Machine Leaning
Model Evaluation

Prof. Anil Singh Parihar
Department of Computer Science & Engineering
Delhi Technological University, Delhi

After building your ML model..

- How well is my model doing? Is it a useful model?
- Will training my model on more data improve its performance?
- Do I need to include more features?

Why Evaluation is necessary for successful model?

- Overfitting and under-fitting are the two biggest causes
- for poor performance of machine learning algorithms.

Why Evaluation is necessary for successful model?

- Overfitting: Occurs when the Model performs well for a particular set of data (Known Data), and

- may therefore fail to fit additional data (Unknown Data) or predict future observations reliably.

Why Evaluation is necessary for successful model?

- Overfitting refers to a model that has:
- learned the training dataset too well,
- including the statistical noise or random fluctuations in the training dataset.

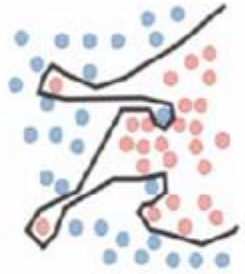
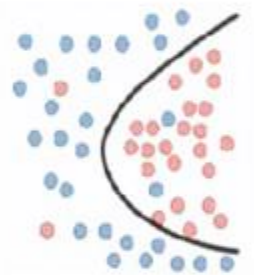
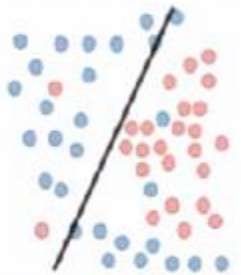
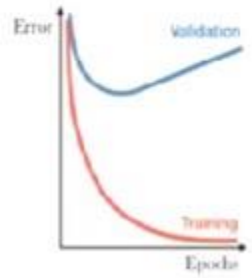
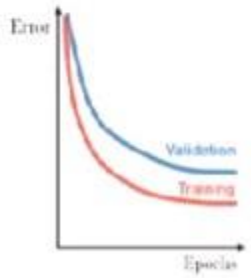
Why Evaluation is necessary for successful model?

- Under-fitting: Occurs when the model cannot adequately capture the underlying structure of the data.

Why Evaluation is necessary for successful model?

- Generalization: refers to how well the underlying structure of the data, learned by a machine learning model apply to

- specific examples not seen by the model when it was learning.

	Overfitting	Generalization	Underfitting
Regression Illustration			
Classification Illustration			
Deep Learning Illustration			

Bias & Variance

- Bias and variance are related to the predictive performance and generalization ability of machine learning models.
- They represent two different sources of errors in model predictions and are critical for understanding model behavior and making improvements.

Bias & Variance

- Some machine learning models provide the framework for generalization by suggesting the underlying structure of that knowledge.

- For example, a linear regression model imposes a framework to learn linear relationships between the information we feed it.

Bias

- Bias is the error introduced by approximating a real-world problem, which may be complex, by a much simpler model.
- In other words, it's a measure of how closely the model's predictions match the true values in the training data.

Bias

- In some cases, a model with too much pre-built structure can limit the model's ability to learn from the examples

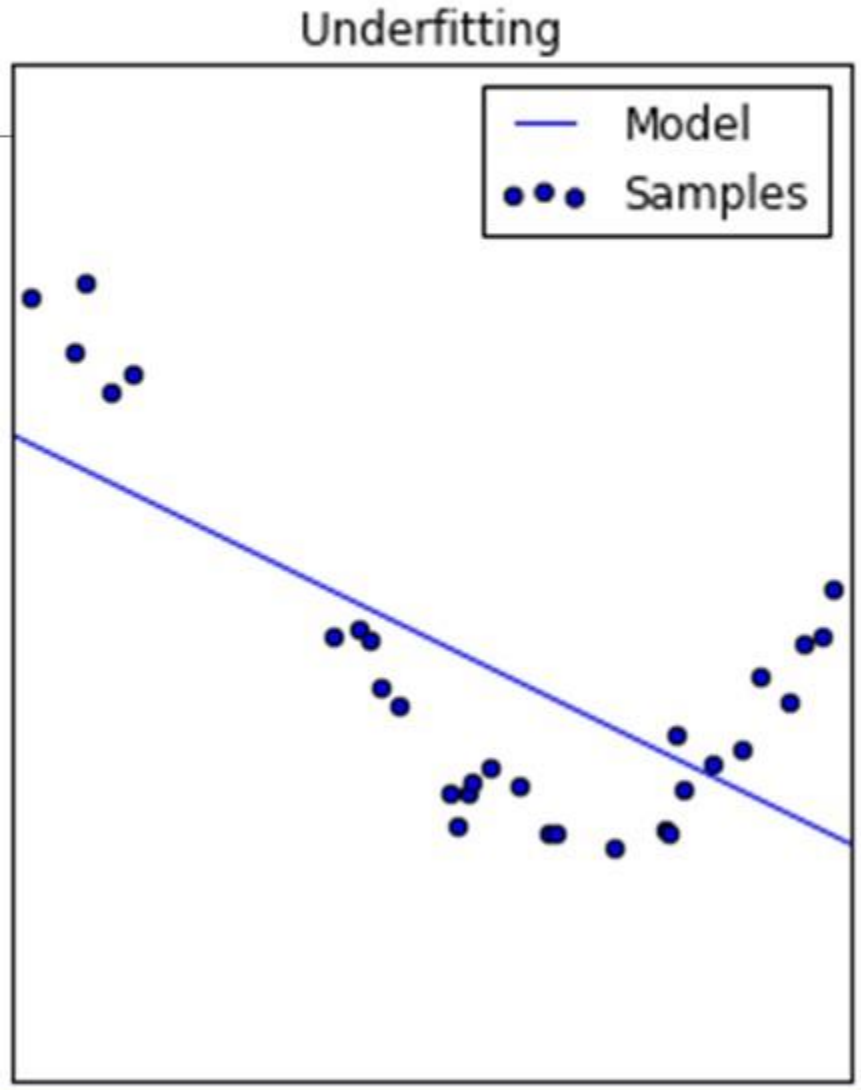
- Such as the case where we train a linear model on a exponential dataset.

Bias

- In this case, our model is **biased** by the pre-imposed structure and relationships (linear relation in above example).
- Models with high bias pay little attention to the data (underlying behavior/trends) presented;

Bias

- A model with high bias is said to underfit the data.
- As, it doesn't capture the underlying patterns and relationships in the data because it's too simplistic or too constrained.



Bias

Examples of high-bias models:

- Linear regression models on highly nonlinear data.
- Models with too few parameters to capture the complexity of the data.

Bias

- Reducing bias often involves:
- Using more complex models or increasing the model's capacity to capture intricate patterns in the data.

Variance

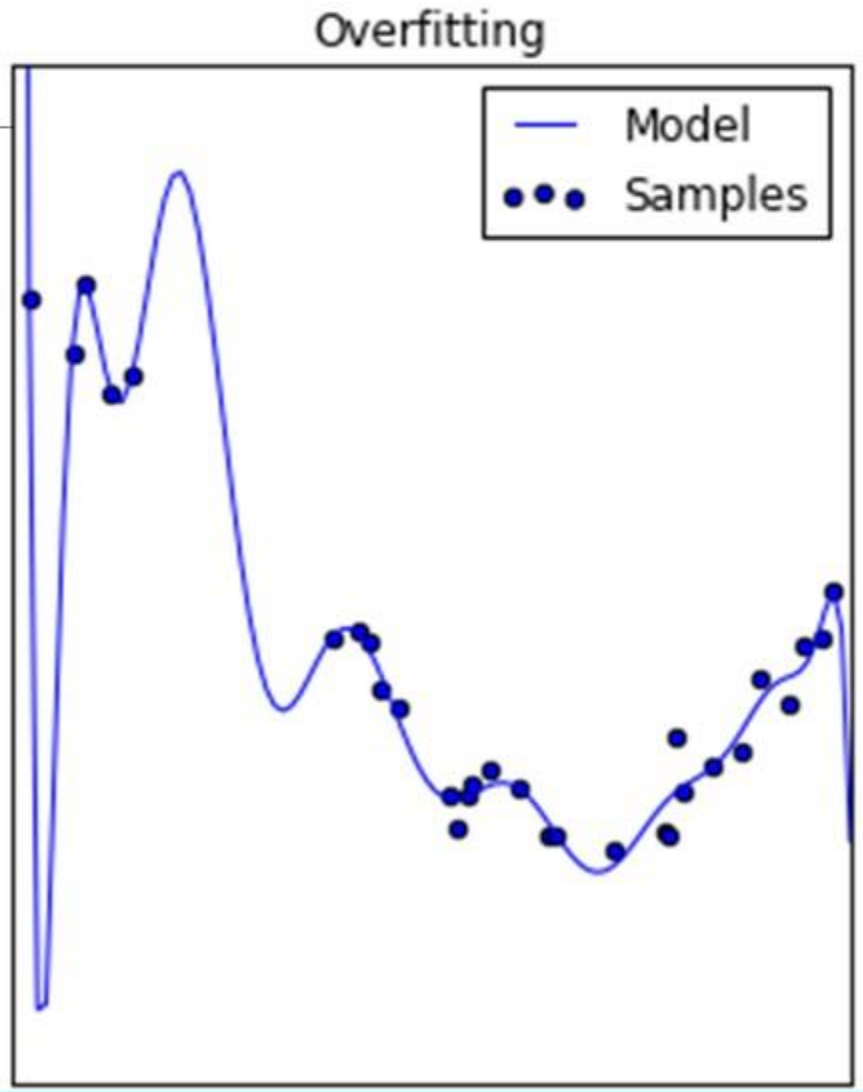
- Variance is the error introduced by the model's sensitivity to small fluctuations or noise in the training data.
- It quantifies how much the model's predictions vary when trained on different subsets of the data.

Variance

- A model with high variance is said to overfit the training data.
- This means it learns to capture not only the underlying patterns (true trends) but also the noise (wild fluctuations) in the training data.
- In other words, such, learns too much from the training data even noise.

Variance

- In this case, model does not generalize well because
- it pays too much attention to the training data.
- High variance often leads to models that perform very well on the training data but poorly on unseen data.



Variance

Reducing variance often involves techniques like:

- Regularization (e.g., L1 or L2 regularization),
- Reducing model complexity,
- increasing the amount of training data, or
- Using ensemble methods like bagging or boosting.

Summary: Bias & Variance

- A model with **high bias** is limited from learning the true trend
  - and underfits the data.

