Machine Learning Techniques

ML Component Framework

Dr. Ashish Tendulkar

IIT Madras

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

- 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

- 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

- 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

- 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

- 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.

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.

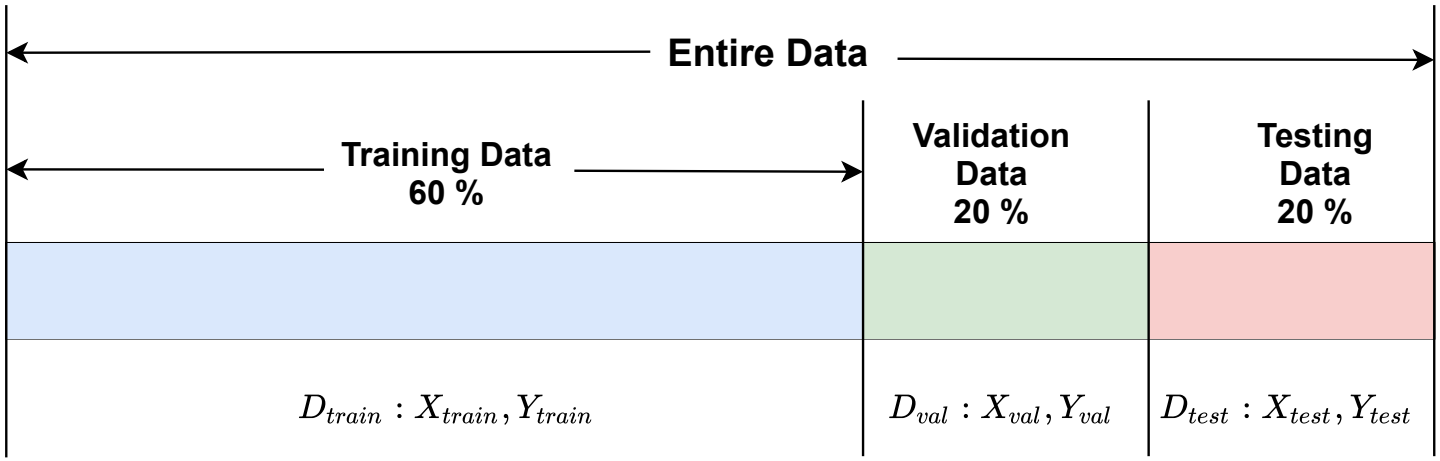
Data

- 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.

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.

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.

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.

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.

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.

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.

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.

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.

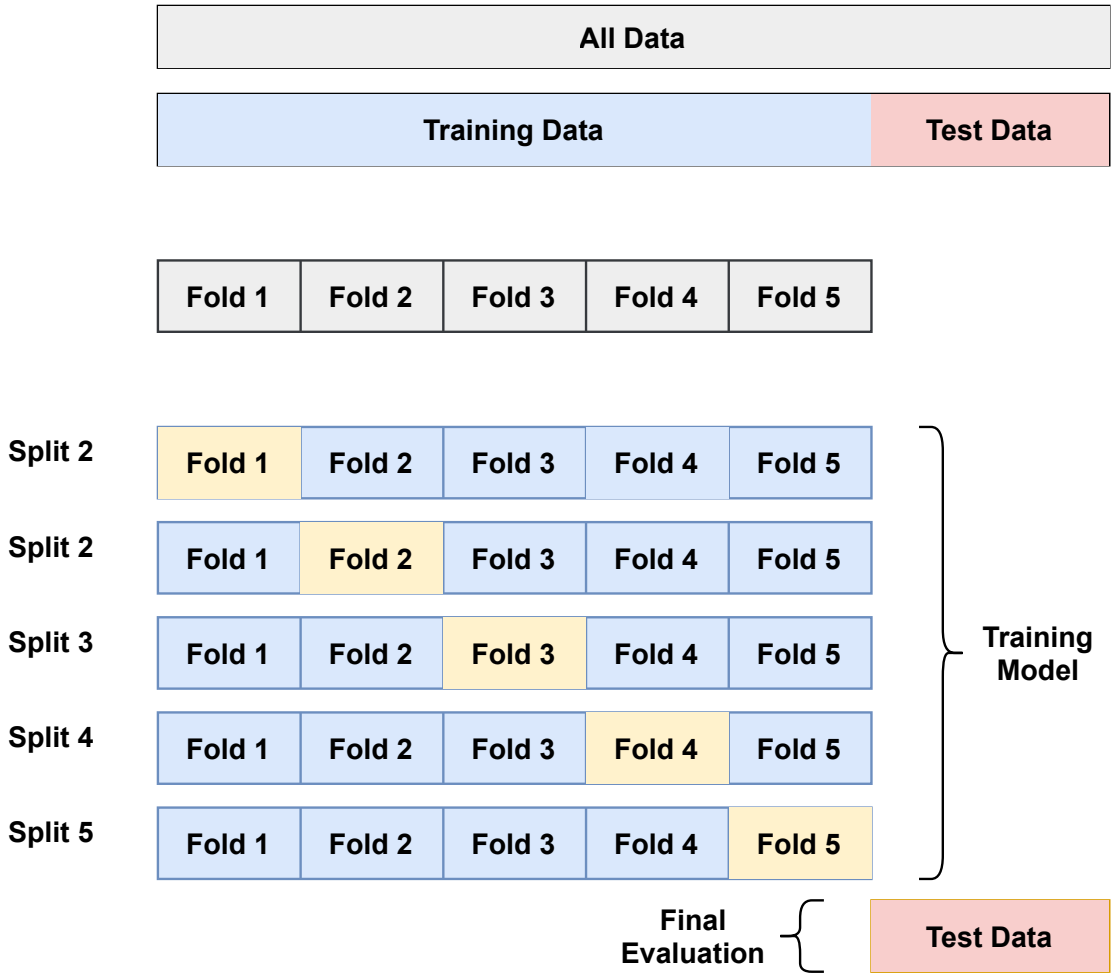
K-fold Cross Validation (CV)

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.

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

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

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

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

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

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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:

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

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:

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

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:

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

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:

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

Loss Function

- Loss function: Measurement of difference between the actual and the predicted labels.
- The loss function is represented by 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

- Loss function: Measurement of difference between the actual and the predicted labels.
- The loss function is represented by 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

- Loss function: Measurement of difference between the actual and the predicted labels.
- The loss function is represented by 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

- Loss function: Measurement of difference between the actual and the predicted labels.
- The loss function is represented by 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

- Loss function: Measurement of difference between the actual and the predicted labels.
- The loss function is represented by 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

- Loss function: Measurement of difference between the actual and the predicted labels.
- The loss function is represented by 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.

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.

Loss Function

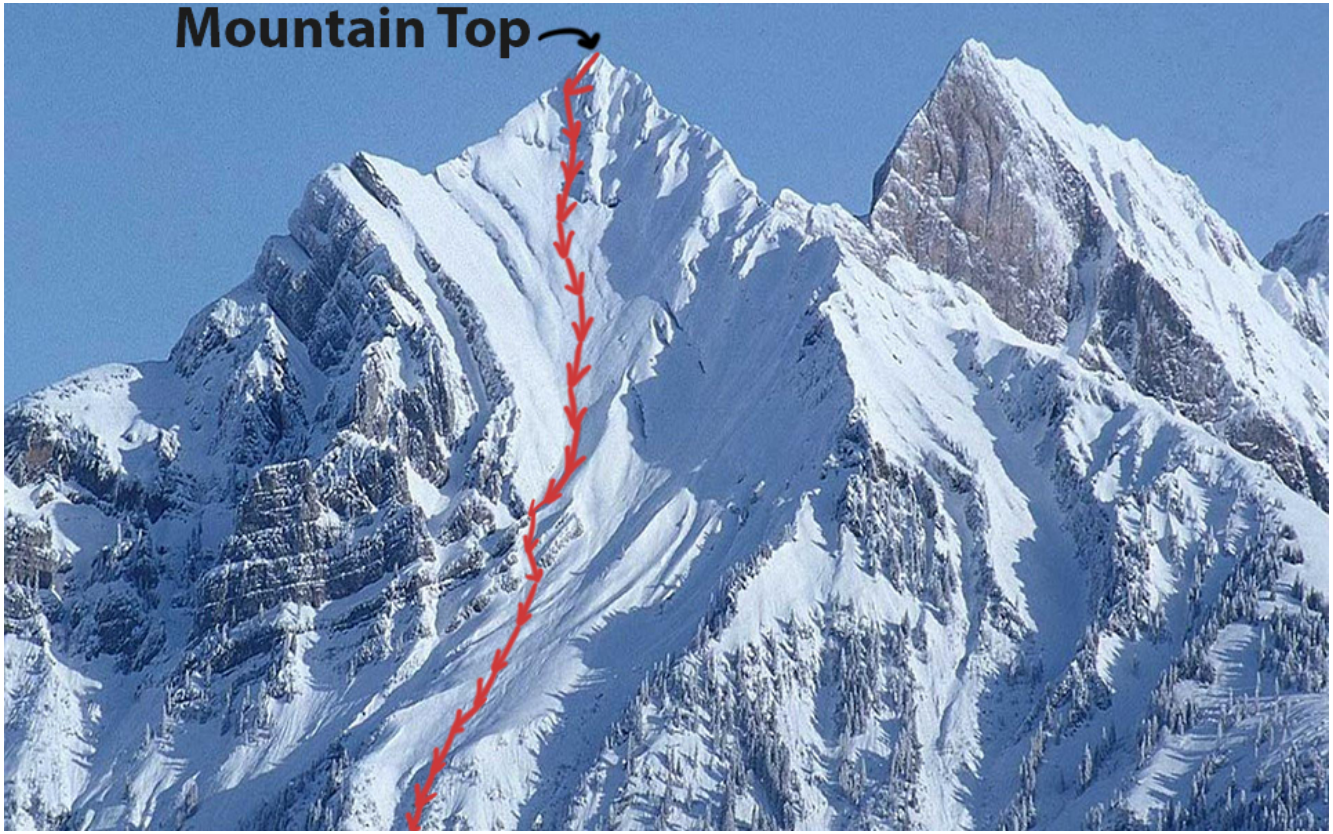
- The loss function can be convex or non-convex depending on the model function class.
- Linear/logistic regressions, Support Vector Machines have convex loss functions, while the loss functions for neural networks are non-convex in nature.
- Convex loss functions are easier to optimize than the non-convex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

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.

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.

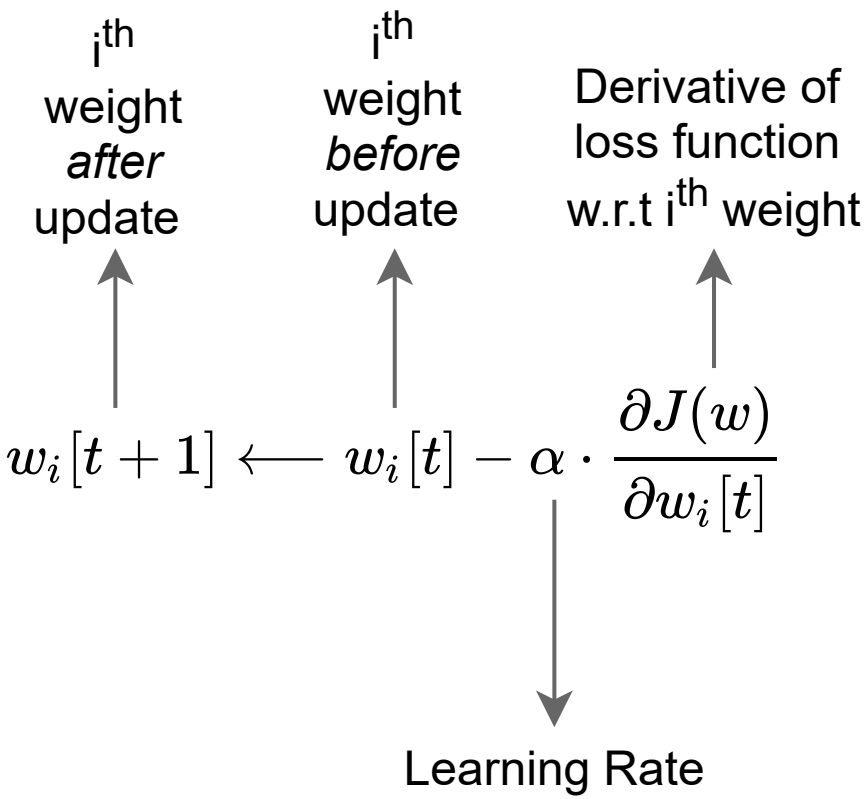
Gradient Descent: Mathematical Framework

- 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

- 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



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: 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: 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: 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: 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: 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: 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: 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?**

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?

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?

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?**

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

- 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

- 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

- 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**.

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**.

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**.

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**.

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

- 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

- 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

- 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

- 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.

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.

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.

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.

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.

