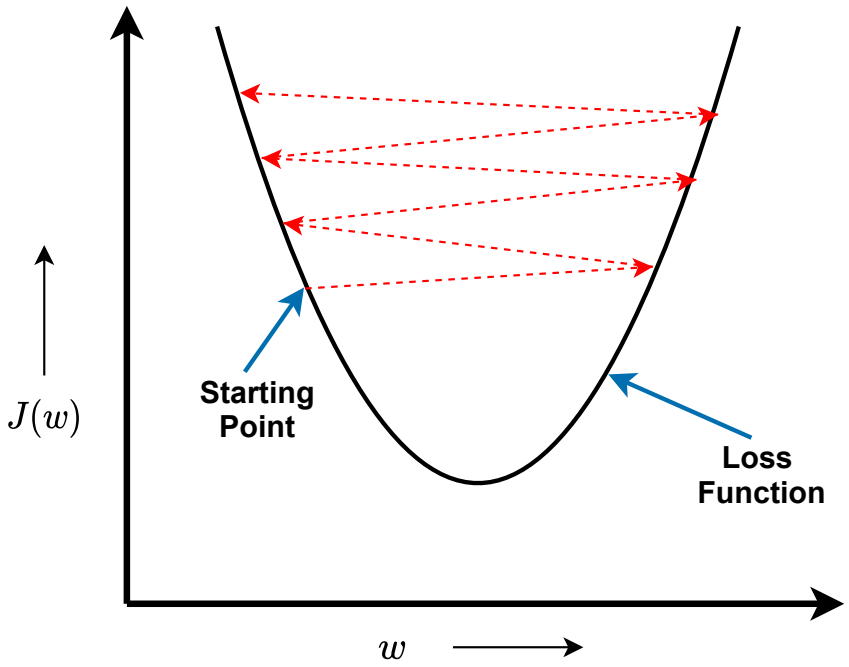
GD vs Mini batch GD and SGD

- Everything except the gradient calculation is exactly the same as batch gradient descent.
- While calculating the gradient, we iterate over examples in the given batch and use it for the weight update.
- SGD and mini-batch have the same code - they only differ in the batch sizes.
- Choose batch sizes that are power of 2 to attain optimal disk read and hence optimal IO performance.

GD vs Mini batch GD and SGD

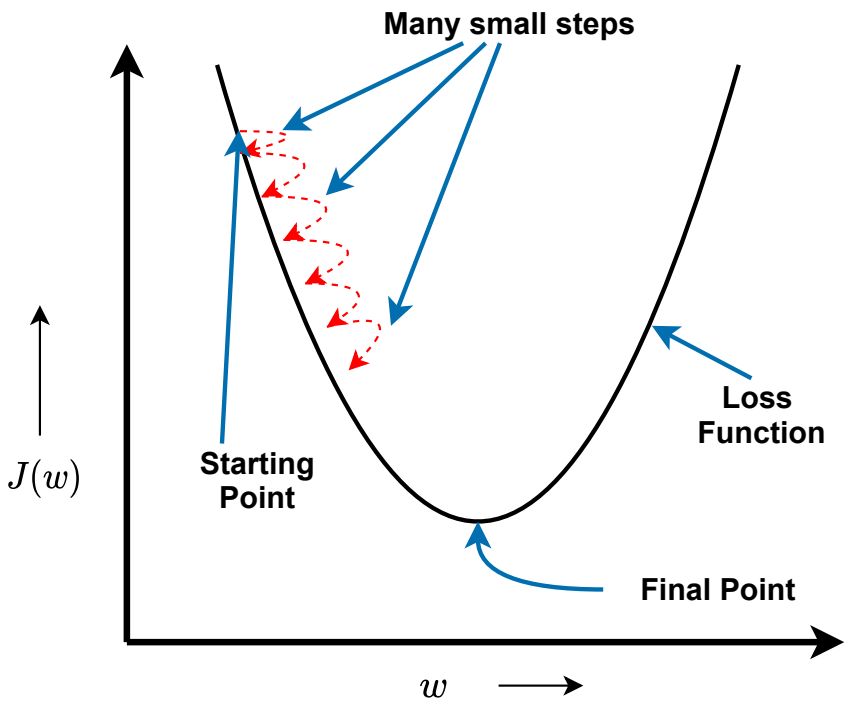
- As the number of examples in the batch go up, we get a more stable estimate of the gradient, whereas with smaller sizes our estimate is generally unstable.
- It changes quite a lot from iteration to iteration.
- This is the price for faster gradient updates.
- Assumption: the training examples are **IID** - that is **Independent and Identically Distributed**.
- Learning happens faster with mini-batch or stochastic GDs.
- This is because we are expected to see representative samples from the distribution and hence the loss optimization proceeds faster than classical GD.

Choosing learning rate



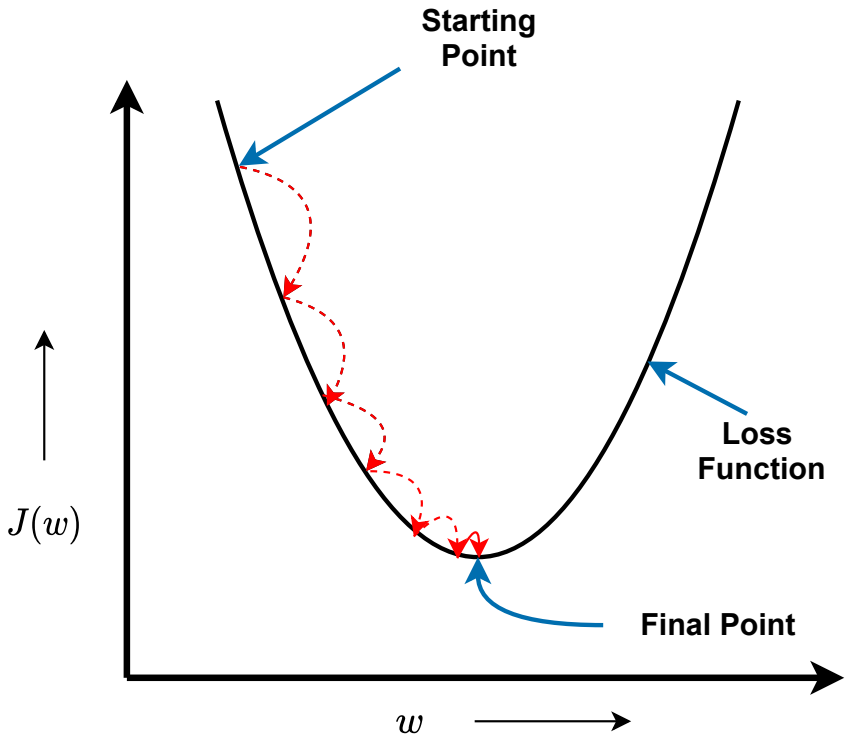
- Larger the learning rate, the longer the steps we take.
- Too large steps can cause to jump over the valley and move to the other side.
- Such jumps would continue with each iteration.
- This would never lead to the valley.

Choosing learning rate



- Too small learning rate causes to take many small steps to reach the valley.
- In each iteration, more gradient computations and weight updates.

Choosing learning rate: Learning curves



- Choose a rate that is not too small or too big.
- You would wonder how to do that.

Choosing learning rate: Learning curves

- Learning curves help to diagnose issues in training.
- It helps you to determine if the learning is progressing as intended or there are some hiccups which need our attention.
- The learning curve has iteration numbers on x-axis and loss on y-axis.
- When training the model with gradient descent or its variants, the loss is expected to come down with every step.
- This indicates that we are proceeding in the correct direction to the valley.
- If this doesn't happen, try reducing the learning rate and check again.

Choosing learning rate: Learning curves

If learning curve is going down ***very slowly***:

- It means loss is reducing very slowly.
- It could be due to small steps that we are taking due to the small learning rate.
- Try increasing the learning rate.

Choosing learning rate: Learning curves

- Sometimes there are a lot of ups and downs in the learning curve - that means the loss sometimes comes down or sometimes goes up.
- This could be due to a large learning rate or a small batch size.
- Try adjusting these quantities and recheck the learning curve.

1 Training Data

2 Model

3 Loss Function

4 Optimization Procedure

5 Evaluation

Overfitting and Underfitting
Fixing Overfitting and Underfitting
Metrics for Regression
Metrics for classification

Evaluation

- The model is evaluated on validation data after training
- Many metrics can be used depending on the problem type.
- Take into account metrics on validation examples for evaluating accuracy of the model.
- Calculate both training and validation metrics and compare them for diagnostics related to over/underfitting.
- There are different metrics for regression and classification problems.
- First and the simplest one is to compare the loss function between training and validation points.

Evaluation

Observing training loss and validation loss:

Symptoms	Diagnosis
Training Loss - low Validation Loss - low	Just the Right fit
Training Loss - high Validation Loss - high	Underfitting
Training Loss - low Validation Loss - high	Overfitting

1 Training Data

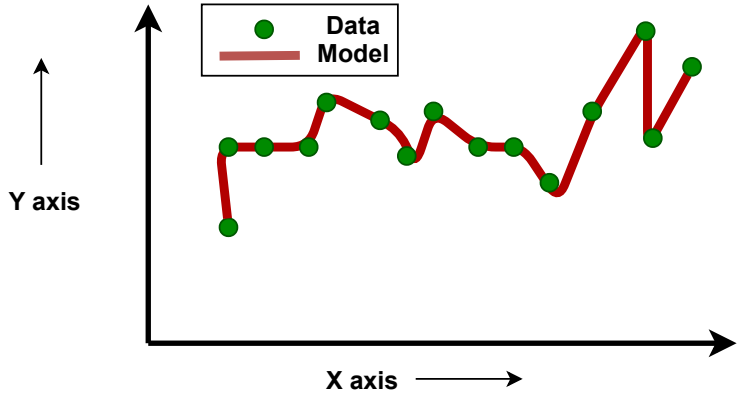
2 Model

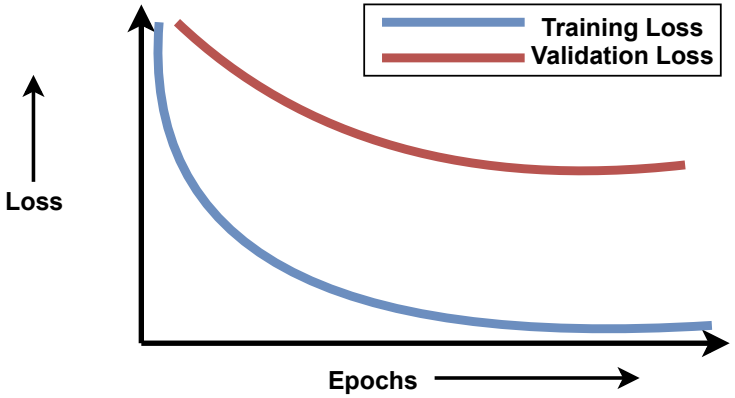
3 Loss Function

4 Optimization Procedure

5 Evaluation
- Overfitting and Underfitting
- Fixing Overfitting and Underfitting
- Metrics for Regression
- Metrics for classification

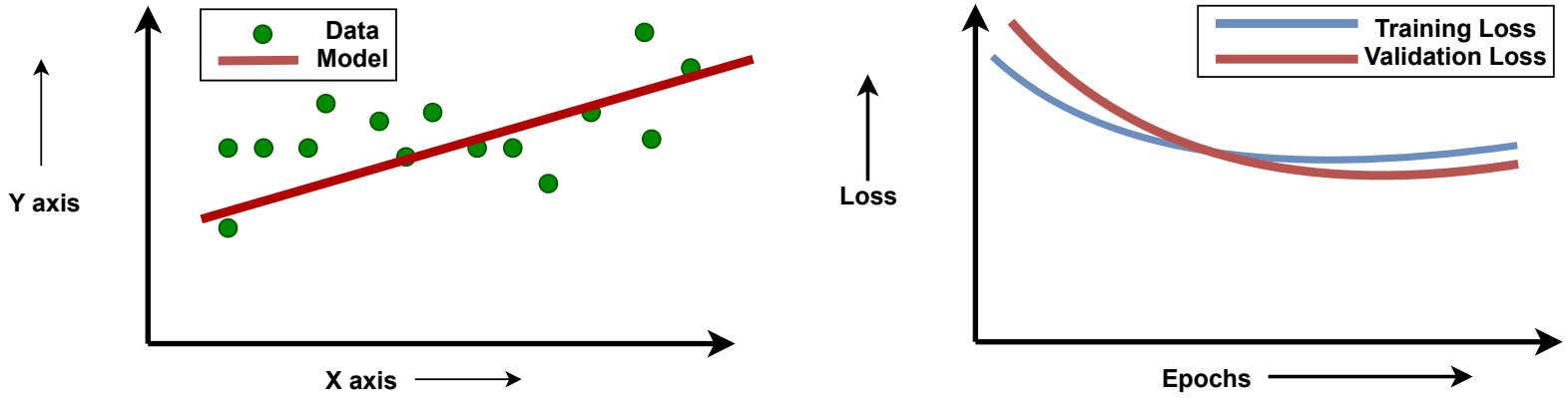
Overfitting





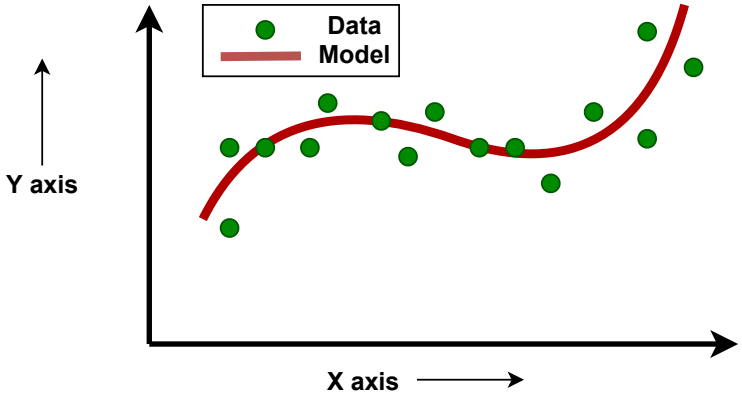
- An example: - when the kids are given a question bank for a topic and the kids only prepare questions from the question bank and nothing outside it.
- They are able to answer the questions from the question bank but are unable to answer questions outside.
- This is overfitting to the questions in the question bank.

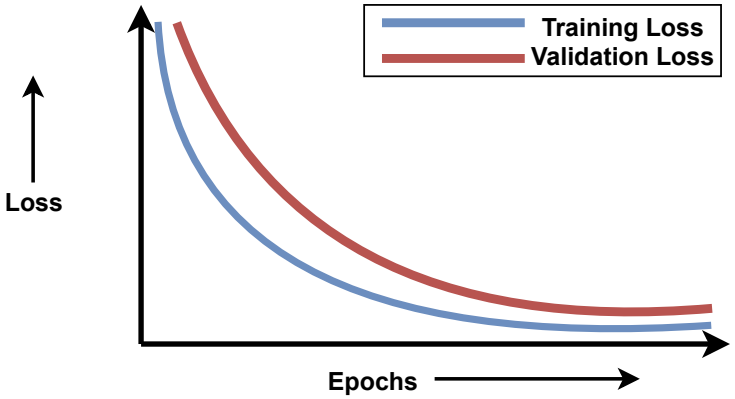
Underfitting



- The underfitting here would make kids ignore the questions and only prepare a few topics.
- They answer questions based on those few topics, but their performance remains poor in the exam which is based on all the topics.

The right fit





- The right fit here would mean - that the kids prepare questions from the question bank and also prepare similar questions on other concepts and try to understand the basics.
- They are likely to do well in the exam as they took into account the data and what would generalize it.

1 Training Data

2 Model

3 Loss Function

4 Optimization Procedure

5 Evaluation

Overfitting and Underfitting

Fixing Overfitting and Underfitting

Metrics for Regression

Metrics for classification

Fixing underfitting

- Underfitting is caused due to lack of representation capacity or power in the model,
  - One simple way to enhance capacity is by adding polynomial features - We will study how to incorporate polynomial features later in this module in the data preprocessing video.
  - At times despite having enough representation power, a large regularization rate can cause underfitting.
- Getting more data won't solve the underfitting problem as the model does not have any representation power.

Fixing overfitting

Overfitting is caused by excessive representation capacity that allows the model to memorize the training set and it can't perform well on the unseen examples.

To fix it:

- Supply more data for training.
- Introduce regularization for avoiding overfitting. In case it is already being used, increase the regularization rate.

1 Training Data

2 Model

3 Loss Function

4 Optimization Procedure

5 Evaluation

Overfitting and Underfitting

Fixing Overfitting and Underfitting

Metrics for Regression

Metrics for classification

Metrics for regression

Metric Name	Mathematical Formula
Mean Squared Error (MSE)	$\frac{1}{n} * \sum_{i=1}^{n} \left(\hat{y}^{(i)} - y^{(i)}\right)^2$
Mean Absolute Error	$\frac{1}{n} * \sum_{i=1}^{n} \left\lvert \hat{y}^{(i)} - y^{(i)} \right\rvert$

1 Training Data

2 Model

3 Loss Function

4 Optimization Procedure

5 Evaluation

Overfitting and Underfitting

Fixing Overfitting and Underfitting

Metrics for Regression

Metrics for classification

Metrics for classification

Metrics:

- Precision
- Recall
- F-1 Score
- AUC-ROC
- AUC-PR

Accuracy is not the best metric

- Accuracy is **not** an ideal metric whenever we have imbalance in the labels.
- For example, if we have 70% examples from one class, say class 1 and 30% from the other, say class 0.
- Then one can easily get 70% accuracy by always predicting the majority class label, i.e. from class 1 in this case.

Confusion matrix

Predicted / Actual	False(0)	True(1)
False(0)	True Negative (TN)	False Positive (FP)
True(1)	False Negative (FN)	True Positive (TP)

- It literally captures the confusion of our model in deciding between the two classes.

True Positive

Actual \ Predicted	False(0)	True(1)
False(0)	True Negative (TN)	False Positive (FP)
True(1)	False Negative (FN)	True Positive (TP)

- True positives are those examples where both the actual and predicted labels are 1 - which is a desired outcome. Their count is recorded in TP.

True Negative

Actual \ Predicted	False(0)	True(1)
False(0)	True Negative (TN)	False Positive (FP)
True(1)	False Negative (FN)	True Positive (TP)

- True negatives are those examples where both the actual and predicted labels are 0, which again is a desired outcome. And their count is recorded in TN.

False Positive

Actual \ Predicted	False(0)	True(1)
False(0)	True Negative (TN)	False Positive (FP)
True(1)	False Negative (FN)	True Positive (TP)

- One of them is false positives where the actual label is 0, it is predicted to be 1 by the current model. I remember it as a false prediction and it is falsely predicted to be positive.

False Negative

Actual \ Predicted	False(0)	True(1)
False(0)	True Negative (TN)	False Positive (FP)
True(1)	False Negative (FN)	True Positive (TP)

- The other one is false negative. These denote cases where the actual label is 1, but it is predicted to be 0 by the current model. Similar to FP, I remember it as a false prediction and it is falsely predicted to be negative.

Precision

- **Precision**:ratio of correctly predicted positives by the total predicted positives.

$$Precision = \frac{TP}{TP + FP}$$

- Percentage of correct positive identifications.

Recall

- **Recall**: the ratio of correctly predicted positives to the total actual positives.

$$Recall = \frac{TP}{TP + FN}$$

- Percentage of actual positive examples correctly classified.

F1 Score

- **F1-score**: harmonic mean of precision and recall.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

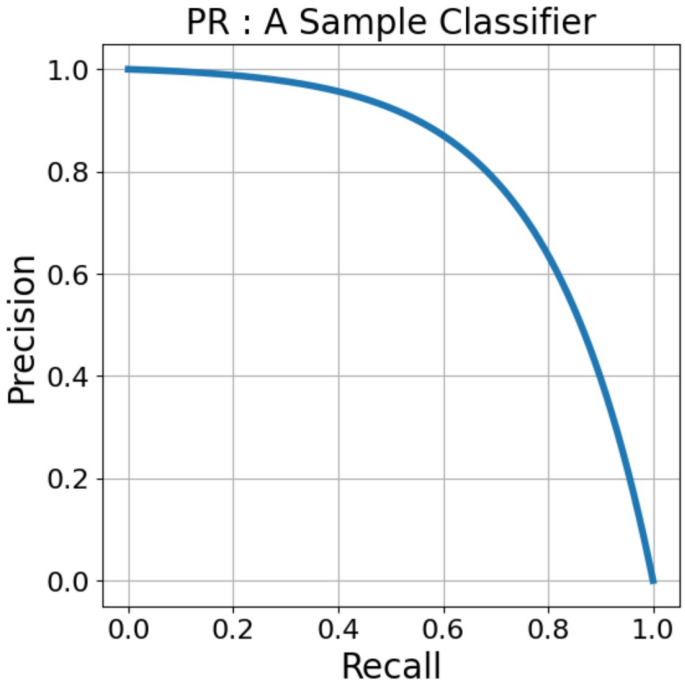
- When the model outputs the probability of a positive class - we typically decide a probability threshold above which all examples are positive i.e. they belong to the positive class and below ones belong to the negative class i.e. class 0.
- Instead of evaluating the classifier only at a particular threshold, it would be useful to understand behavior of classification metrics across different probability thresholds.

Precision-Recall (PR) curve

Steps:

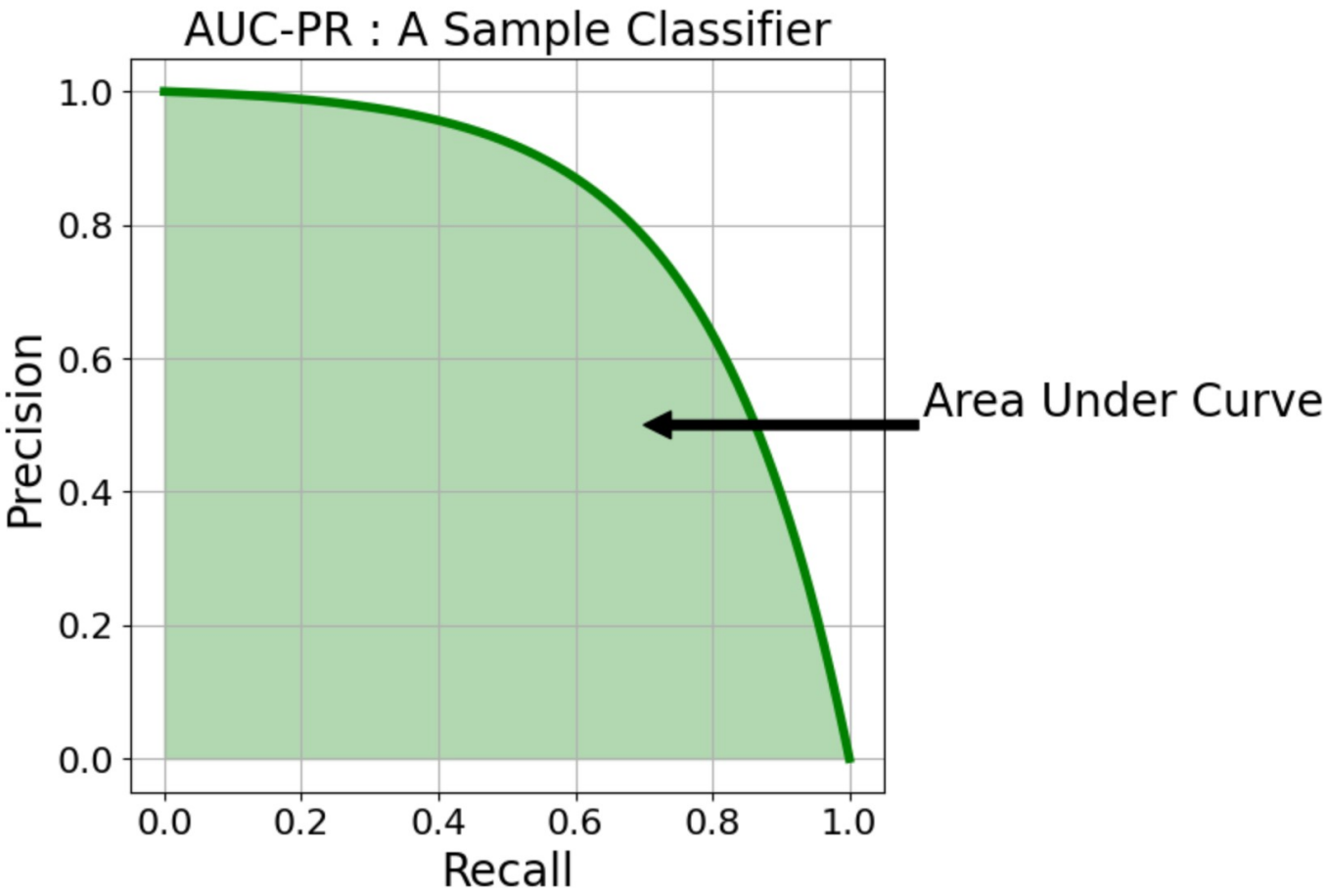
- Fix the probability threshold, say r.
- Get predictions for positive and negative classes as per r.
- Calculate the confusion matrix at this threshold and calculate precision and recall metrics.
- Store these metrics for the threshold.
- Choose multiple r s between 0 to 1 and for each value of r, calculate precision/recall values.
- Plot these values in ascending order of r. This chart is called a PR curve.

PR curve



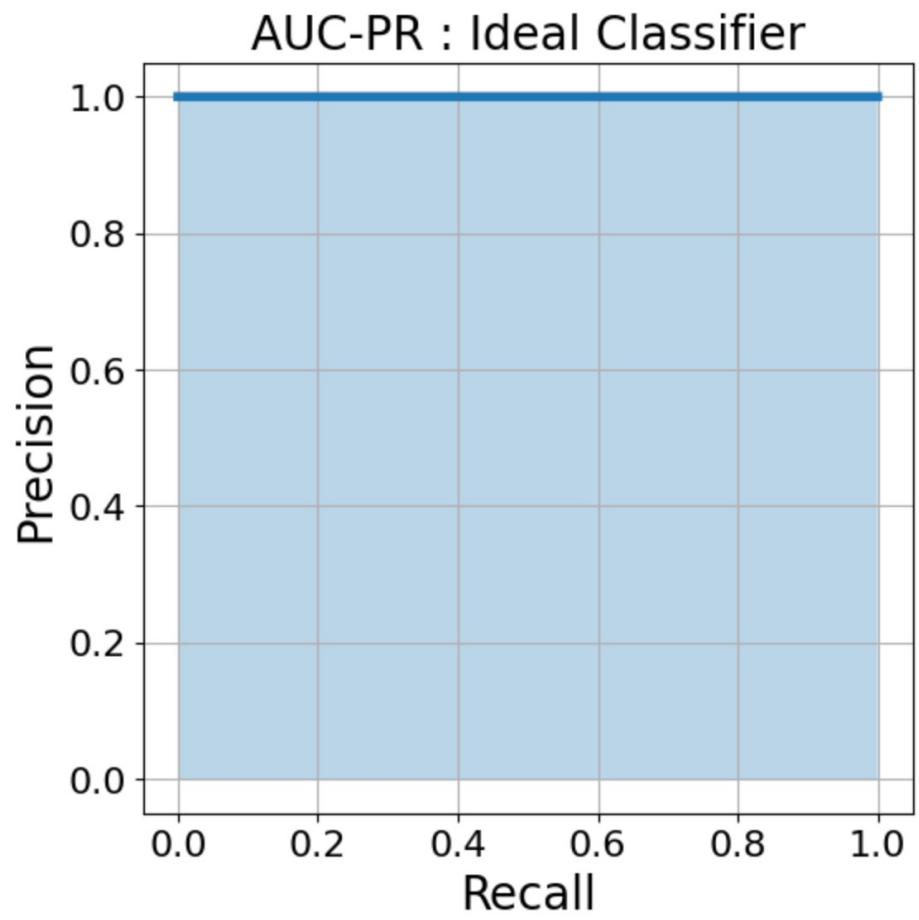
- PR curve has precision on y-axis and recall on x-axis.
- As precision increases, recall drops and vice versa.
- Plot values of recall and precision for each probability threshold from 0 to 1.