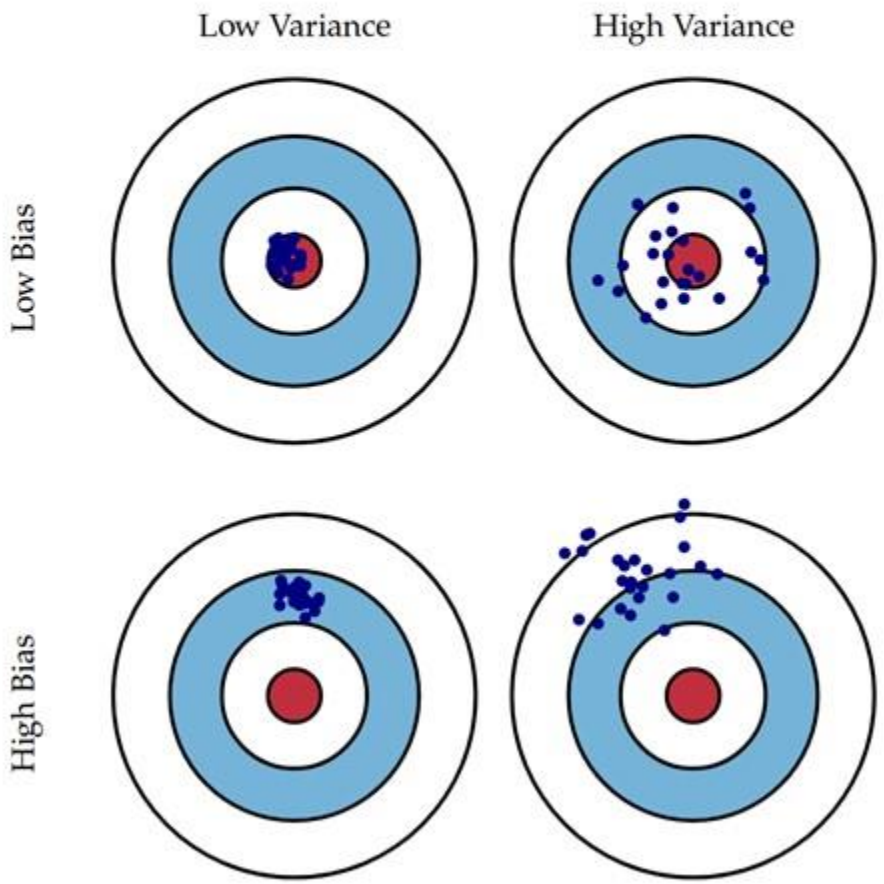
- A model with **high variance** learns too much from the training data
  - and overfits the data.

The best model sits somewhere in the middle of the two extremes.

Bias & Variance

- High Bias: Model is too simplistic and fails to capture patterns (underfitting).

- High Variance: Model is too sensitive to noise and training data (overfitting).



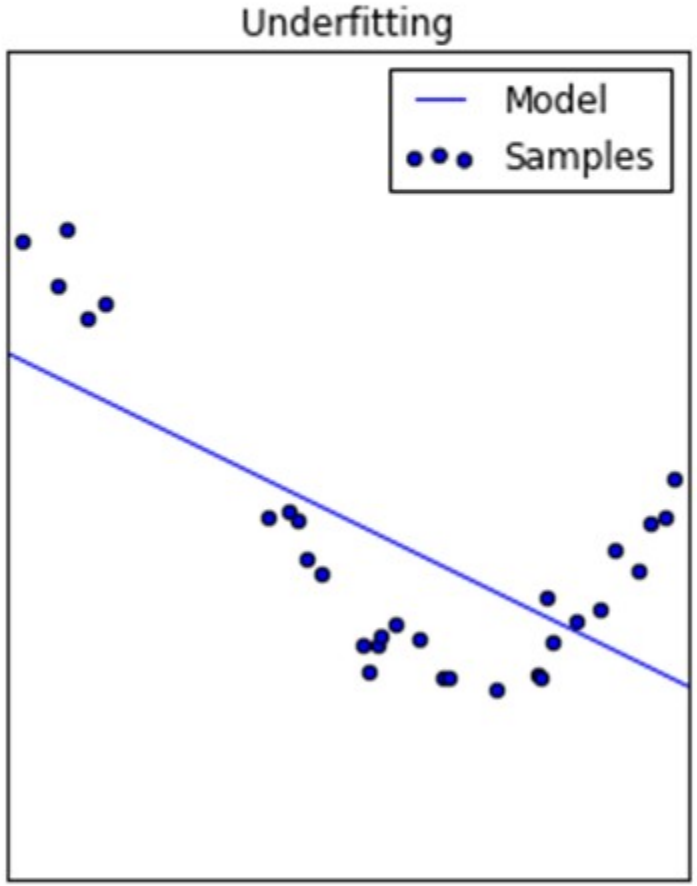
Bias-Variance Trade-off

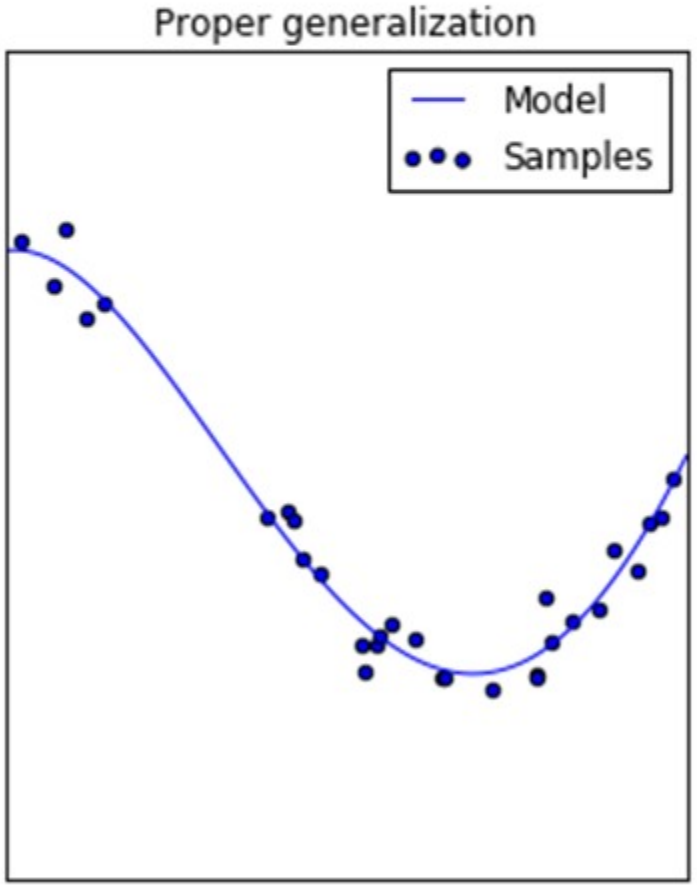
- There is a trade-off between bias and variance.
- As you decrease bias (e.g., by using more complex models), you tend to increase variance.
- Conversely, as you reduce variance (e.g., by using simpler models or regularization), you may increase bias.

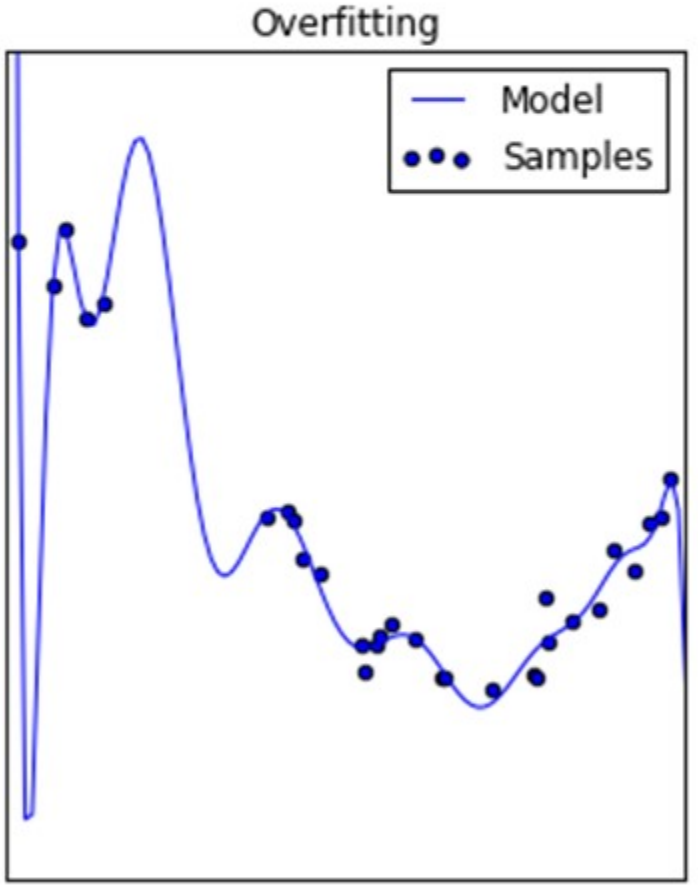
Bias-Variance Trade-off

- The goal is to find the right balance between bias and variance to achieve good generalization performance on unseen data.
- This is often referred to as the bias-variance trade-off.

Summary: Bias vs Variance







The train/test split

- Do not train your ML model on the entire dataset.
- A typical train/test split would be to use 70% of the data for training and 30% of the data for testing.
- It's important to use new data when testing our model
  - to prevent the likelihood of overfitting to the training set.

The train/test/validation split

- However, sometimes it's useful
  - to evaluate our model as we're building it
  - to find that best parameters of a model
- But, we can't use the test set for this evaluation
  - else we'll end up selecting the parameters that perform best on the test data
  - but maybe not the parameters that generalize best.

The train/test/validation split

- To evaluate the model while still building and tuning the model,
  - we create a third subset of the data known as the cross-validation set (development set or hold-out validation set).

- A typical train/test/validation split would be to use
  - 60% of the data for training,
  - 20% of the data for validation, and
  - 20% of the data for testing.

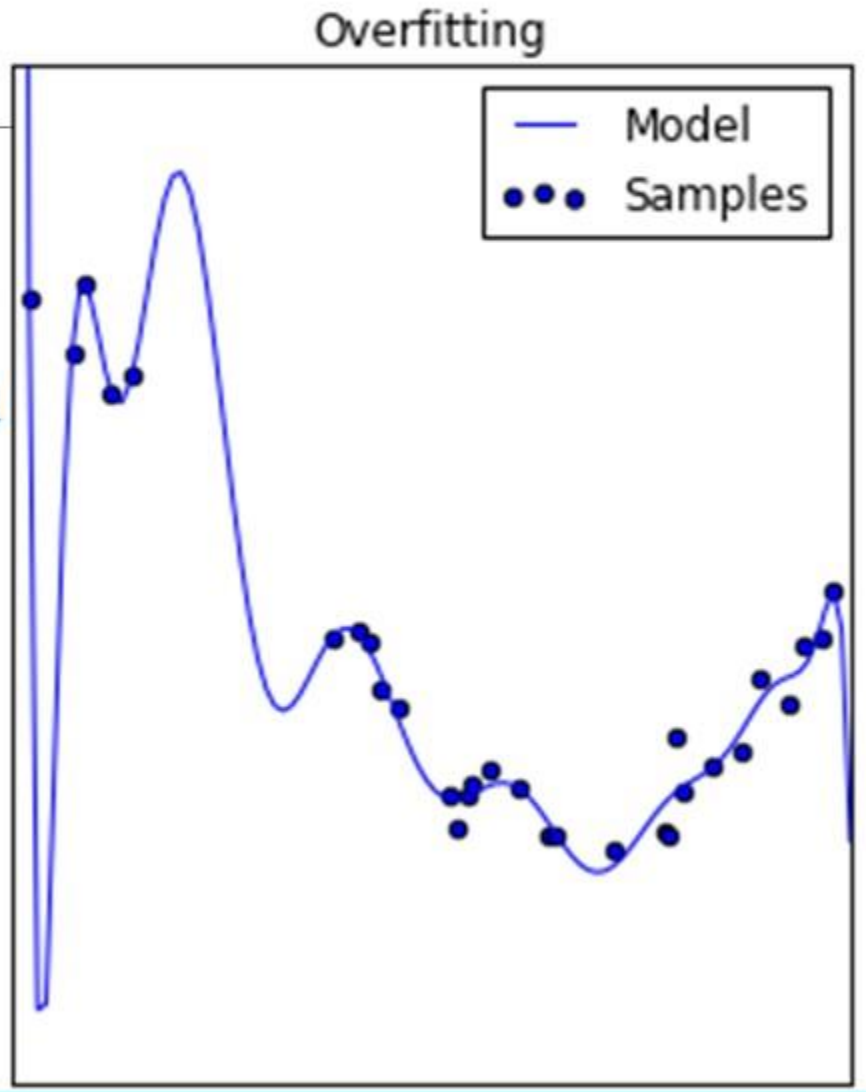
Model Selection

- Consider the example of polynomial fitting to a dataset (linear regression), then:
  - How to choose the degree of polynomial?
  - How to select value of λ (tuning parameter for regularization)?
  - ..
- These are called model selection problems.