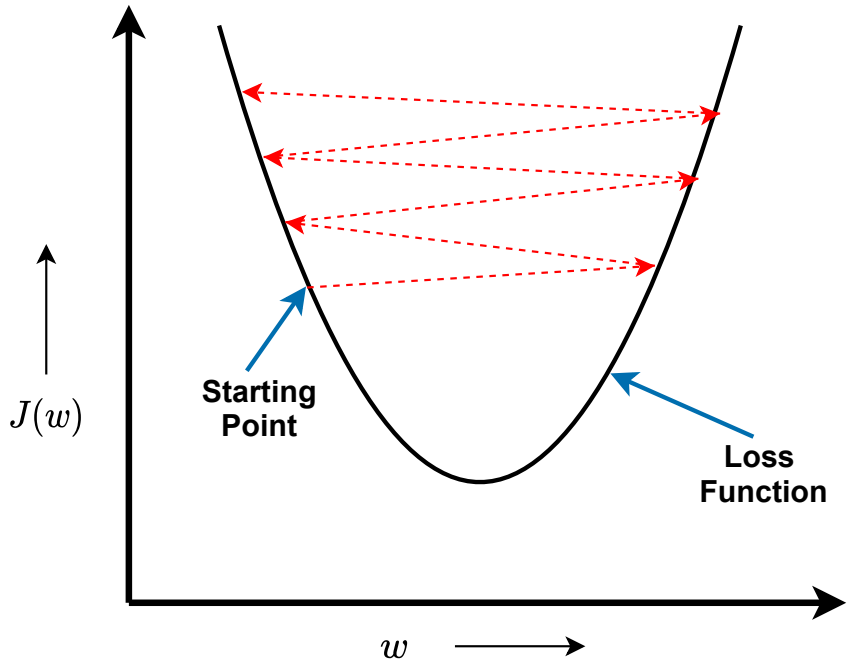
GD vs Mini batch GD and SGD

- 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.

GD vs Mini batch GD and SGD

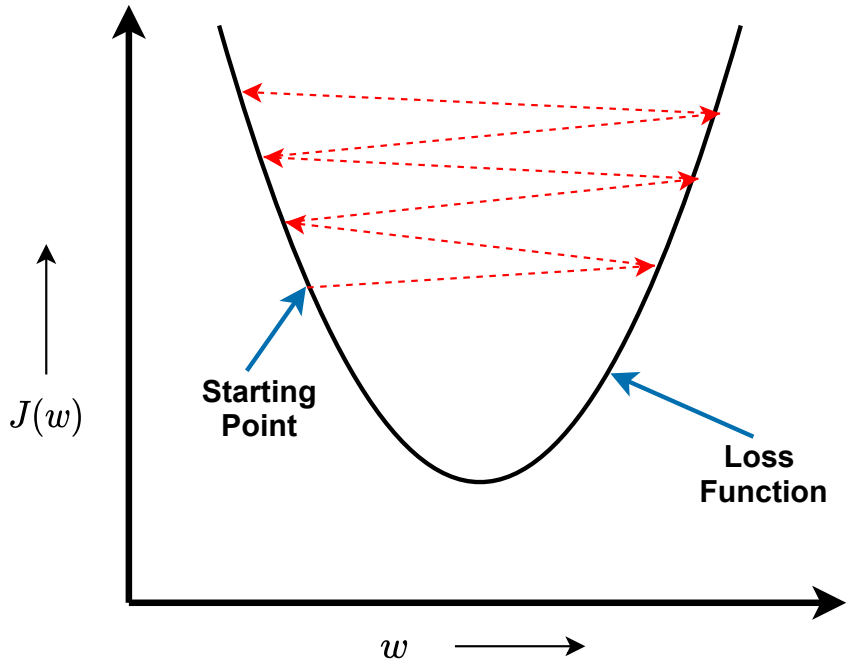
- As the number of examples in the batch go up, we get a more stable estimate of the gradient, whereas with smaller sizes our estimate is generally unstable.
- It changes quite a lot from iteration to iteration.
- This is the price for faster gradient updates.
- Assumption: the training examples are **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



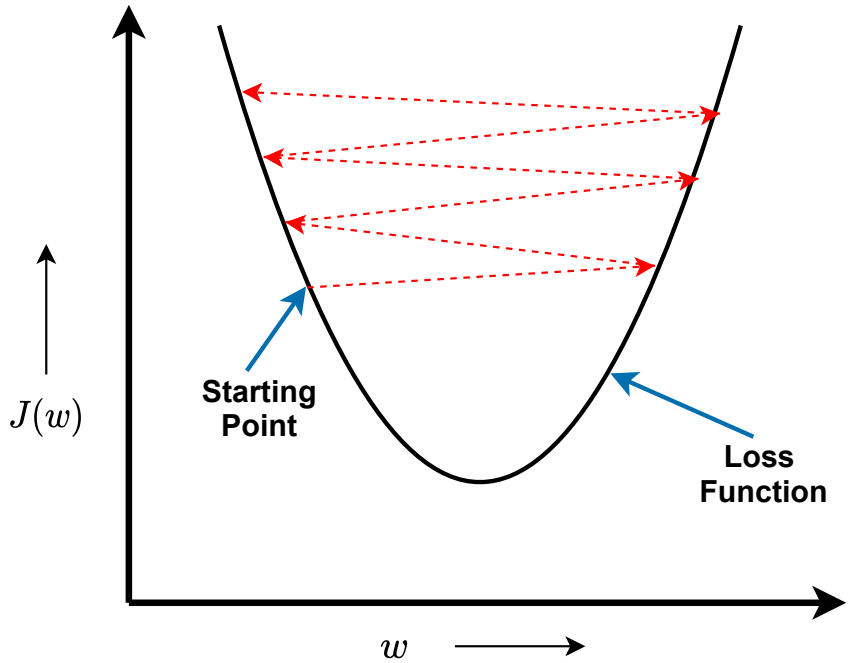
- Larger the learning rate, the longer the steps we take.
- Too large steps can cause to jump over the valley and move to the other side.
- Such jumps would continue with each iteration.
- This would never lead to the valley.

Choosing learning rate



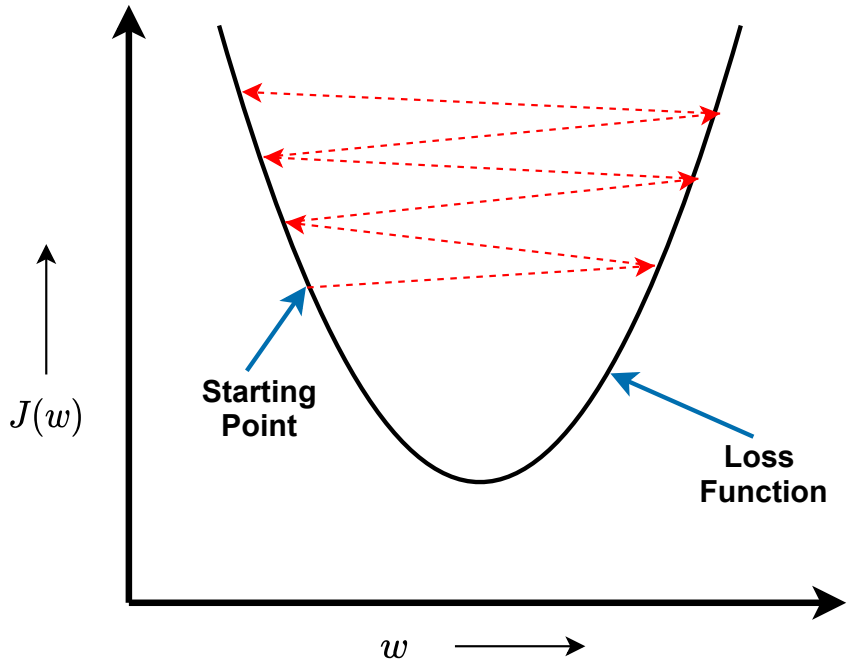
- Larger the learning rate, the longer the steps we take.
- Too large steps can cause to jump over the valley and move to the other side.
- Such jumps would continue with each iteration.
- This would never lead to the valley.

Choosing learning rate



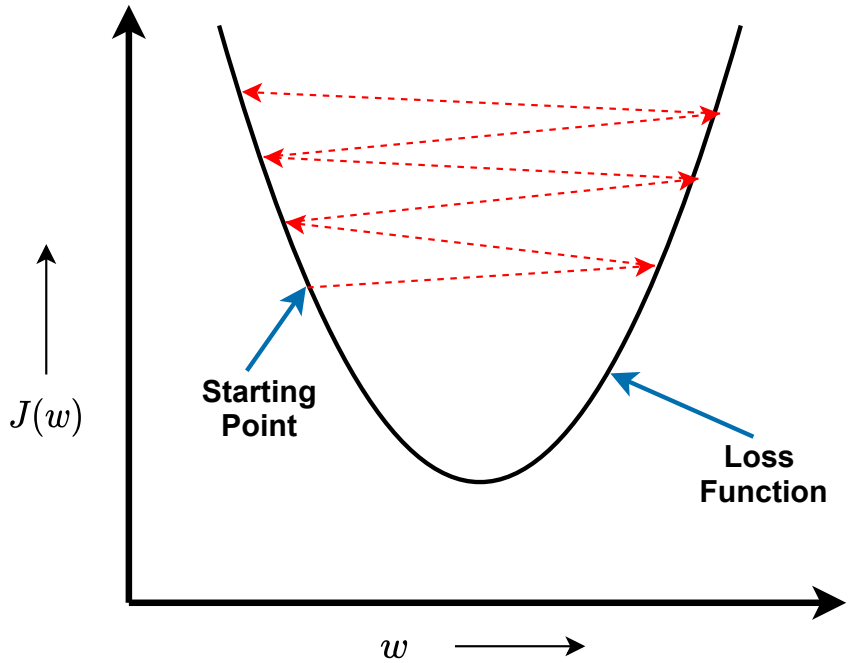
- Larger the learning rate, the longer the steps we take.
- Too large steps can cause to jump over the valley and move to the other side.
- Such jumps would continue with each iteration.
- This would never lead to the valley.

Choosing learning rate



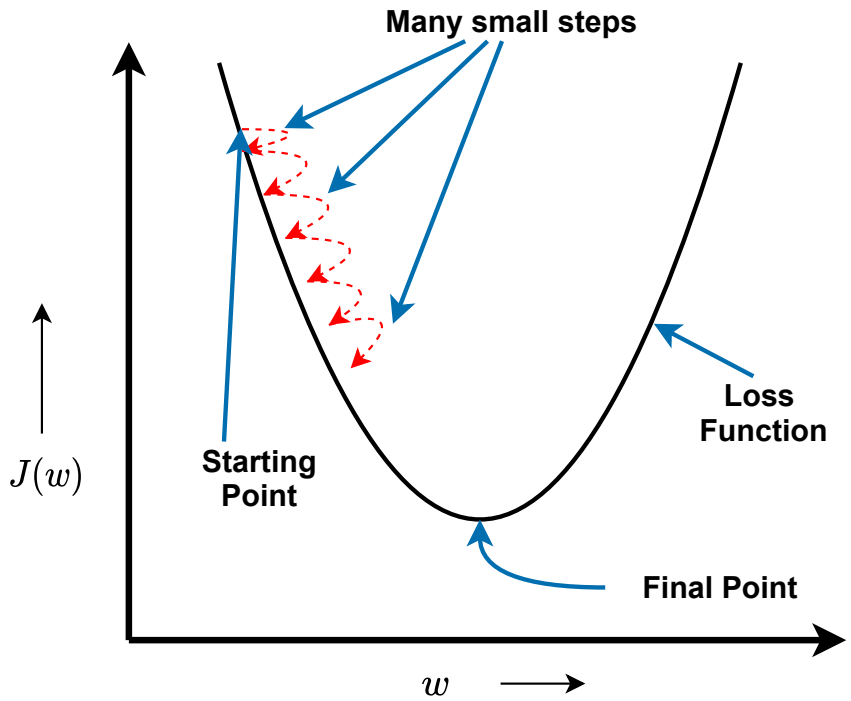
- Larger the learning rate, the longer the steps we take.
- Too large steps can cause to jump over the valley and move to the other side.
- Such jumps would continue with each iteration.
- This would never lead to the valley.