PR curve: AUC



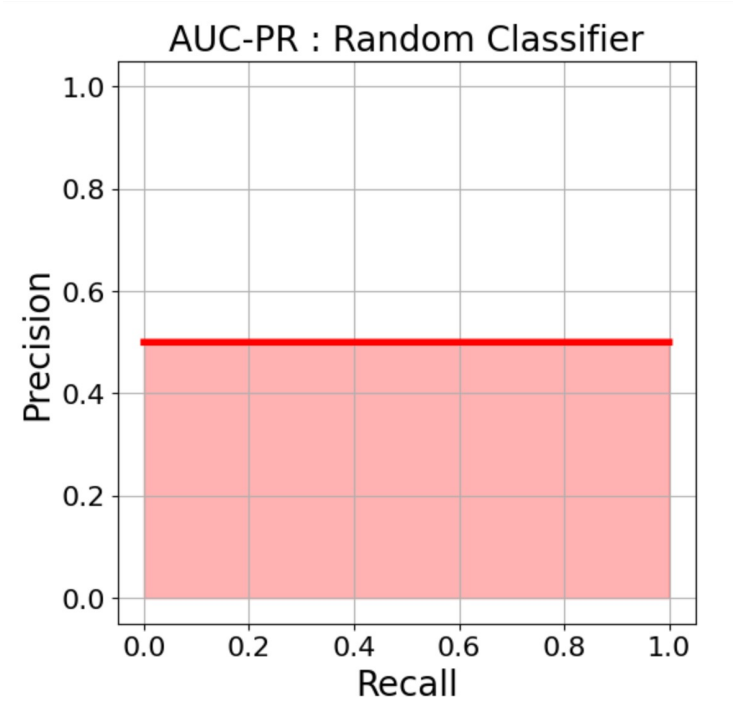
- Area under the curve (AUC) measures effectiveness of the model. It is called AUC-PR.

PR curve: An ideal classifier



- Ideal classifier has AUC-PR equal to 1.

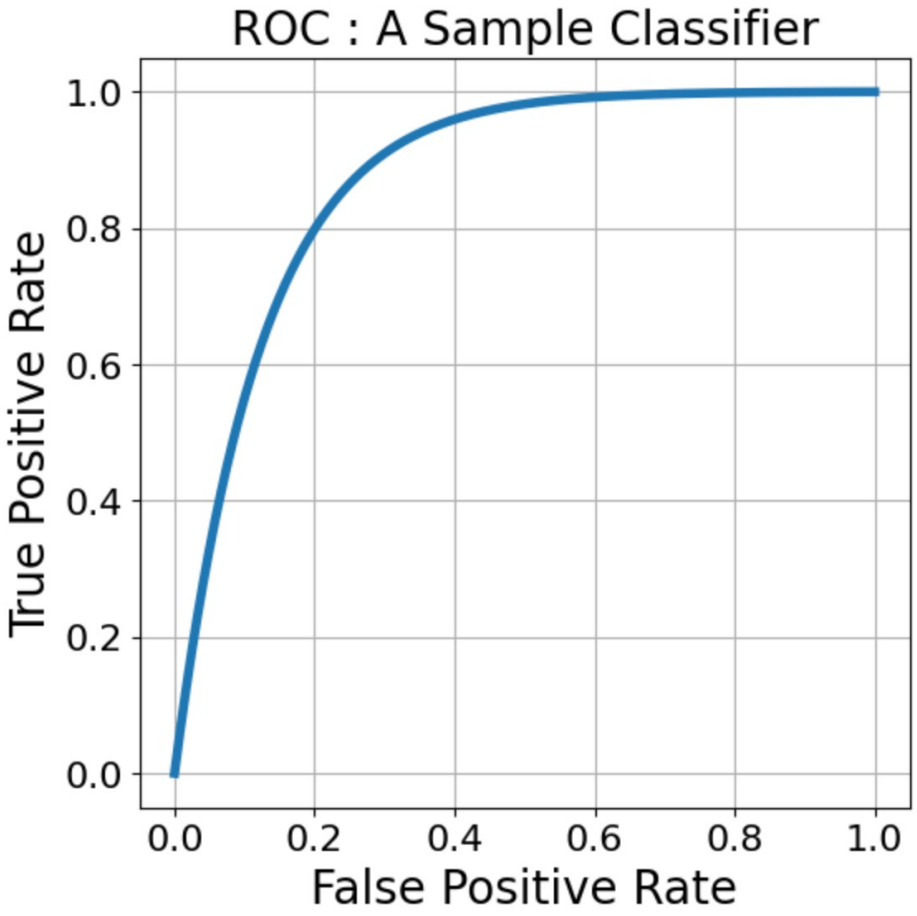
PR Curve: A random classifier



- Above is the PR curve of a random or no-skill classifier. It is a horizontal line at precision equal to the percentage of the majority class.
- For a balanced class setting, the baseline is 0.5, as displayed in the figure above.

Receiver Operating Characteristic (ROC) curve

- **ROC**: Receiver Operating Characteristics



ROC curve

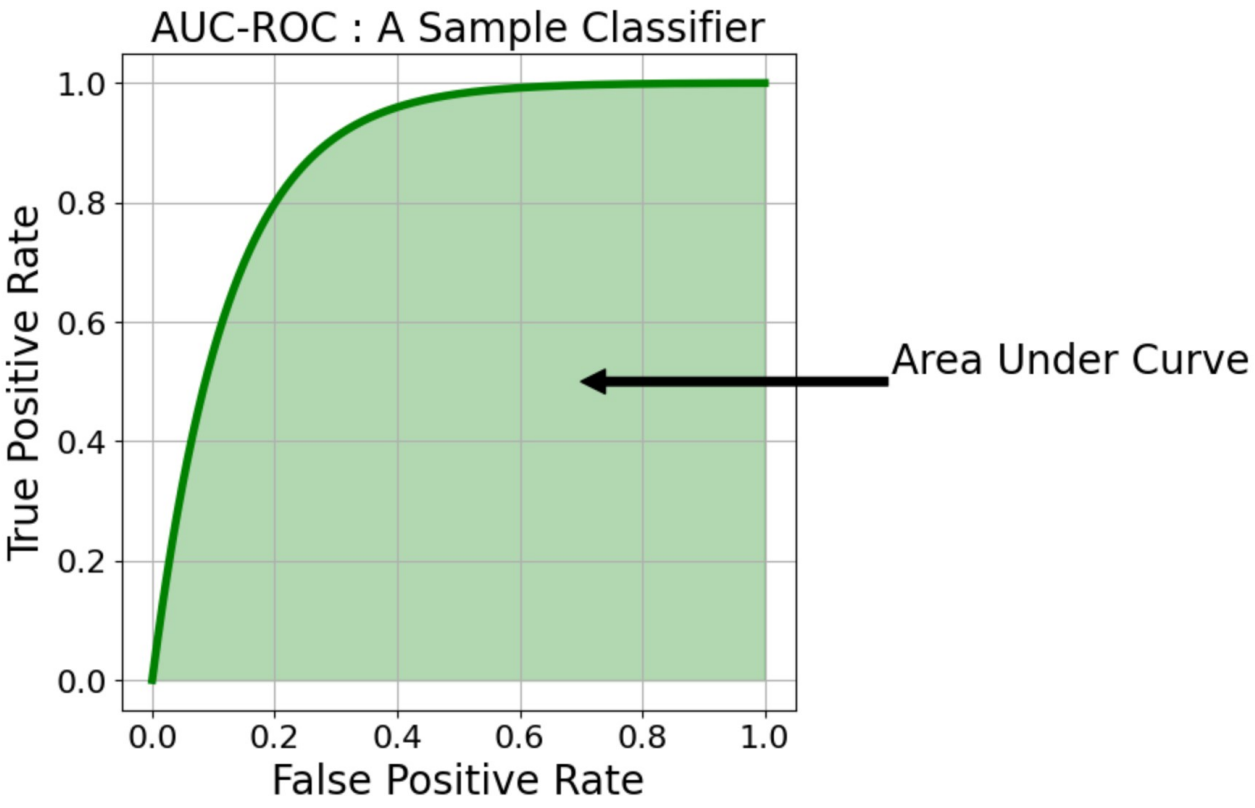
- **FPR**: False positive rate:

$$FPR = 1 - Specificity = \frac{FP}{FP + TN}$$

- **TPR**: True positive rate:

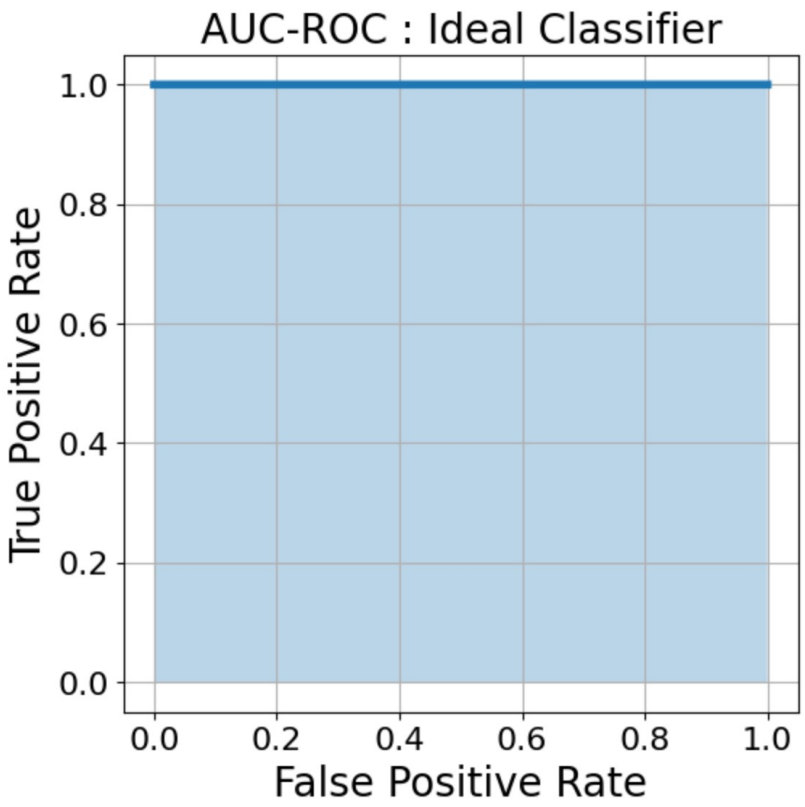
$$TPR = Sensitivity = \frac{TP}{FN + TP}$$

ROC curve: AUC



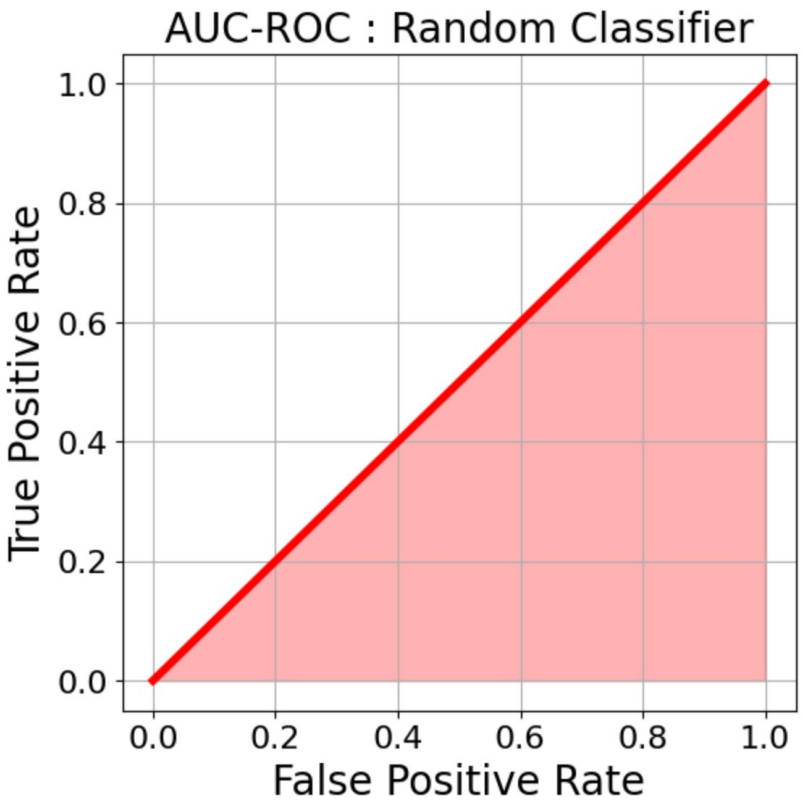
- Area under the curve (AUC) measures performance of the classifier. Higher the better.

AUC-ROC : Ideal Classifier



- The ideal classifier has AUC-ROC=1.

AUC-ROC : A random classifier



- A random or no-skill classifier has an AUC of 0.5.

AUC-ROC : A good classifier

- Good classifiers or classifiers with some skills learnt from the training examples are expected to have AUC-ROC > 0.5.

When to use AUC-ROC and AUC-PR

- For balanced classification, AUC-ROC is preferred.
- For moderate to large imbalanced classification setting, AUC-PR is used.

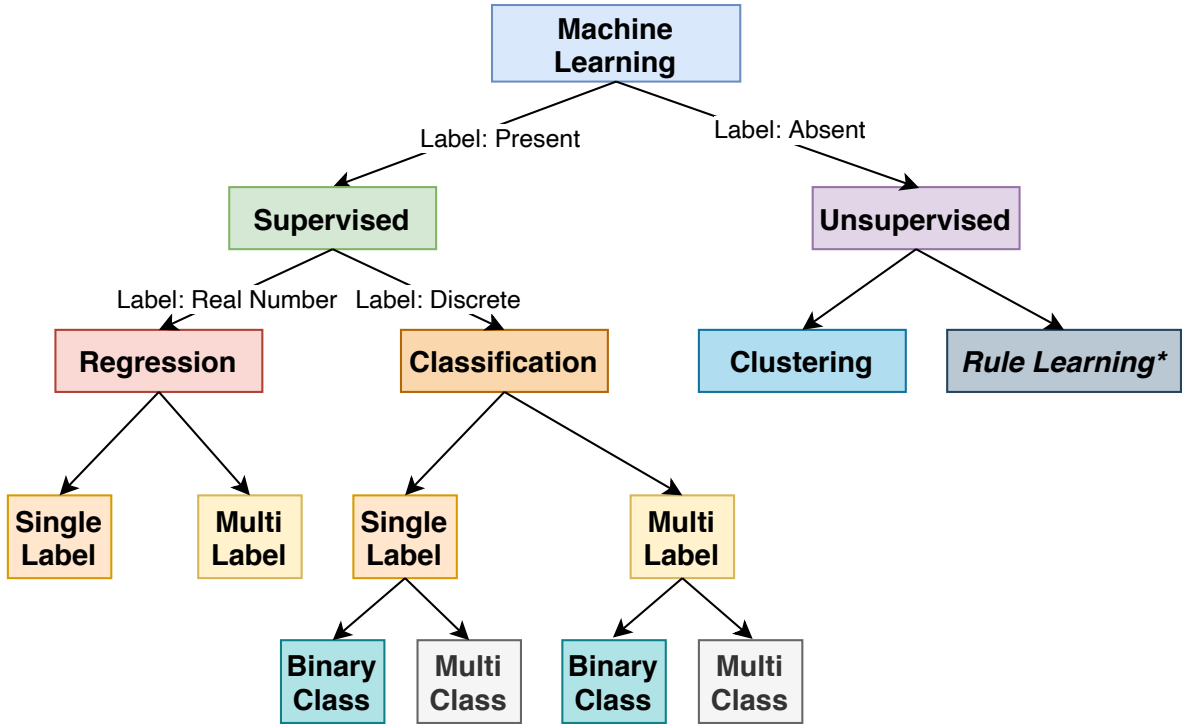
‘

Models of Regression

Machine Learning Techniques

Dr. Ashish Tendulkar
IIT Madras

Regression



It is a supervised learning problem where the output label is a real number.

Two types of regression

Single label regression

Single output label: $y \in \mathbf{R}$

e.g. Predict temperature for tomorrow based on weather features of past days

Multi label regression

Multiple output labels - total labels k: $\mathbf{y} \in \mathbf{R}^k$

e.g. Predict temperature for next 5 days based on weather features of past days

What will be covered in this module?

- Linear regression
- Polynomial regression
- Regularization

All concepts will be discussed for single label regression problem set up.

In the end, we will extend them to multi-label regression problem set up.

Linear Regression

It is a machine learning algorithm that predicts real valued output label based on linear combination of input features.

- Input: Feature vector $\mathbf{x}_{m\times 1}$.
- Label: $y \in \mathbb{R}$, which is a scalar. (*Single label set up)*

Part I: Linear Regression

Data

Data

A set of n ordered pairs of a feature vector, $\mathbf{x}$ and a label y representing examples. We denote it by D:

$$D = \{(\mathbf{X}, \mathbf{y})\} = \left\{(\mathbf{x}^{(i)}, y^{(i)})\right\}_{i=1}^{n}$$

Features

$\mathbf{X}$ is a feature matrix corresponding to n training examples, each represented with m features and has shape $n \times m$. In this matrix, each feature vector is transposed and represented as a row.

$$\mathbf{X}_{n \times m} = \begin{bmatrix} - \left(\mathbf{x}^{(1)}\right)^T - \\ - \left(\mathbf{x}^{(2)}\right)^T - \\ \vdots \\ - \left(\mathbf{x}^{(n)}\right)^T - \end{bmatrix}$$

Feature vector for i-th training example $\mathbf{x}^{(i)}$ can be obtained by $\mathbf{X}[i]$

Labels

For single label problem, $\mathbf{y}$ is a label vector of shape $n \times 1$.

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

The i-th entry in this vector, $\mathbf{y}[i]$ gives label for i-th example, which is denoted by $y^{(i)} \in \mathbb{R}$

Data: Example

$$D = \{(\mathbf{X}, \mathbf{y})\} = \left\{(\mathbf{x}^{(i)}, y^{(i)})\right\}_{i=1}^{n}$$

Sample House Pricing data is presented below.

$$\mathbf{X}_{n\times m} = \begin{bmatrix} - \left(\mathbf{x}^{(1)}\right)^T - \\ - \left(\mathbf{x}^{(2)}\right)^T - \\ \vdots \\ - \left(\mathbf{x}^{(n)}\right)^T - \end{bmatrix}$$

Area | No. of Floors | Age

Features (m=3)

Examples (n=4)

$$\mathbf{X}_{4\times 3} = \begin{bmatrix} 1000 & 1 & 15 \\ 1500 & 2 & 25 \\ 1100 & 3 & 10 \\ 2000 & 1 & 40 \end{bmatrix}$$

House 1

House 2

House 3

House 4

Data: Example

House Pricing data, labels (price in Rs.)

Price

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix} \qquad \mathbf{y}_{4\times 1} = \begin{bmatrix} 1000000 \\ 2000000 \\ 1580000 \\ 4000000 \end{bmatrix}$$

House 1

House 2

House 3

House 4

Data: Example

$$D = \{(\mathbf{X}_{4\times3}, \mathbf{y}_{4\times1})\} = \left\{(\mathbf{x}^{(i)}, y^{(i)})\right\}_{i=1}^{4}$$

Area | No. of Floors | Age

Price

$$\mathbf{X}_{4\times3} = \begin{bmatrix} 1000 & 1 & 15 \\ 1500 & 2 & 25 \\ 1100 & 3 & 10 \\ 2000 & 1 & 40 \end{bmatrix} \qquad \mathbf{y}_{4\times1} = \begin{bmatrix} 1000000 \\ 2000000 \\ 1580000 \\ 4000000 \end{bmatrix}$$

House 1

House 2

House 3

House 4

Model

Model

Label y is computed as a linear combination of features $x_1, x_2, ..., x_m$:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m$$

Here $w_0, w_1, w_2, \ldots, w_m$ are weights

- Each feature x_j has a corresponding weight w_j.
- Let's introduce a dummy feature $x_0 = 1$ for weight w_0:

$$y = w_0 x_0 + w_1 x_1 + \ \ldots + w_m x_m$$

It can be written compactly as

$$y = \sum_{i=0}^{m} w_i x_i$$

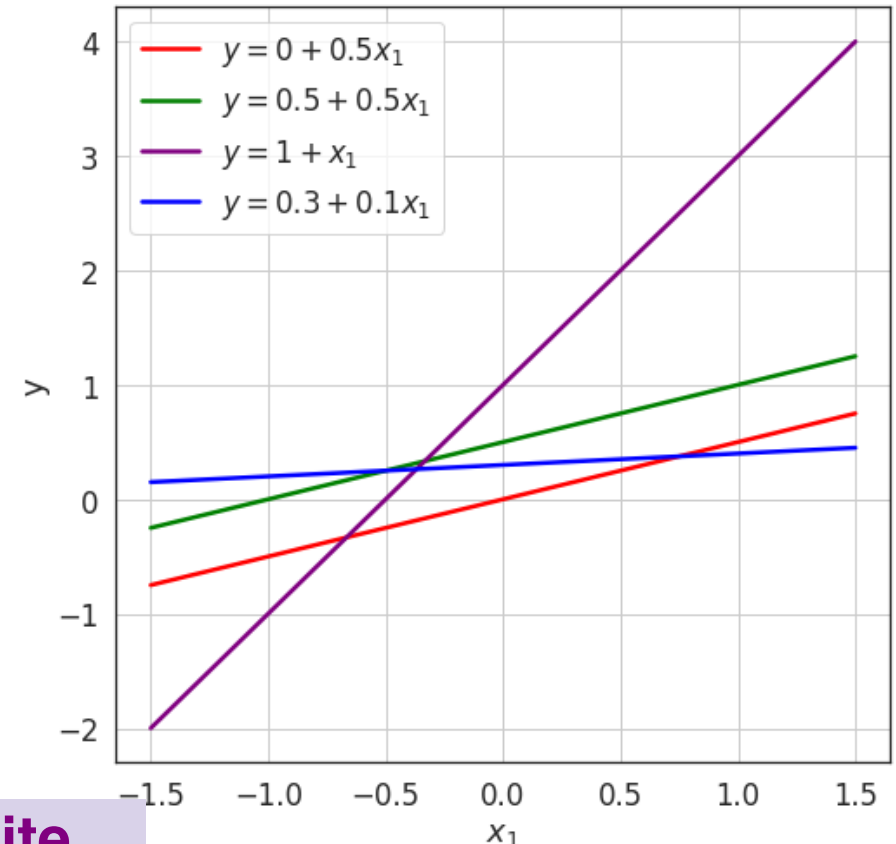
How does the model look like?

Take a simple model with a single feature x_1:

$$y = w_0 + w_1 x_1$$

model parameters (2): w_0, w_1

Each new combination of w_0, w_1 leads to a new model.



Total # of possible models = infinite

Model: geometry and # parameters depends on # features

# Features (m)	Model	Geometry	# Params (# w)
1	$y = w_0 + w_1 x_1$	line	2
2	$y = w_0 + w_1 x_1 + w_2 x_2$	plane	3
≥ 3	$y = \sum_{i=0}^{m} w_i x_i$	hyperplane in (m +1)-D	$m + 1$

Model: Vectorized Form

$$y = \sum_{j=0}^{m} w_j x_j = \mathbf{w}^T \mathbf{x}$$

Exercise: Check $\mathbf{w}^T\mathbf{x} = \sum w_j x_j$

Solve vector matrix product and verify it corresponds to the linear combination of features.

All weights $w_0, w_1, \ldots, w_m$ can be arranged in a vector $\mathbf{w}$ with shape $(m+1) \times 1$

$$\mathbf{w}_{(m+1)\times 1} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_m \end{bmatrix}$$

Exercise: Check $\mathbf{w}^T\mathbf{x} = \sum w_j x_j$

The features for an i-th example can also be represented as a feature vector $\mathbf{x}^{(i)}$. Note that we add a special feature x_0.

$$\mathbf{x}^{(i)}_{(m+1)\times 1} = \begin{bmatrix} x_0 \\ x_1^{(i)} \\ \vdots \\ x_m^{(i)} \end{bmatrix}$$

Setting $x_0 = 1$

$$\mathbf{x}^{(i)}_{(m+1)\times 1} = \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_m^{(i)} \end{bmatrix}$$

Exercise: Check $\mathbf{w}^T\mathbf{x} = \sum w_j x_j$

Let's multiply these vectors to obtain $y^{(i)}$:

$$
\begin{aligned}
y^{(i)} &= \mathbf{w}^T\mathbf{x} \\
&= \left(\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_m \end{bmatrix}\right)^T \times \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_m^{(i)} \end{bmatrix} \\
&= \begin{bmatrix} w_0 & w_1 & \dots & w_m \end{bmatrix} \times \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_m^{(i)} \end{bmatrix}
\end{aligned}
$$

Exercise: Check $\mathbf{w}^T\mathbf{x} = \sum w_j x_j$

$$y^{(i)} = \begin{bmatrix} w_0 & w_1 & \dots & w_m \end{bmatrix} \times \begin{bmatrix} 1 \\ x_1^{(i)} \\ \vdots \\ x_m^{(i)} \end{bmatrix}$$

$$y^{(i)} = w_0 \times 1 + w_1 \times x_1^{(i)} + \dots + w_m \times x_m^{(i)}$$

$$y^{(i)} = w_0 x_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \dots + w_m x_m^{(i)}$$

$$y^{(i)} = \sum_{j=0}^{m} w_j {x_j}^{(i)}$$

Vectorized implementation: Multiple Examples

A weight vector $\mathbf{w}$ with shape $(m+1) \times 1$.