Model Selection

- Fitting well of learning algorithm (model) to a dataset (training and testing) doesn't mean a hypothesis is good.

$$h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots$$

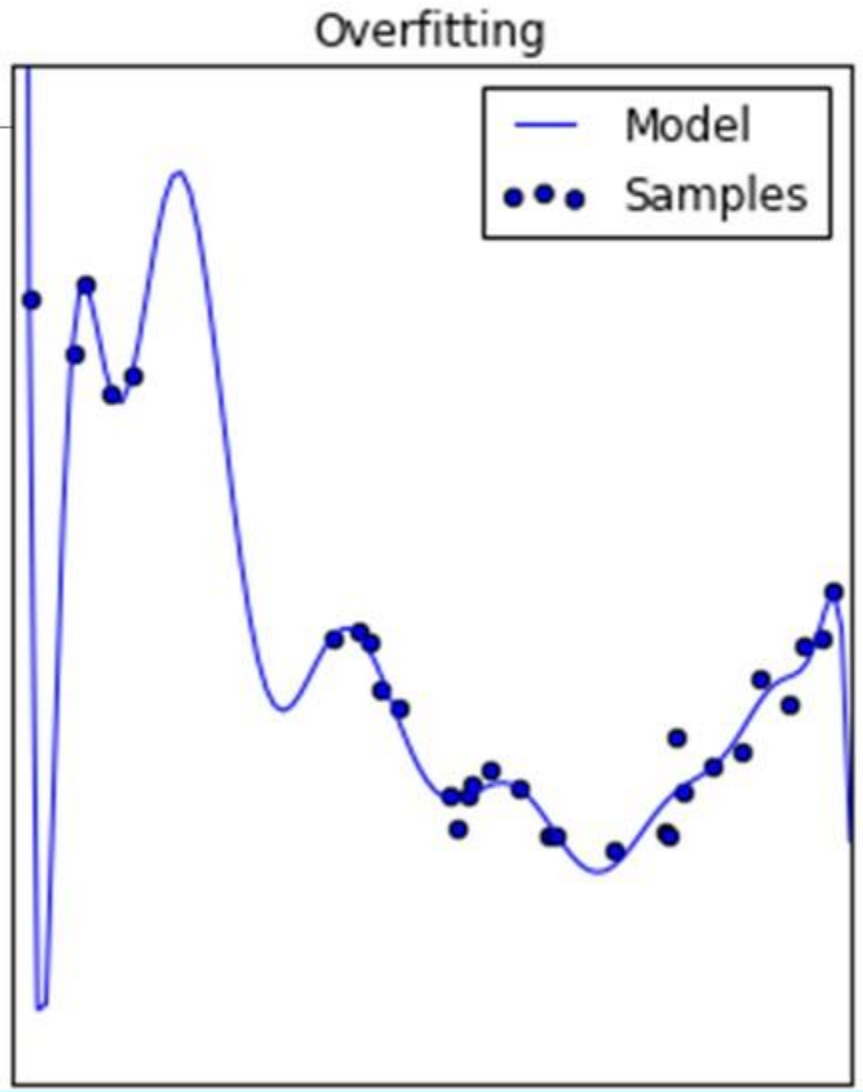


Model Selection

$$h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots$$

- i.e. low training/testing error* doesn't mean that model give good predictions for new examples (unseen).

(* if we use only train and test split)



Model Selection

- Let's consider the selection of degree of a polynomial.

1. $h_{\mathbf{w}}(x) = w_0 + w_1 x$
2. $h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2$
3. $h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$

:

10. $h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots + w_{10} x^{10}$

- Now, fit (train) each model (polynomial) on data set (training) by minimizing cost function $J(\boldsymbol{w})$ [may be using gradient descent]
- and get the optimal set of parameters $\mathbf{w}^{(i)}$ for each model ($i = 1, 2, \ldots, 10$).

Model Selection

- Compute testing error for each model $J_{test}(\mathbf{w}^{(i)})$ and
  - choose the model with *min* $J_{test}(\mathbf{w}^{(i)})$ for all i

- Let $J_{test}(\mathbf{w}^{(5)})$ is minimum among the all,
  - So the hypothesis $h_{\mathbf{w}}(x) = w_0 + w_1 x + \ldots + w_5 x^5$ is selected,

Model Selection

- However, test error $J_{test}(\mathbf{w}^{(5)})$ will not reflect:
  - how well does the model generalize.
- Since $J_{test}(\mathbf{w}^{(5)})$ is optimize over test set
  - It is unfair to evaluate (for generalization) hypothesis on test set again.
- Just like set parameters $\mathbf{w}$ obtained by minimizing $J(\mathbf{w})$ over training set does not give indicative of performance over new examples.

Model Selection

- In other words, we have fitted the parameters of chosen the degree of polynomial using test set,
  - it is likely to do better on test set but may not for unseen examples.
- Thus, this performance over test set is not an indicative of
  - how the hypothesis will generalize.

The train/test/validation split

- To address this problem in model selection setting,

  we generally **split** our data in three sets (instead of training/test):

  - *Train set:* To train the model and estimate the model parameters.
  - *Cross-validation set or development (dev) set or hold out validation set*: To evaluate the model while still building and tuning the model.
  - *Test Set*: To test the confidence in overall performance of your model.

Model Selection…

- Now, lets define performance measures (error) for all three sets:
- Training error: $J_{train}(\boldsymbol{w}) = \frac{1}{2m_{tr}} \sum_{i=1}^{m_{tr}} \left\{ h\left(x^{(i)}, \boldsymbol{w}\right) - y^{(i)} \right\}^2$ same as $J(\mathbf{w})$
- Cross Validation error: $J_{cv}(\boldsymbol{w}) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} \left\{ h\left(x_{cv}^{(i)}, \boldsymbol{w}\right) - y_{cv}^{(i)} \right\}^2$
- Test error: $J_{test}(\boldsymbol{w}) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left\{ h\left(x_{test}^{(i)}, \boldsymbol{w}\right) - y_{test}^{(i)} \right\}^2$

Model Selection…

- Fit (train) each model (polynomial) on data set (training) by minimizing cost function $J(\mathbf{w})$ or $J_{train}(\mathbf{w})$,
  - and get the optimal set of parameters $\mathbf{w}^{(i)}$ for each model.

- Now instead of testing these hypothesis on test set, we will test them on cross-validation set.

Model Selection…

- Thus, compute cross-validation error for each hypothesis $J_{cv}(\mathbf{w}^{(i)})$ using validation set, instead of test set
  - choose the model with *min* $J_{cv}(\mathbf{w}^{(i)})$ for all I

- Let $J_{cv}(\mathbf{w}^{(4)})$ is minimum among the all,
  - So the hypothesis $h_{\mathbf{w}}(x) = w_0 + w_1 x + \ldots + w_4 x^4$ is selected.

Model Selection…

- Now, estimate generalization error of final model

$$h_{\mathbf{w}}(x) = w_0 + w_1 x + \ldots + w_4 x^4$$

on test set i.e. $J_{test}(\mathbf{w}^{(4)})$

- Note that $h_{\mathbf{w}}(x) = w_0 + w_1 x + \ldots + w_4 x^4$ is not fitted on test data,
- Thus, performance over test set is indicative of hypothesis generalization.

The train/test/validation split

- Note that above division (percentage) is roughly useful for a data set of size upto10,000 instances (roughly).

- However, incase of Big-data (say 1,00,00,000 instances), specially in Deep Learning,

- generally, most of the data is used in training (98%), 1% dev, and 1% test.

The train/test/validation split

- Note that it's very important to shuffle the data before making these splits
  - so that each split has an accurate representation of the dataset.

- Note: the validation set and test set data must come from same distribution.
- In other words, choose a validation and test set
  - to reflect data you expect to get in future and consider important to do well.

Size of test set

- Set your test set to be big enough
  - to give high confidence in overall performance of your model.

- For some application you may not need a high confidence in overall model performance,
  - In that case having train and validation set is enough.

test and train only?

- Normally, when we call a train and test split only and iterate over test to get best model.
- Here actually, we refer validation/dev set as test set (which is actually dev set).
- However, it is always advisable to have separate test set.

Tools to diagnose: bias or variance

Validation curves

- Validation curves allow us to find
  - the a spot between underfitting and overfitting a model
  - to build a model that generalizes well.

- A typical validation curve is a plot of
  - the model's error as a function of some model hyper-parameter
    - which controls the model's tendency to overfit or underfit the data.

Validation curves

- The parameter you choose depends on the specific model you're evaluating.
- for example, you might choose to plot the degree of polynomial features for a linear regression model.
- Generally, the chosen parameter will have some degree of control over the model's complexity.