Choosing learning rate



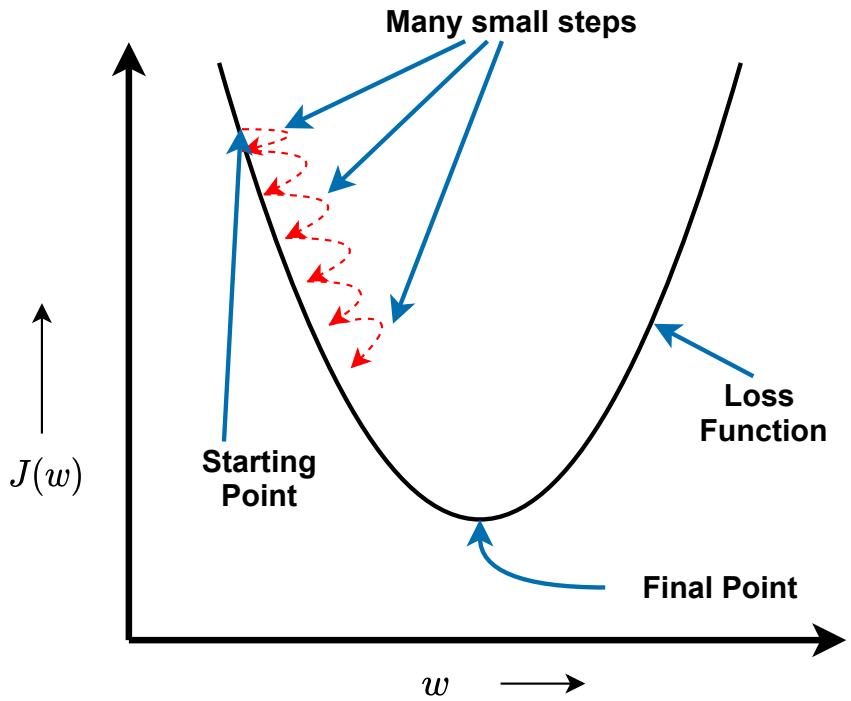
- Larger the learning rate, the longer the steps we take.
- Too large steps can cause to jump over the valley and move to the other side.
- Such jumps would continue with each iteration.
- This would never lead to the valley.

Choosing learning rate



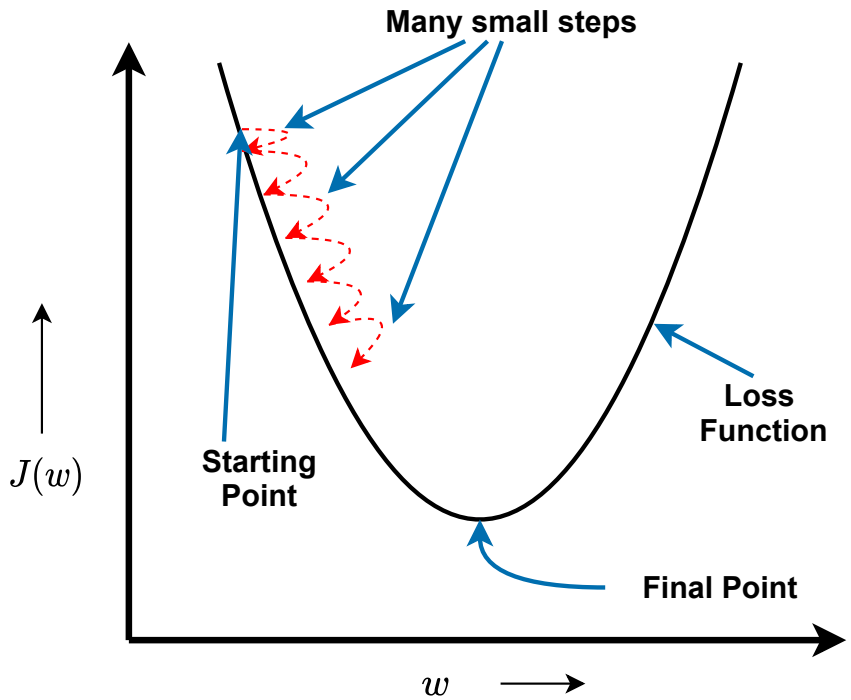
- Too small learning rate causes to take many small steps to reach the valley.
- In each iteration, more gradient computations and weight updates.

Choosing learning rate



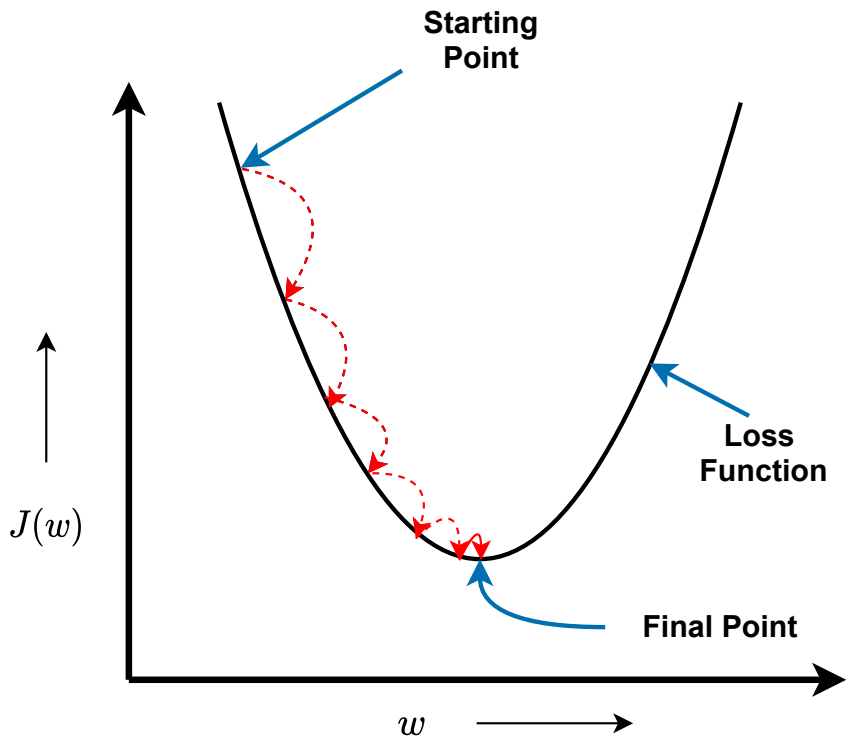
- Too small learning rate causes to take many small steps to reach the valley.
- In each iteration, more gradient computations and weight updates.

Choosing learning rate



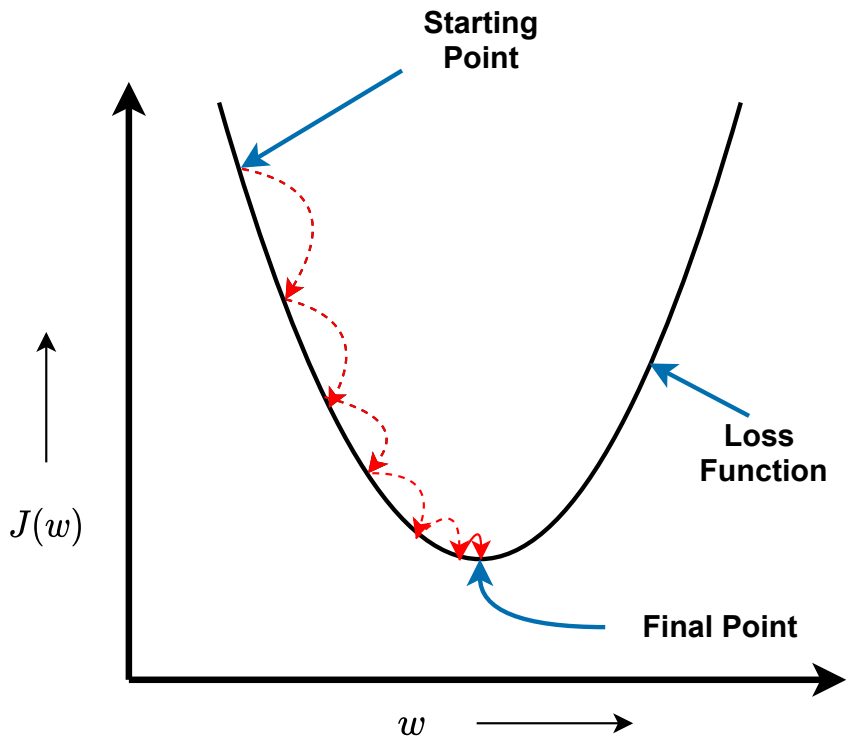
- Too small learning rate causes to take many small steps to reach the valley.
- In each iteration, more gradient computations and weight updates.

Choosing learning rate: Learning curves



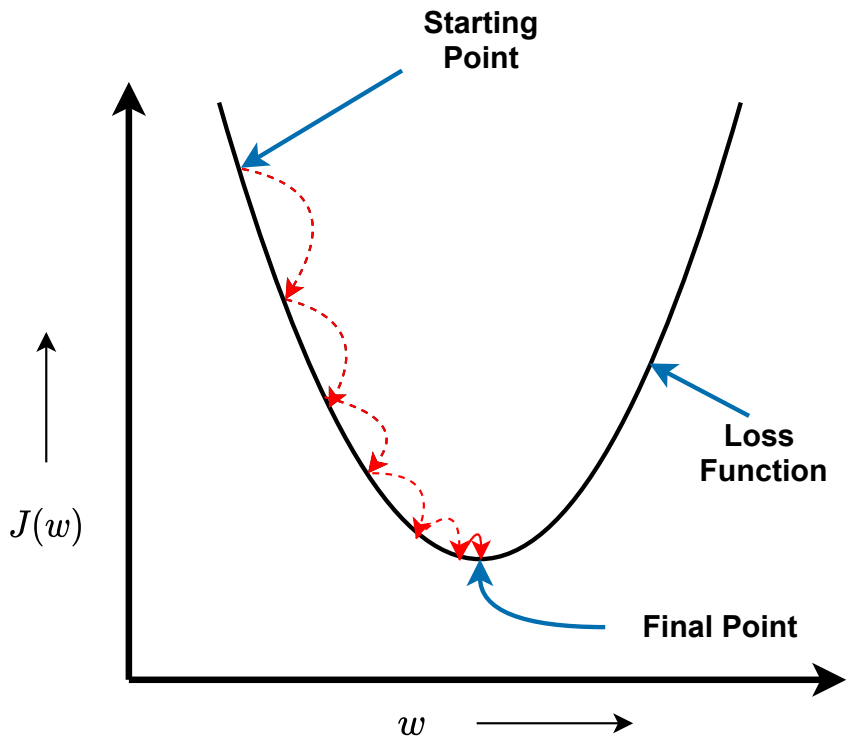
- Choose a rate that is not too small or too big.
- You would wonder how to do that.

Choosing learning rate: Learning curves



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

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

- 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

- 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

- 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

- 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

- 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

- 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

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

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

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.

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.

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.

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.

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

- 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

- 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

- 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

- 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

- 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

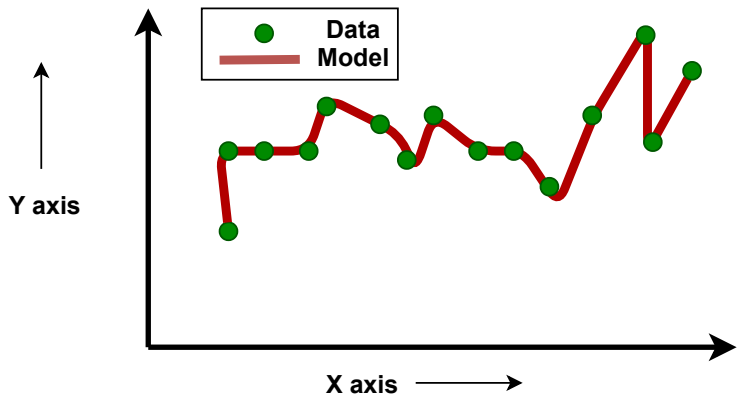
- The model is evaluated on validation data after training
- Many metrics can be used depending on the problem type.
- Take into account metrics on validation examples for evaluating accuracy of the model.
- Calculate both training and validation metrics and compare them for diagnostics related to over/underfitting.
- 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.