A feature matrix $\mathbf{X}$ has shape $(n, m+1)$ containing features for n examples.

$$\mathbf{w}_{(m+1)\times 1} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_m \end{bmatrix}$$

$$\mathbf{X}_{n\times(m+1)} = \begin{bmatrix} - \left(\mathbf{x}^{(1)}\right)^T - \\ - \left(\mathbf{x}^{(2)}\right)^T - \\ \vdots \\ - \left(\mathbf{x}^{(n)}\right)^T - \end{bmatrix}$$

The label vector $\widehat{\mathbf{y}}$ containing labels for all the n examples can be computed as follows:

$$\widehat{\mathbf{y}}_{n\times 1} = \mathbf{X}_{n\times(m+1)}\mathbf{w}_{(m+1)\times 1}$$

$$\widehat{\mathbf{y}} = \mathbf{X}\mathbf{w}$$

Model Inference

$$\mathbf{y} = \mathbf{X}\mathbf{w}$$

Implement the model inference in vectorized form:

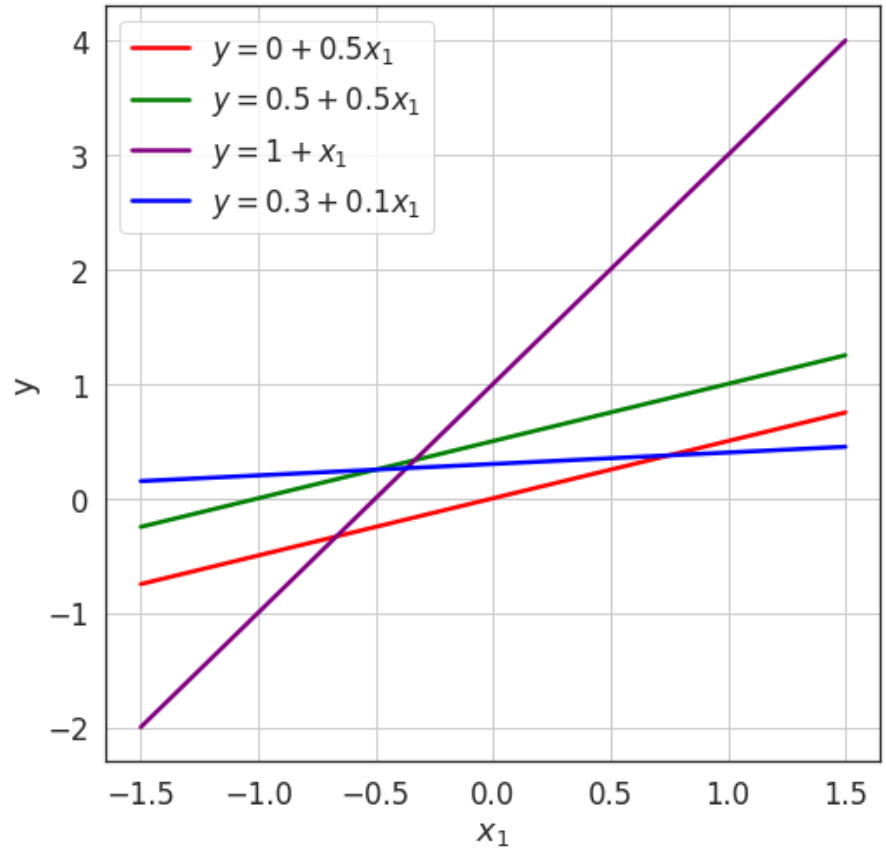
```
def predict(X, w):
  # Make sure features and weights have compatible shape.
  # Check to make sure that the shapes are compatible.
  assert X.shape[-1]==w.shape[0], "X and w don't have comp

  y = X @ w
  return y
```

Choosing best-fit model

- Let's choose a simple model for exploring this question:

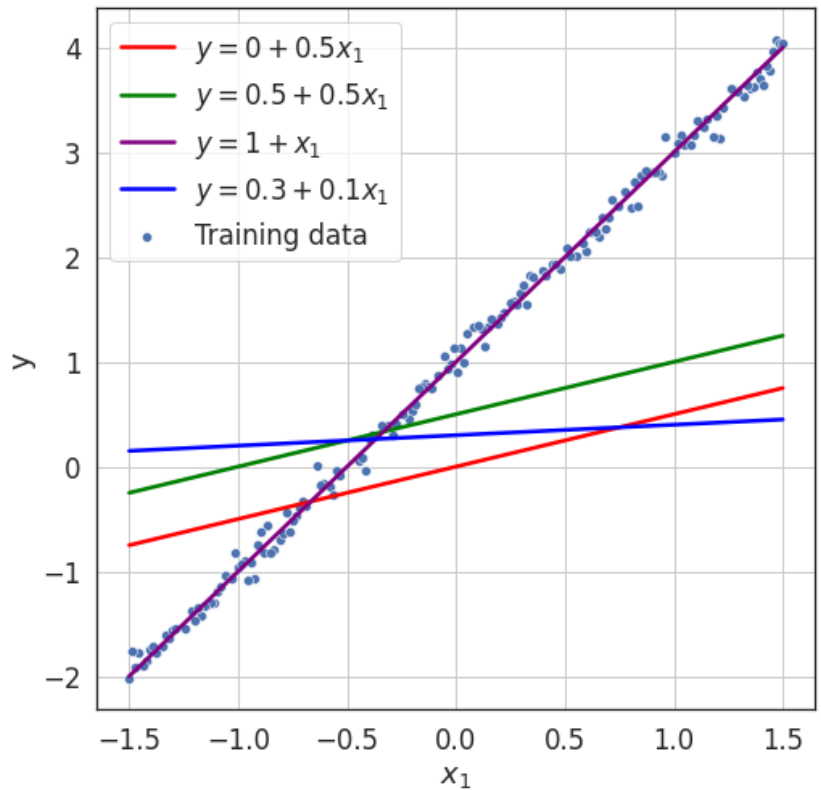
$$y = w_0 + w_1 x_1$$

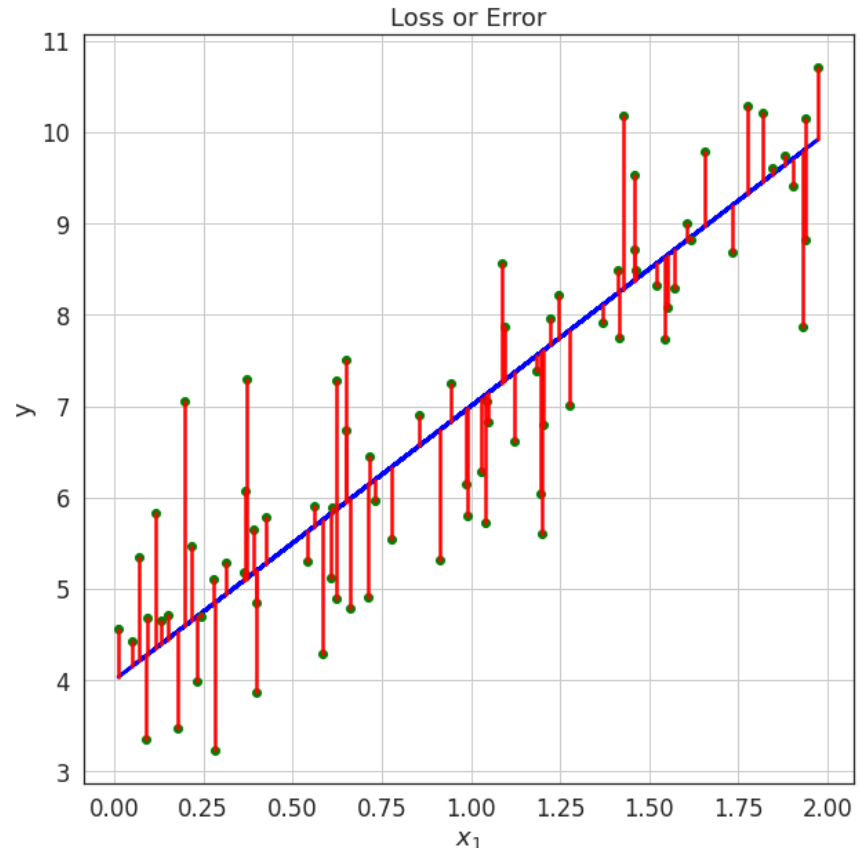
- As we vary (w_0, w_1), we get a new model.



Evaluate data fitment of model

Loss or error function

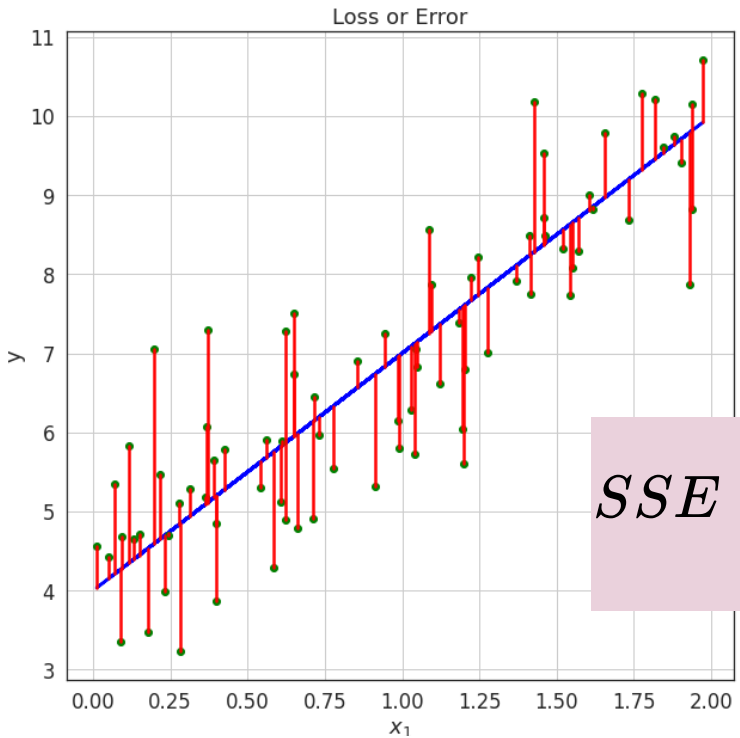




- Which one fits the training data better than all other models?
- We need a measure of fitment or mis-fitment.

Loss Function

Loss Function: Sum of Squared Error (SSE)



$$SSE = \sum_{i=1}^{n} \left(\text{predicted label}^{(i)} - \text{actual label}^{(i)} \right)^2$$

SSE is the sum of square of difference between actual and predicted labels for each training point.

Error at i-th training point

$$
\begin{aligned}
e^{(i)} &= (\text{predicted label} - \text{actual label})^2 \\
&= \left(h_{\mathbf{w}}(\mathbf{x}^{(i)}) - y^{(i)}\right)^2 \\
&= \left(\mathbf{w}^T \mathbf{x}^{(i)} - y^{(i)}\right)^2
\end{aligned}
$$

- The error $e^{(i)}$ is non-negative or $e^{(i)} \geq 0$.
- $e^{(i)} = 0$, whenever the actual label and the predicted label are identical.

Loss Function: SSE

The total loss $J(\mathbf{w})$ is sum of errors at each training point:

$$J(\mathbf{w}) = \sum_{i=1}^{n} e^{(i)}$$

We divide this by $\frac{1}{2}$ for mathematical convenience in later use:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n} e^{(i)} = \frac{1}{2}\sum_{i=1}^{n}\left(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)}\right)^2$$

The loss is dependent on the value of $\mathbf{w}$ - as these values change, we get a new model, which will result in different prediction and hence different error at each training point.

SSE vectorization for efficient computation

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}\left(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)}\right)^2$$

$$= \frac{1}{2}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^T\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)$$

Recall, $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$

```python
def loss(features, labels, weights):

  # Compute error vector
  e =  predict(features, weights) - labels

  # Compute loss
  loss = (1/2) * (np.transpose(e) @ e)

  return loss
```

Loss function visualization

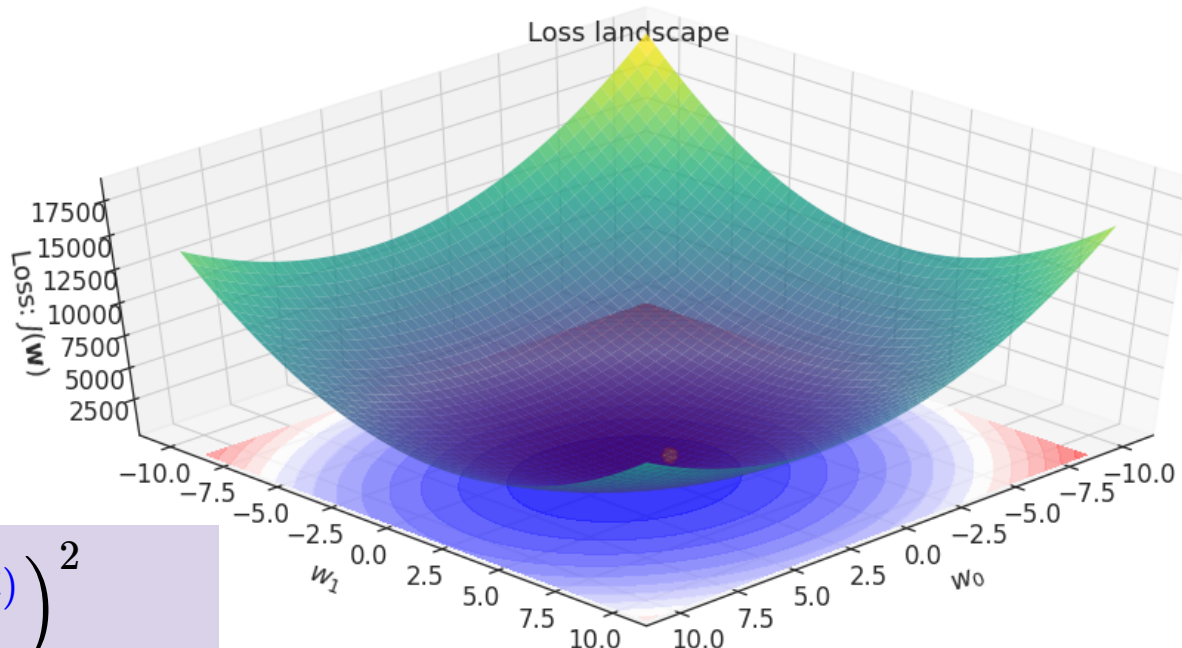
Model

$$h_{\mathbf{w}}(\mathbf{x}) : y = w_0 + w_1 x_1$$

Loss (SSE)

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}\left(h_{\mathbf{w}}(\mathbf{x}^{(i)}) - y^{(i)}\right)^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}\left((w_0 + w_1 x_1^{(i)}) - y^{(i)}\right)^2$$



Duality of loss and model spaces

https://www.geogebra.org/material/iframe/id/evjby7mb/width/1020/height/500/border/888888/sfsb/true/smb/false/stb/false/stbh/false/ai/false/asb/false/sri/false/rc/false/ld/false/sdz/true/ctl/false

How do we find the model with the least loss or error?

Optimization

Optimization

Key calculation is the derivative of loss function $J(\mathbf{w})$ with respect to $\mathbf{w}$.

Let's look at a couple of useful identities for this:

Derivative of linear function: $\frac{\partial}{\partial \mathbf{w}}(\mathbf{w}\mathbf{x}) = \frac{\partial}{\partial \mathbf{w}}\left(\mathbf{x}^T\mathbf{w}\right) = \frac{\partial}{\partial \mathbf{w}}\left(\mathbf{w}^T\mathbf{x}\right) = \mathbf{x}$

Similar to $\frac{d}{dw}(xw) = x$

Derivative of quadratic function: $\frac{\partial}{\partial \mathbf{w}}\left(\mathbf{w}^T\mathbf{X}\mathbf{w}\right) = 2\mathbf{X}\mathbf{w}$

Similar to $\frac{d}{dw}(xw^2) = 2xw$

Computing gradient of loss function

$$
\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}} \left\{ \frac{1}{2} \left(\mathbf{X}\mathbf{w} - \mathbf{y} \right)^T \left(\mathbf{X}\mathbf{w} - \mathbf{y} \right) \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \left((\mathbf{X}\mathbf{w})^T - \mathbf{y}^T \right) \left(\mathbf{X}\mathbf{w} - \mathbf{y} \right) \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \left(\mathbf{w}^T \mathbf{X}^T - \mathbf{y}^T \right) \left(\mathbf{X}\mathbf{w} - \mathbf{y} \right) \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \mathbf{w}^T \mathbf{X}^T \left(\mathbf{X}\mathbf{w} - \mathbf{y} \right) - \mathbf{y}^T \left(\mathbf{X}\mathbf{w} - \mathbf{y} \right) \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \left(\mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} \right) - \left(\mathbf{y}^T \mathbf{X}\mathbf{w} - \mathbf{y}^T \mathbf{y} \right) \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\mathbf{w} + \mathbf{y}^T \mathbf{y} \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} + \mathbf{y}^T \mathbf{y} \right\}
\end{aligned}
$$

Computing gradient of loss function

$$
\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \frac{1}{2}\frac{\partial}{\partial \mathbf{w}}\left\{\mathbf{w}^T\mathbf{X}^T\mathbf{w}\mathbf{X} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{y}^T\mathbf{y}\right\} \\
&= \frac{1}{2}\left\{\frac{\partial}{\partial \mathbf{w}}\left\{\mathbf{w}^T\mathbf{X}^T\mathbf{w}\mathbf{X}\right\} - \frac{\partial}{\partial \mathbf{w}}\left\{2\mathbf{w}^T\mathbf{X}^T\mathbf{y}\right\} + \frac{\partial}{\partial \mathbf{w}}\left\{\mathbf{y}^T\mathbf{y}\right\}\right\} \\
&= \frac{1}{2}\left\{2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} + 0\right\} \\
&= \frac{1}{2}\left\{2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y}\right\} \\
&= \frac{1}{2} \times 2 \times \left\{\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y}\right\} \\
&= \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} \\
&= \mathbf{X}^T\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)
\end{aligned}
$$

How to obtain $\mathbf{w}$?

- Normal equation

  Set the partial derivative to 0 and solve with analytical method to obtain the weight vector

- Gradient descent

  Iteratively change the weights based on the partial derivative of the loss function until convergence (Iterative optimization)

Normal Equation

Let's set $\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$ to 0 and solve for $\mathbf{w}$:

$$\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= 0 \\
\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} &= 0 \\
\mathbf{X}^T\mathbf{X}\mathbf{w} &= \mathbf{X}^T\mathbf{y} \\
\left(\mathbf{X}^T\mathbf{X}\right)\mathbf{w} &= \mathbf{X}^T\mathbf{y} \\
\mathbf{w} &= \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}
\end{aligned}$$

Recall $\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y}$

Gradient Descent: Non Vectorized

1. Randomly initialize the weight vector $\mathbf{w}$. One possible initialization can be: $w_0 = 0, w_1 = 0, \ldots, w_m = 0$.

Gradient Descent: Non Vectorized

2. Iterate until convergence:

   a. Calculate gradient of loss function w.r.t. the weights w_0, w_1 .... w_m, which are $\frac{\partial J(\mathbf{w})}{\partial w_0}$, $\frac{\partial J(\mathbf{w})}{\partial w_1}$ .... $\frac{\partial J(\mathbf{w})}{\partial w_m}$ respectively.

Gradient Descent: Non Vectorized

$$\frac{\partial J(\mathbf{w})}{\partial w_0} = \sum_{i=1}^{n} (w_0 + w_1 x_1^{(i)} + \ldots + w_m x_m^{(i)} - y^{(i)}) x_0^{(i)}$$

$$= \sum_{i=1}^{n} (w_0 + w_1 x_1^{(i)} + \ldots + w_m x_m^{(i)} - y^{(i)}) 1$$

$$\frac{\partial J(\mathbf{w})}{\partial w_1} = \sum_{i=1}^{n} (w_0 + w_1 x_1^{(i)} + \ldots + w_m x_m^{(i)} - y^{(i)}) x_1^{(i)}$$

$$\vdots$$

$$\frac{\partial J(\mathbf{w})}{\partial w_m} = \sum_{i=1}^{n} (w_0 + w_1 x_1^{(i)} + \ldots + w_m x_m^{(i)} - y^{(i)}) x_m^{(i)}$$

Gradient Descent: Non Vectorized

b. Calculate updated parameters:

$$
\begin{aligned}
w_0(\text{new}) &:= \quad w_0 - \alpha \frac{\partial J(\mathbf{w})}{\partial w_0} \\
w_1(\text{new}) &:= \quad w_1 - \alpha \frac{\partial J(\mathbf{w})}{\partial w_1} \\
&\vdots \\
w_m(\text{new}) &:= \quad w_m - \alpha \frac{\partial J(\mathbf{w})}{\partial w_m}
\end{aligned}
$$

Here α is learning rate.

Gradient Descent: Non Vectorized

c. Update parameters simultaneously:

$$
\begin{aligned}
w_0 &= w_0(\text{new}) \\
w_1 &= w_1(\text{new}) \\
&\vdots \\
w_m &= w_m(\text{new})
\end{aligned}
$$

After this step, we have a new weight vector.

Gradient Descent: Non Vectorized

- Note that the weights are updated to their new values simultaneously in the last step of GD .

- Until then the old values are used for predicting labels of examples while calculating the parameter updates.

Gradient Descent: Vectorized

$$
\begin{aligned}
\mathbf{w}_{k+1} &:= \mathbf{w}_k - \alpha \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \\
&:= \mathbf{w}_k - \alpha \left(\mathbf{X}^T \mathbf{X} \mathbf{w} - \mathbf{X}^T \mathbf{Y} \right) \\
&:= \mathbf{w}_k - \alpha \mathbf{X}^T \left(\mathbf{X} \mathbf{w} - \mathbf{Y} \right) \\
&:= \mathbf{w}_k - \alpha \mathbf{X}^T \left(\text{predicted output} - \text{actual output} \right)
\end{aligned}
$$

This vectorized implementation will make sure that all the parameters are updated in one go.

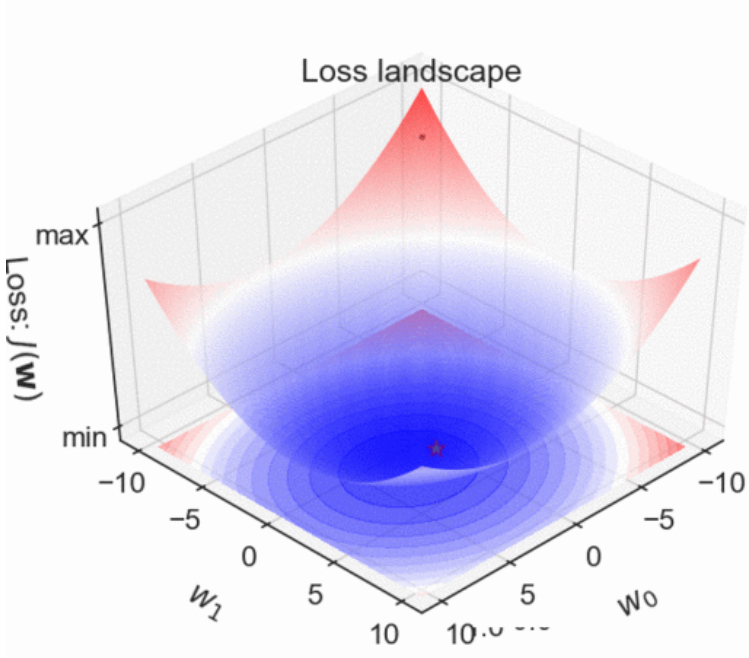
Gradient Descent Visualization

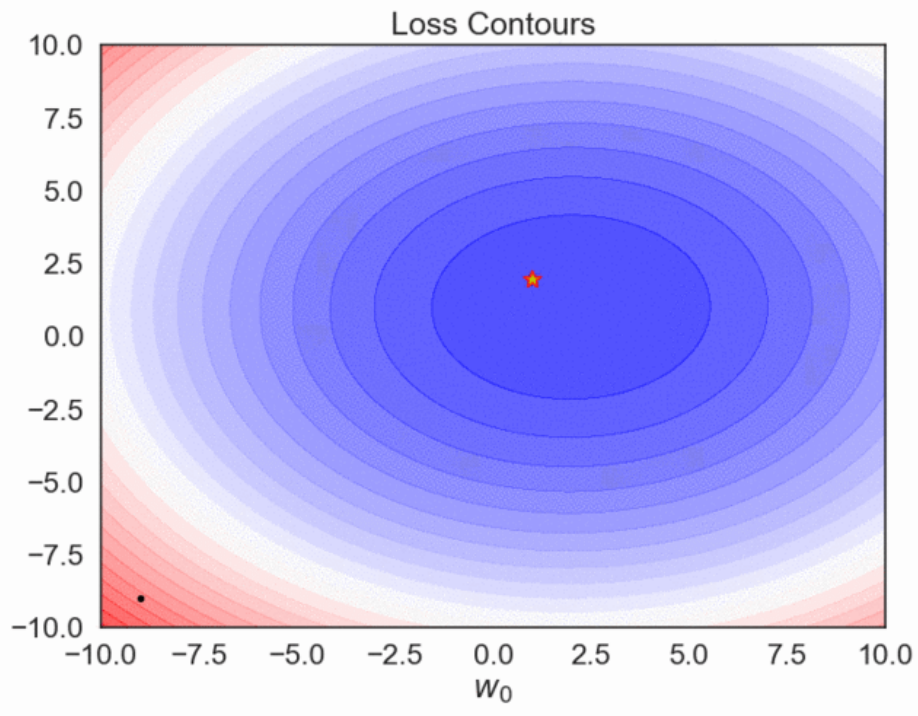
Model

$$h_{\mathbf{w}}(\mathbf{x}) : y = w_0 + w_1 x_1$$

Loss (SSE)

$$J(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} \left((w_0 + w_1 x_1^{(i)}) - y^{(i)} \right)^2$$





Gradient Descent: Salient Features

- GD is a very generic optimization algorithm that can be used for learning weight vectors of most of the ML models.

- GD obtains optimal weight vector by making small changes to their values in each iteration proportional to the gradient of the loss function.

- Once GD reaches the minima of the loss function, we obtain the optimal weights corresponding to the minima.

Gradient Descent Convergence

- The weight vector changes by a very small margin in successive iterations or in last few iterations.

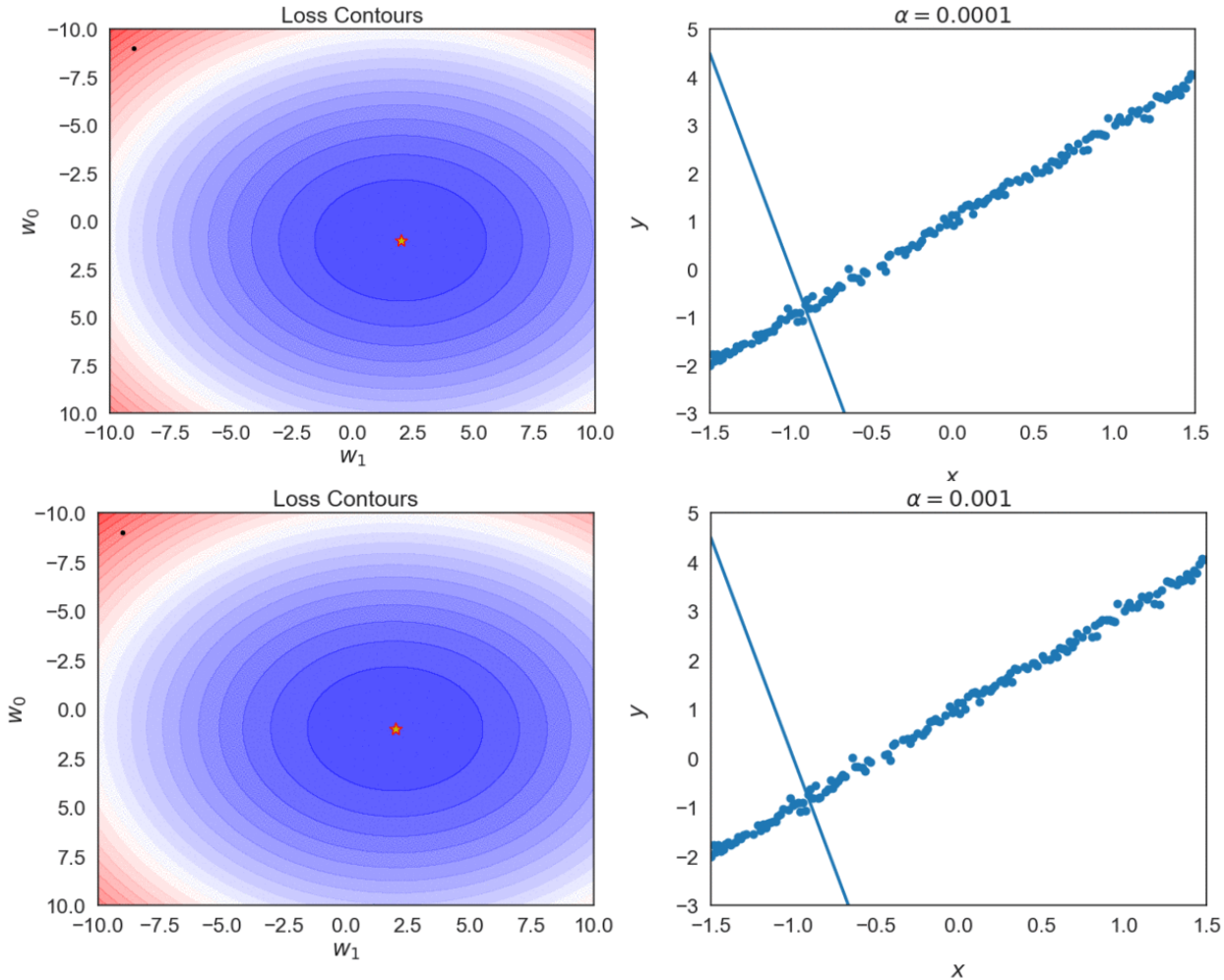
- After completing a fixed number of iterations.

Gradient Descent: Practical considereations

1. How do we set the learning rate α ?

2. How do we decide the number of iterations?

Fixing learning rate α

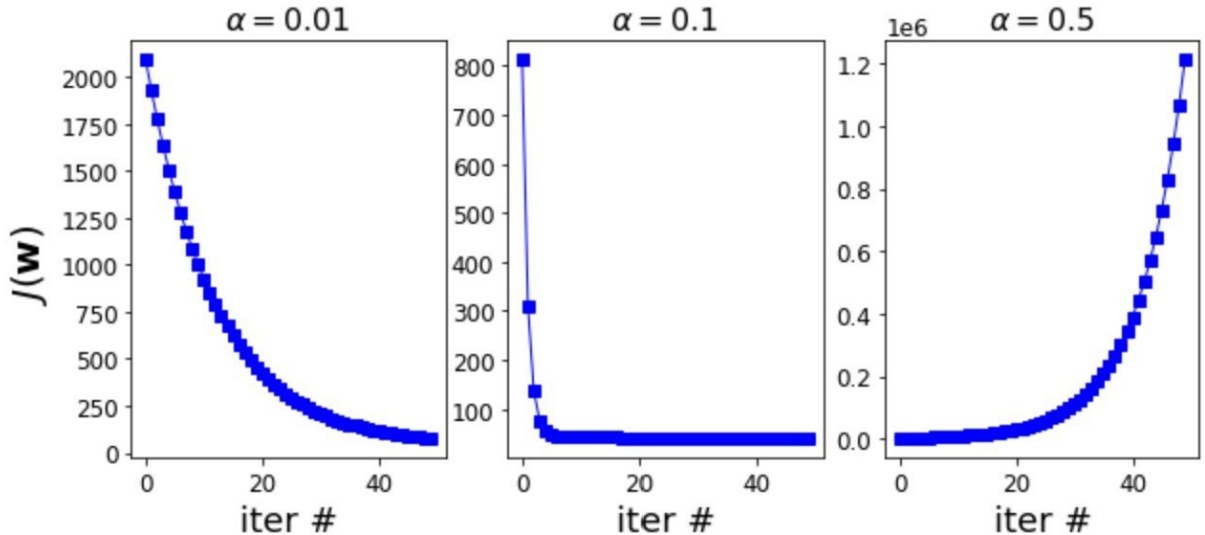


Try different values of α: $\{0.0001, 0.001, 0.01, 0.1, 1\}$

Top one, with $\alpha = 0.0001$ takes longer to reach optimal point, compared with $\alpha = 0.001$

Visualization based diagnosis is not possible for real-word problems with more features, learning curves are our best friends in general case.