Validation curves

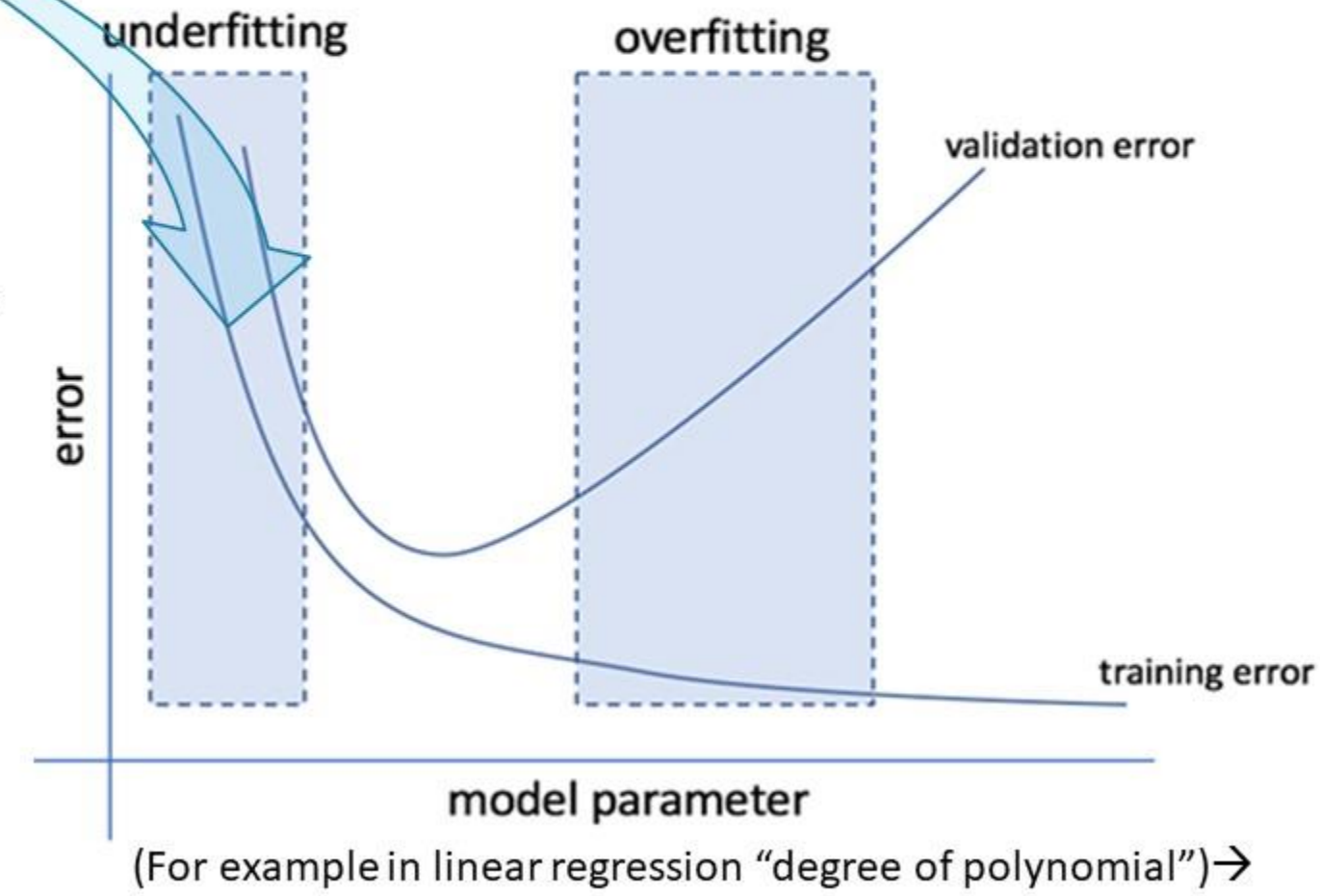
- On this curve, we plot both the training error and the validation error of the model.

- Using both of these errors combined,
  - we can diagnose whether a model is suffering from high bias or high variance

Validation curves

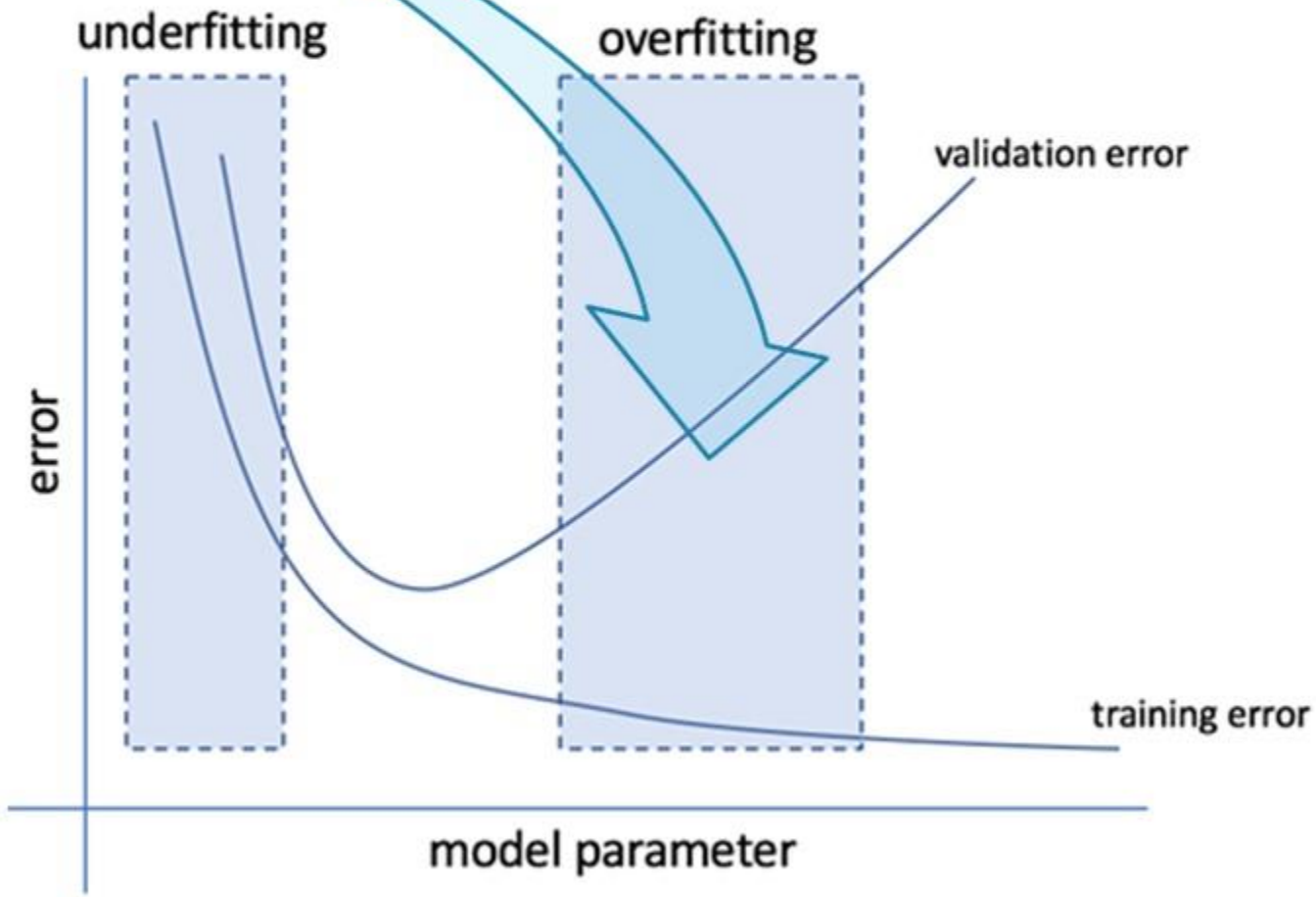
- Here both the training error and validation error are high,
  - the model is subject to high bias.

- Here, it was not able to learn from the data and it performing poorly.



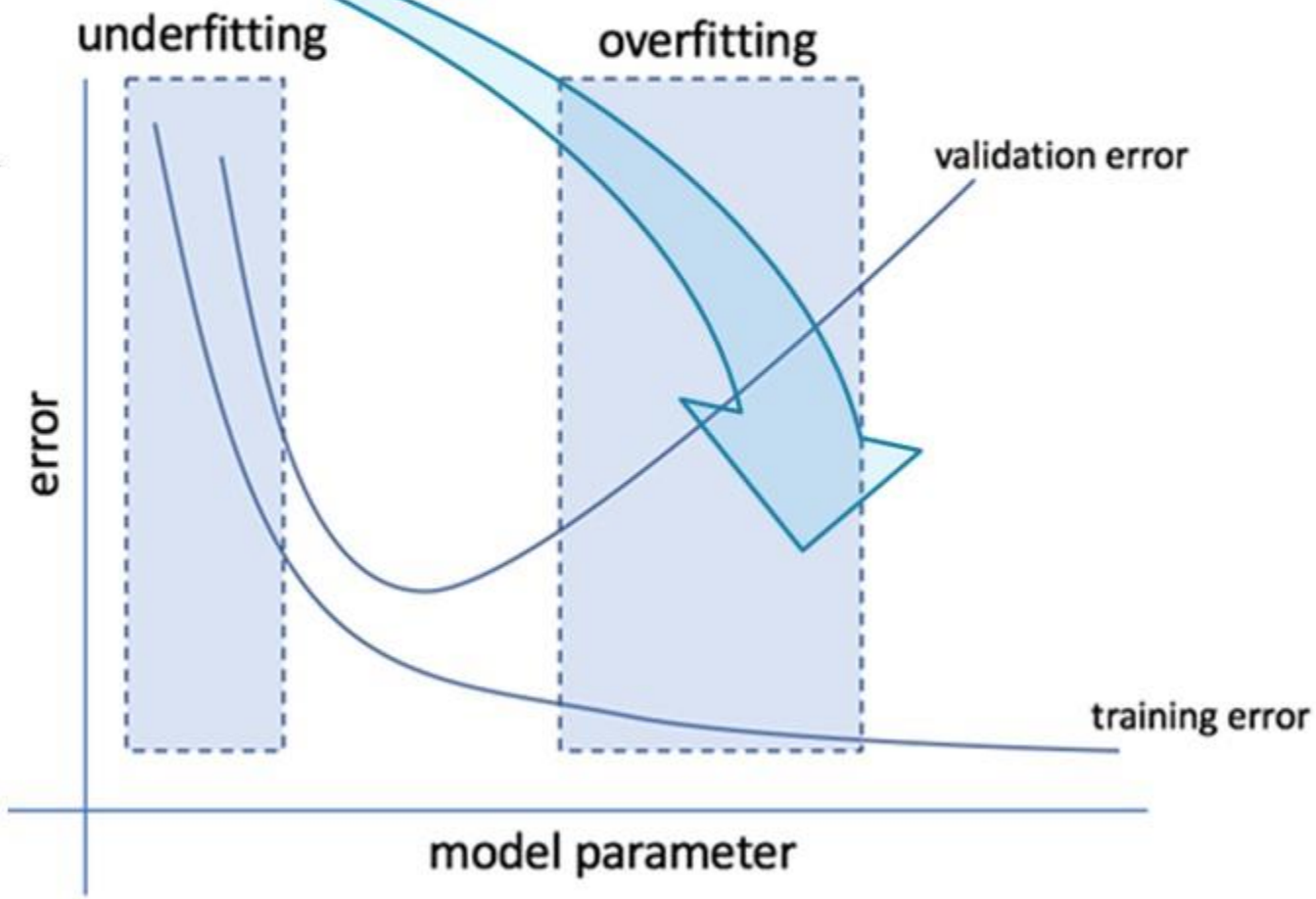
Validation curves

- Here the training error and validation error diverge,
- with the training error staying low and validation error increasing,
  - we're beginning to see the effects of high variance.



Validation curves

- The training error is low
  - because we're overfitting the data and learning too much from the training examples,
- while the validation error is high
  - because our model isn't able to generalize from the training data to new data.



Learning curves

- A learning curve is a plot of
  - model learning performance vs experience or time (number of training examples).

- During the training of a machine learning model,
  - the learning curves are used to decide if more data, more features will help in model performance improvement.

Learning curves

- Train Learning Curve: It can be evaluated on the training dataset
  - to give an idea of how well the model is "learning."

- Validation Learning Curve: It can also be evaluated on a hold-out validation dataset
  - to an idea of how well the model is "generalizing."

Learning curves

- It is common to create dual learning curves for a machine learning model
  - during training on both the training and validation datasets.

- These plots can give an insight about where the training is
  - suffering from high bias or high variance issues.

- Thus, learning curves are diagnostic tools for rectifying or improving performance of the ML models.