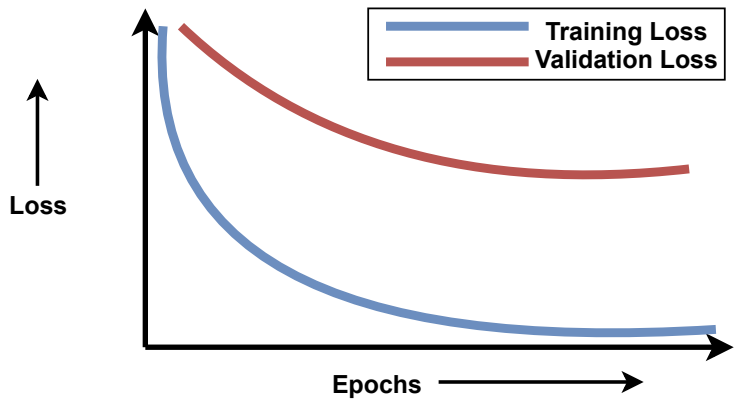
Evaluation

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

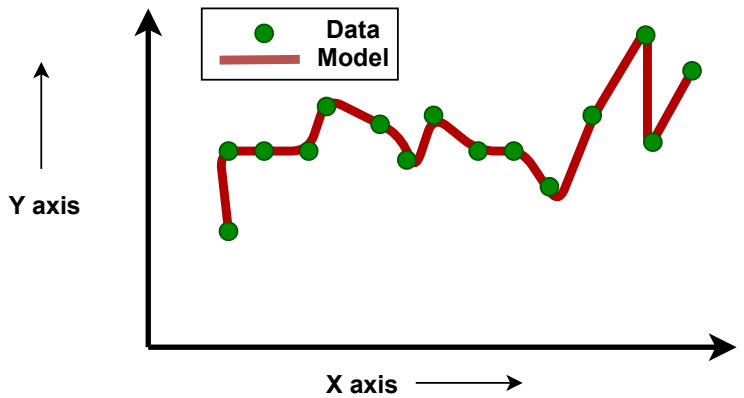
Overfitting

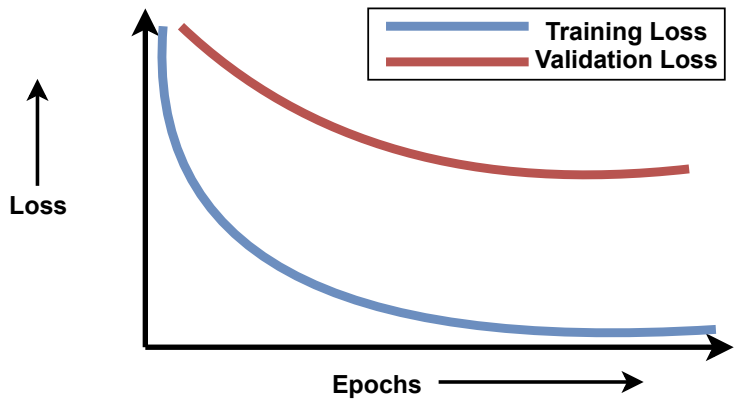




- An example: - when the kids are given a question bank for a topic and the kids only prepare questions from the question bank and nothing outside it.
- They are able to answer the questions from the question bank but are unable to answer questions outside.
- This is overfitting to the questions in the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

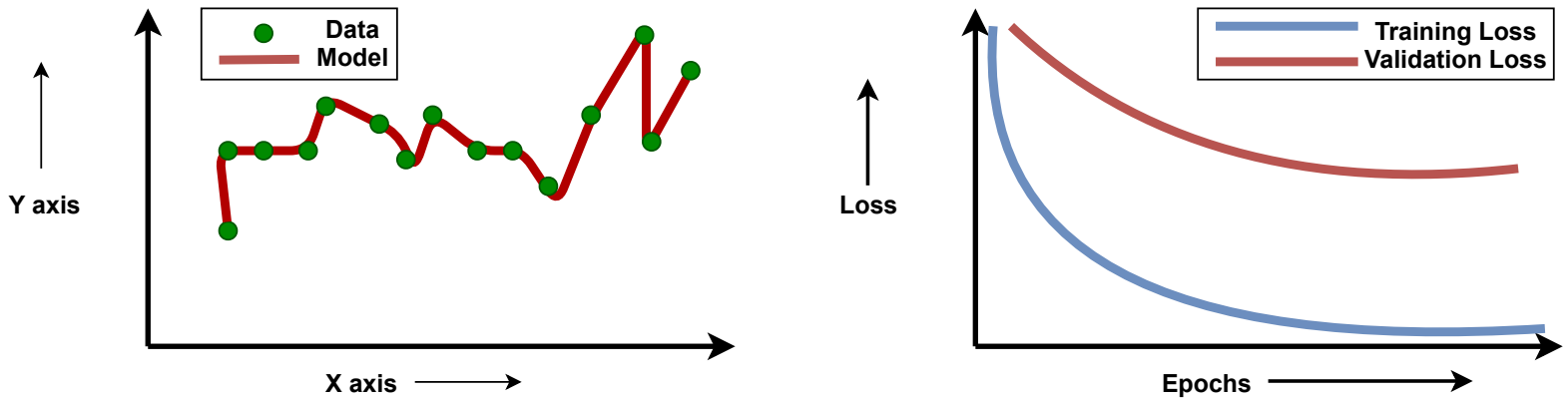
Overfitting





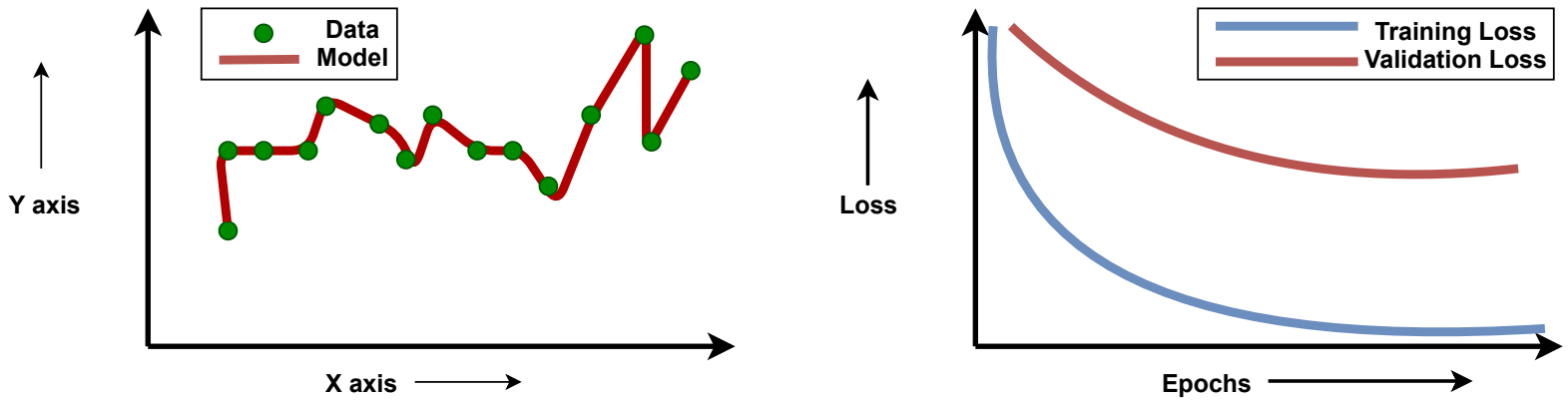
- An example: - when the kids are given a question bank for a topic and the kids only prepare questions from the question bank and nothing outside it.
- They are able to answer the questions from the question bank but are unable to answer questions outside.
- This is overfitting to the questions in the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

Overfitting



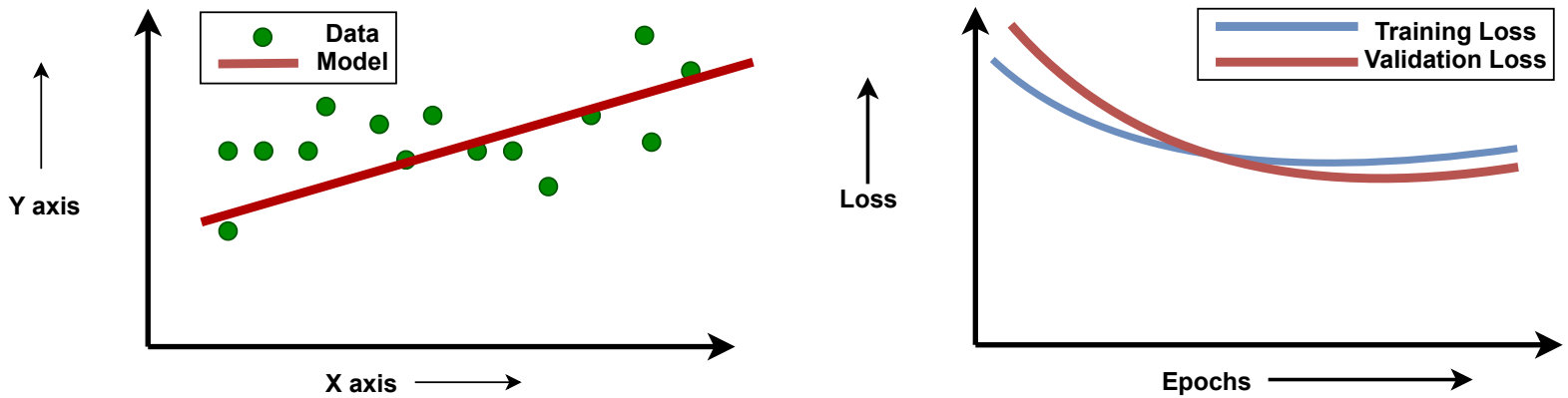
- An example: - when the kids are given a question bank for a topic and the kids only prepare questions from the question bank and nothing outside it.
- They are able to answer the questions from the question bank but are unable to answer questions outside.
- This is overfitting to the questions in the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

Overfitting



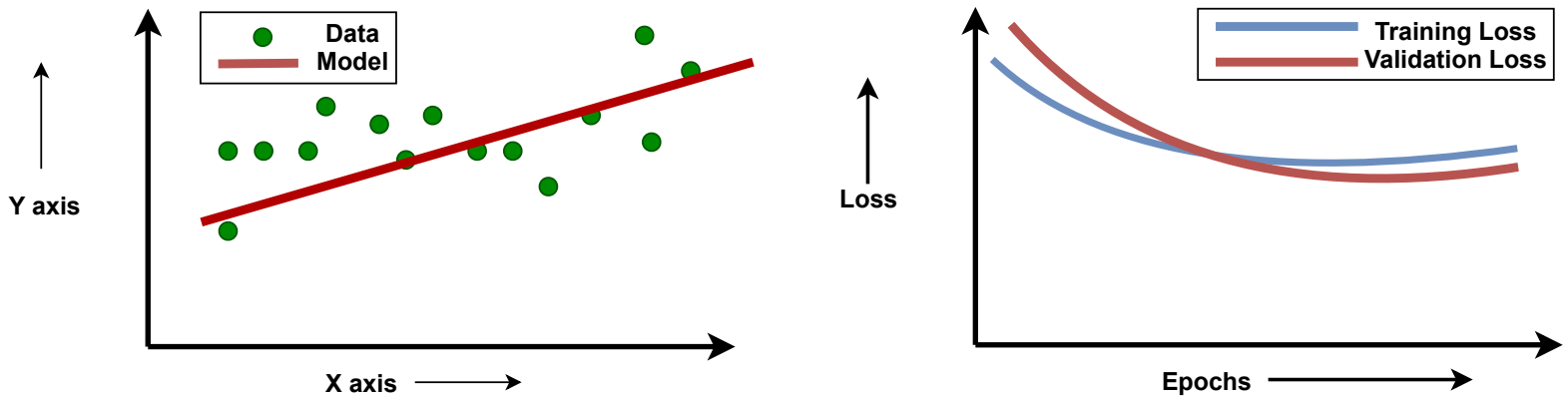
- An example: - when the kids are given a question bank for a topic and the kids only prepare questions from the question bank and nothing outside it.
- They are able to answer the questions from the question bank but are unable to answer questions outside.
- This is overfitting to the questions in the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

Underfitting



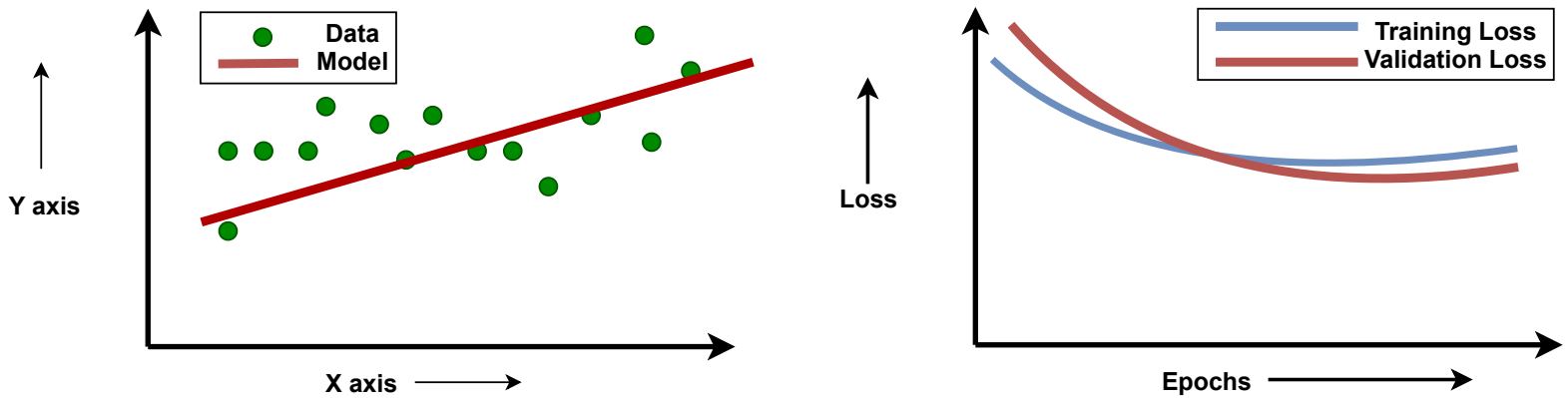
- The underfitting here would make kids ignore the questions and only prepare a few topics.
- They answer questions based on those few topics, but their performance remains poor in the exam which is based on all the topics.