Fixing α: Diagnosis based on learning curves



- After first 50 iterations, $\alpha = 0.1$(middle one) has the lowest loss. Loss decreases rapidly.
- The loss for $\alpha = 0.01$ slowly decreases.
- The loss in case of $\alpha = 0.5$ rather increases.

just right α

small α

too large α

Fixing α

- [Small α]
  - Loss either does not reduce or reduces very slowly after initial few iterations points to too small α.
  - Stop the training, increase α (typically by 10x), restart the training and again observe learning curves.

- [Too large α]
  - Loss increases rather than decreasing points to too large α.
  - Stop the training, decrease α (typically by 10x), restart the training and again observe the learning curves.

Fixing the number of iterations

- **Too few iterations:** not enough iterations to reach the optimal solution.
- **Too many iterations:** once we reach the optimal solution, we end up wasting computational cycles as subsequent runs hardly result in any changes in $\mathbf{w}$.
- In practice, we set the number of iterations to a sufficiently large number, but add a convergence criteria which terminates GD loop as soon as the gradient vector becomes smaller than some threshold ϵ.

Variations of GD

A couple of variations for faster weight updates and hence faster convergence. Note that GD uses all n training examples for weight updates:

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \sum_{i=1}^{n} (w_0 + w_1 x_1^{(i)} + \ldots + w_m x_m^{(i)} - y^{(i)}) x_j^{(i)}$$

Mini-batch gradient descent (MBGD)

Uses $k << n$ examples for weight update in each iteration.

Stochastic gradient descent (SGD)

Uses $k = 1$ examples for weight update in each iteration.

Mini batch Gradient Descent (MBGD)

The key time consuming step in GD is the the gradient computation, which performs summation over all training examples:

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = \sum_{i=1}^{n} (w_0 + w_1 x_1^{(i)} + \ldots + w_m x_m^{(i)} - y^{(i)}) x_j^{(i)}$$

- This summation is carried out for each $m+1$ weight in each iteration.
- Exploits the fact that training examples are independent and identically distributed thereby uses $k << n$ examples for faster updates.

Mini batch Gradient Descent (MBGD)

- The key idea is to use a small batch of examples for calculating the gradient in each iteration.
- For that the training set is partitioned into batches of $k << n$ examples.
- In each iteration, MBGD performs the weight update using k examples.
- It needs total of $\frac{n}{k}$ iterations to process the entire training set.
- One full pass over the training set is called an **epoch** and it has $\frac{n}{k}$ iterations and hence performs $\frac{n}{k}$ weight updates.

Mini batch Gradient Descent Algorithm

- Randomly initialize the parameter vector $\mathbf{w}$.
- Iterate until convergence:
  1. **for every batch**:
     1. Calculate gradient of loss function w.r.t. the weights.
     2. Set weights to their new values.
     3. Update weights simultaneously.

All steps are same as GD except *each step processes a small number of examples*.

Stochastic Gradient Descent (SGD)

When we use $k = 1$ in mini-batch GD, it is called stochastic gradient descent (SGD).

Total weight updates after processing full training set once:

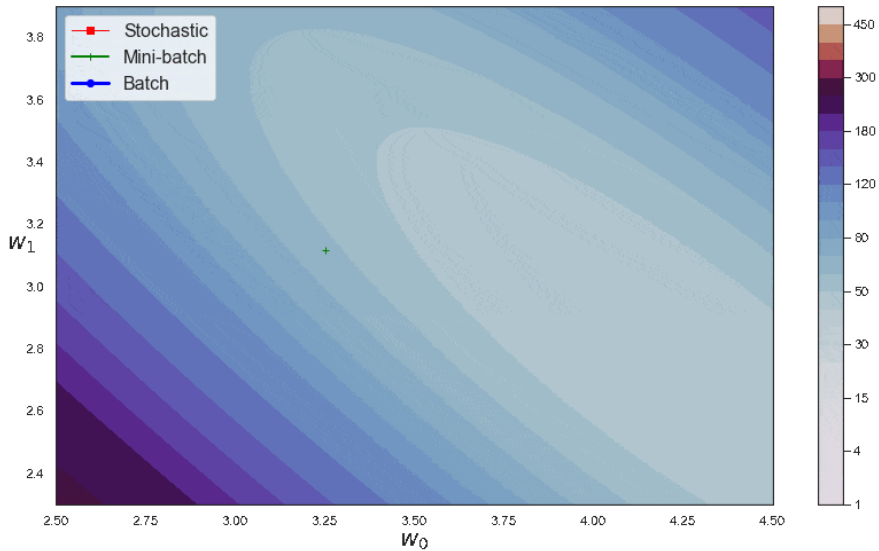
- GD: 1
- MBGD: $\frac{n}{k}$
- SGD: n

GD, MBGD and SGD: Convergence characteristics

- Plot the trajectory of optimization algorithm in the weights' space. In other words, plot the weights as obtained from the first to the last iteration in that order.

GD, MBGD and SGD: Convergence trajectories

Step by step trajectories



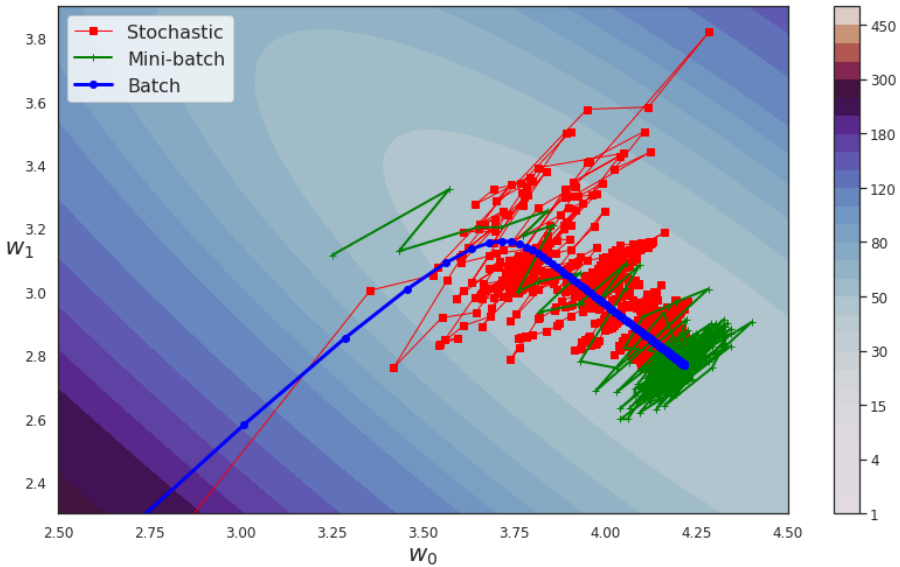
- Batch GD have a smooth path to the minima, though it takes longer time to reach there.

- SGD is more erratic in its path than mini-batch GD. Why?

SGD computes weight updates based on a single example as against k examples used in mini-batch GD.

GD, MBGD and SGD: Convergence trajectories

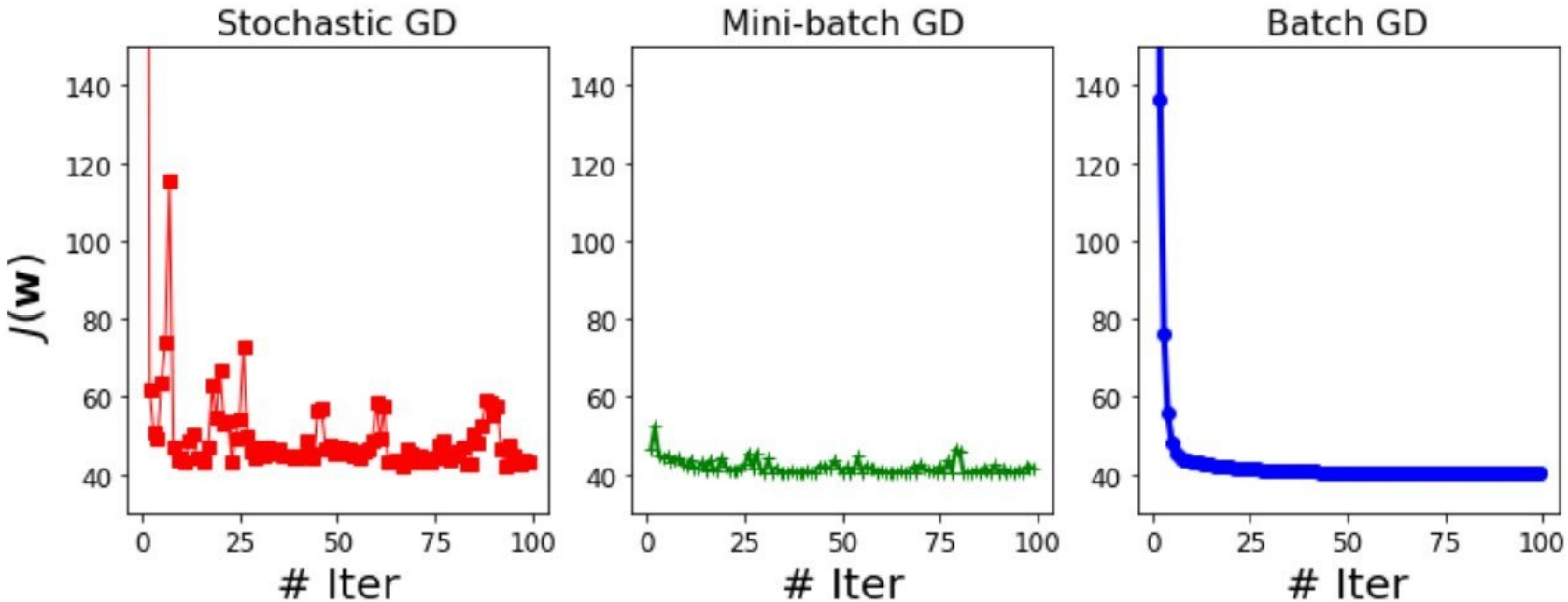
- Observe the trajectory of three variants and compare it with one another.



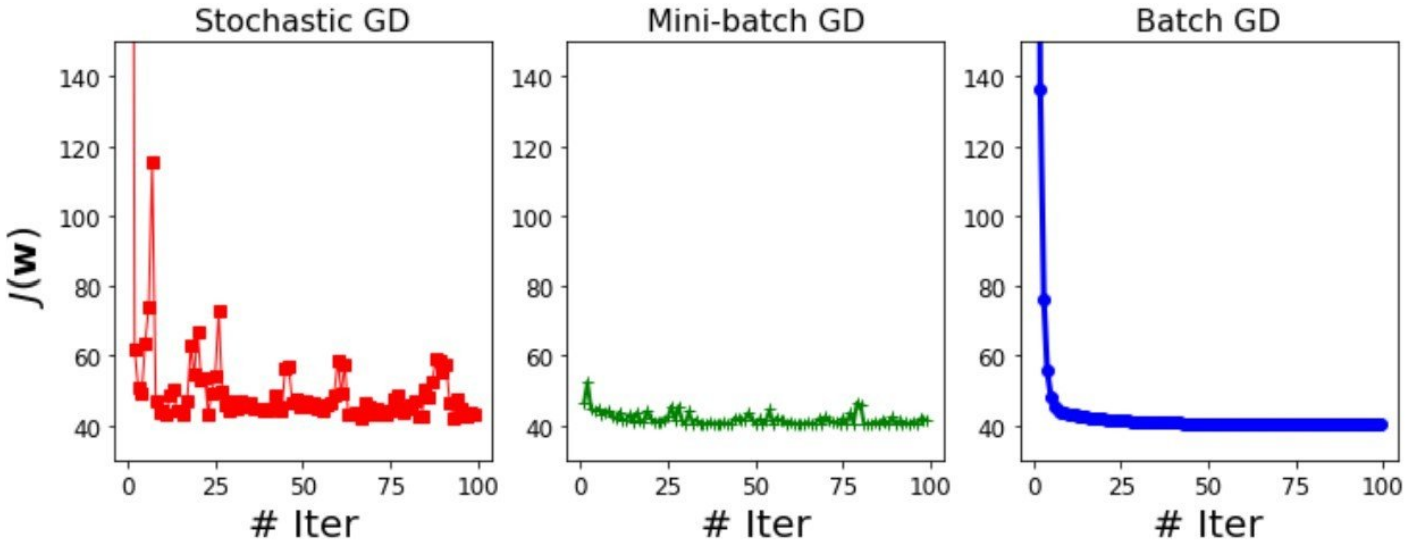
- All optimizers end up around the actual minima.
- Batch GD ends in the minima, while the other two end up around minima. They can reach the minima with appropriate learning schedule.

Visualization based diagnosis is not possible for real-word problems with more features, learning curves are our best friends in general case.

GD vs MBGD vs SGD: Learning curve comparison

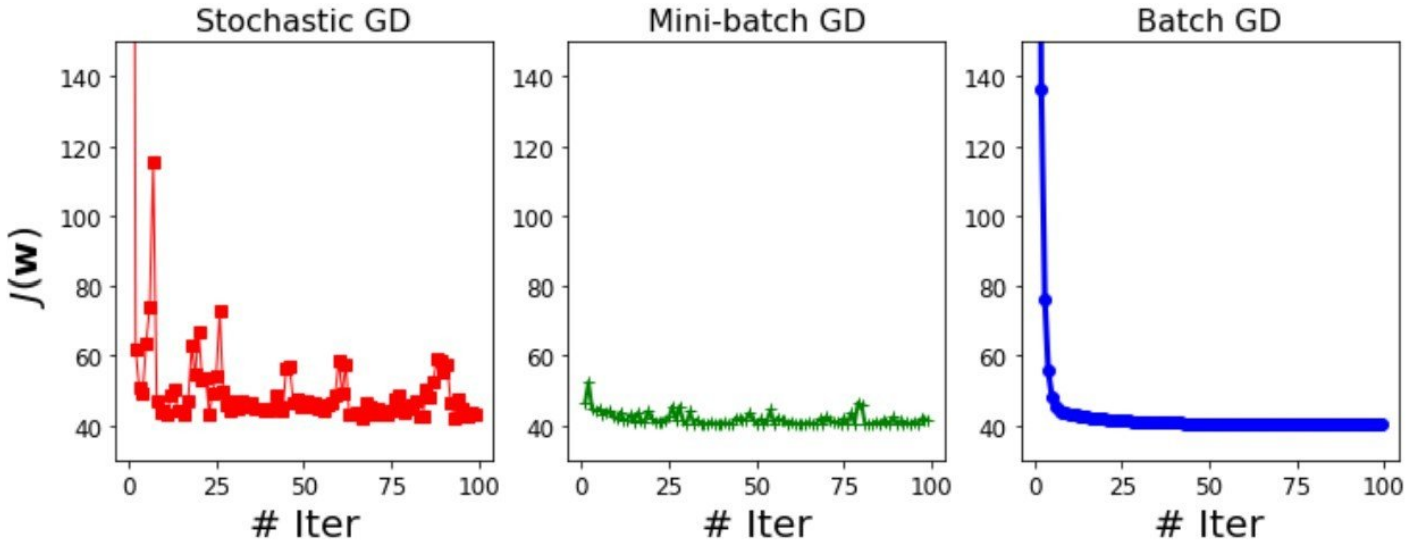


GD vs MBGD vs SGD: Learning curve comparison



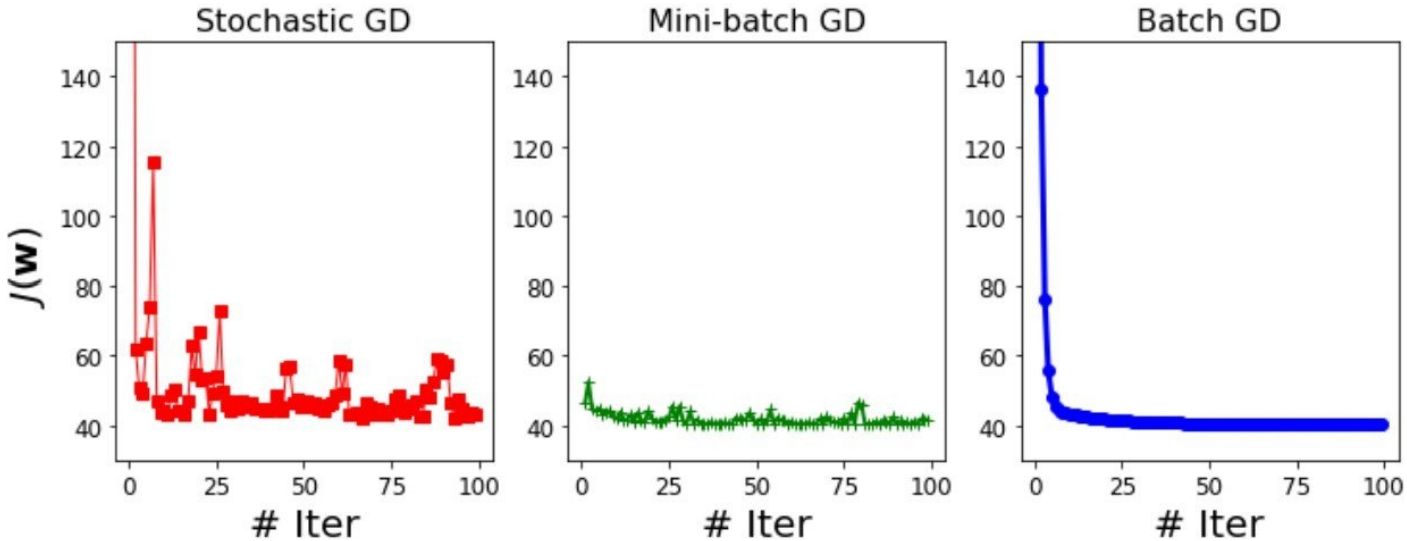
- Batch GD has the smoothest learning curve: the loss reduces continuously iterations after iterations.

GD vs MBGD vs SGD: Learning curve comparison



- The learning curves corresponding to mini-batch GD and SGD show ups and downs in loss.
- In some iteration, the loss reduces and then in the next one it goes up.
- Overall the losses are on downward trajectory.

GD vs MBGD vs SGD: Learning curve comparison



- As the batch size increases, the resulting learning curves become smoother for a given configuration of hyper-parameters like learning rate.

Evaluation

Evaluation

Uses **root-mean-squared-error (RMSE)** measure, which is derived from SSE.

$$\mathrm{SSE} = \frac{1}{2}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^T\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)$$

$$\begin{aligned}\mathrm{RMSE} &= \sqrt{\frac{2}{n}\mathrm{SSE}} \\ &= \sqrt{\frac{1}{n}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^T\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)} \\ &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}^{(i)} - y^{(i)}\right)^2}\end{aligned}$$

Square-root is used to compare the error on the same unit and the same scale as that of the output label.

Division by n, the size of the dataset, enables us to compare model performance on datasets of difference sizes on the same footing.

Linear regression: Recap

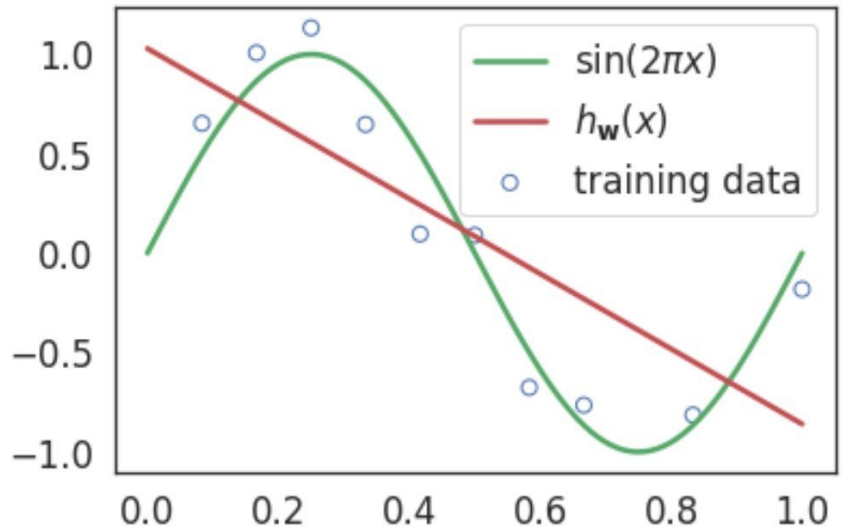
(1) Data	Features and label that is real number.
(2) Model	Linear combination of features
(3) Loss function	Sum of squared error (SSE)
(4) Optimization procedure	(1) Normal eq. (2) GD/MBGD/SGD
(5) Evaluation	Root mean squared error (RMSE)

Linear regression: Recap

(1) Data	$D = \{(\mathbf{X}, \mathbf{y})\} = \left\{(\mathbf{x}^{(i)}, y^{(i)})\right\}_{i=1}^{n}$
(2) Model	$h_{\mathbf{w}} : \mathbf{y} = \mathbf{X}\mathbf{w}$
(3) Loss function	$J(\mathbf{w}) = \frac{1}{2}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})$
(4) Optimization procedure	(1) Normal eq. (2) GD/MBGD/SGD
(5) Evaluation	$\mathrm{RMSE} = \sqrt{\frac{1}{n}(\mathbf{X}\mathbf{w} - \mathbf{y})^T(\mathbf{X}\mathbf{w} - \mathbf{y})}$

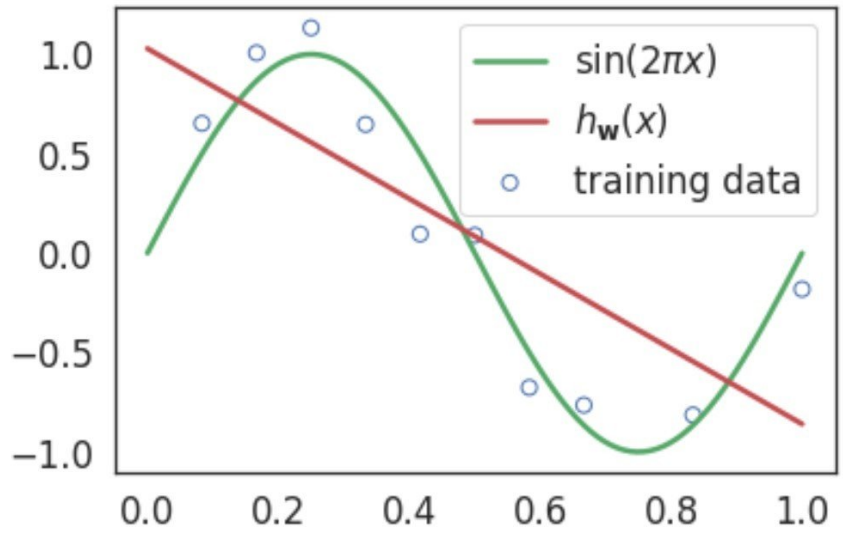
Part II: Polynomial Regression

Why polynomial regression?



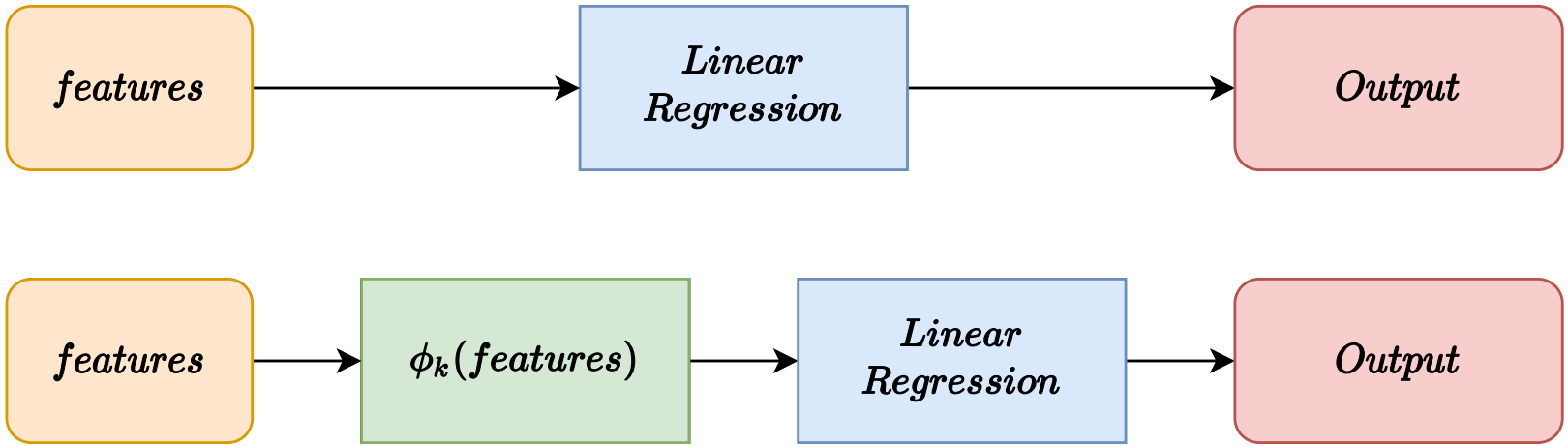
Many times the relationship between the input features and the output label is non-linear and simple linear models are not adequate to learn such mappings.

Polynomial regression: key idea



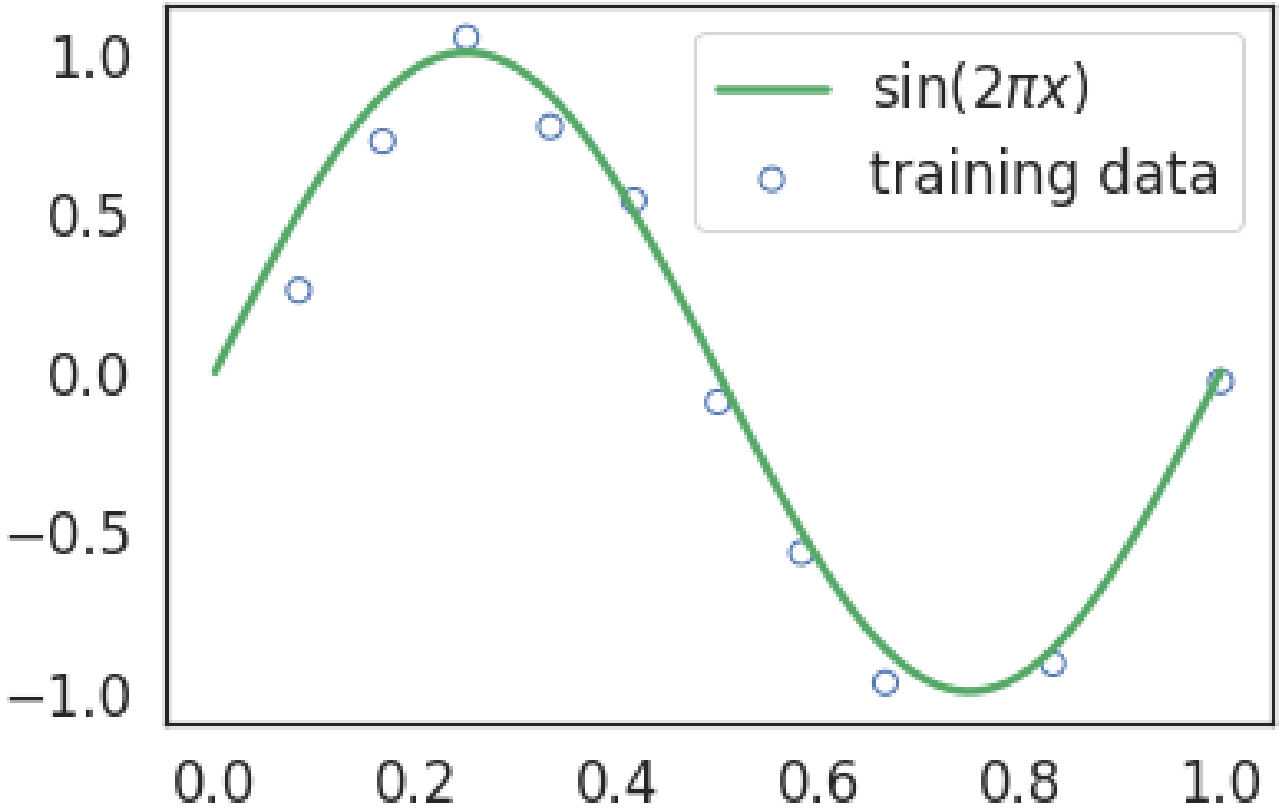
- The key idea is to create polynomial features by combining the existing input features.
- And then apply linear regression model on the polynomial feature representation.