Learning curves

- Consider the case of linear regression: We are fitting a quadratic to a data (training set: m_{tr} samples; validation set: m_{cv} samples).

$$h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2$$

- Training error:

$$J_{train}(\mathbf{w}) = \frac{1}{2m_{tr}} \sum_{i=1}^{m_{tr}} \left\{ h\left(x^{(i)}, \mathbf{w}\right) - y^{(i)} \right\}^2$$

- Cross Validation error:

$$J_{cv}(\mathbf{w}) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} \left\{ h\left(x_{cv}^{(i)}, \mathbf{w}\right) - y_{cv}^{(i)} \right\}^2$$

Learning curves

- Horizontal axis: No. of training samples in the increasing order.
- Vertical axis:

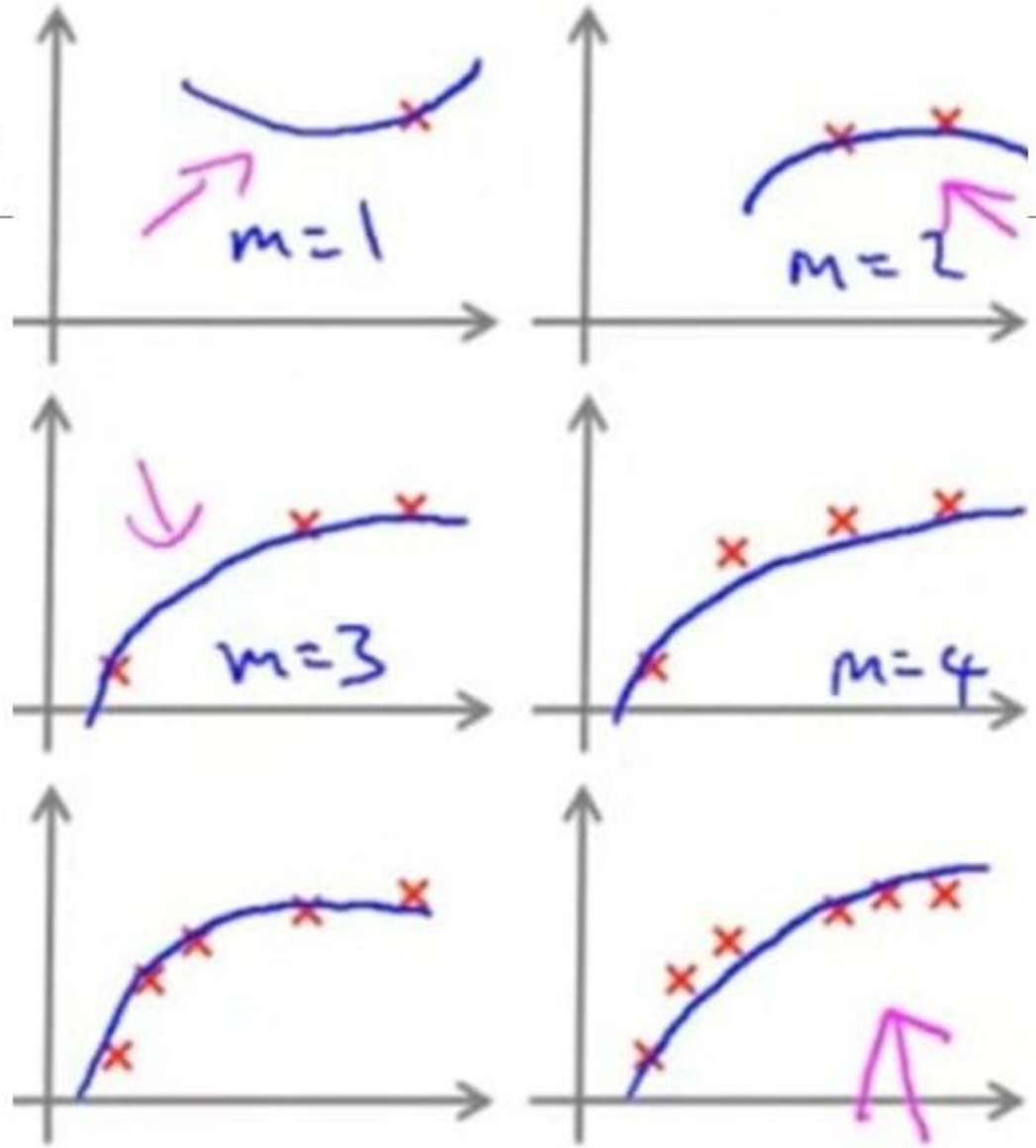
I. Training error $J_{train}(\mathbf{w})$ w.r.t number samples,

for example, we train model with 1 sample and compute $J_{train}(\mathbf{w})$, then train with 2 samples and compute $J_{train}(\mathbf{w})$, and so on..

II. Validation error $J_{cv}(\mathbf{w})$ w.r.t (model trained with) number samples

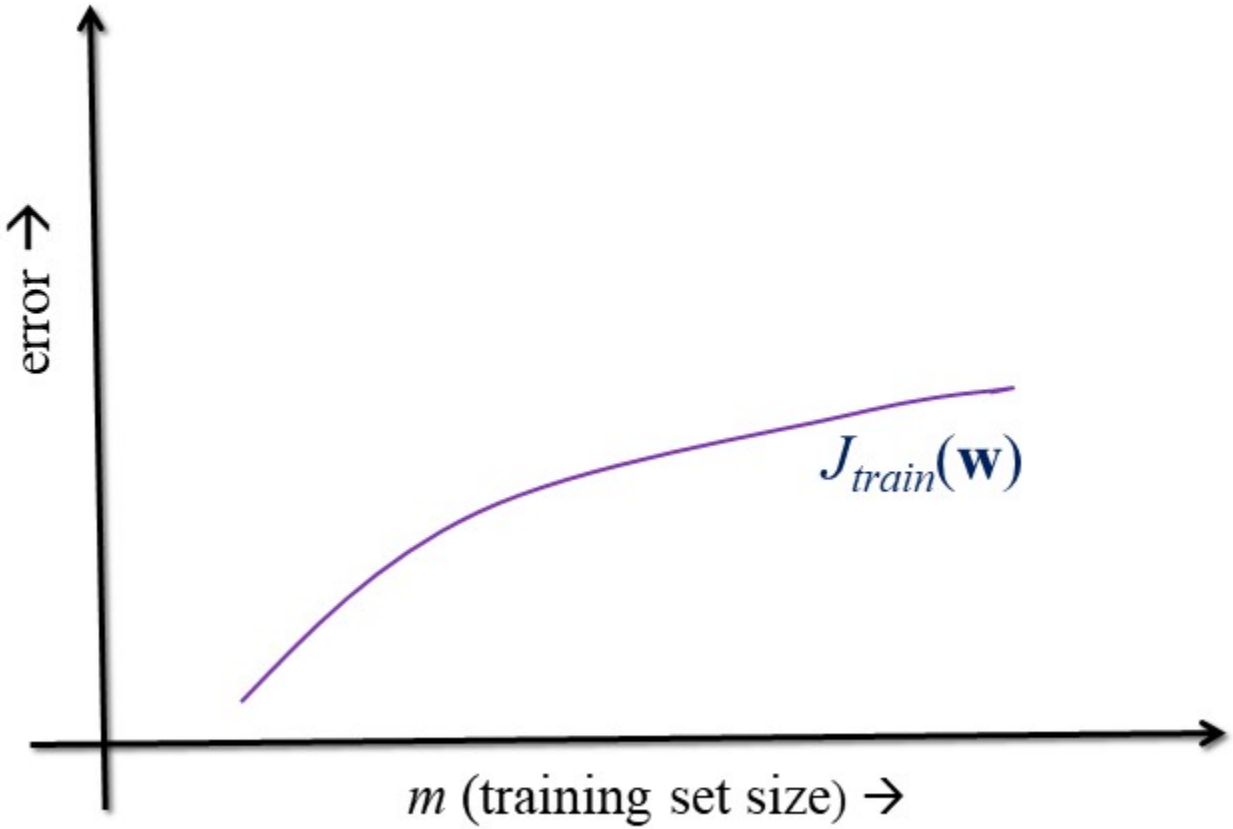
for example, we compute $J_{cv}(\mathbf{w})$ for model trained with 1 sample, then for model trained with 2 samples, and so on..

$$h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2$$



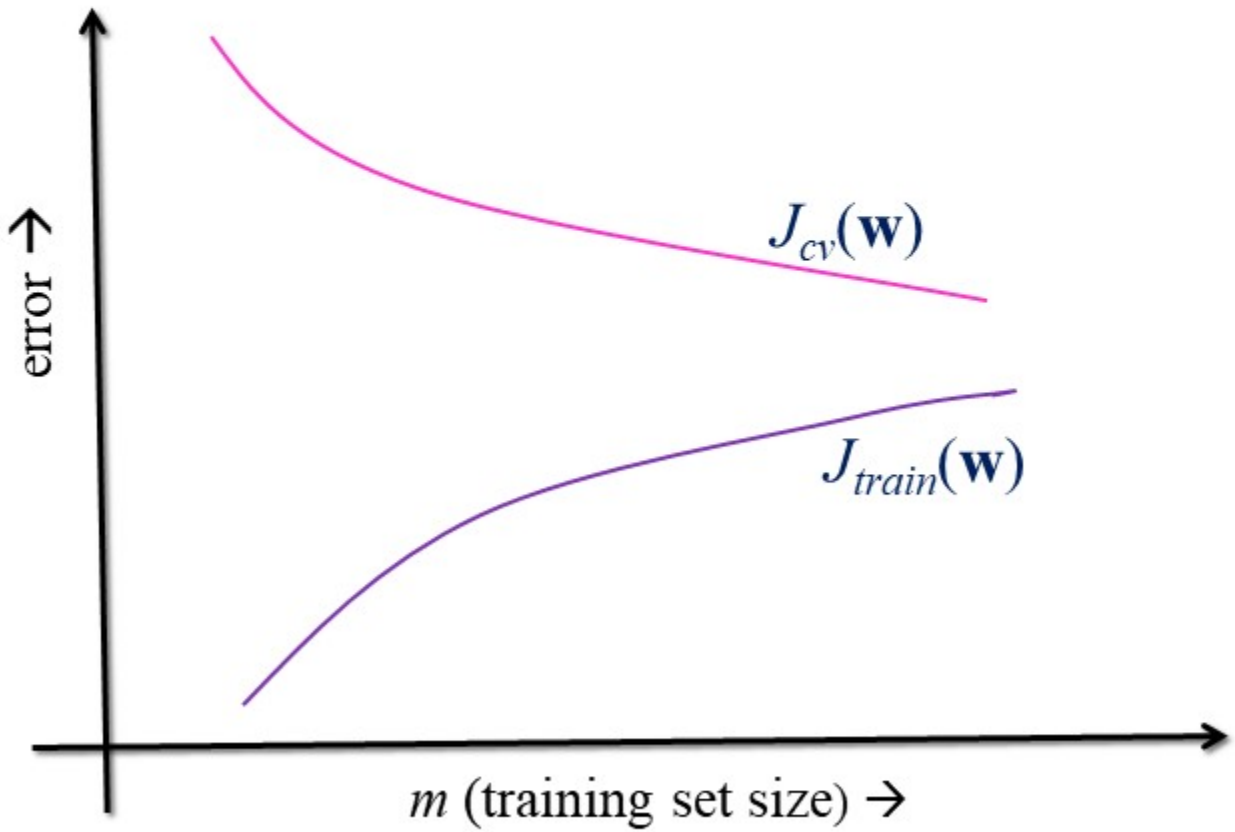
Learning curves

- Initially, for small training set size (no. of samples), quadratic curve will fit easily,
- Thus, very low training error $J_{train}(\mathbf{w})$, nearly zero.
- As the training set size ↑, it becomes more and more difficult to fit quartic,
- Thus, $J_{train}(\mathbf{w})$ ↑ with increase in training set size.