Underfitting



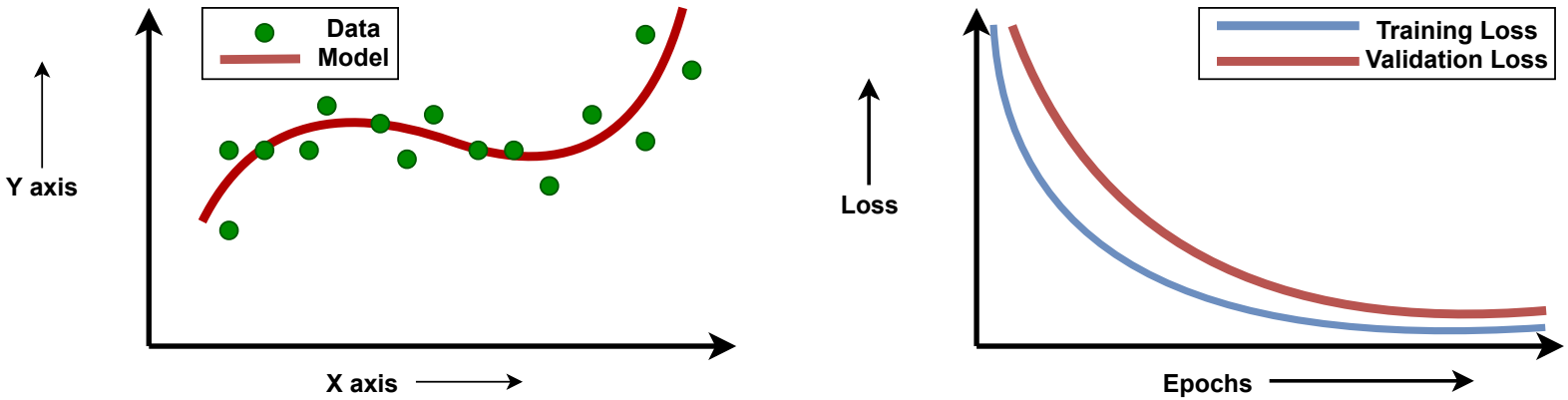
- The underfitting here would make kids ignore the questions and only prepare a few topics.
- They answer questions based on those few topics, but their performance remains poor in the exam which is based on all the topics.

Underfitting



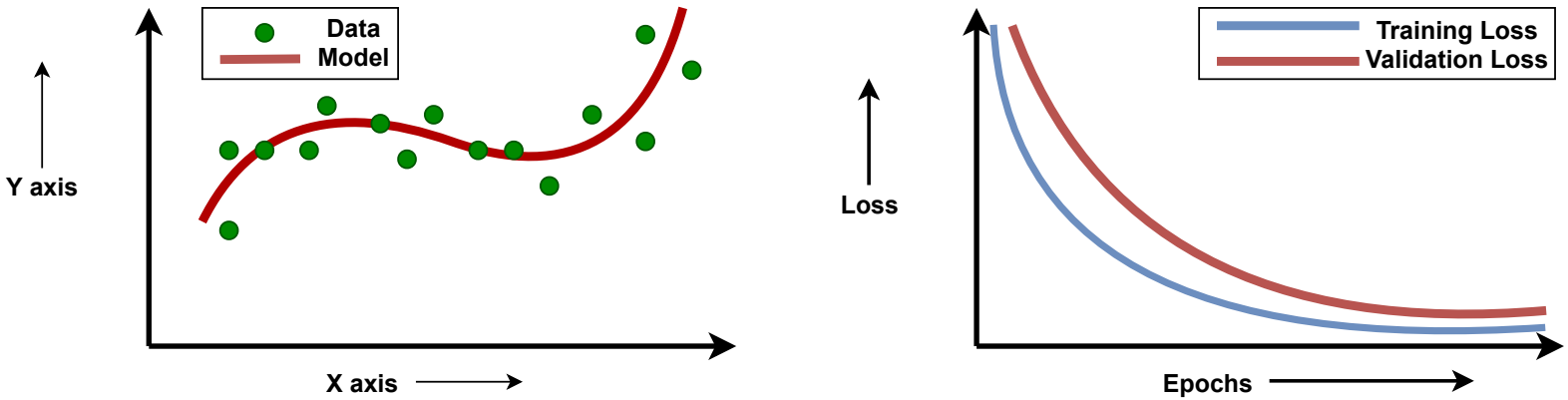
- The underfitting here would make kids ignore the questions and only prepare a few topics.
- They answer questions based on those few topics, but their performance remains poor in the exam which is based on all the topics.

The right fit



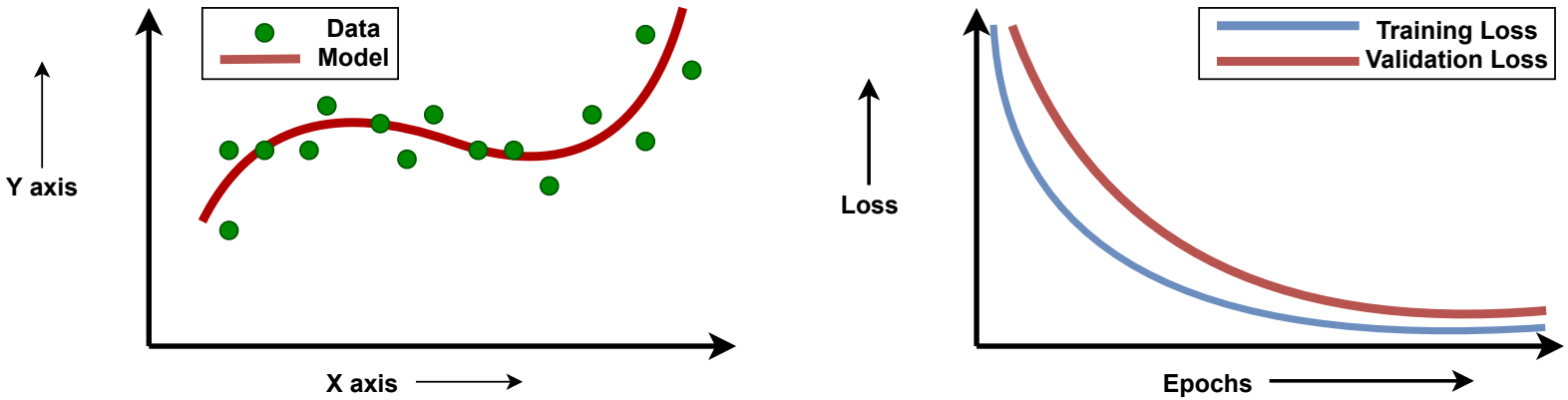
- The right fit here would mean - that the kids prepare questions from the question bank and also prepare similar questions on other concepts and try to understand the basics.
- They are likely to do well in the exam as they took into account the data and what would generalize it.

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.

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.

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 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 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 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 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.

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.

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.

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	$\frac{1}{n} * \sum_{i=1}^{n} \left\lvert\hat{y}^{(i)} - y^{(i)}\right\rvert$

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.

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.

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.

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.

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 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.

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 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.

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.

Precision

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

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

- Percentage of correct positive identifications.

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.

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.

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.

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

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.

- 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.

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.

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.

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.

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.

Precision-Recall (PR) curve

Steps:

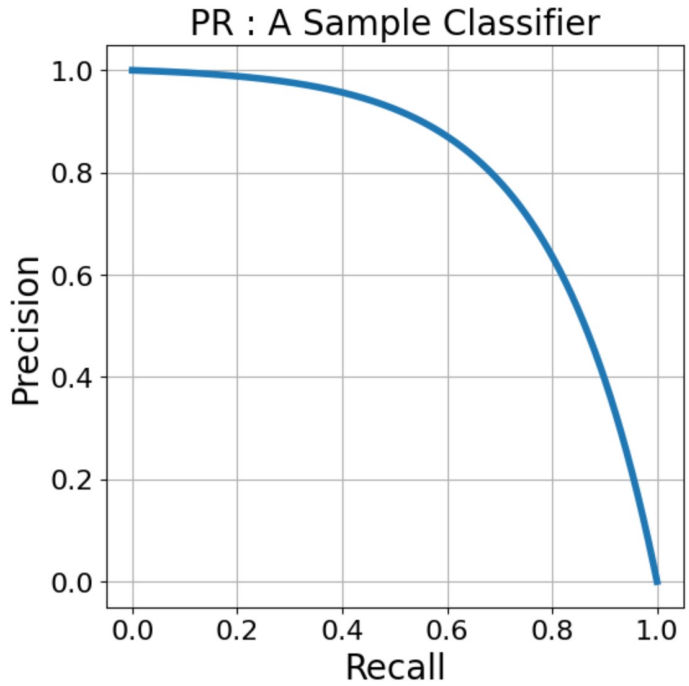
- Fix the probability threshold, say r.
- Get predictions for positive and negative classes as per r.
- Calculate the confusion matrix at this threshold and calculate precision and recall metrics.
- Store these metrics for the threshold.
- Choose multiple r s between 0 to 1 and for each value of r, calculate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

Precision-Recall (PR) curve

Steps:

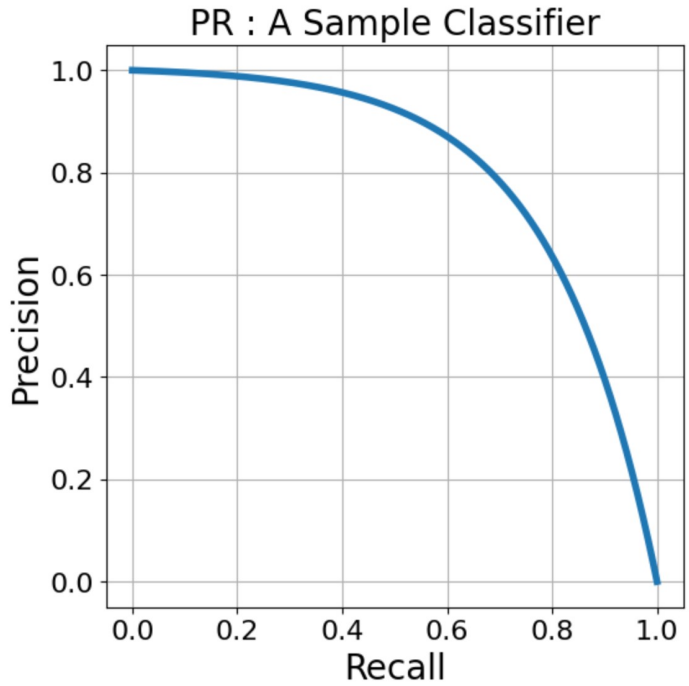
- Fix the probability threshold, say r.
- Get predictions for positive and negative classes as per r.
- Calculate the confusion matrix at this threshold and calculate precision and recall metrics.
- Store these metrics for the threshold.
- Choose multiple r s between 0 to 1 and for each value of r, calculate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

PR curve



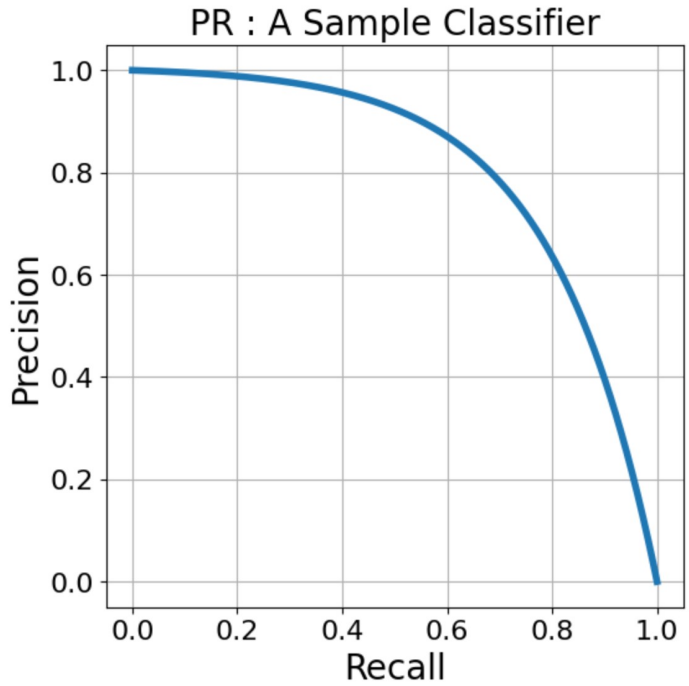
- PR curve has precision on y-axis and recall on x-axis.
- As precision increases, recall drops and vice versa.
- Plot values of recall and precision for each probability threshold from 0 to 1.