Comparison: Linear and polynomial regression



ϕ_k(features) is called **polynomial transformation** of k-th order.

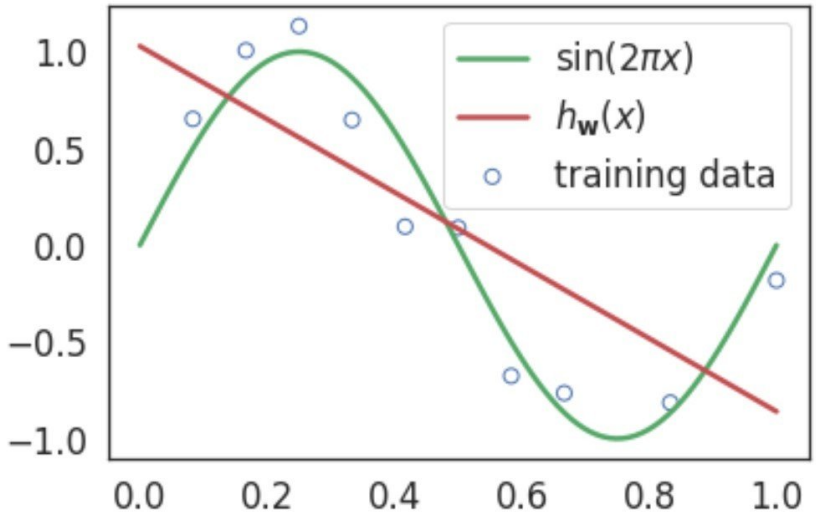
Polynomial regression: sample data



Why Polynomial Regression?

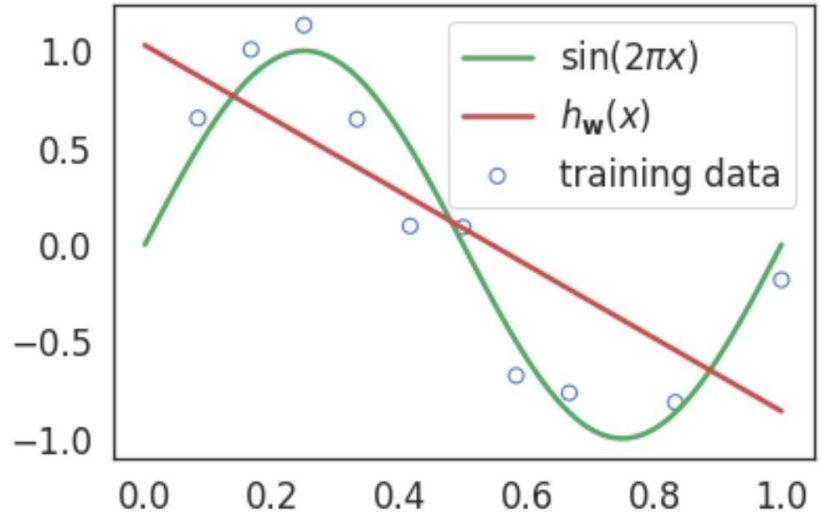
Linear regression without polynomial transformation.

$$h_{\mathbf{w}}(x) = w_0 + w_1 x$$



Clearly, this is not a great fit and we need to search for a better model.

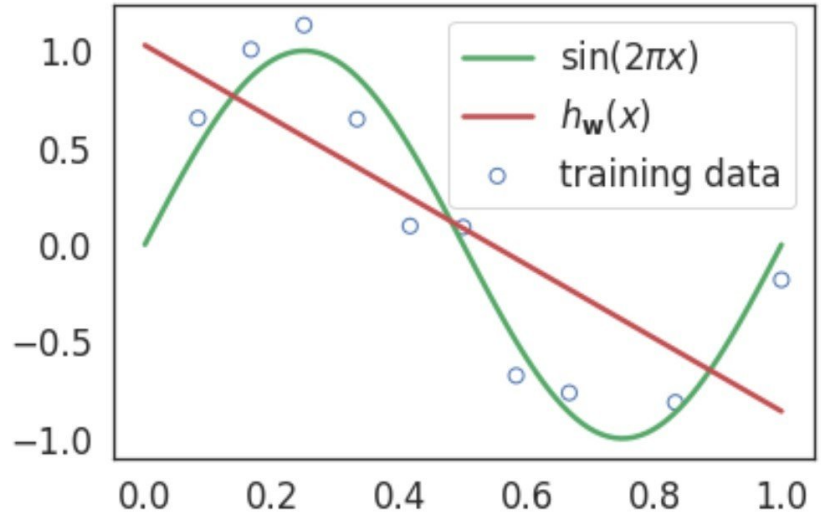
Fitness diagnostics



In this case, we are able to visualize the model fitment since we are dealing with data in 1D feature space.

As the number of features grow, it won't be possible for us to visualize the fitment in this manner.

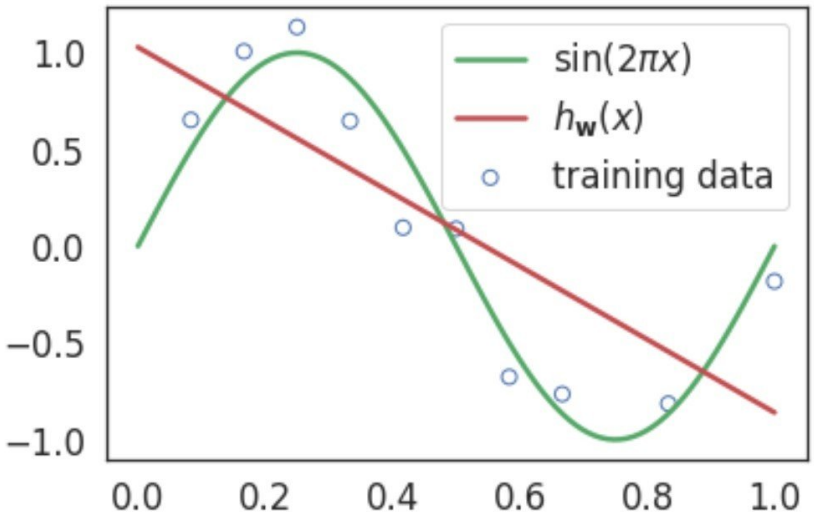
Fitness diagnostics

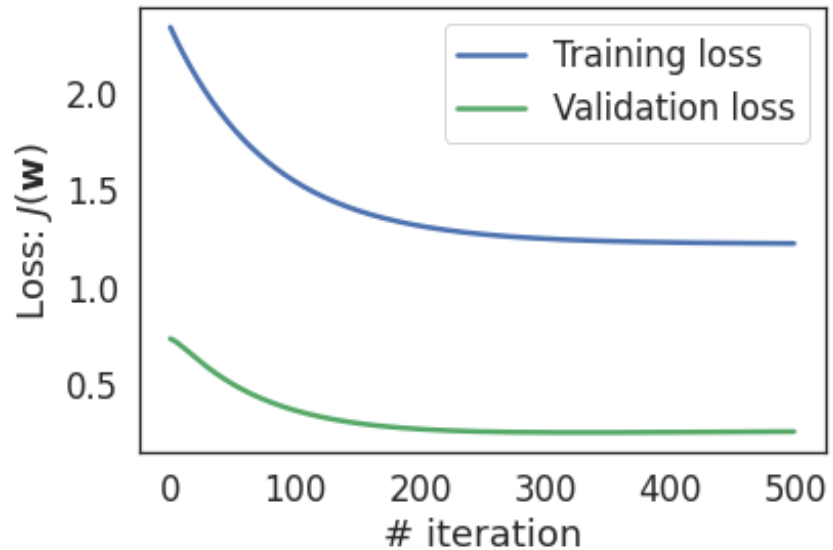


We rely on learning curve to determine quality of the fitment:

- Underfitting - both training and validation losses are high.
- Overfitting - training and validation losses decrease initially but then validation loss increases while training loss keeps decreasing.

Learning curve of linear regression





Linear model underfits: it is not enough to model the relationship between features and labels as present in the training data.

In this case, validation loss is less than the training loss (which is unusal), this is due to small amount of the validation data.

How can we fix underfitting?

- By increasing the capacity of the model to learn non-linear relationship between the features and the label.

- One way to achieve it is through polynomial transformation that constructs new features from the existing features.

Polynomial regression in single feature

For training data with a single feature x_1,

- First use k-th order polynomial transformation to create new features like $x^2, x^3, \ldots, x^k$ and
- Use linear regression model to learn relationship between x_1 and y.

$$
\begin{aligned}
y &= w_0 + w_1 x_1 + w_2 x_1^2 + \ldots + w_k x_1^k \\
&= w_0 1 + w_1 x_1 + w_2 x_1^2 + \ldots + w_k x_1^k \\
&= w_0 {x_1}^0 + w_1 x_1 + w_2 x_1^2 + \ldots + w_k x_1^k \\
&= \sum_{i=0}^{k} w_i x_1^i
\end{aligned}
$$

Note that the model is a non-linear function of x, but is a linear function of weight vector $\boldsymbol{w} = [w_0, w_1, \ldots, w_k]$.

Polynomial transformation

Let's represent this transformation in form of a vector ϕ with k components.

$$\phi = \begin{bmatrix} \phi_0 \\ \phi_1 \\ \vdots \\ \phi_k \end{bmatrix}$$

Each component denotes a specific transform to be applied to the input feature.

The polynomial regression model becomes:

$$y = \sum_{i=0}^{k} w_i (\phi(\mathbf{x}))_i$$

$$y = \mathbf{w}^T \phi(\mathbf{x})$$

Polynomial transformation: examples

- For a single feature x_1, $\phi_k = x_1^k$
- Two features (x_1, x_2), $\phi_2(x_1, x_2) = \left[1, x_1, x_2, x_1^2, x_2^2, x_1 x_2\right]$.
- Two features (x_1, x_2), $\phi_3(x_1, x_2) = \left[1, x_1, x_2, x_1^2, x_2^2, x_1 x_2, x_1^3, x_2^3, x_1 x_2^2, x_1^2 x_2\right]$
- ϕ_3 contains features from ϕ_2.

Polynomial Transformation: Implementation

```python
import itertools
import functools

def polynomial_transform(x, degree):
  '''Performs transformation of input x into deg d polynomial features.

  Arguments:
    x: Data of shape (n,)
    degree: degree of polynomial

  Returns:
    Polynomial transformation of x
  '''
  if x.ndim == 1:
    x = x[:, None]

  x_t = x.transpose()
  features = [np.ones(len(x))]
  for degree in range(1, degree + 1):
    for items in itertools.combinations_with_replacement(x_t, degree):
      features.append(functools.reduce(lambda x, y: x * y, items))
  return np.asarray(features).transpose()
```

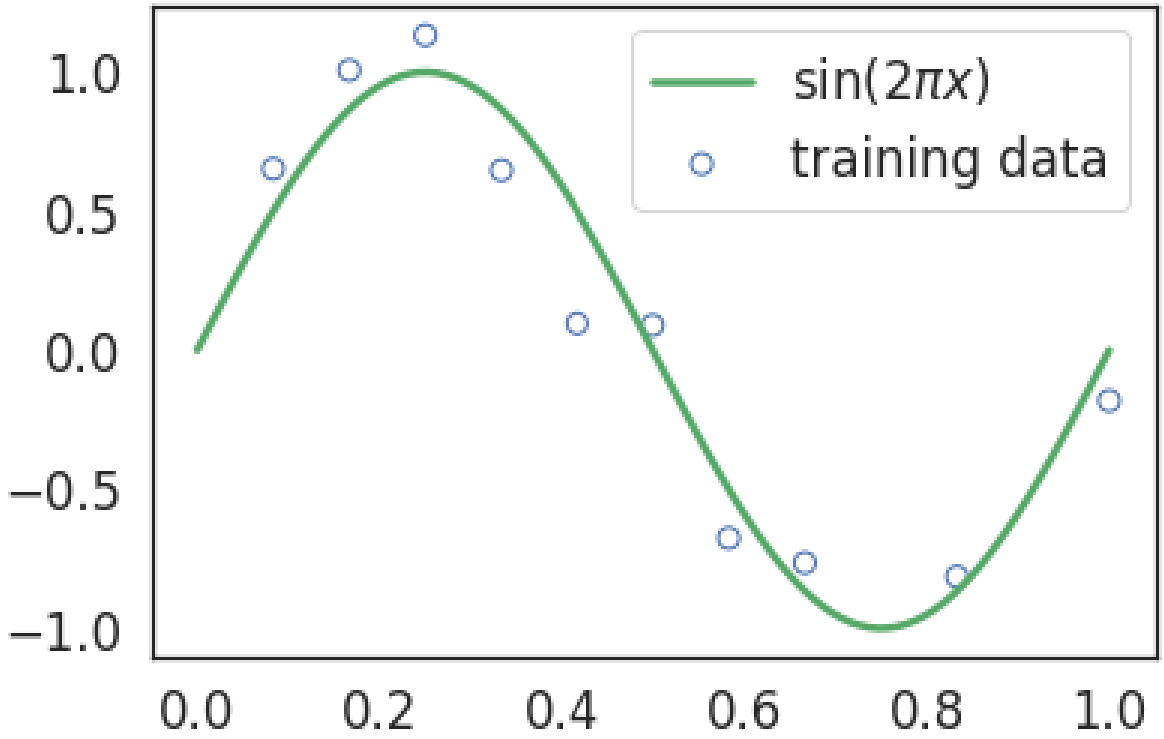
Polynomial transformation: implementation

An example:

```
polynomial_transform(np.array([[1, 2], [3, 4]]), degree=3)
```

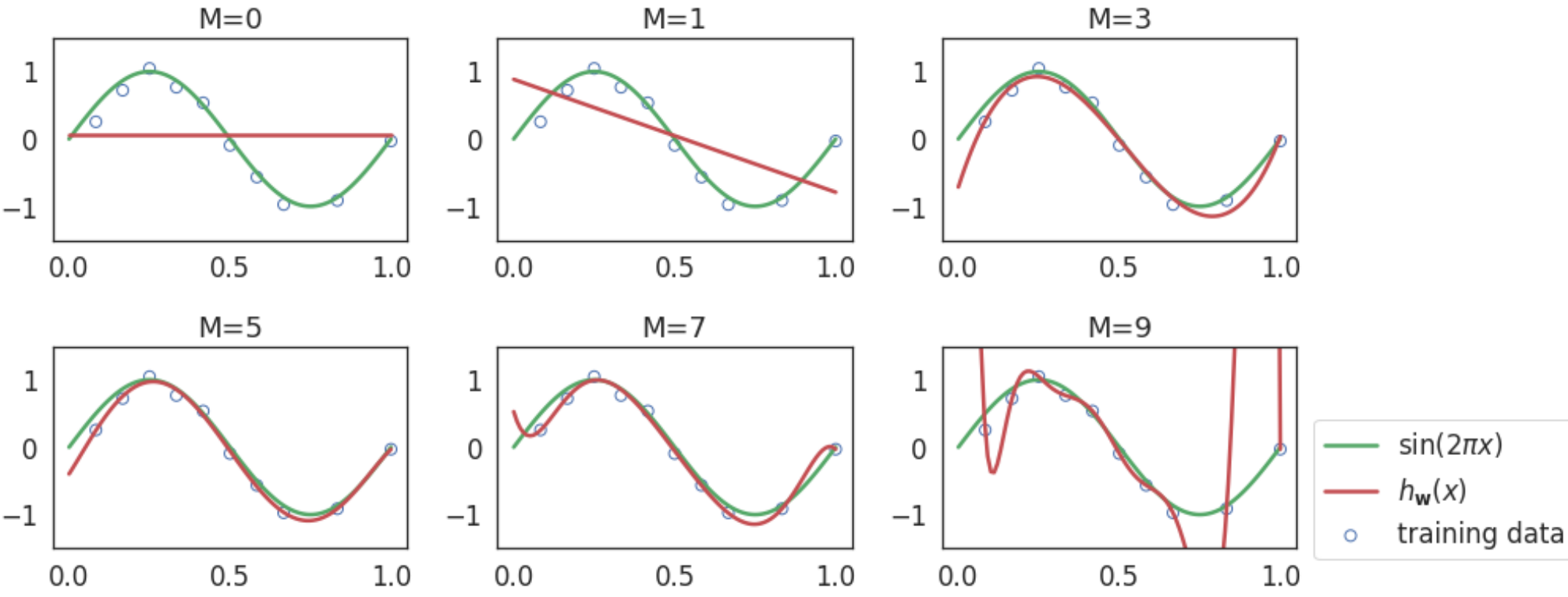
Output:

```
array([[ 1.,  1.,  2.,  1.,  2.,  4.,  1.,  2.,  4.,  8.],
       [ 1.,  3.,  4.,  9., 12., 16., 27., 36., 48., 64.]])
```

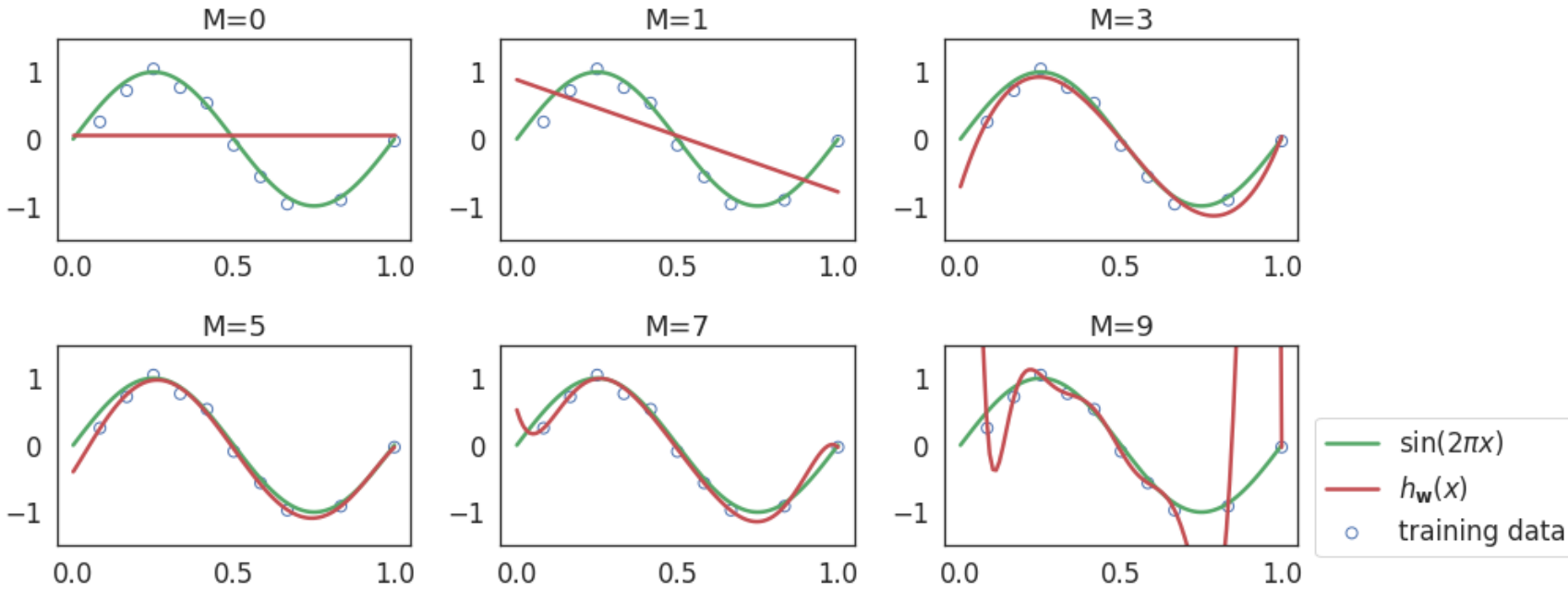


Let's fit a polynomial model of different orders $k = \{0, 1, 2, \ldots, 9\}$ on this data.

Fitting Polynomial Regression Models



Fitting Polynomial Regression Models



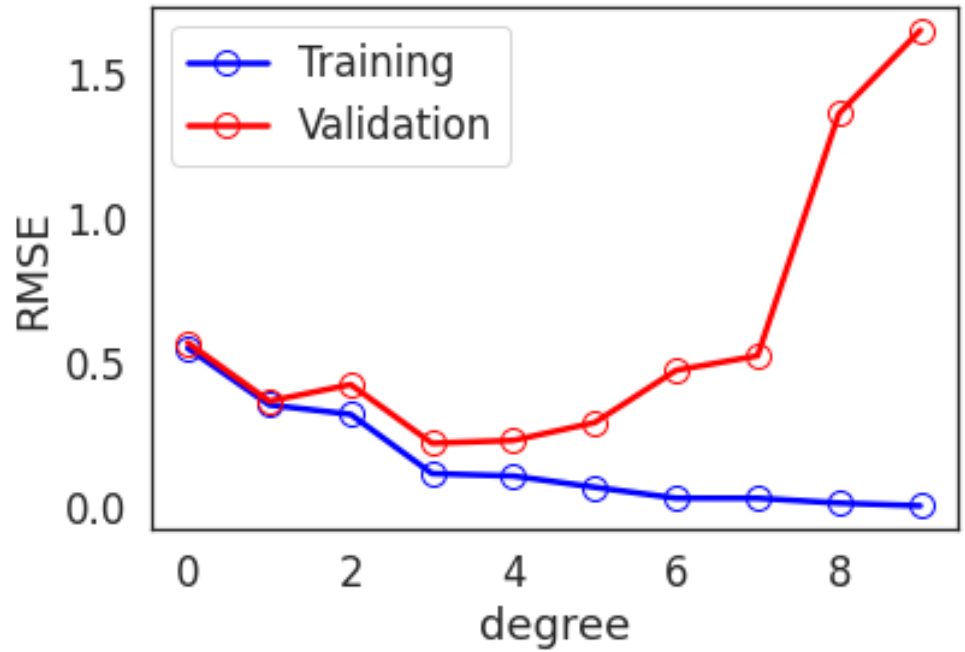
- Lower degree (0 and 1) polynomial model: Underfitting.
- Third order (degree=3) polynomial model: Reasonable fit.
- Higher degree ≥ 7 polynomial model: Overfitting.

Model diagnostics: learning curves

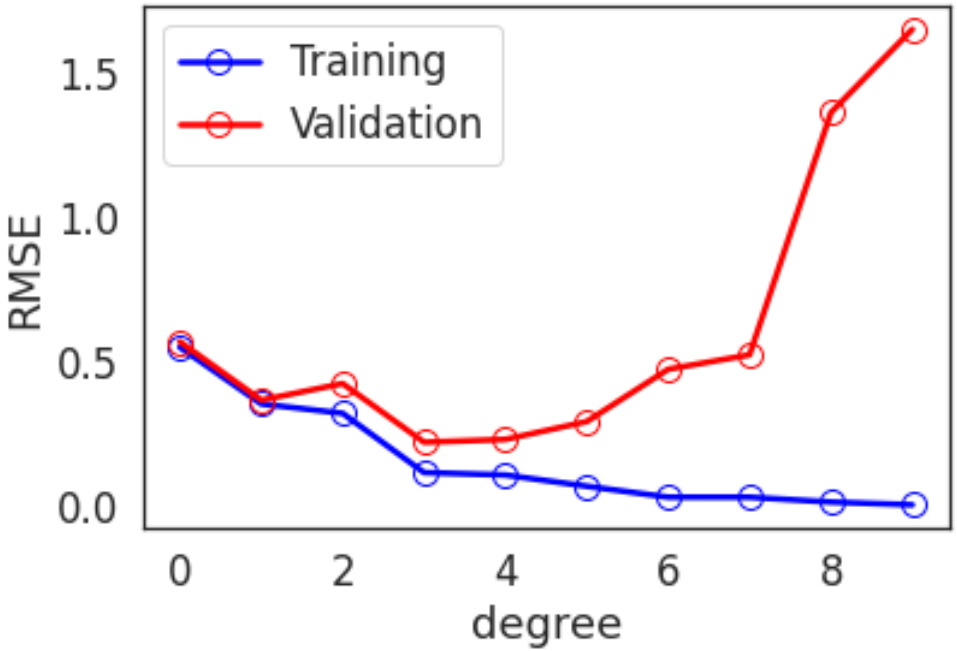
(1) Train polynomial model of degrees: $\{0, 1, \ldots, k\}$

(2) Calculate training and validation RMSE.

(3) Plot degree vs. RMSE plot.



Model diagnostics: learning curves



- The training and validation errors are close until certain degree.
- After a point, the training error continues to reduce, while the validation error keeps increasing.
- RMSE increases sharply for degrees $\geq$ 7. This is a signature of overfitting.

Issues with polynomial regression

- Higher order polynomial models are very flexible, or in other words, they have higher capacity compared to lower order models.

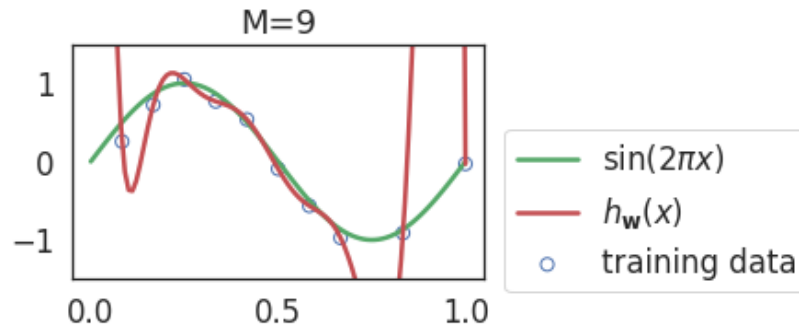
- Hence they are prone to overfitting compared to the lower degree polynomials.

- Perfect fit to training data, but poor prediction accuracy on validation data.

Demonstration of overfitting: deg = 9

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \\ w_8 \\ w_9 \end{bmatrix} = \begin{bmatrix} 1.76 \\ -115.74 \\ 2458.97 \\ -21739.10 \\ 103583.95 \\ -293329.12 \\ 507610.57 \\ -525717.33 \\ 298590.02 \\ -71343.99 \end{bmatrix}$$

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{j=0}^{j=9} w_j x^j$$

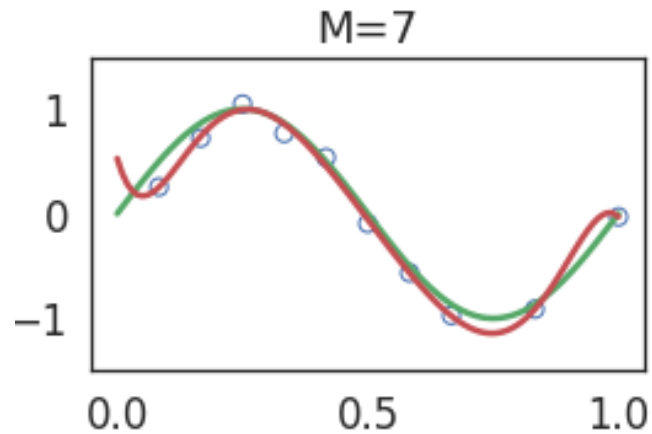


Observe that higher degree polynomial features have higher weights than others.