Learning curves

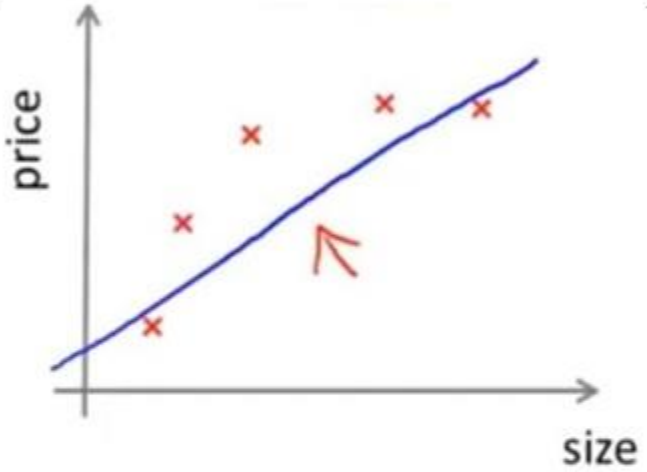
- Initially, for small training set size (no. of samples), model is not learnt significantly,
- Thus, very high validation error $J_{cv}(\mathbf{w})$.
- As the training set size ↑, model achieves better learning,
- Thus, $J_{cv}(\mathbf{w})$ ↓ with increase in training set size.



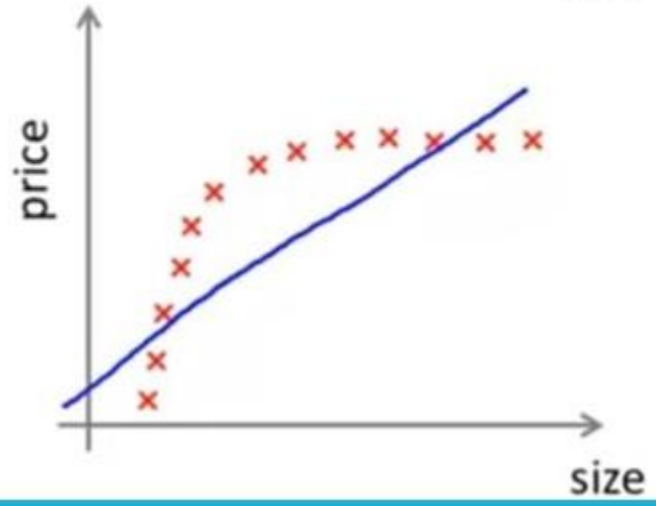
Learning curves: High Bias

- Suppose we try to a line to the given data

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

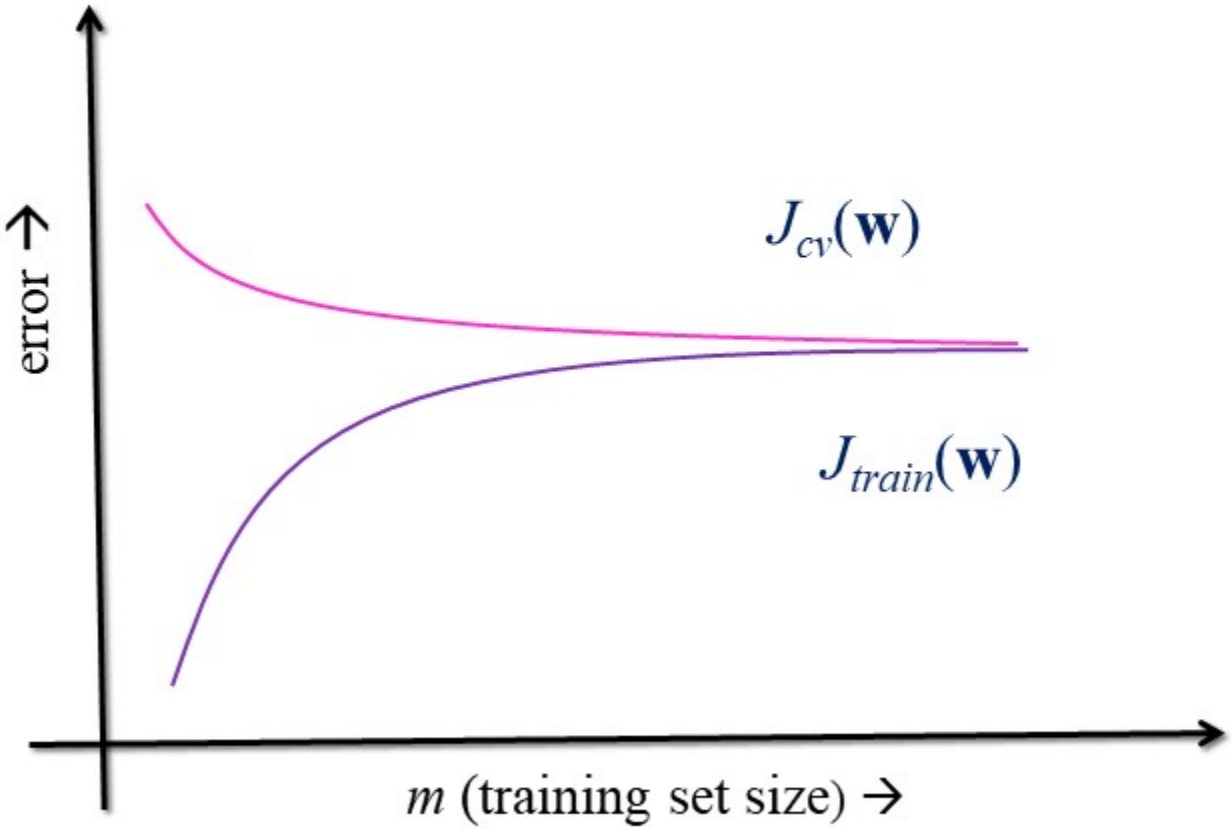


- After certain number, increasing the no. sample will not change the line much.



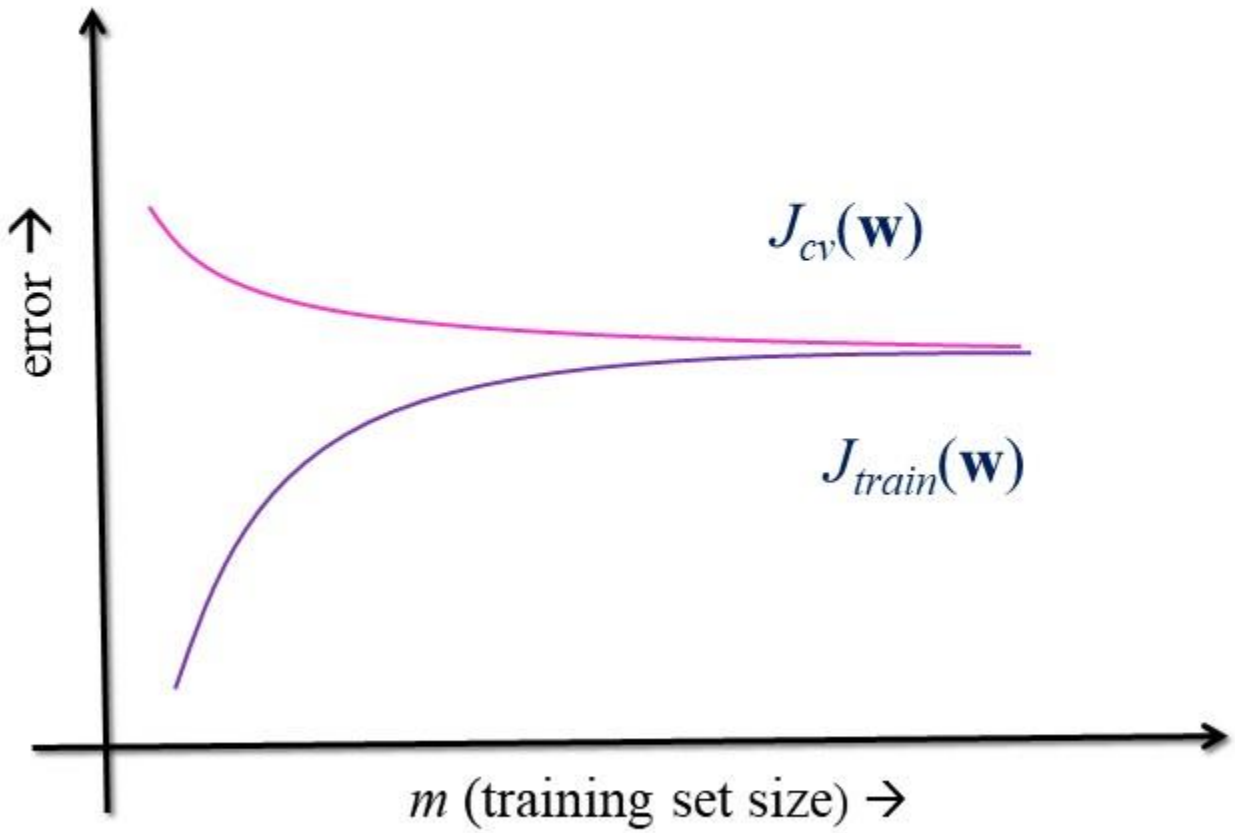
Learning curves: High Bias

- we'll observe fairly quick convergence to
- a high error for the validation and training datasets.
- If a model is suffering from high bias,
- Getting more training data will not (by itself) help.