PR curve



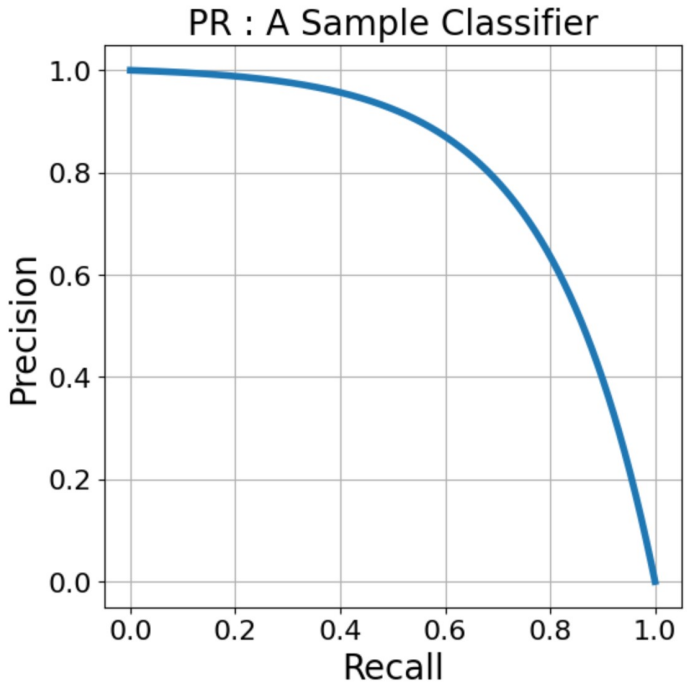
- PR curve has precision on y-axis and recall on x-axis.
- As precision increases, recall drops and vice versa.
- Plot values of recall and precision for each probability threshold from 0 to 1.

PR curve



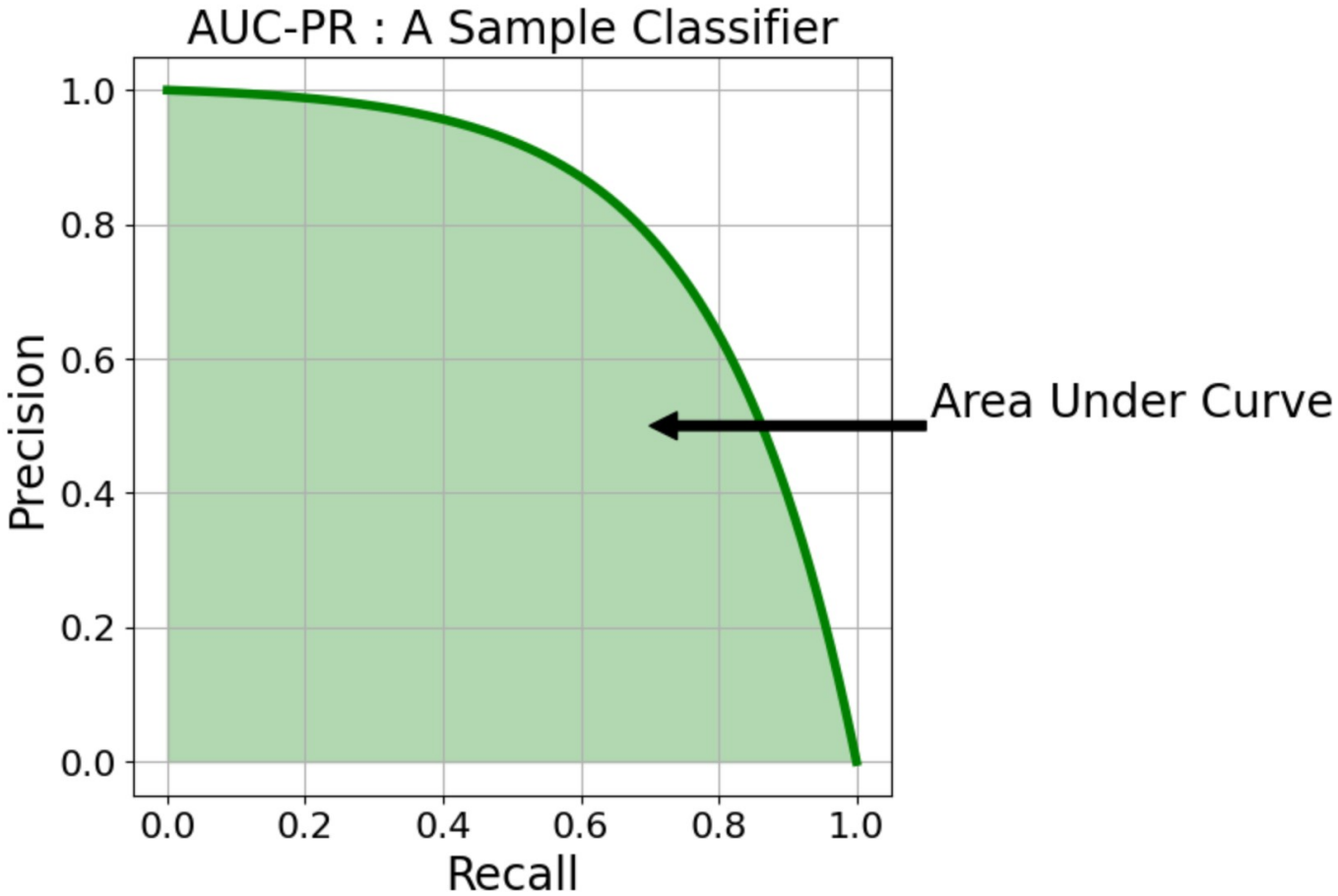
- PR curve has precision on y-axis and recall on x-axis.
- As precision increases, recall drops and vice versa.
- Plot values of recall and precision for each probability threshold from 0 to 1.

PR curve



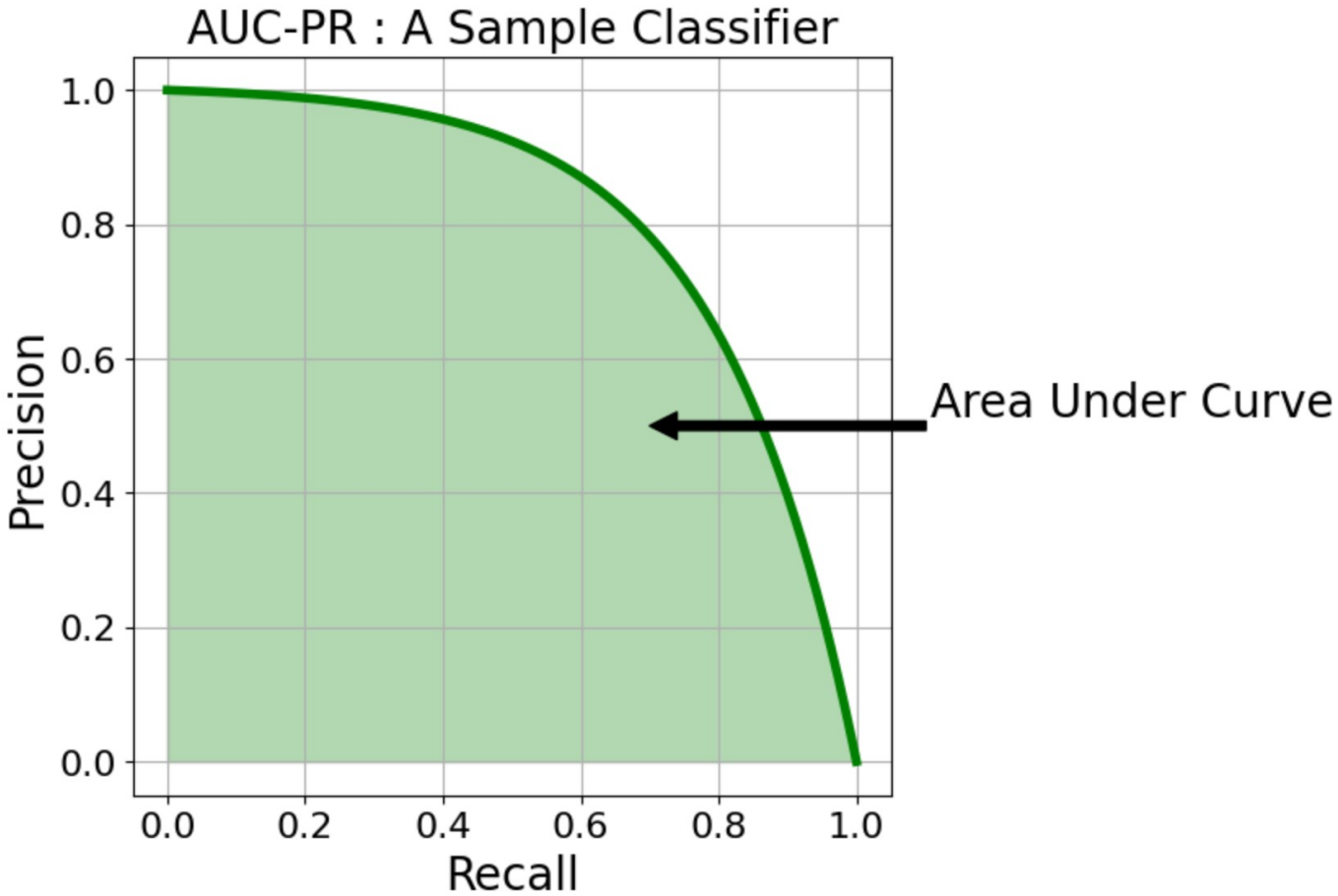
- PR curve has precision on y-axis and recall on x-axis.
- As precision increases, recall drops and vice versa.
- Plot values of recall and precision for each probability threshold from 0 to 1.

PR curve: AUC



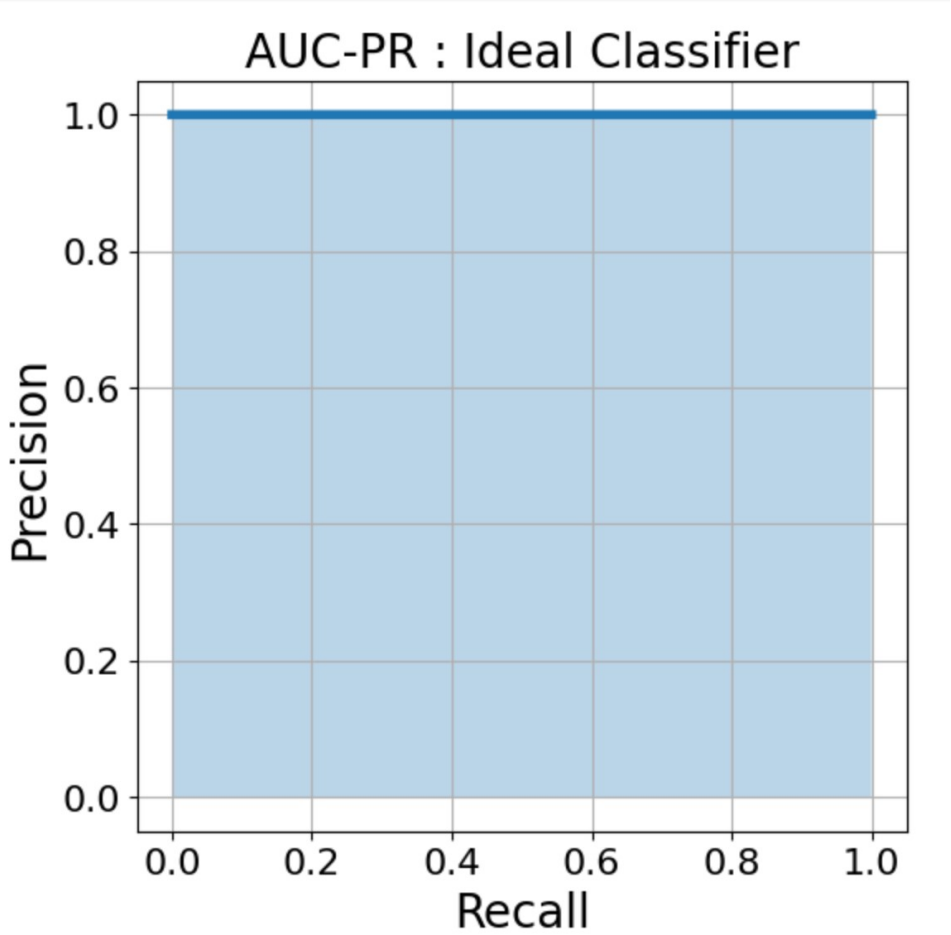
- Area under the curve (AUC) measures effectiveness of the model. It is called AUC-PR.