Demonstration of overfitting: deg = 7

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \\ w_7 \end{bmatrix} = \begin{bmatrix} -4.23 \\ 99.31 \\ -641.24 \\ 1988.00 \\ -3332.88 \\ 2993.67 \\ -1304.04 \\ 201.43 \end{bmatrix}$$

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{j=0}^{j=7} w_j x^j$$



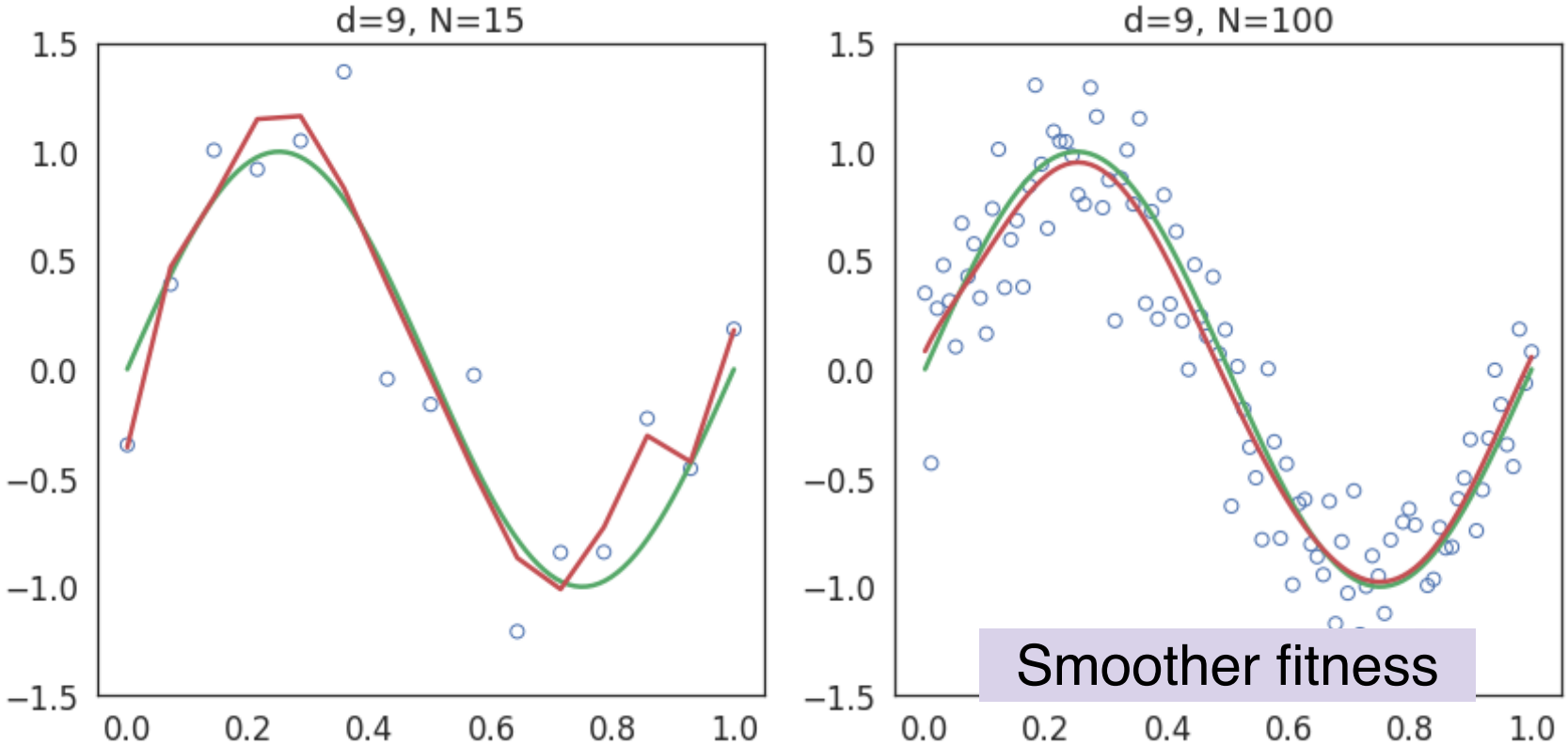
Observe that higher degree polynomial features have higher weights than others.

How to address overfitting?

- Using larger training sets.
- Controlling model complexity through regularization, which allows us to fit complex model on relatively smaller dataset without overfitting.

Fixing overfitting with more data

Train a polynomial model of degree 9 with two datasets - (i) with 15 points and (ii) with 100 points.



Fixing overfitting through regularization

Overfitting is caused by larger weights assigned to the higher order polynomial terms.

Regularization: Add a penalty term in the loss function for such large weights to control this behaviour.

At a high level, let's modifies loss function by adding a penalty term as follows:

$$J(\mathbf{w}) = \frac{1}{2}\left(\mathbf{X}\mathbf{w} - \mathbf{Y}\right)^T\left(\mathbf{X}\mathbf{w} - \mathbf{Y}\right) + \lambda\,\mathrm{penalty}$$

Regularization modifies the loss function, which leads to the change in derivative of the loss function and hence a change in the weight update rule in gradient descent procedure.

Two components of regularization

$$J(\mathbf{w}) = \frac{1}{2}\left(\mathbf{X}\mathbf{w} - \mathbf{Y}\right)^T\left(\mathbf{X}\mathbf{w} - \mathbf{Y}\right) + \lambda\,\mathrm{penalty}$$

- Penalty is a function of weight vector

- Regularization rate λ controls amount of penalty to be added to the loss function.

Regularization types

- L2 regularization (Ridge regression)
- L1 regularization (Lasso regression)
- Combination of L1 and L2: Elastic net regularization

Ridge Regression

Ridge regression uses second norm of the weight vector as a penalty term:

$$J(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)})^2 + \frac{\lambda}{2}\sum_{i=1}^{m} w_i^2$$

This can be written in a vectorized form as follows

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{2}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^T\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) + \frac{\lambda}{2}\|\mathbf{w}\|_2 \\ &= \frac{1}{2}\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^T\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \end{aligned}$$

Gradient calculation in ridge regression

$$
\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial}{\partial \mathbf{w}} \left\{ \frac{1}{2} \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^T \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w} \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \left((\mathbf{X}\mathbf{w})^T - \mathbf{y}^T \right) \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) + \lambda \mathbf{w}^T \mathbf{w} \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \left(\mathbf{w}^T \mathbf{X}^T - \mathbf{y}^T \right) \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) + \lambda \mathbf{w}^T \mathbf{w} \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \mathbf{w}^T \mathbf{X}^T \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) - \mathbf{y}^T \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) + \lambda \mathbf{w}^T \mathbf{w} \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \left(\mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} \right) - \left(\mathbf{y}^T \mathbf{X}\mathbf{w} - \mathbf{y}^T \mathbf{y} \right) + \lambda \mathbf{w}^T \mathbf{w} \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\mathbf{w} + \mathbf{y}^T \mathbf{y} + \lambda \mathbf{w}^T \mathbf{w} \right\} \\
&= \frac{1}{2} \frac{\partial}{\partial \mathbf{w}} \left\{ \mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} + \mathbf{y}^T \mathbf{y} + \lambda \mathbf{w}^T \mathbf{w} \right\}
\end{aligned}
$$

These steps are same as gradient calculation in linear regression except for the penalty term.

Gradient calculation in ridge regression

$$
\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \frac{1}{2}\frac{\partial}{\partial \mathbf{w}}\left\{\mathbf{w}^T\mathbf{X}^T\mathbf{w}\mathbf{X} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{y} + \mathbf{y}^T\mathbf{y} + \lambda\mathbf{w}^T\mathbf{w}\right\} \\
&= \frac{1}{2}\left\{\frac{\partial}{\partial \mathbf{w}}\left\{\mathbf{w}^T\mathbf{X}^T\mathbf{w}\mathbf{X}\right\} - \frac{\partial}{\partial \mathbf{w}}\left\{2\mathbf{w}^T\mathbf{X}^T\mathbf{y}\right\} + \frac{\partial}{\partial \mathbf{w}}\left\{\mathbf{y}^T\mathbf{y}\right\} + \frac{\partial}{\partial \mathbf{w}}\left\{\lambda\mathbf{w}^T\mathbf{w}\right\}\right\} \\
&= \frac{1}{2}\left\{2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} + 0 + 2\lambda\mathbf{w}\right\} \\
&= \frac{1}{2}\left\{2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{y} + 2\lambda\mathbf{w}\right\} \\
&= \frac{1}{2} \times 2 \times \left\{\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} + \lambda\mathbf{w}\right\} \\
&= \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} + \lambda\mathbf{w}
\end{aligned}
$$

Normal Equation

Let's set $\dfrac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$ to 0 and solving for $\mathbf{w}$:

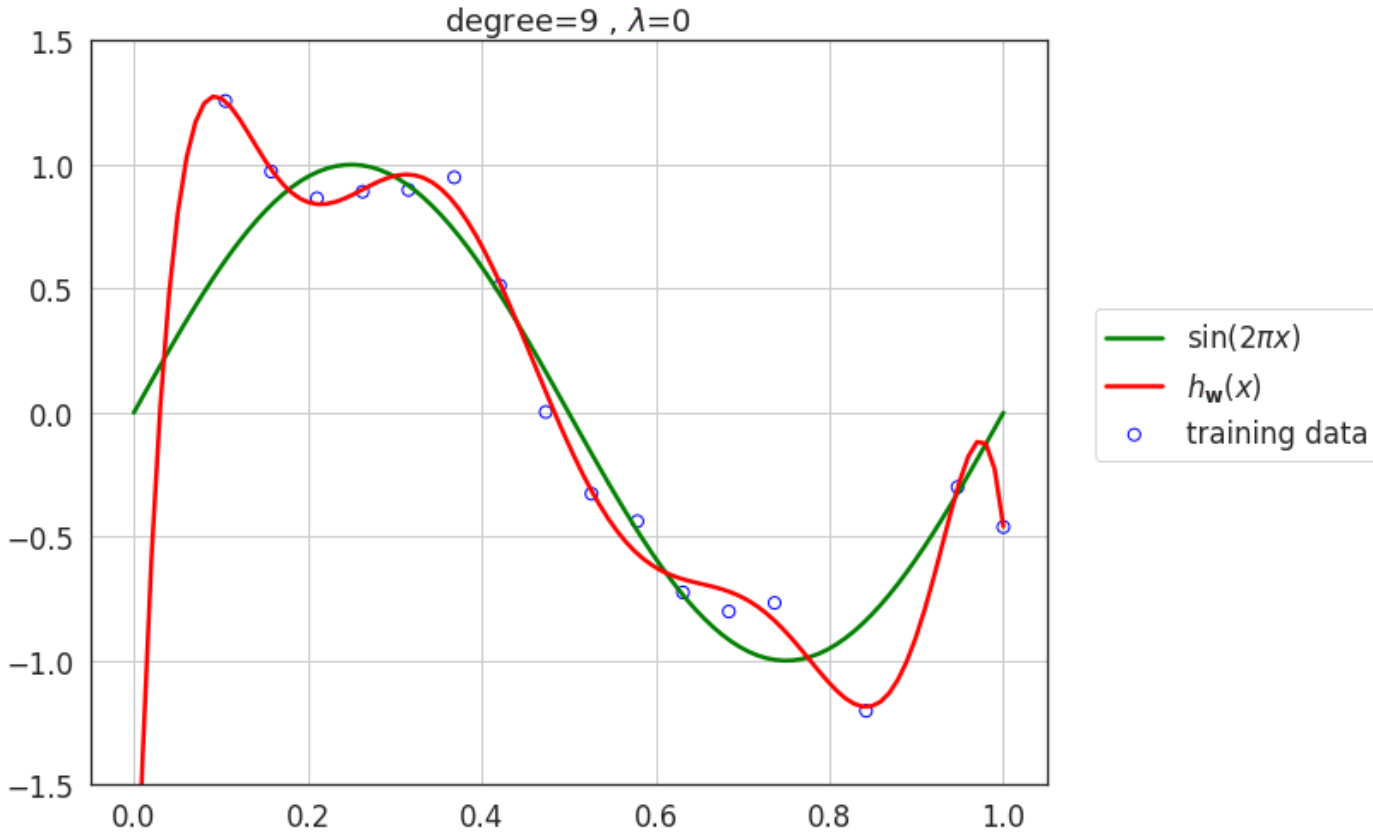
$$
\begin{aligned}
\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} + \lambda\mathbf{w} = 0 \\
\mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w} &= \mathbf{X}^T\mathbf{y} \\
\left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)\mathbf{w} &= \mathbf{X}^T\mathbf{y} \\
\mathbf{w} &= \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y}
\end{aligned}
$$

- [No regularization] When $\lambda \to 0$, $\mathbf{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$ (Same as the linear regression solution)
- [Infinite regularization] When $\lambda \to \infty$, $\mathbf{w} = 0$

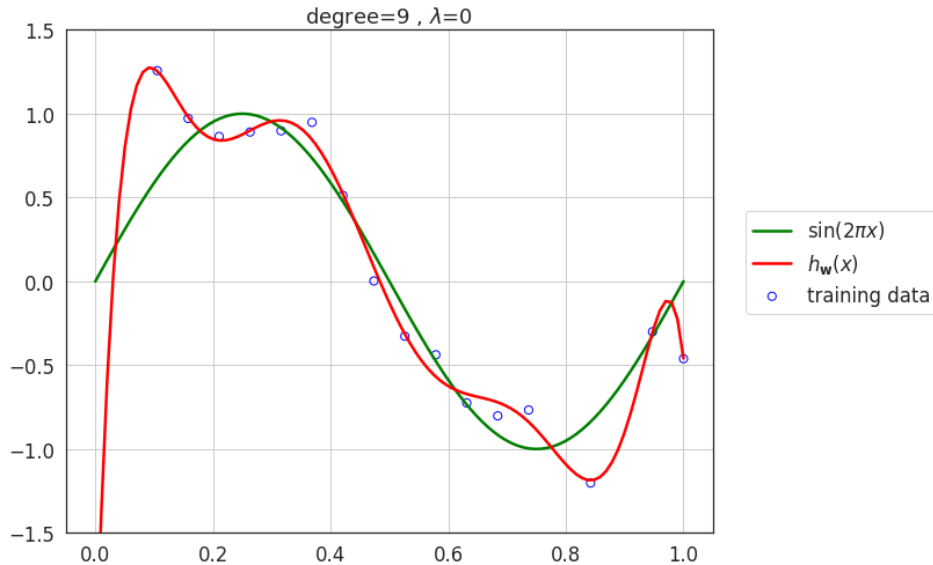
Gradient Descent Update Rule

$$
\begin{aligned}
\mathbf{w}_{k+1} &:= \mathbf{w}_k - \alpha \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \\
&:= \mathbf{w}_k - \alpha \left(-\mathbf{X}^T\mathbf{y} + \mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w}\right) \\
&:= \mathbf{w}_k - \alpha \left(\mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{y} + \lambda\mathbf{w}\right) \\
&:= \mathbf{w}_k - \alpha \left\{\mathbf{X}^T\left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) + \lambda\mathbf{w}\right\}
\end{aligned}
$$

Effect of regularization λ



Effect of regularization λ

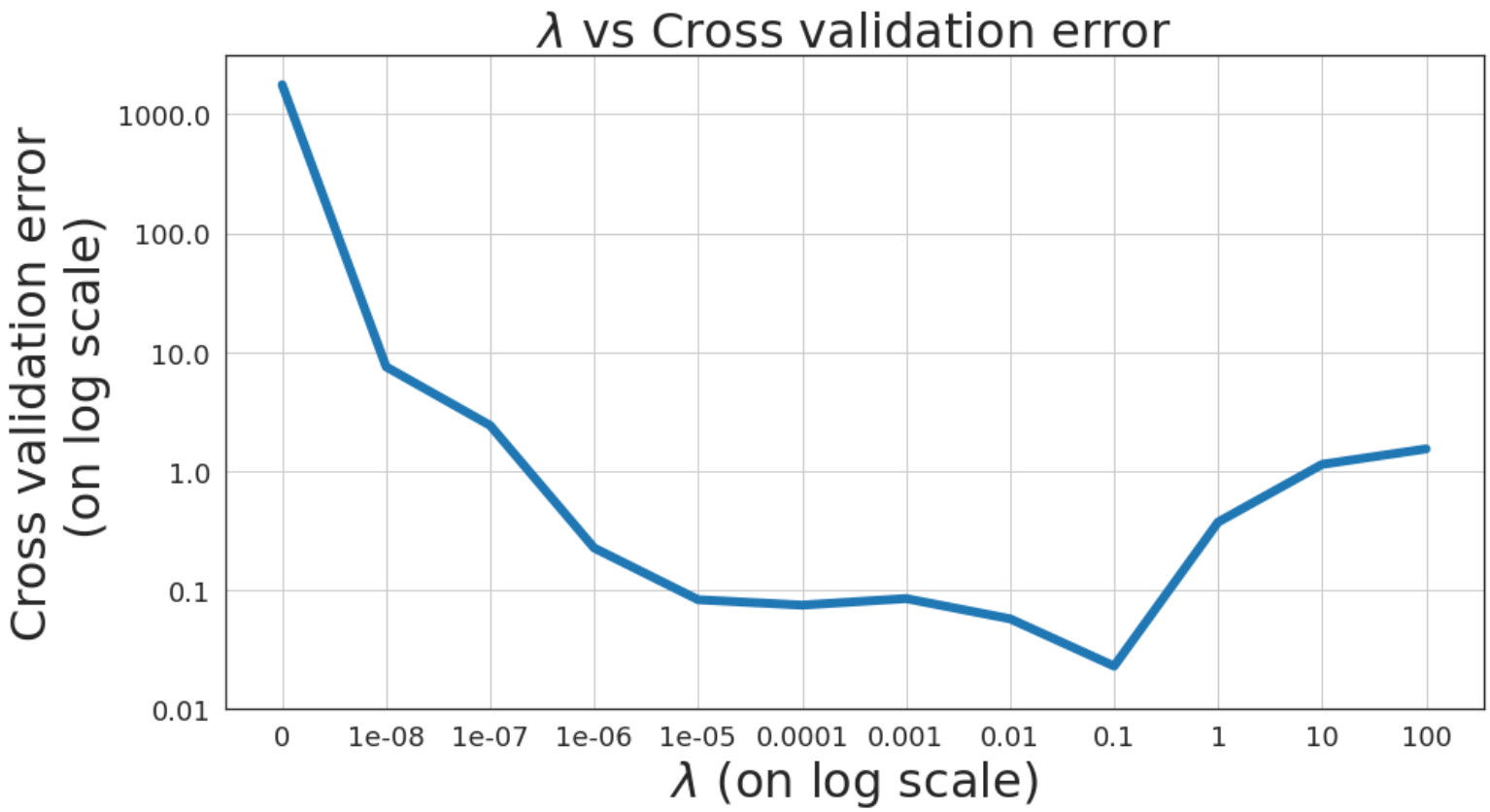


- As value of λ increases, the model loses its capacity and eventually for very large values of λ, the model underfits the data.

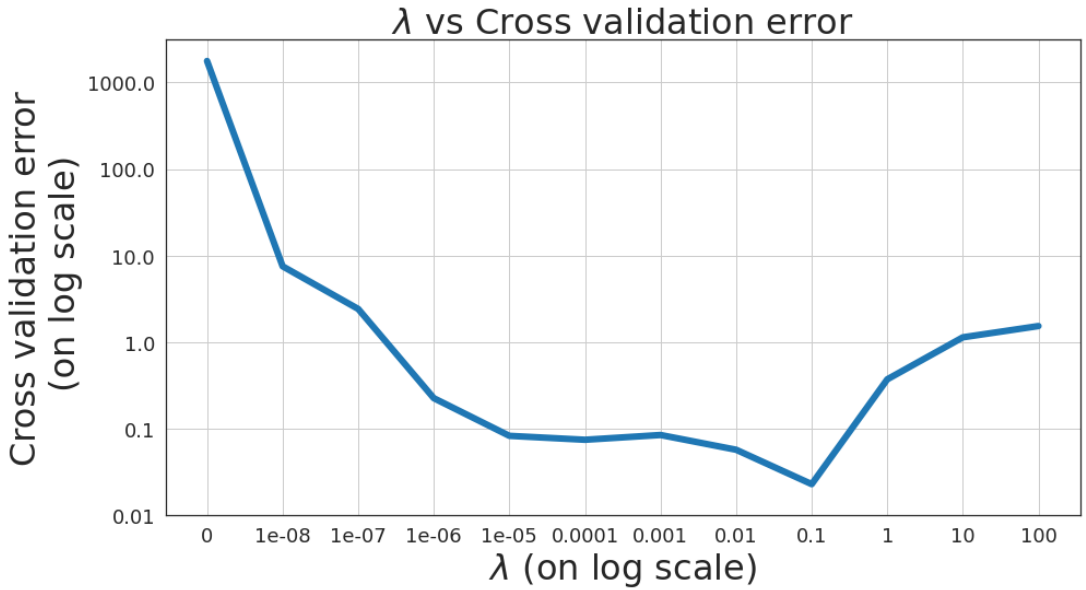
Choosing λ

1. Construct a set of values of λ that we want to experiment with.
2. For each candidate value of λ:
    1. Train the model and calculate cross validation error on validation set.
    2. Choose the the value of λ which results in the least cross validation error.
3. With the chosen value of λ, train the model on entire training set.
4. Report the model performance on the test set.

λ vs Cross validation error

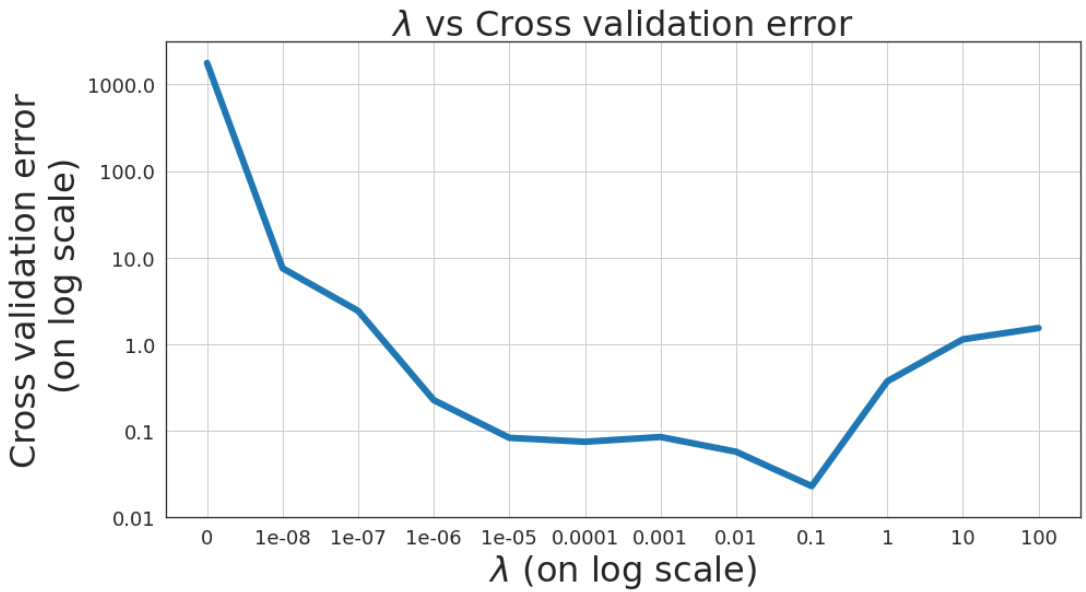


λ vs Cross validation error



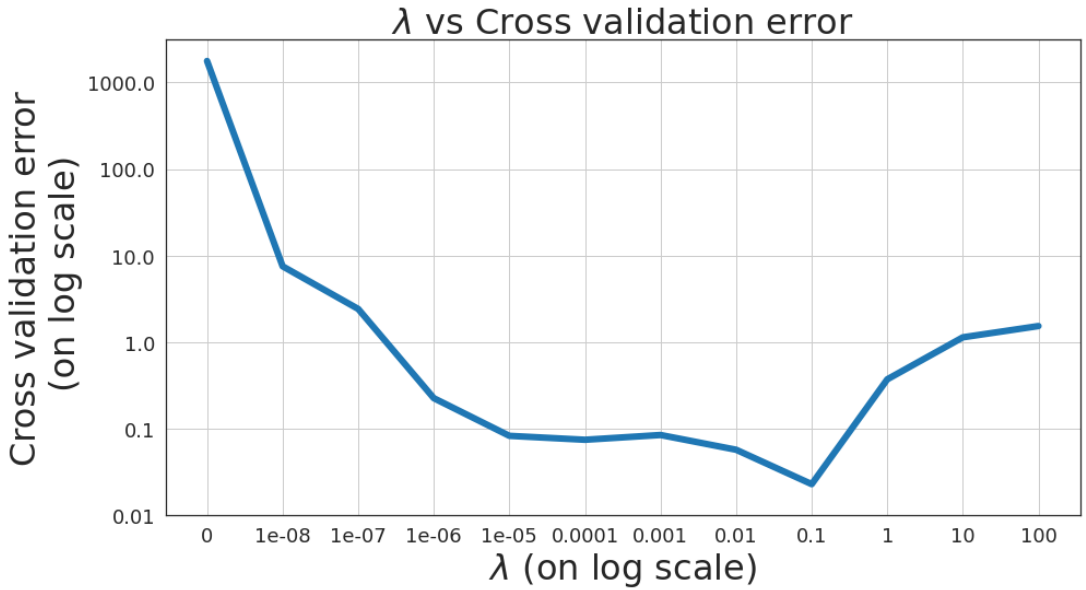
Typically above chart looks like a bowl shape, however, due to small number of features and samples, the chart takes shape like above.

λ vs Cross validation error



The most appropriate value of λ results in lowest cross validation error.

λ vs Cross validation error



Most appropriate value of λ from above figure is 0.1.

Lasso Regression

$$
\begin{aligned}
J(\mathbf{w}) &= \frac{1}{2}\left(\mathbf{X}\mathbf{w}-\mathbf{y}\right)^T\left(\mathbf{X}\mathbf{w}-\mathbf{y}\right) + \frac{\lambda}{2}\|\mathbf{w}\|_1 \\
&= \frac{1}{2}\left(\mathbf{X}\mathbf{w}-\mathbf{y}\right)^T\left(\mathbf{X}\mathbf{w}-\mathbf{y}\right) + \frac{\lambda}{2}\sum_{j=1}^{m}|w_j|
\end{aligned}
$$

We need specialized optimization algorithms to estimate weight vector in lasso regression, which are beyond scope of this course.

We will use sklearn implementation of Lasso in ML practice course.

Why does ridge and lasso work?

Models of Classification

Machine Learning Techniques

Dr. Ashish Tendulkar
IIT Madras

What topics will be discussed?

1. Discriminant functions: Learn direct mapping between feature vector $\mathbf{x}$ and label y.
2. Generative and discriminative models:
   - Generative classifiers model class conditional densities $p(\mathbf{x}|y)$ for features and prior probabilities of classes $p(y)$ and then through Bayes theorem, calculate $p(y|\mathbf{x})$.
   - Discriminative classifiers learn conditional probability distribution $p(y|\mathbf{x})$ through parameteric models.
3. Instance based models - Compare the test examples with the training examples and assigns class labels based on certain measure of similarity.

Part I: Classification setup

Classification set up

- Predict class label y of an example based on the feature vector $\mathbf{x}$.
- Class label y is a discrete quantity unlike a real number in regression set up.

Nature of class label

- Label is a discrete quantity - precisely an element in some finite set of class labels.
- Depending on the nature of the problem, we have one or more labels assigned to each example.

Types of classification

1. **Single label classification** - where each example has exactly one label.
   - e.g. is *the applicant eligible for loan?*
   - Label set: {yes, no}.
   - Label either *yes* or *no*.
2. **Multi-label classification** - where each example has more than one label.
   - e.g. identifying different types of fruits in a picture.

Label representation: Single example

1. **Single label classification:** Label is a scalar quantity and is represented by y.
2. **Multi-label classification:** More than one label hence vector representation $\mathbf{y}$.

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix}$$

Label set: $y = \{y_1, y_2, \ldots, y_k\}$ has k elements/labels.

Depending on the presence of the label, the corresponding label is set to 1.