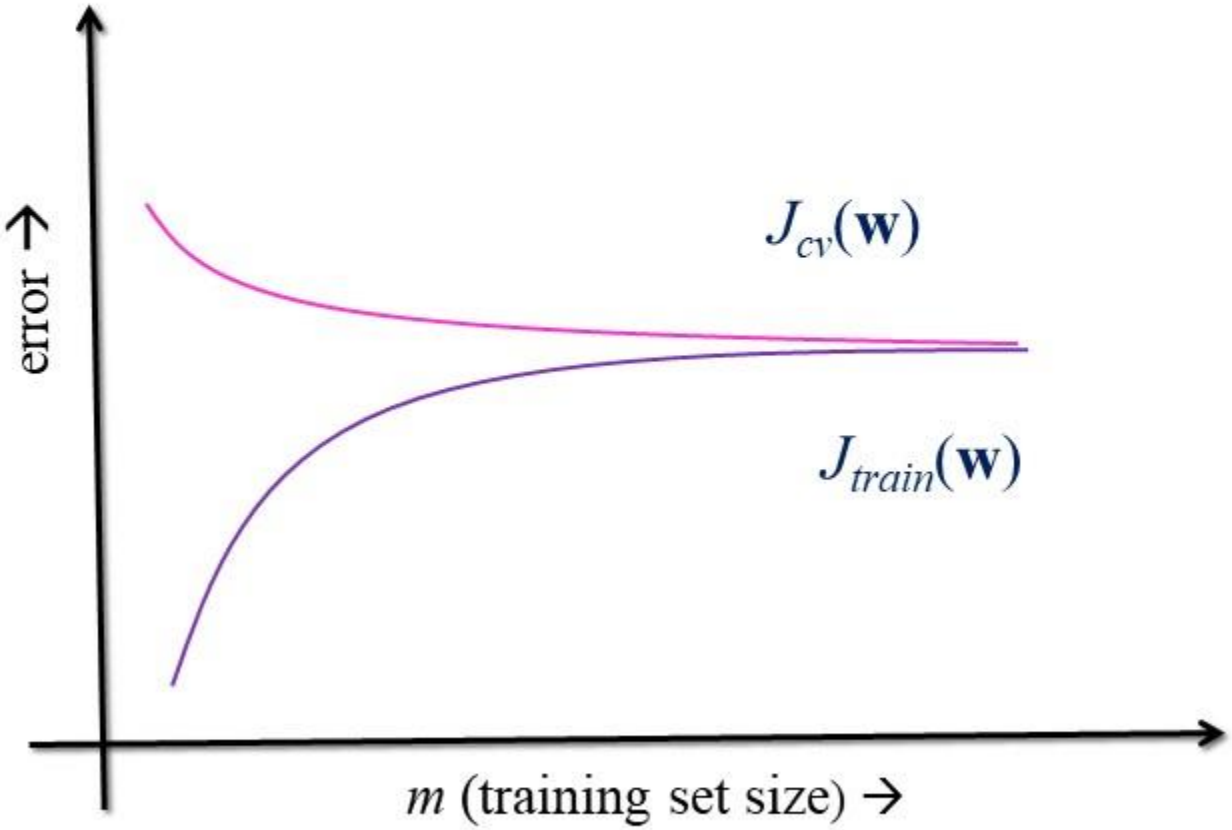
Learning curves: High Bias

- In learning curve, after certain size of training set,
- Both training error, and validation error becomes almost same.



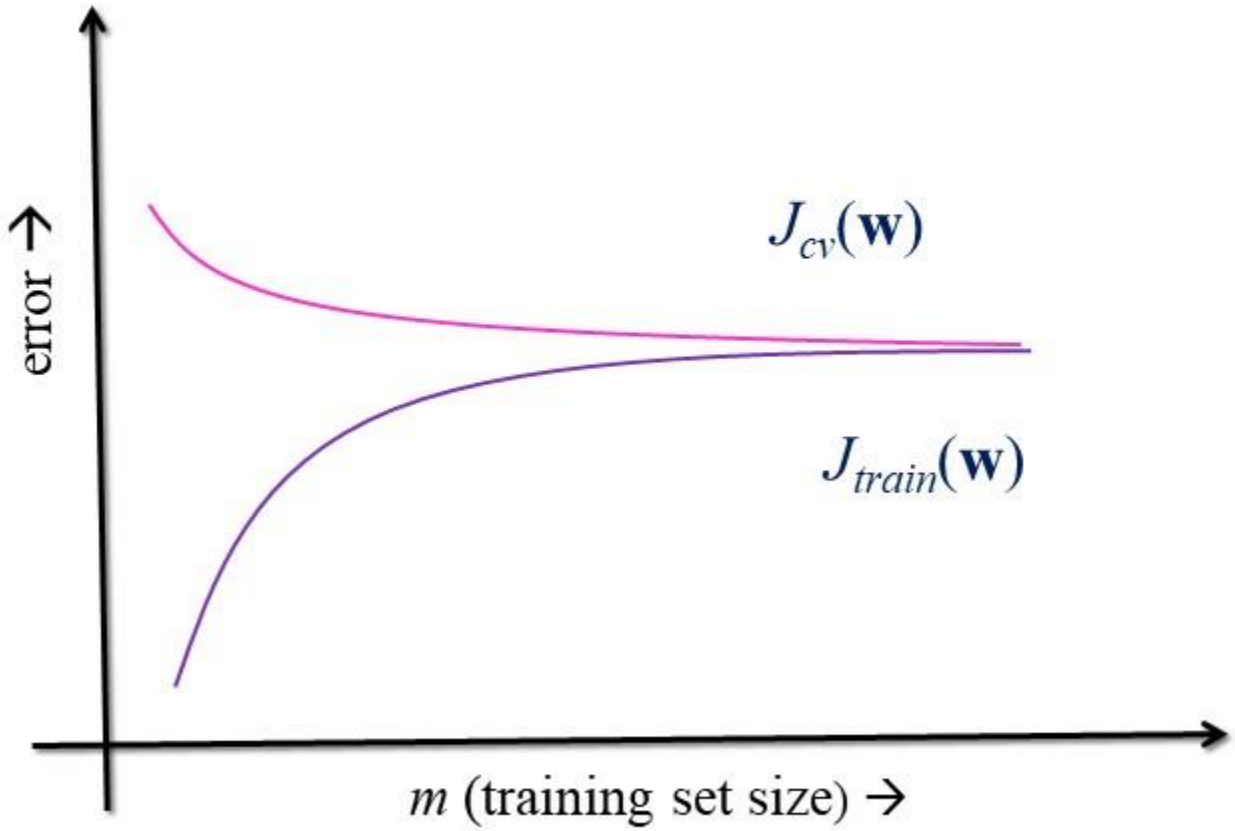
Learning curves: High Bias

- This is because, increasing training size not improving learning (incapability of model), and
- Training error behaves like validation error



Learning curves: High Bias

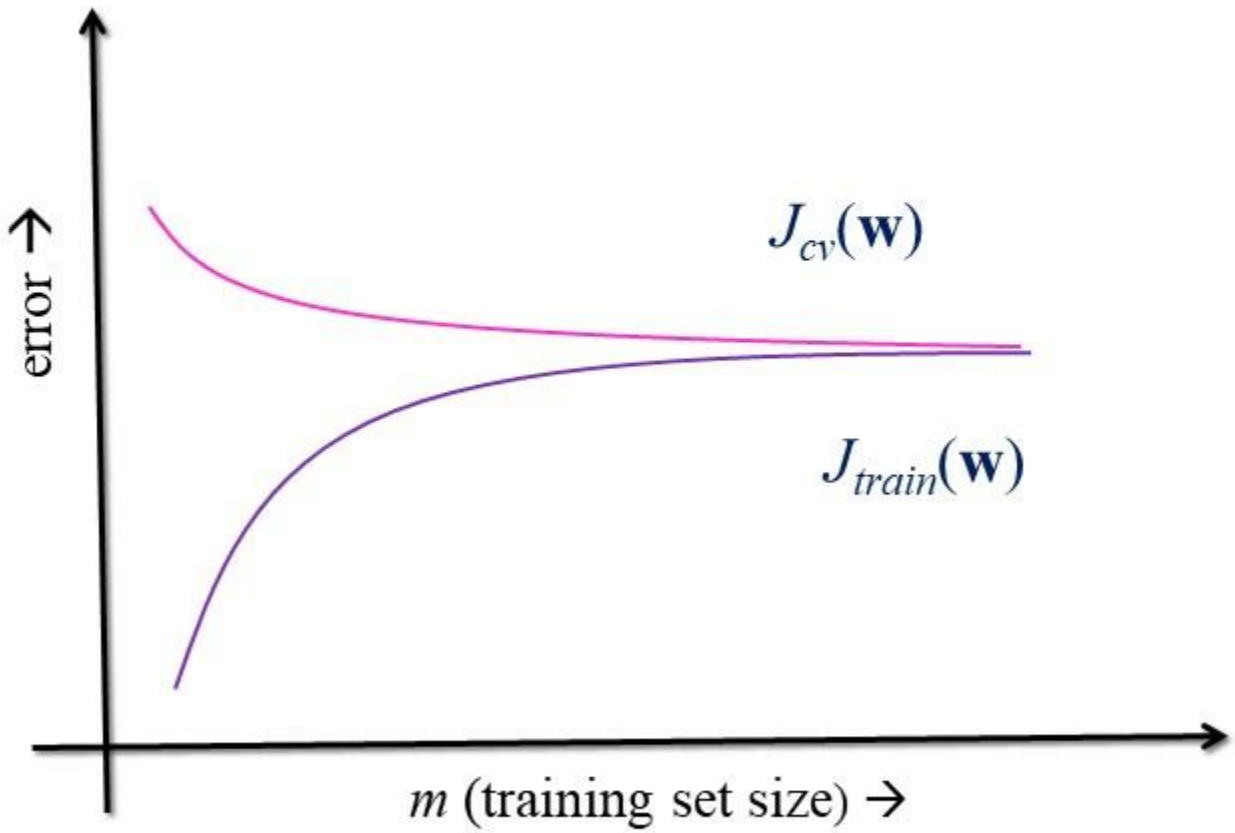
- Models which underfit the data pay little attention to the data,
- So feeding in more data will be useless.



Learning curves: High Bias

- A better approach to improving models (with high bias) is to consider adding additional features to the dataset

- So that the model can be more equipped to learn the proper relationships.