PR curve: AUC



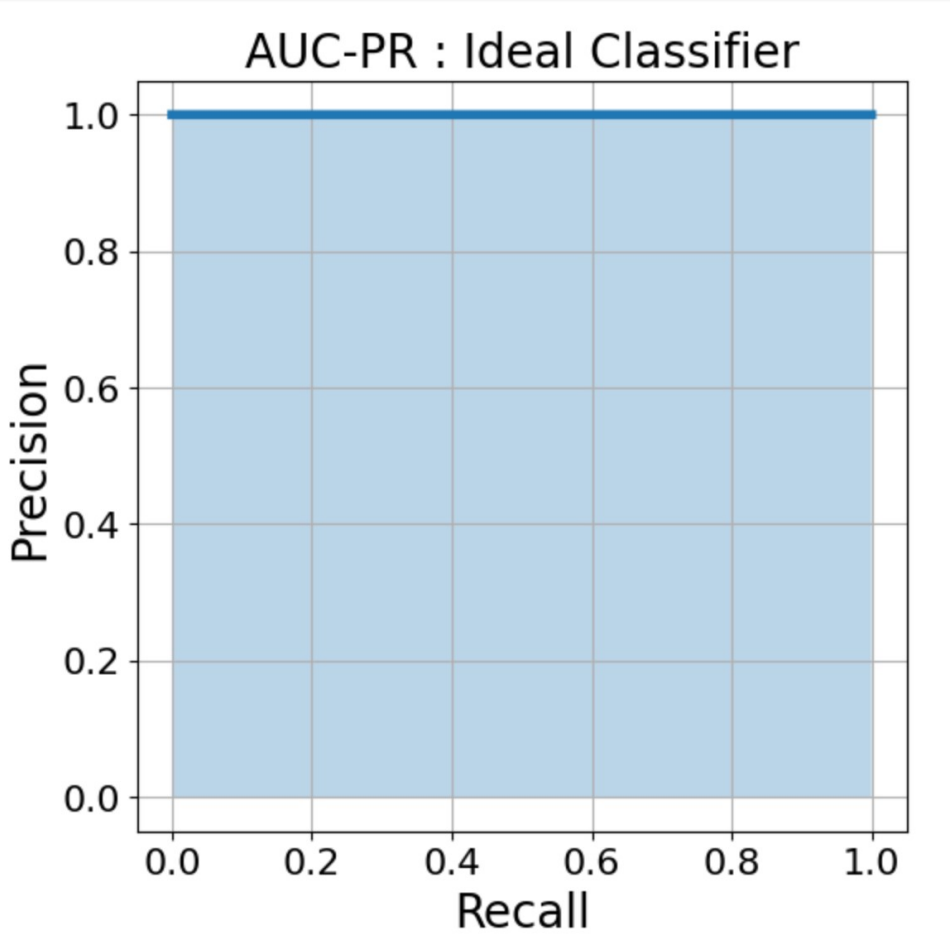
- Area under the curve (AUC) measures effectiveness of the model. It is called AUC-PR.

PR curve: An ideal classifier



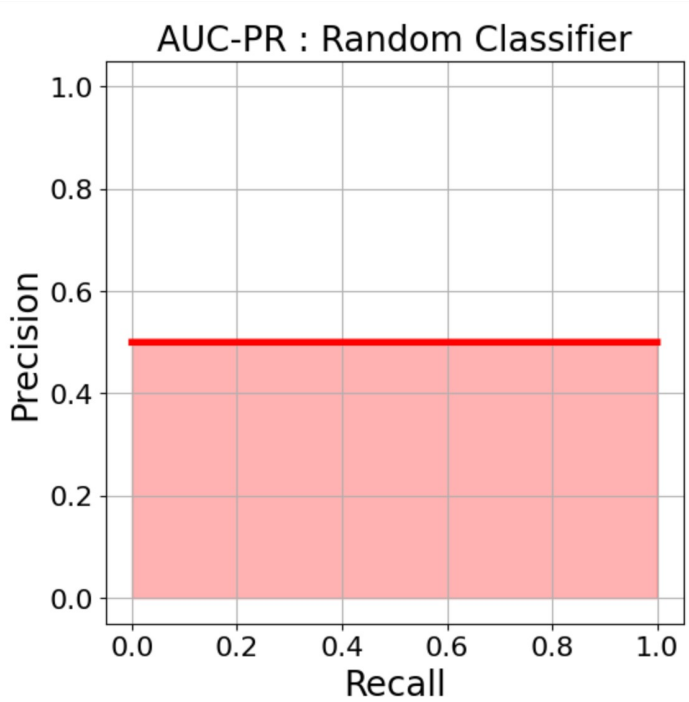
- Ideal classifier has AUC-PR equal to 1.

PR curve: An ideal classifier



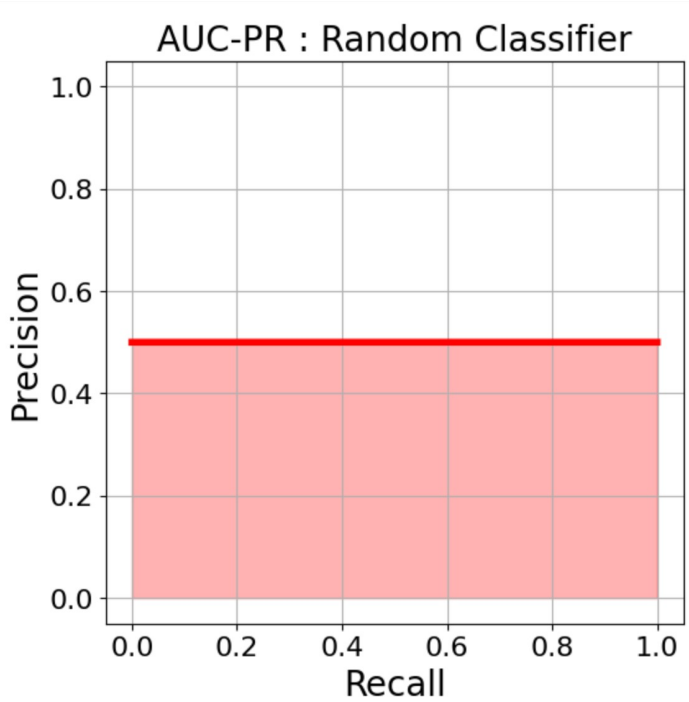
- Ideal classifier has AUC-PR equal to 1.

PR Curve: A random classifier



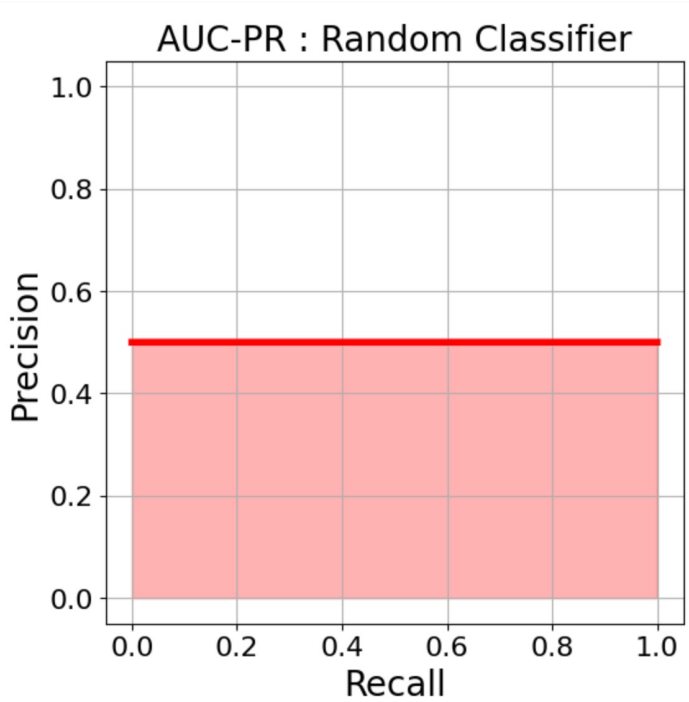
- Above is the PR curve of a random or no-skill classifier. It is a horizontal line at precision equal to the percentage of the majority class.
- For a balanced class setting, the baseline is 0.5, as displayed in the figure above.

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.

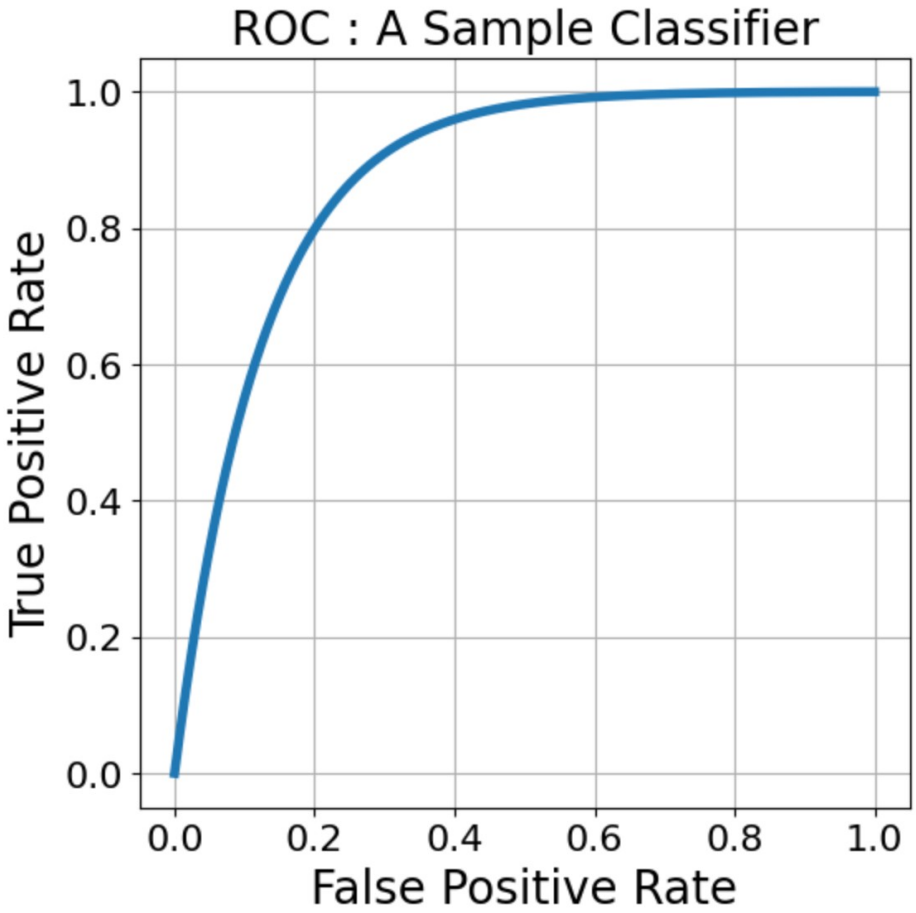
PR Curve: A random classifier



- 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

- **FPR**: False positive rate:

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

- **TPR**: True positive rate:

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

ROC curve

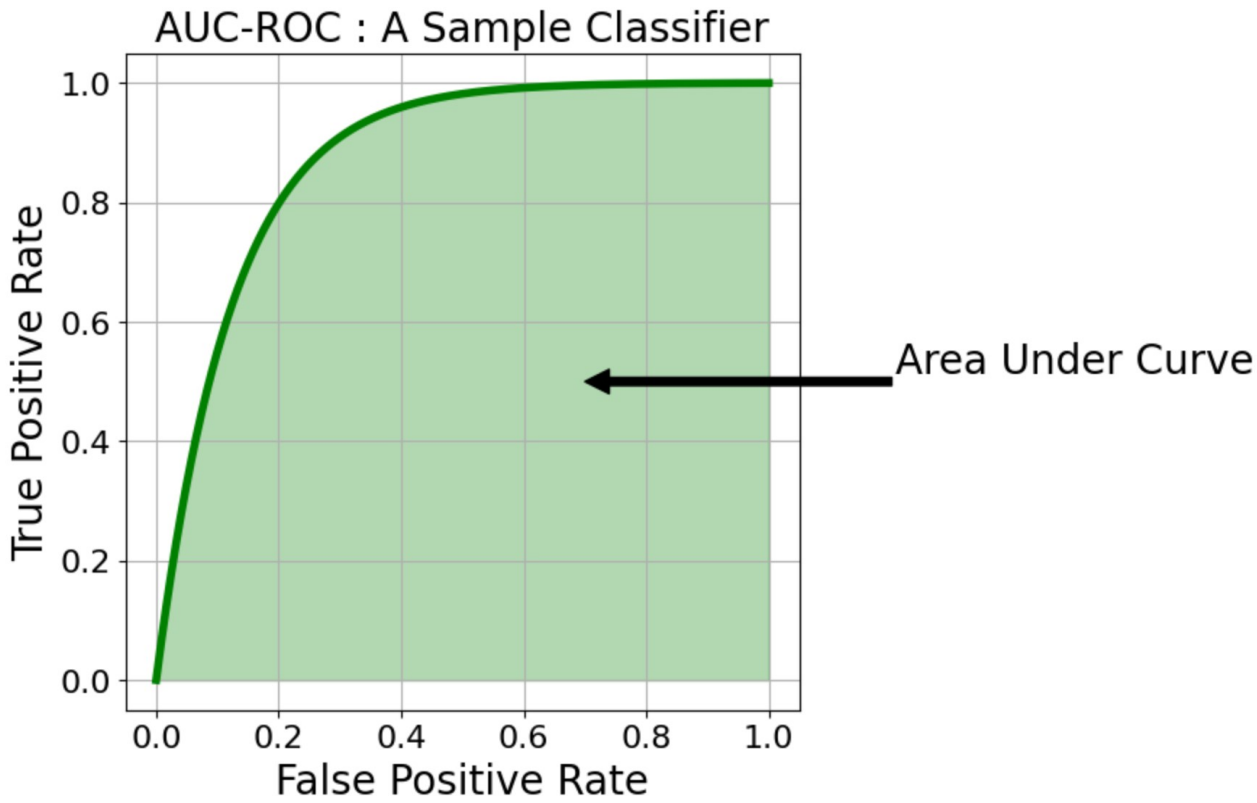
- **FPR**: False positive rate:

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

- **TPR**: True positive rate:

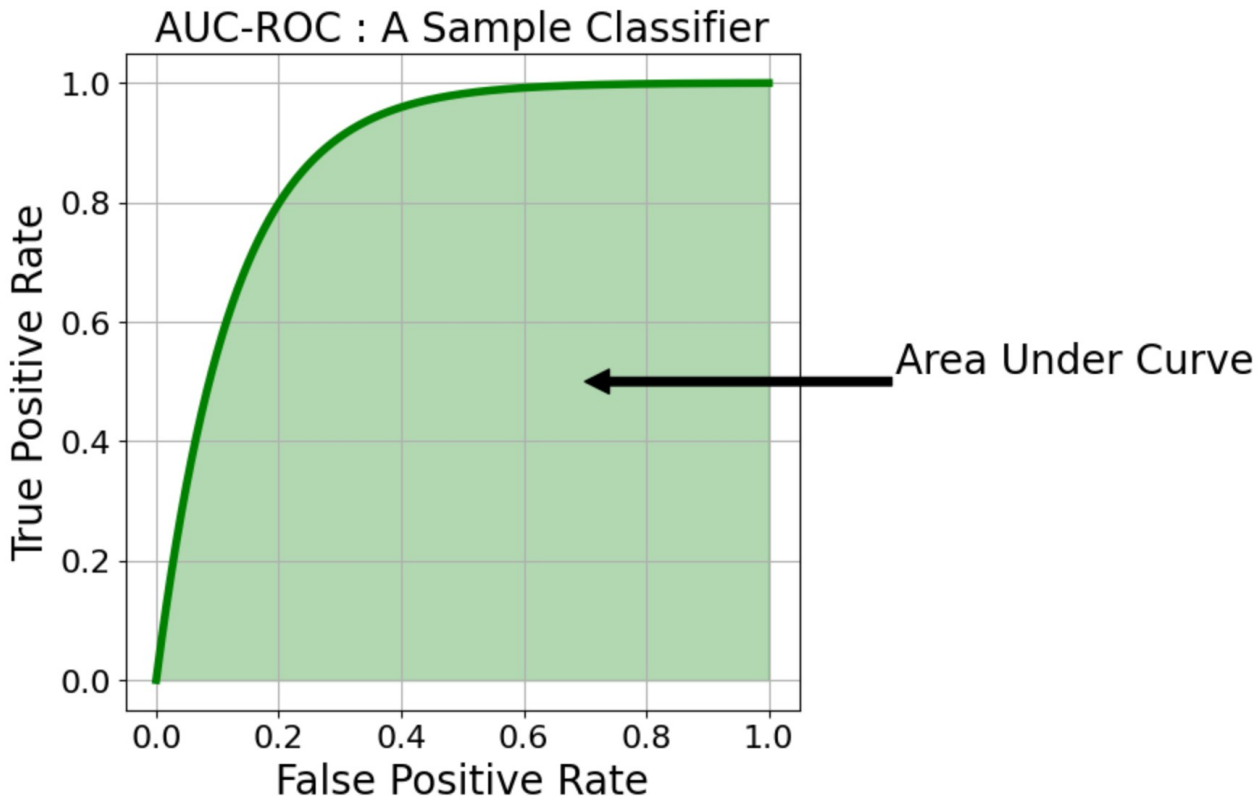
$$TPR = Sensitivity = \frac{TP}{FN + TP}$$

ROC curve: AUC



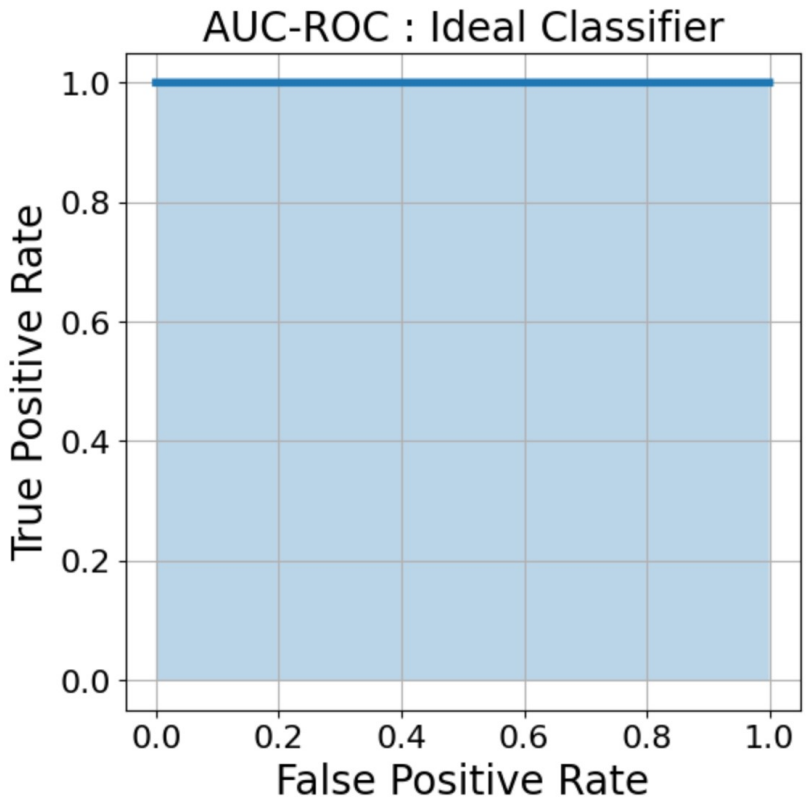
- Area under the curve (AUC) measures performance of the classifier. Higher the better.

ROC curve: AUC



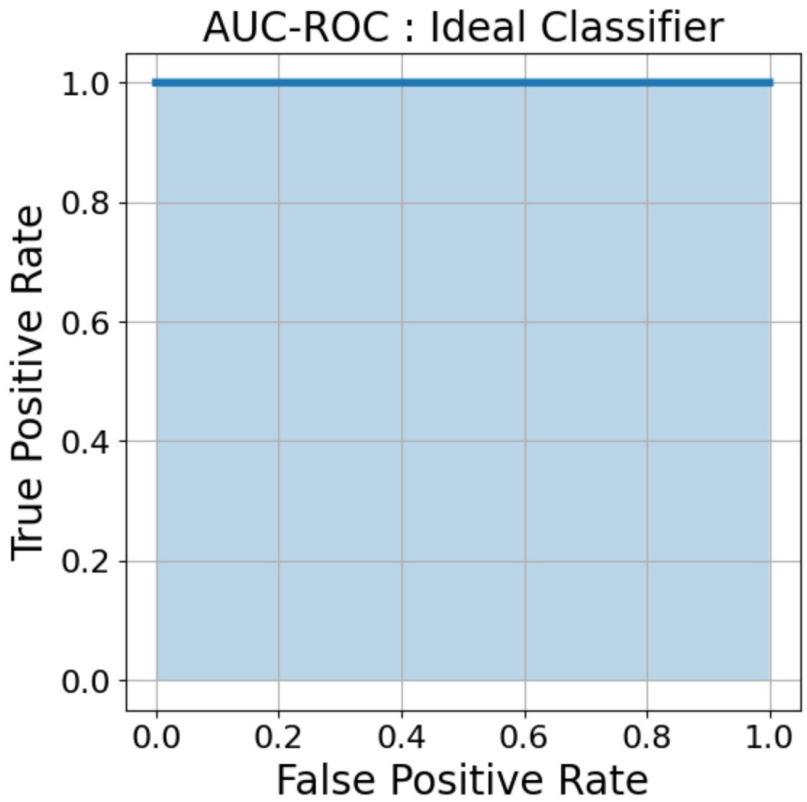
- Area under the curve (AUC) measures performance of the classifier. Higher the better.

AUC-ROC : Ideal Classifier



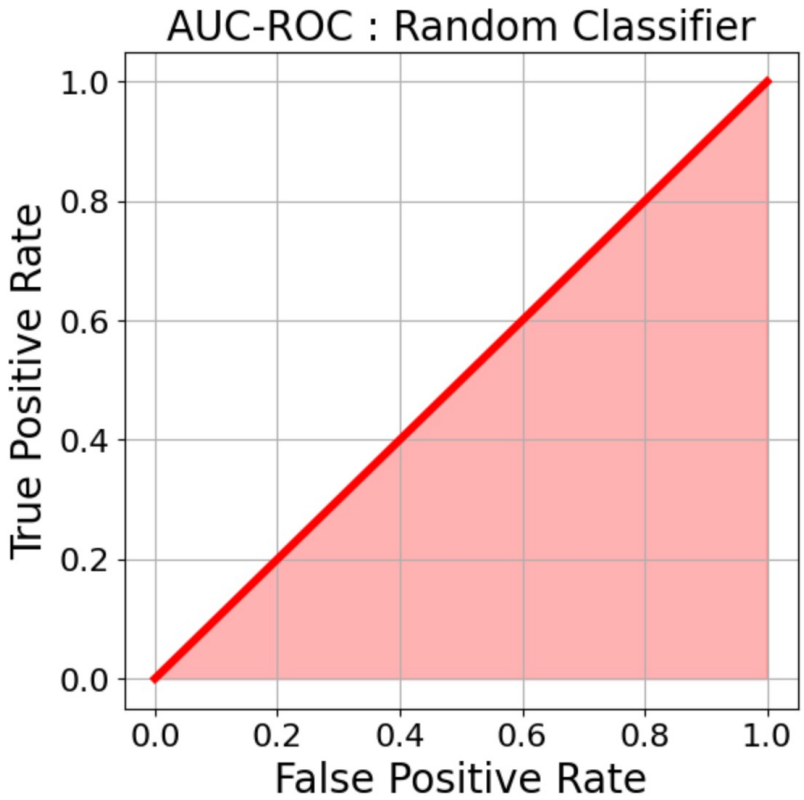
- The ideal classifier has AUC-ROC=1.

AUC-ROC : Ideal Classifier



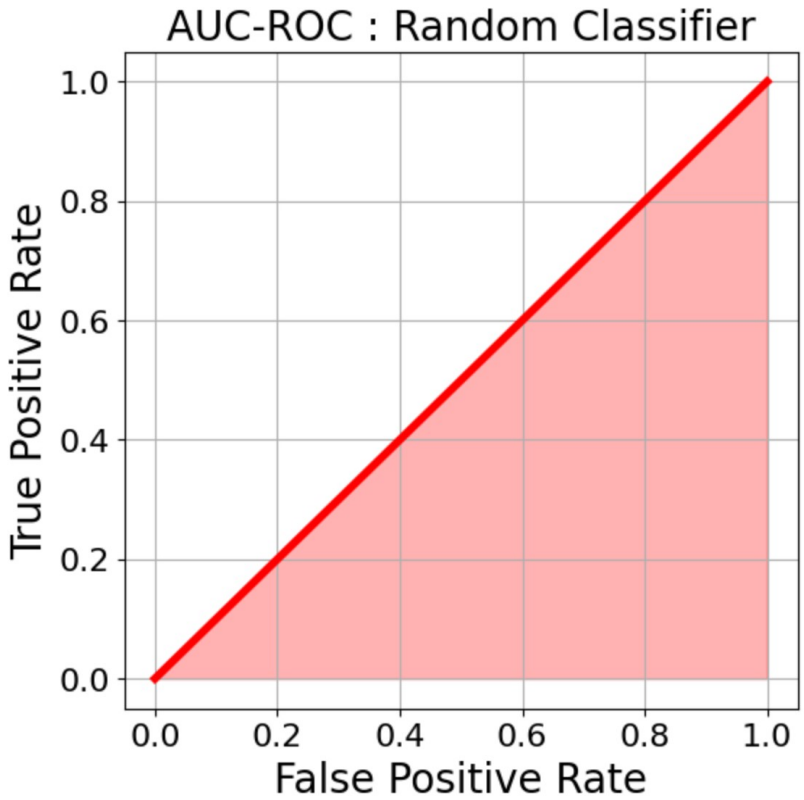
- The ideal classifier has AUC-ROC=1.

AUC-ROC : A random classifier



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

AUC-ROC : A 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.

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.

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.

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.

‘