Example: Single label classification (Binary)

- *Is the application eligible for loan?*
- Label set: $\{yes, no\}$, usually converted to $\{1, 0\}$
  - Label: either yes (1) or $no(0)$.
- Training example:
  - Feature vector: $\mathbf{x}$ - features for loan application like *age of applicant, income, number of dependents* etc.
  - Label: y

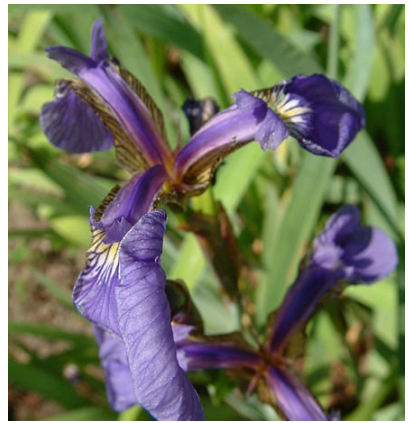
Example: Single label classification (Multiclass)

- *Types of iris flower*
- Label set: $C = \{versicolor, setosa, virginica\}$
- Label: exactly one label from set C.

Example: Single label classification (Multiclass)

- *Types of iris flower*

versicolor



setosa

virginica

Image Source: Wikipedia.org

Label encoding in multiclass setup

Use one-hot encoding scheme for label encoding.

- Make use of a vector $\mathbf{y}$ with components equal to the number of labels in the label set.
- In iris example, this would become:

$$\mathbf{y} = \begin{bmatrix} y_{versicolor} \\ y_{setosa} \\ y_{virginica} \end{bmatrix}$$

Example: Label encoding (single label)

Let's assume that the flower has label *versicolor*, we will encode it as follows:

$$\mathbf{y} = \begin{bmatrix} y_{versicolor} = 1 \\ y_{setosa} = 0 \\ y_{virginica} = 0 \end{bmatrix}$$

Note that the component of $\mathbf{y}$ corresponding to the label *versicolor* is 1, every other component is 0.

$$\mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Example: Multi-label Classification

- *Label all fruits from an image*.
- Label set: List of fruits e.g. $\{apple, guava, mango, orange, banana, strawberry, \}$
- Label: One or more fruits as they are present in the image.

$$\mathbf{y} = \begin{bmatrix} y_{apple} \\ y_{guava} \\ y_{mango} \\ y_{orange} \\ y_{banana} \\ y_{strawberry} \end{bmatrix}$$

Example: Multi-label Classification

Sample image

Image source: Wikipedia.org

Example: Multi-label Classification

Different fruits in the images are:

Apple

Orange

Banana

Example: Multi-label Classification

- Let's assume that the labels are $\mathbf{apple}$, $\mathbf{orange}$ and $\mathbf{banana}$.

$$\mathbf{y} = \begin{bmatrix} y_{apple} = 1 \\ y_{guava} = 0 \\ y_{mango} = 0 \\ y_{orange} = 1 \\ y_{banana} = 1 \\ y_{strawberry} = 0 \end{bmatrix} \quad \text{becomes} \quad \mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Training Data: Binary Classification

- Let's denote D as a set of n pairs of a features vector $\mathbf{x}_{m\times 1}$ and a label y, to represent examples.

$$D = \{(\mathbf{X}, \mathbf{y})\} = \left\{(\mathbf{x}^{(i)}, y^{(i)})\right\}_{i=1}^{n}$$

- $\mathbf{X}$ is a feature matrix corresponding to all the training examples and has shape $n \times m$. In this matrix, each feature vector is transposed and represented as a row in this matrix.

Training Data: Binary Classification

$$D = \{(\mathbf{X}, \mathbf{y})\} = \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$$

Concretely, the feature vector for i-th training example $\mathbf{x}^{(i)}$ can be obtained by $\mathbf{X}[i]$:

$$\mathbf{X}_{n\times m} = \begin{bmatrix} - \left(x^{(1)}\right)^T - \\ - \left(x^{(2)}\right)^T - \\ \vdots \\ - \left(x^{(n)}\right)^T - \end{bmatrix}$$

Training Data: Binary Classification

$$D = \{(\mathbf{X}, \mathbf{y})\} = \left\{(\mathbf{x}^{(i)}, y^{(i)})\right\}_{i=1}^{n}$$

$\mathbf{y}$ is a label vector of shape $n \times 1$. The i-th entry in this vector gives label for i-th example, which is denoted by $y^{(i)}$.

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

Training Data: Multi-class classification

A set of n pairs of a feature vector $\mathbf{x}$ and a label vector $\mathbf{y}$ representing examples.

We denote it by D:

$$D = \{(\mathbf{X}, \mathbf{Y})\} = \left\{\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)\right\}_{i=1}^{n}$$

where

$\mathbf{X}$ is an $n \times m$ feature matrix:

$$\mathbf{X}_{n\times m} = \begin{bmatrix} - \left(\mathbf{x}^{(1)}\right)^T - \\ - \left(\mathbf{x}^{(2)}\right)^T - \\ \vdots \\ - \left(\mathbf{x}^{(n)}\right)^T - \end{bmatrix}$$

Training Data: Multi-class classification

$$D = \{(\mathbf{X}, \mathbf{Y})\} = \left\{\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)\right\}_{i=1}^{n}$$

$\mathbf{Y}$ is a label matrix of shape $n \times k$, where k is the total number of classes in label set.

$$\mathbf{Y} = \begin{bmatrix} - \left(\mathbf{y}^{(1)}\right)^T - \\ - \left(\mathbf{y}^{(2)}\right)^T - \\ \vdots \\ - \left(\mathbf{y}^{(n)}\right)^T - \end{bmatrix}$$

Multi-class and multi-label classification label vector

$\mathbf{Y}$ is a label matrix of shape $n \times k$, where k is the total number of classes in label set.

$$\mathbf{Y} = \begin{bmatrix} - \left(\mathbf{y}^{(1)}\right)^T - \\ - \left(\mathbf{y}^{(2)}\right)^T - \\ \vdots \\ - \left(\mathbf{y}^{(n)}\right)^T - \end{bmatrix}$$

Multi-class and multi-label classification label vector

- Multi-class classification: For $\left(\mathbf{y}^{(i)}\right)^T$, **exactly one entry** corresponding to the class label is 1.
- Multi-label classification: For $\left(\mathbf{y}^{(i)}\right)^T$, **more than one entries** corresponding to the class labels can be 1.

Part II: Discriminant Functions

Overview

Example: Two classes

Simplest discriminant function is very similar to the linear regression:

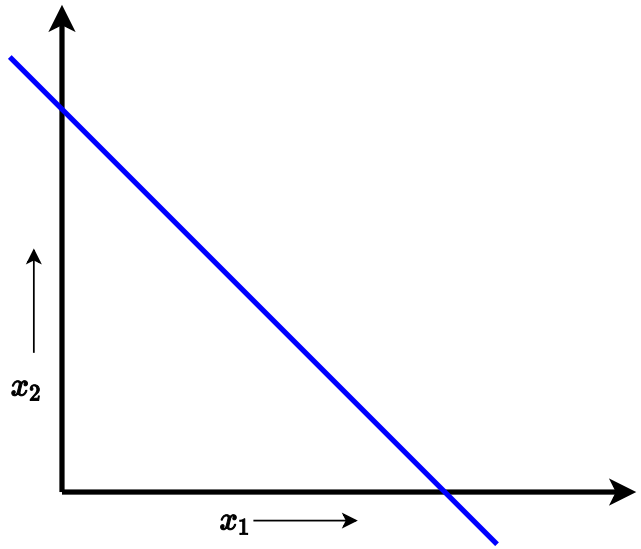
$$
\begin{aligned}
y &= w_0 + w_1 x_1 + \ldots + w_m x_m \\
&= w_0 + \mathbf{w}^T \mathbf{x}
\end{aligned}
$$

where,

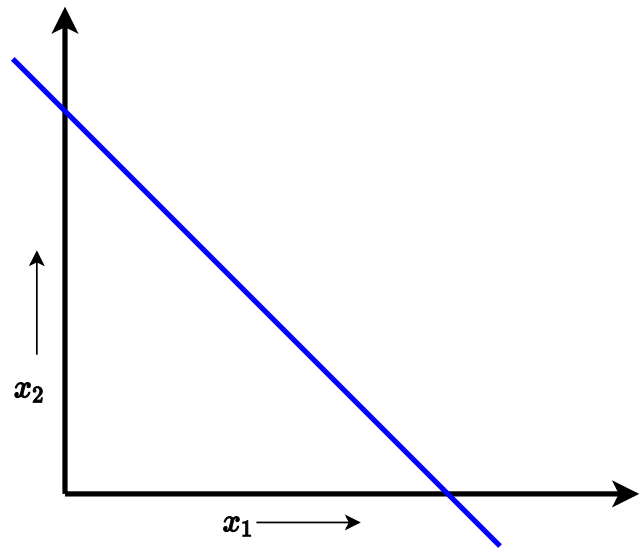
- w_0: Bias [*Keeping this separately for a reason*]
- $\mathbf{w}$: Weight vector
- $\mathbf{x}$: Feature vector
- y: label

Geometric Interpretation

The simplest discriminant function $y = w_o + \mathbf{w}^T\mathbf{x}$ represents a hyperplane in $m-1$ dimensional space where m is the number of features.

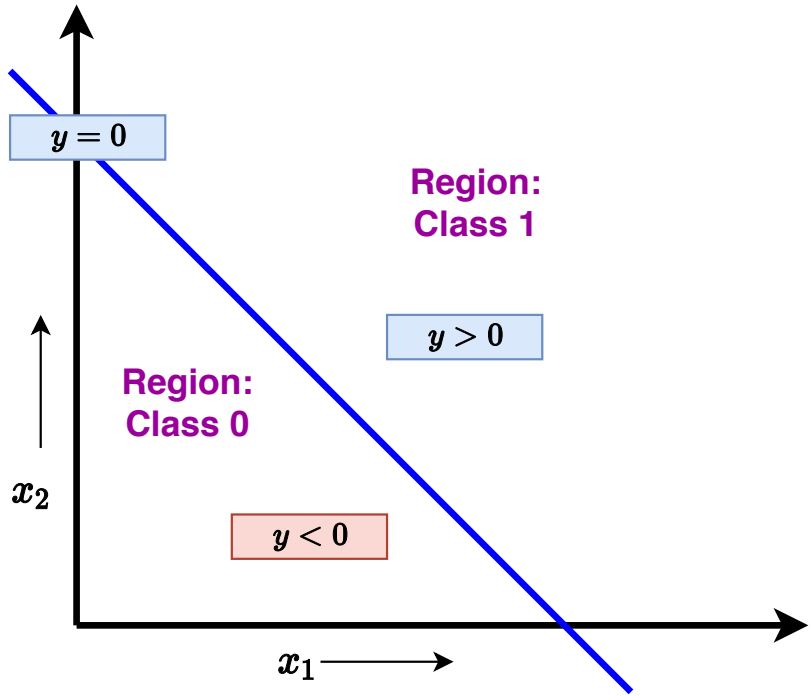


Geometric Interpretation

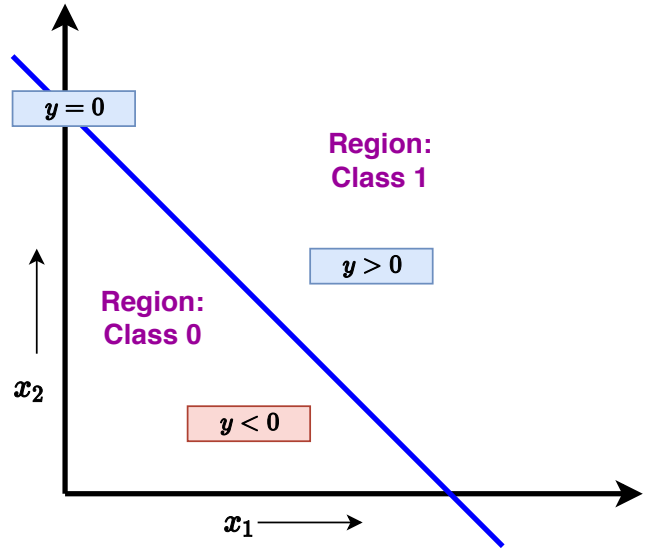


The discriminant function is a hyperplane in (m-1)-D space i.e. $2 - 1 = 1$-D space, which is a line. Note that here $m = 2$ features.

Classification with discriminant functions



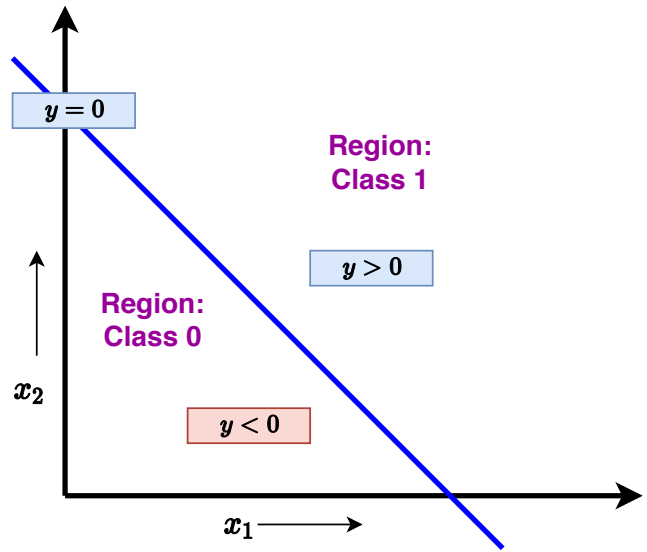
Classification with discriminant functions



Classification is performed as follows:

$$y = \begin{cases} 1, \text{ if } w_0 + \mathbf{w}^T\mathbf{x} > 0 \\ 0, \text{ otherwise} \end{cases}$$

Classification with discriminant functions



The decision boundary is defined by

$$w_0 + \mathbf{w}^T \mathbf{x} = 0$$

What does $\mathbf{w}$ represent?

Consider two points $x^{(A)}$ and $x^{(B)}$ on the decision surface, we will have

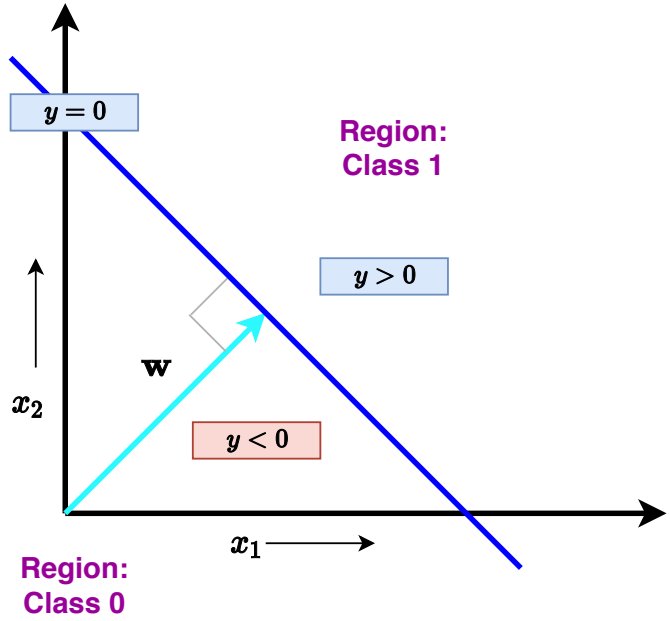
$$y^{(A)} = w_0 + \mathbf{w}^T \mathbf{x}^{(A)} = 0$$
$$y^{(B)} = w_0 + \mathbf{w}^T \mathbf{x}^{(B)} = 0$$

Since $y^{(A)} = y^{(B)} = 0$, $y^{(A)} - y^{(B)}$ results into the following equation:

$$\mathbf{w}^T (\mathbf{x}^{(A)} - \mathbf{x}^{(B)}) = 0$$

What does $\mathbf{w}$ represent?

The vector $\mathbf{w}$ is orthogonal to every vector lying within the decision surface, hence it determines the **orientation of the decision surface**.



What does w_0 represent?

For points on decision surface, we have

$$w_0 + \mathbf{w}^T\mathbf{x} = 0$$
$$\mathbf{w}^T\mathbf{x} = -w_0$$

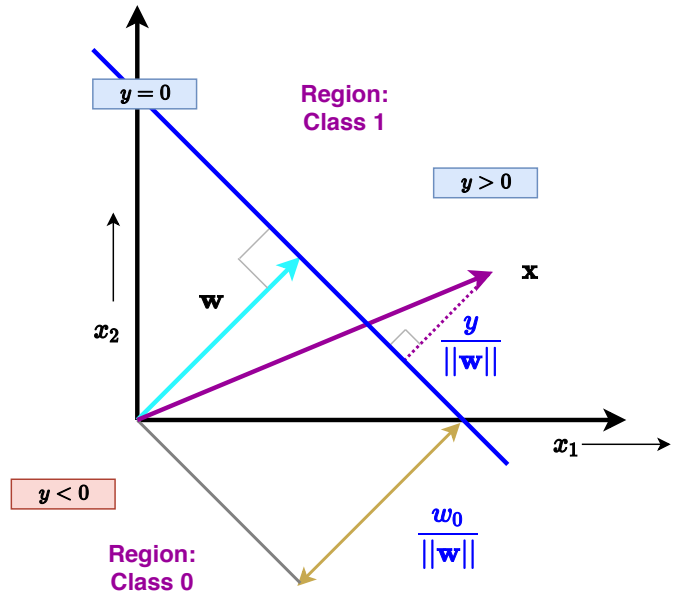
What does w_0 represent?

Normalizing both sides with the length of the vector $||\mathbf{w}||$, we get normal distance from the origin to the decision surface:

$$\frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||} = -\frac{w_0}{||\mathbf{w}||}$$

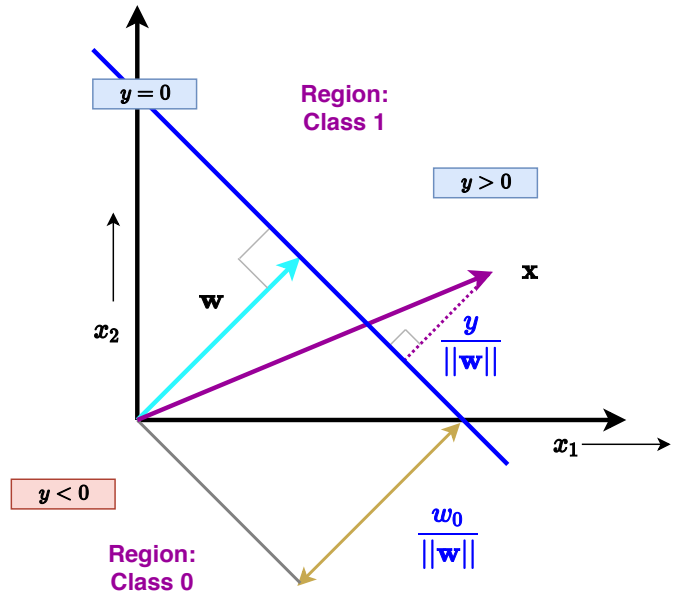
w_0 determines the location of the decision surface

What does y represent?



y gives signed measure of perpendicular distance of the point $\mathbf{x}$ from the decision surface.

What does y represent?



- w_0 determines the **location** of the decision surface.
- $\mathbf{w}$ is orthogonal to every vector lying within the decision surface, hence it determines the **orientation** of the decision surface.

Alternate interpretation

By using a dummy feature x_0 and setting it to 1, we get the following equation:

$$\begin{aligned} y &= w_0 x_0 + w_1 x_1 + \ldots + w_m x_m \\ &= \mathbf{w}^T \mathbf{x} \end{aligned}$$

This represents a decision surface that is m-D hyperplane passing through the origin of $(m+1)$-D space.

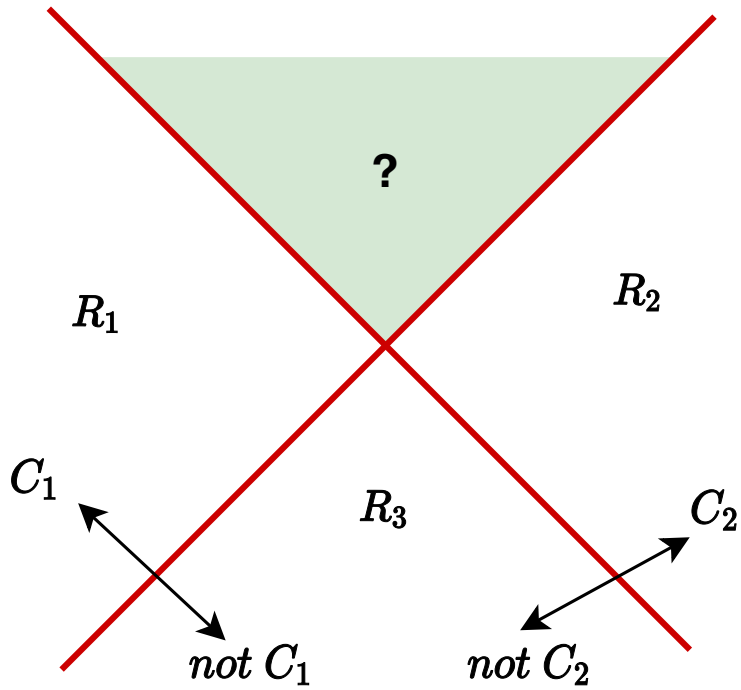
Multiple classes

Assuming the number of classes to be $k > 2$, we can build discriminant functions in two ways:

- **One-vs-rest**: Build $k - 1$ discriminant functions. Each discriminant function solves two class classification problem: class C_k vs $not\ C_k$.
- **One-vs-one**: One discriminant function per pair of classes. Total functions = $\binom{k}{2} = \frac{k\,(k-1)}{2}$

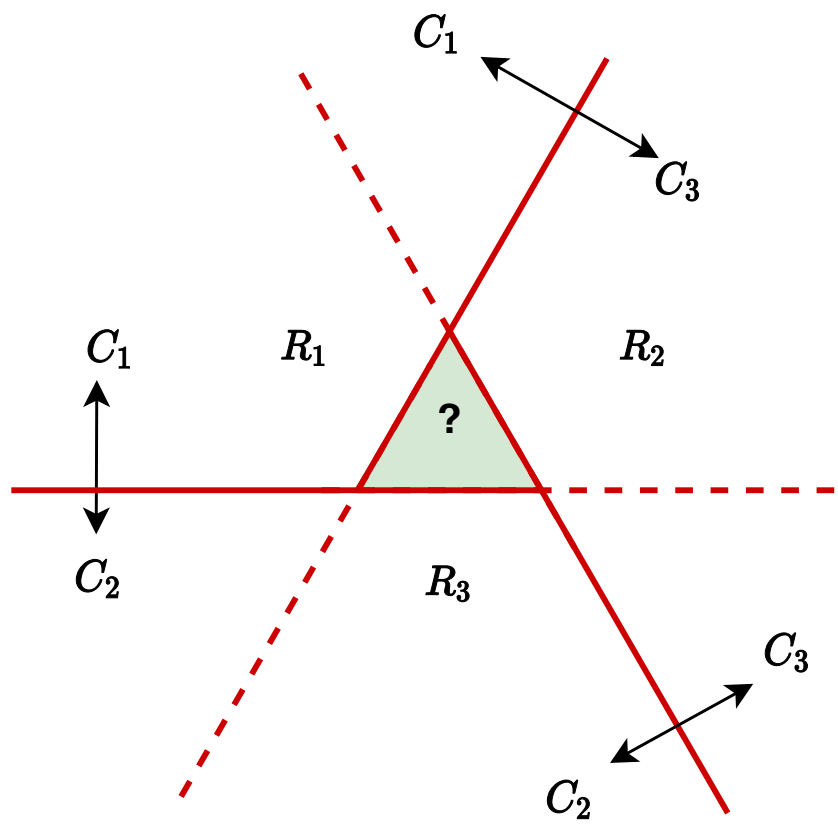
Issues with *one-vs-rest*

- Two discriminant functions for each class C_1 and C_2.
- Each discriminant function separates C_k and not C_k.
- Region of ambiguity is in green.



Issues with *one-vs-one*

- $k(k-1)/2$ discriminant functions for each class pair C_i and C_j.
- Each discriminant function separates C_i and C_j.
- Each point is classified by majority vote.
- Region of ambiguity is in green.



How do we fix it?

A single k-class discriminant comprising k linear functions as follows:

$$\begin{aligned} y_k &= w_{k0} + w_{k1}x_1 + \ldots + w_{km}x_m \\ &= w_{k0} + \mathbf{w_k}^T\mathbf{x} \end{aligned}$$

How do we fix it?

Concretely:

$$
\begin{aligned}
y_1 &= \quad w_{10} + \mathbf{w_1}^T \mathbf{x} \\
y_2 &= \quad w_{20} + \mathbf{w_2}^T \mathbf{x} \\
&\vdots \\
y_k &= \quad w_{k0} + \mathbf{w_k}^T \mathbf{x}
\end{aligned}
$$

Classification in k-discriminant functions

Assign label y_k to example $\mathbf{x}$ if $y_k > y_j, \forall j \neq k$

The decision boundary between classes y_k and y_j corresponds to $m-1$ dimensional hyperplane:

$$(w_{k0} - w_{j0}) + (\mathbf{w}_k - \mathbf{w}_j)^T \mathbf{x} = 0$$

This has the same form as the decision boundary for the two class cases:

$$w_0 + \mathbf{w}^T \mathbf{x} = 0$$

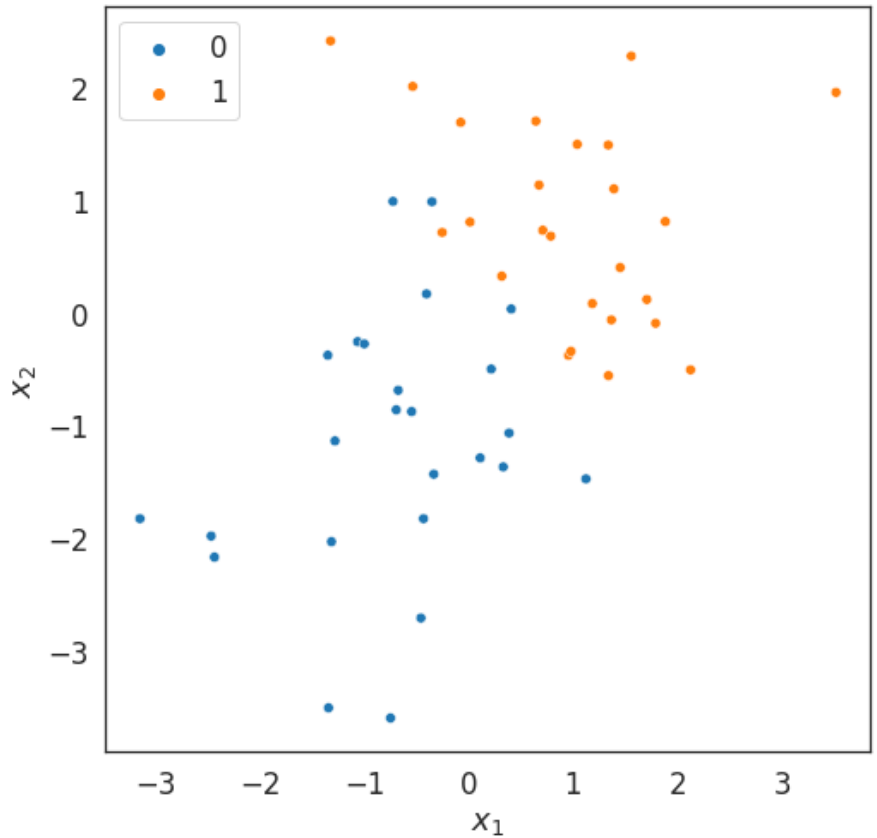
Now that we have a model of linear discriminant functions, we will study two approaches for learning the parameters of the model:

- Least squares
- Perceptron

Least squares classification

Train-test split (TODO)

Sample Training Data



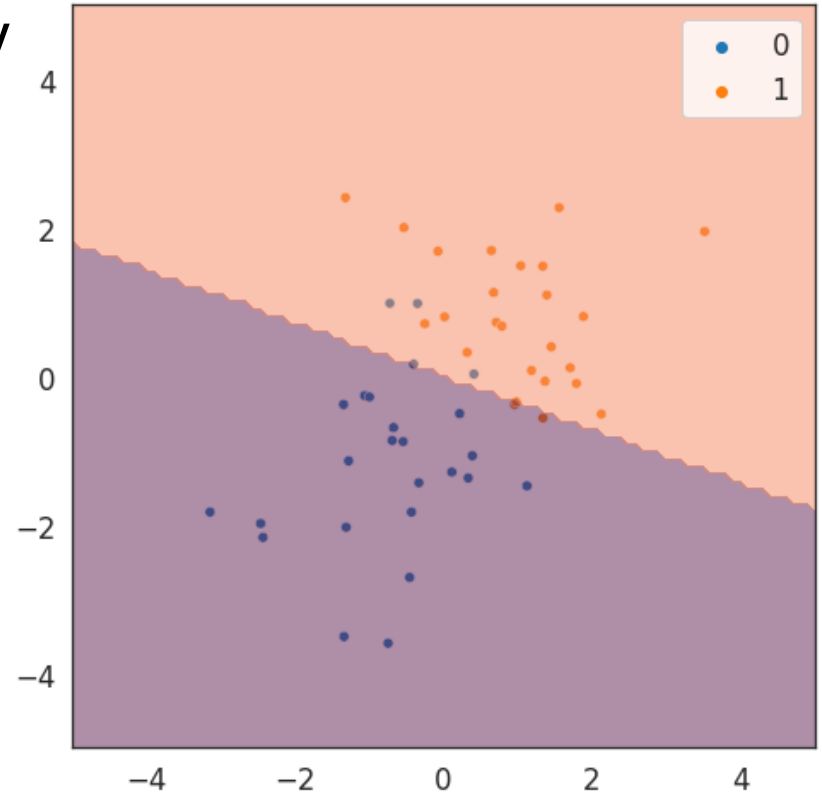
Sample Training Data

Let's implement the model inference function:

```
def predict(x, w):
  z = x @ w
  return np.array([1 if z_val >= 0 else 0 for z_val in z])
```

Decision Boundary Visualization

A random decision boundary



Loss function: Least Square Error

The total loss is the sum of square of errors between actual and predicted labels at each training point.

The error at i-th training point is calculated as follows:

$$
\begin{aligned}
e^{(i)} &= (\text{actual label} - \text{predicted label})^2 \\
&= \left(y^{(i)} - h_{\mathbf{w}}(\mathbf{x}^{(i)})\right)^2 \\
&= \left(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\right)^2
\end{aligned}
$$

Loss function: Least Square Error

The total loss $J(\mathbf{w})$ is sum of errors for each training point:

$$J(\mathbf{w}) = \sum_{i=1}^{n} e^{(i)} = \mathbf{e}^T \mathbf{e}$$

Note that the loss is dependent on the value of $\mathbf{w}$ - as these value changes, we get a new model, which will result in different prediction and hence affects the error at each training point.

Optimization: Normal equation

Calculate derivative of loss function $J(\mathbf{w})$ w.r.t. weight vector $\mathbf{w}$.

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = 2(\mathbf{X}^T\mathbf{X}\mathbf{W} - \mathbf{X}^T\mathbf{Y})$$

Set $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$ to 0 and solve for $\mathbf{W}$:

$$0 = 2(\mathbf{X}^T\mathbf{X}\mathbf{W} - \mathbf{X}^T\mathbf{Y})$$
$$\mathbf{W} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y}$$

Optimization: Normal equation

$$\mathbf{W} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y}$$

Whenever $\mathbf{X}^T\mathbf{X}$ is not full rank, we calculate pseudo-inverse: $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T$

Evaluation metrics

- Confusion matrix
- Precision/Recall/F1 score

Note to Swarnim: Please write one line code for confusion matrix and precision/recall/f1 and report these metrics in a slide: on one side show confusion matrix and on the other side all metric values.

Perceptron

Machine Learning Techniques

Dr. Ashish Tendulkar
IIT Madras

Perceptron is a binary classification algorithm.

We will cover

- Five different components of this algorithm, just like any other ML algorithm along with mathematical details.
- **Implementation** from scratch in Python with Numpy.

Perceptron was motivated from neurons

- Invented in 1958 by Frank Rosenblatt and was intended to be a machine, rather than a program.
- Perceptron was meant to be a rough model of how individual neurons work in the brain.

Let's look at the first component that is training data, which is very similar to other classification algorithms.

Training Data

- Feature matrix: $\mathbf{X}_{n \times m}$
- Label vector: $\mathbf{y}_{n \times 1}$

Note that perceptron can solve only binary classification problems. Hence $y^{(i)} \in \{-1, +1\}$

Let's look at the second component that is model, which is inspired from brain neurons.

Model $h_{\mathbf{w}}(\mathbf{x})$

Weight vector

Non-linear activation function

Feature transformation

$$y = f(\underline{\mathbf{w}^T \phi(\mathbf{x})})$$

Label

Feature vector

Linear combination of features

Model $h_{\mathbf{w}}(\mathbf{x})$

$$y = f(\mathbf{w}^T \phi(\mathbf{x})) = f(z)$$

where, $f(.)$ is a non-linear activation function. Here we use sign or threshold function as $f(.)$:

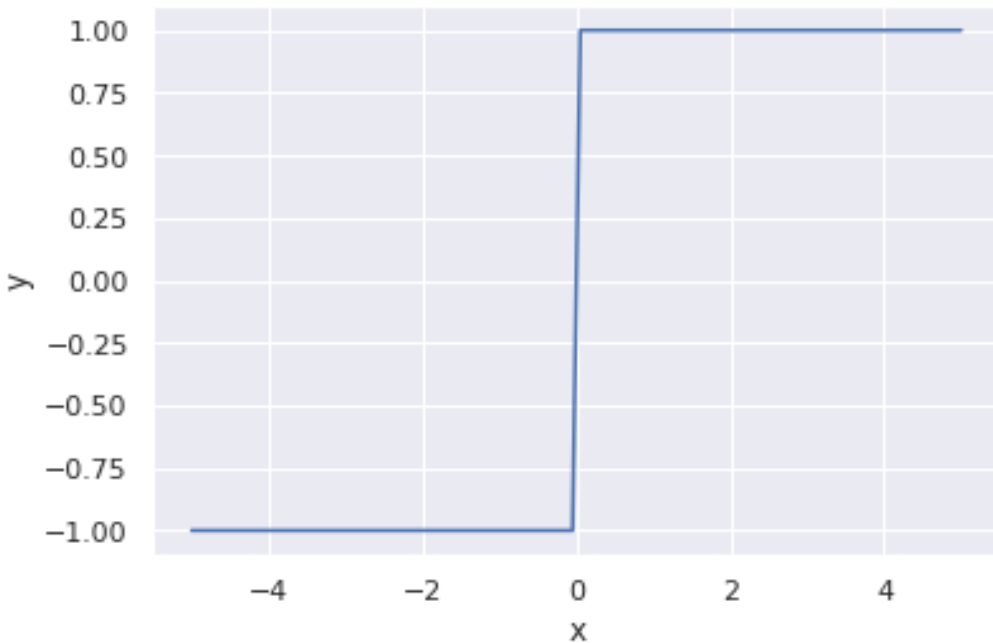
$$f(z) = \begin{cases} +1, \text{ if } z \geq 0 \\ -1, \text{ otherwise (i.e. } z < 0) \end{cases}$$

It can also be written as

$$y = \text{sign}(\mathbf{w}^T \phi(\mathbf{x}))$$

Model visualization

Remember $y = \text{sign}(\mathbf{w}^T \phi(\mathbf{x}))$



Note that for values of $x < 0$, we have $y = -1$

And for values of $x \geq 0$, we have $y = +1$

Now that we have seen the model of perceptron, let's look at the third component that is loss function.

Loss function

Let $\hat{y}^{(i)} \in \{-1, +1\}$ be the prediction from perceptron and $y^{(i)}$ be the actual label for i-th example. The error $e^{(i)}$ is calculated as

$$e^{(i)} = \begin{cases} 0, \text{ if } \hat{y}^{(i)} = y^{(i)} \\ -\mathbf{w}^T \phi(x^{(i)}) y^{(i)}, \text{ otherwise (i.e. } \hat{y}^{(i)} \neq y^{(i)}) \end{cases}$$

For correctly classified examples, the error is 0

For misclassified examples, the error is $-\mathbf{w}^T \phi(x^{(i)}) y^{(i)}$

Loss function

The error can be compactly written as:

$$
\begin{aligned}
e^{(i)} &= \max(0, -\mathbf{w}^T \phi(x^{(i)}) y^{(i)}) \\
&= \max(0, -h_{\mathbf{w}}(x^{(i)}) y^{(i)})
\end{aligned}
$$

Loss function Illustration

$h_{\mathbf{w}}(\mathbf{x}^{(i)}) : y = \text{sign}\left(\mathbf{w}^T \phi(\mathbf{x}^{(i)})\right)$ is either +1 or -1.

If the decision is correct, then

$h_{\mathbf{w}}(\mathbf{x}^{(i)}) = +1$ and $y^{(i)} = +1$ or : $h_{\mathbf{w}}(\mathbf{x}^{(i)}) = -1$ and $y^{(i)} = -1$

$$\begin{aligned} e^{(i)} &= \max(0, -h_{\mathbf{w}}(\mathbf{x}^{(i)})y^{(i)}) \\ &= \max(0, -(1 \times 1)) \\ &= \max(0, -1) \\ &= 0 \end{aligned}$$

$$\begin{aligned} e^{(i)} &= \max(0, -h_{\mathbf{w}}(\mathbf{x}^{(i)})y^{(i)}) \\ &= \max(0, -(-1 \times -1)) \\ &= \max(0, -1) \\ &= 0 \end{aligned}$$

Note that in both the cases the error is 0.

Loss function Illustration

If the decision is wrong, then

$h_{\mathbf{w}}(\mathbf{x}^{(i)}) = +1$ and $y^{(i)} = -1$

$$\begin{aligned} e^{(i)} &= \max(0, -h_{\mathbf{w}}(\mathbf{x}^{(i)})y^{(i)}) \\ &= \max(0, -(+1 \times -1)) \\ &= \max(0, 1) = 1 \end{aligned}$$

$h_{\mathbf{w}}(\mathbf{x}^{(i)}) = -1$ and $y^{(i)} = +1$:

$$\begin{aligned} e^{(i)} &= \max(0, -h_{\mathbf{w}}(\mathbf{x}^{(i)})y^{(i)}) \\ &= \max(0, -(-1 \times +1)) \\ &= \max(0, 1) = 1 \end{aligned}$$

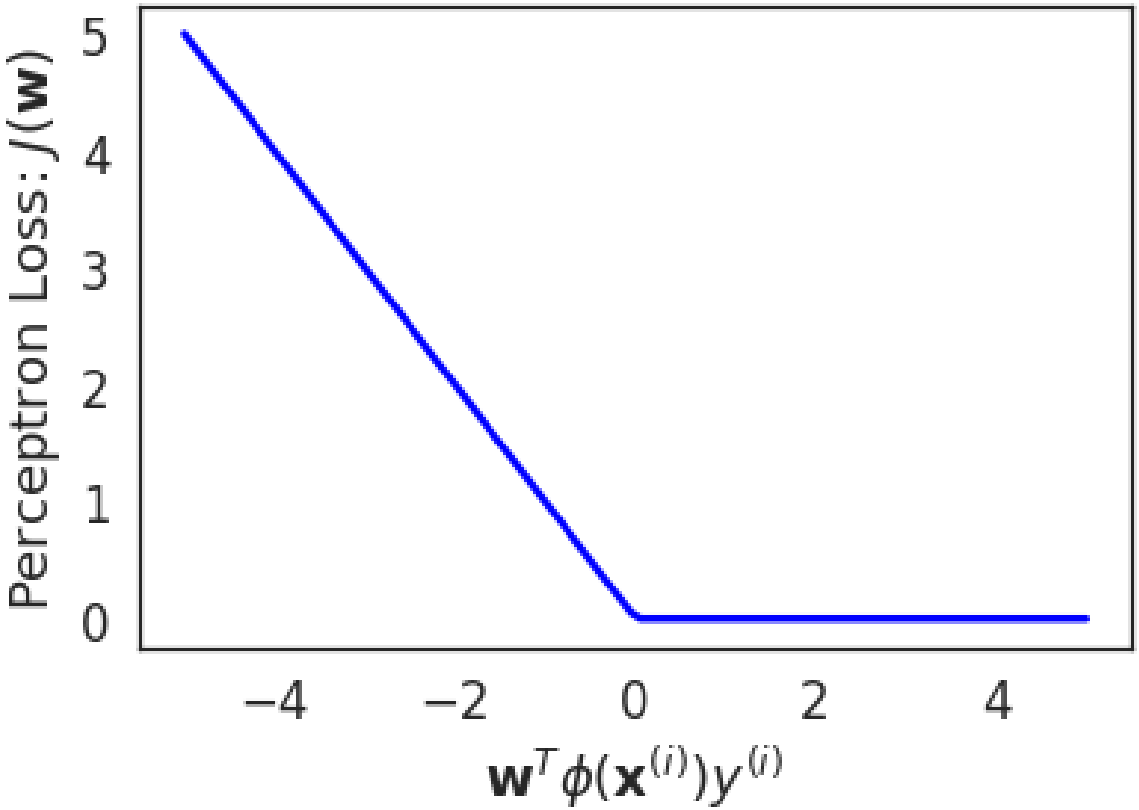
Note that in both the cases the error is 1.

Loss function

$$
\begin{aligned}
J(\mathbf{w}) &= \sum_{i=1}^{n} e^{(i)} \\
&= \sum_{i=1}^{n} \max(0, -\mathbf{w}^T \phi(x^{(i)}) y^{(i)}) \\
&= \sum_{i=1}^{n} \max(0, -y^{(i)} h_{\mathbf{w}}(x^{(i)}))
\end{aligned}
$$

The error is a piecewise linear function: it is zero in the correctly classified regions and a linear function of $\mathbf{w}$ in mis-classified region.

$J(\mathbf{w})$ is not differentiable in $\mathbf{w}$.



We can control the loss is by adjusting the value of $\mathbf{w}$.

The loss is directly proportional to $\mathbf{w}$.

Thus for misclassified example, we can reduce loss by reducing $\mathbf{w}$.

And for correctly classified examples, we leave $\mathbf{w}$ unchanged.

The next task is to obtain the weight vector that minimizes the loss.

We will look at the optimization procedure used in perceptron. This procedure is known as perceptron update rule.

Optimization procedure

1. Initialize $\mathbf{w}^{(0)} = \mathbf{0}$
2. For each training example $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$:

$$\widehat{y}^{(i)} = \text{sign}\left(\mathbf{w}^T \phi(\mathbf{x}^{(i)})\right)$$

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + \alpha \left(y^{(i)} - \widehat{y}^{(i)}\right) \phi(\mathbf{x}^{(i)})$$

Note for correctly classified examples, $\left(y^{(i)} - \widehat{y}^{(i)}\right)$ = 0 and hence there is no change in the weight vector.

Linear separable examples lead to convergence of the algorithm with zero training loss, else it oscillates.

Let's understand perceptron update rule for various values of $\hat{y}^{(i)}$ and $y^{(i)}$:

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + \alpha \left(y^{(i)} - \hat{y}^{(i)}\right) \phi(\mathbf{x}^{(i)})$$

(Case 1: Correct classification) $\hat{y}^{(i)} = y^{(i)}$:

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + \alpha \times 0 \times \phi(\mathbf{x}^{(i)}) = \mathbf{w}^{(t)} + 0 = \mathbf{w}^{(t)}$$

(Case 2: Negative class misclassification) $y^{(i)}$ = -1 and $\hat{y}^{(i)}$ = 1:

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + \alpha \times (-1 - 1) \times \phi(\mathbf{x}^{(i)}) = \mathbf{w}^{(t)} - 2\alpha\phi(\mathbf{x}^{(i)})$$

(Case 3: Positive class misclassification) $y^{(i)}$ = 1 and $\hat{y}^{(i)}$ = -1:

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + \alpha \times (1 - (-1)) \times \phi(\mathbf{x}^{(i)}) = \mathbf{w}^{(t)} + 2\alpha\phi(\mathbf{x}^{(i)})$$

Optimization procedure: Convergence

- On linearly separable data, the optimization procedure eventually converges.

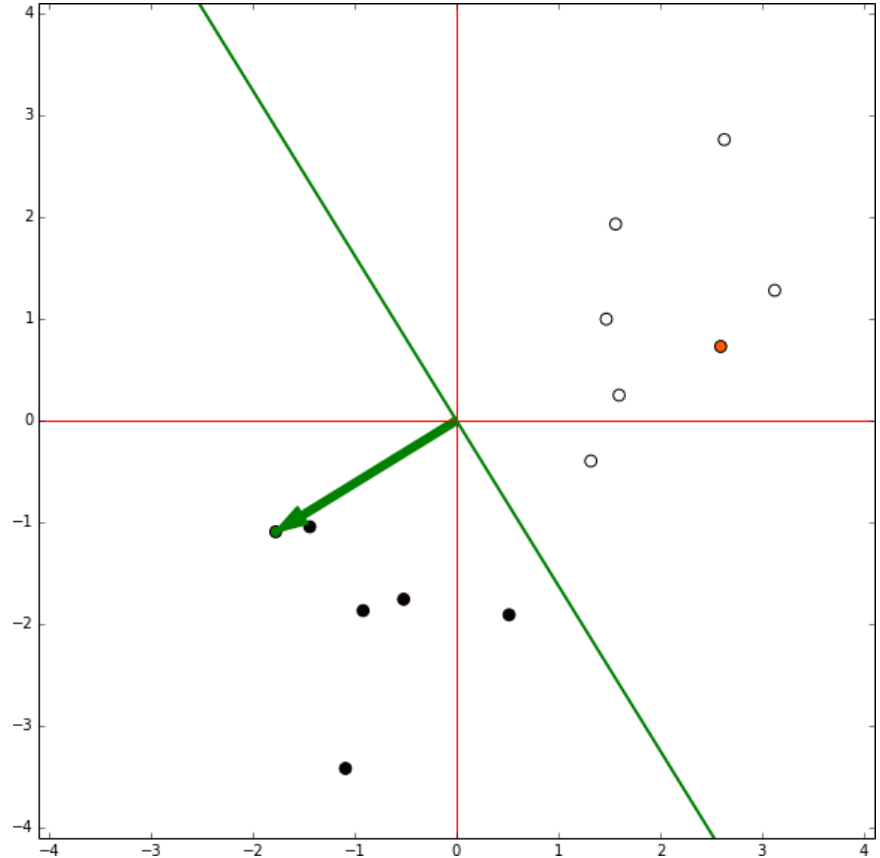
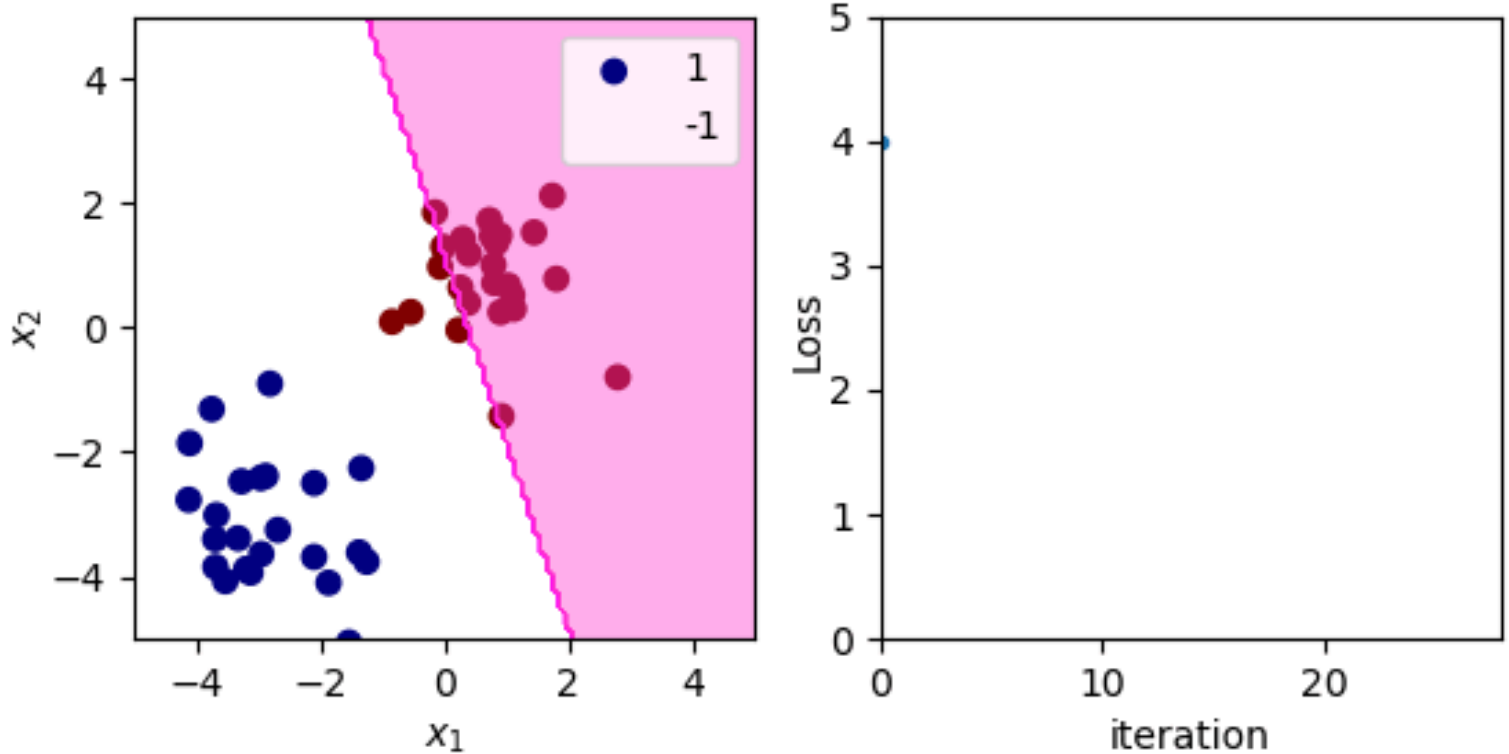


Image source: Wikipedia.org

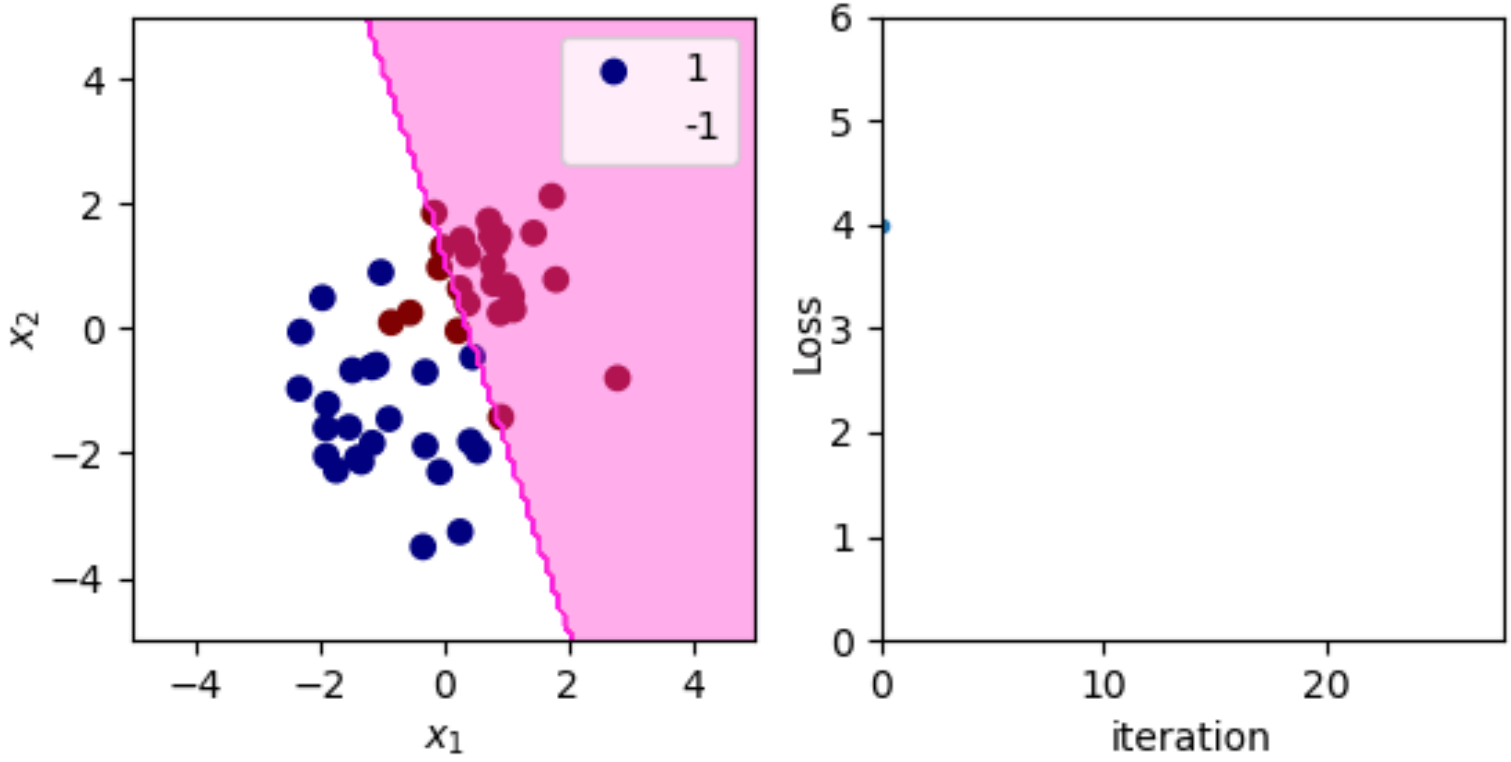
Optimization procedure: Convergence

- On linearly separable data, the optimization procedure converges.



Optimization procedure: Oscillations

- On **non** linearly separable data, the optimization procedure **never** converges.

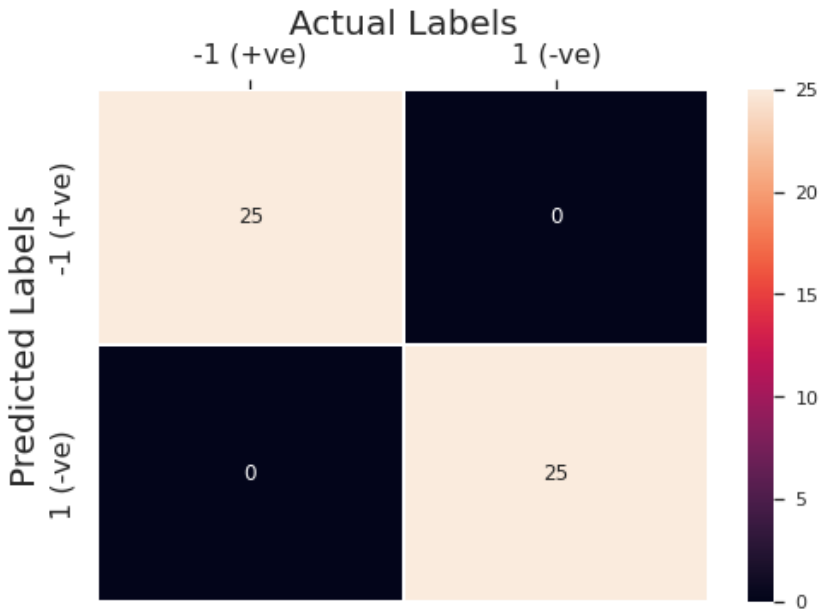


Finally let's look at the evaluation metrics.
They are same as other classification algorithms.

Evaluation metrics

- Calculate confusion matrix based on predicted and actual labels.
- Calculate classification metrics like precision, recall, accuracy, F1-score from confusion matrix.

Confusion matrix and evaluation metrics: Example



Precision (P) = $\frac{TP}{TP + FP} = \frac{25}{25 + 0} = 1.0$

Recall (R) = $\frac{TP}{TP + FN} = \frac{25}{25 + 0} = 1.0$

F1 Score = $\frac{2 \times P \times R}{P + R} = \frac{2 \times 1 \times 1}{1 + 1} = 1.0$

- TP = 25
- FP = 0
- TN = 25
- FN = 0

Confusion matrix implementation

```
def compute_confusion_matrix(y, y_predicted):

  # Create a 2D matrix of size n by n
  confusion_matrix = np.zeros((n,n))

  # Populate entries of confusion matrix
  for i_x,i_y in zip(y_predicted, y):
    confusion_matrix[i_x,i_y] +=1

  return confusion_matrix
```