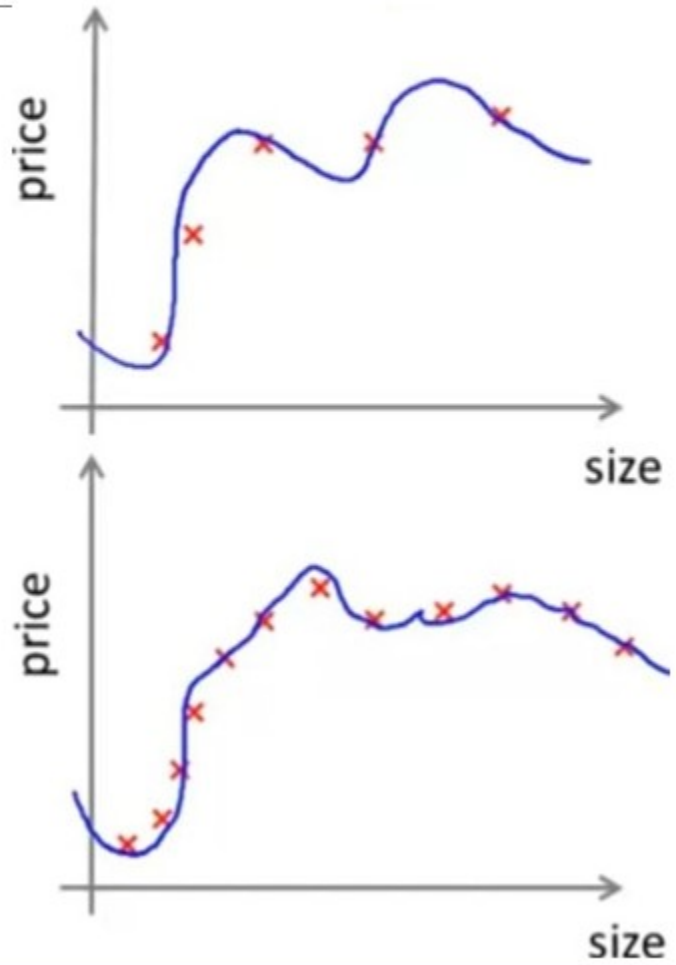
Learning curves: High Variance

- Consider the fitting the following hypothesis on a dataset

$$h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_{100} x^{100}$$

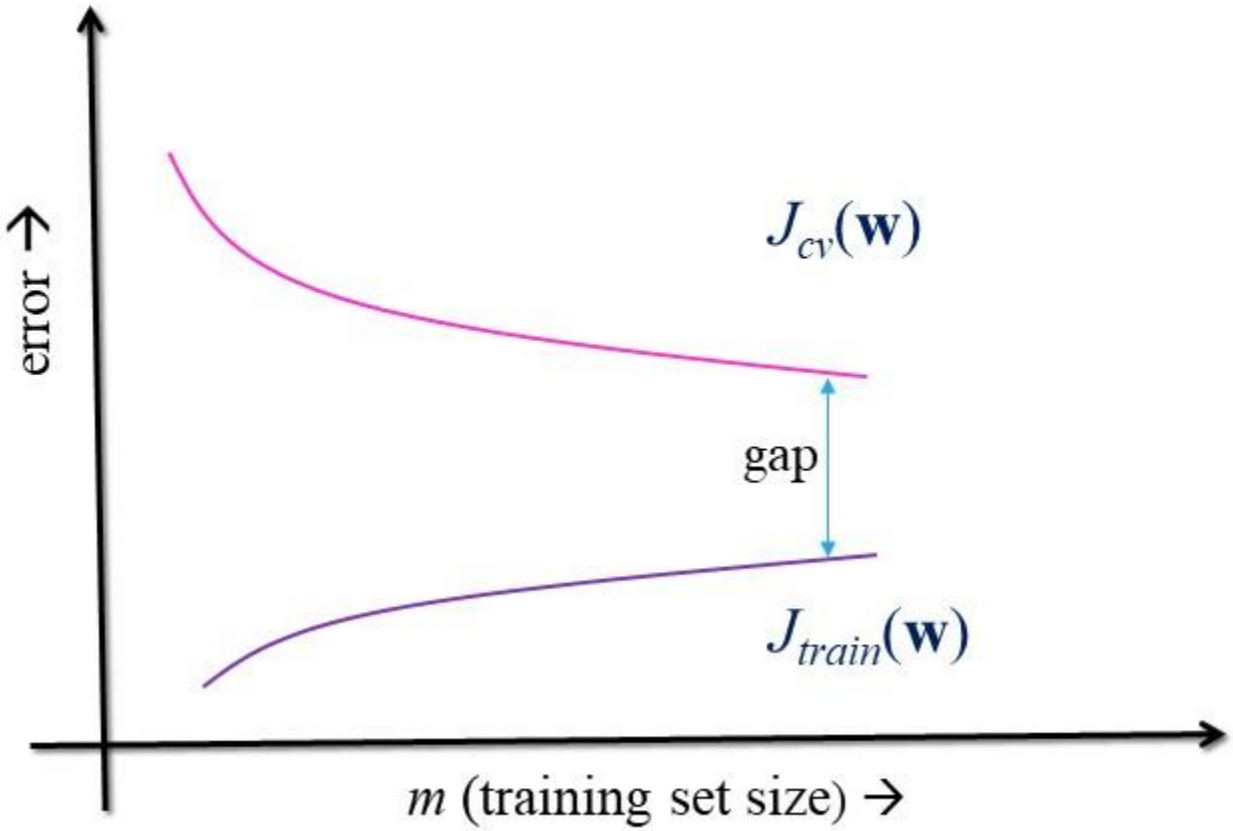
and small λ

- Training error is low, incase of models with high variance due to over fitting.
- However, validation error remains high due to poor generalization.



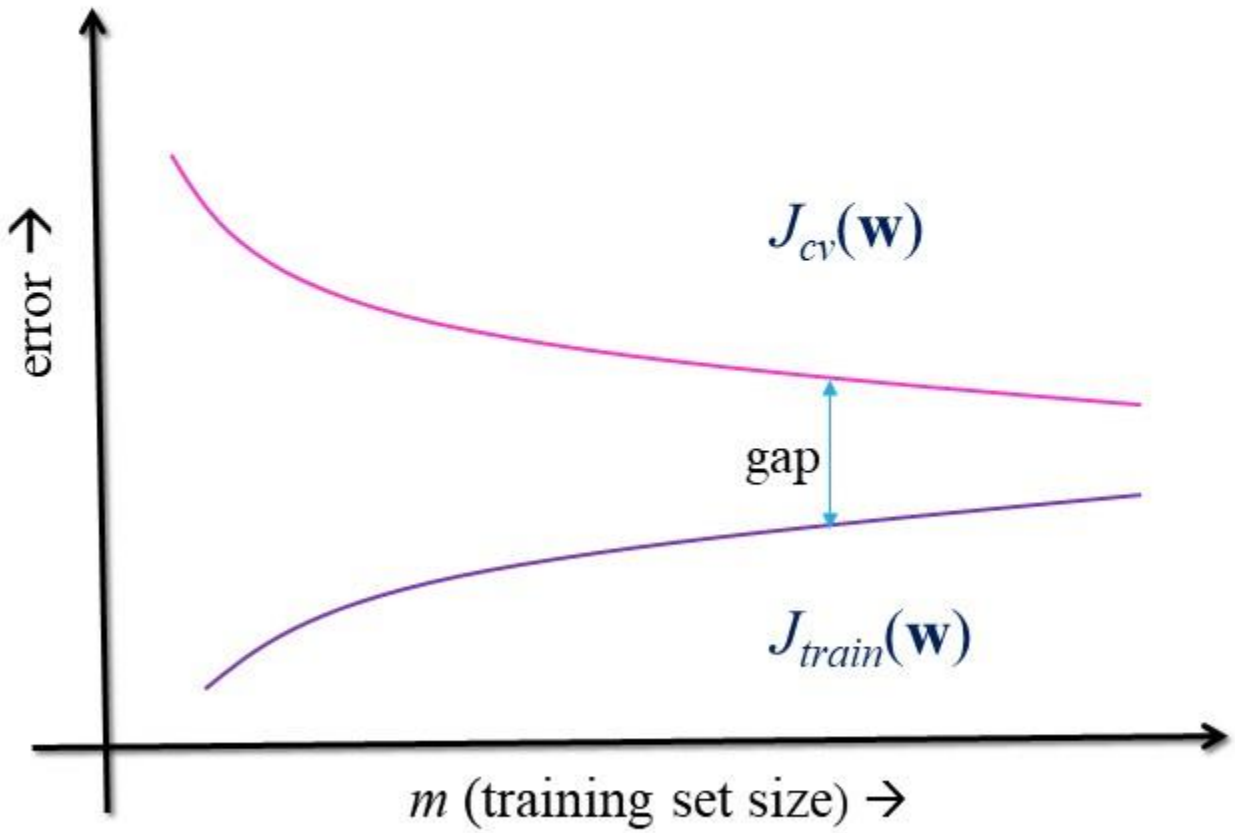
Learning curves: High Variance

- If our model has high variance,
- we'll see a gap between the training and validation error.
- It is due low training error (overfitting) and high cross-validation error (poor generalization).



Learning curves: High Variance

- If our model has high variance,
- Getting more training data is likely to improve the model performance.



Metrics

(To evaluate ML models)

Classification metrics

- When performing classification predictions, there's four types of outcomes that could occur.
  - True positives (TP)
  - True negatives (TN)
  - False positives (FP)
  - False negatives (FN)

Classification metrics

- **True positives:**
  - you predict an observation belongs to a class

    and

  - it actually does belong to that class.
- **True negatives**
  - you predict an observation does not belong to a class

    and

  - it actually does not belong to that class.

Classification metrics

- **False positives:**
  - you predict an observation belongs to a class

    and

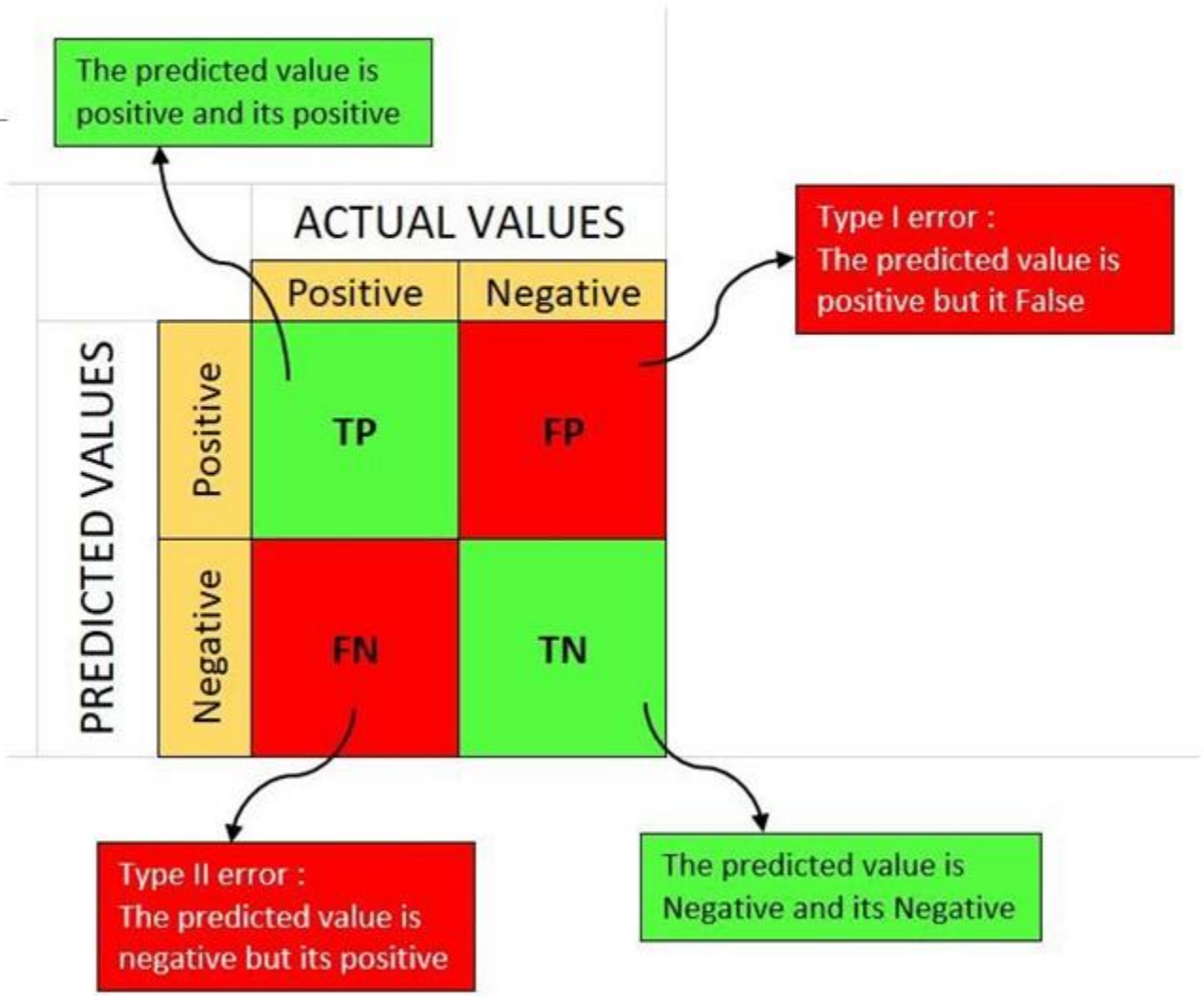
  - it actually does not belong to that class.
- **False negatives**
  - you predict an observation does not belong to a class

    and

  - it actually does belong to that class.

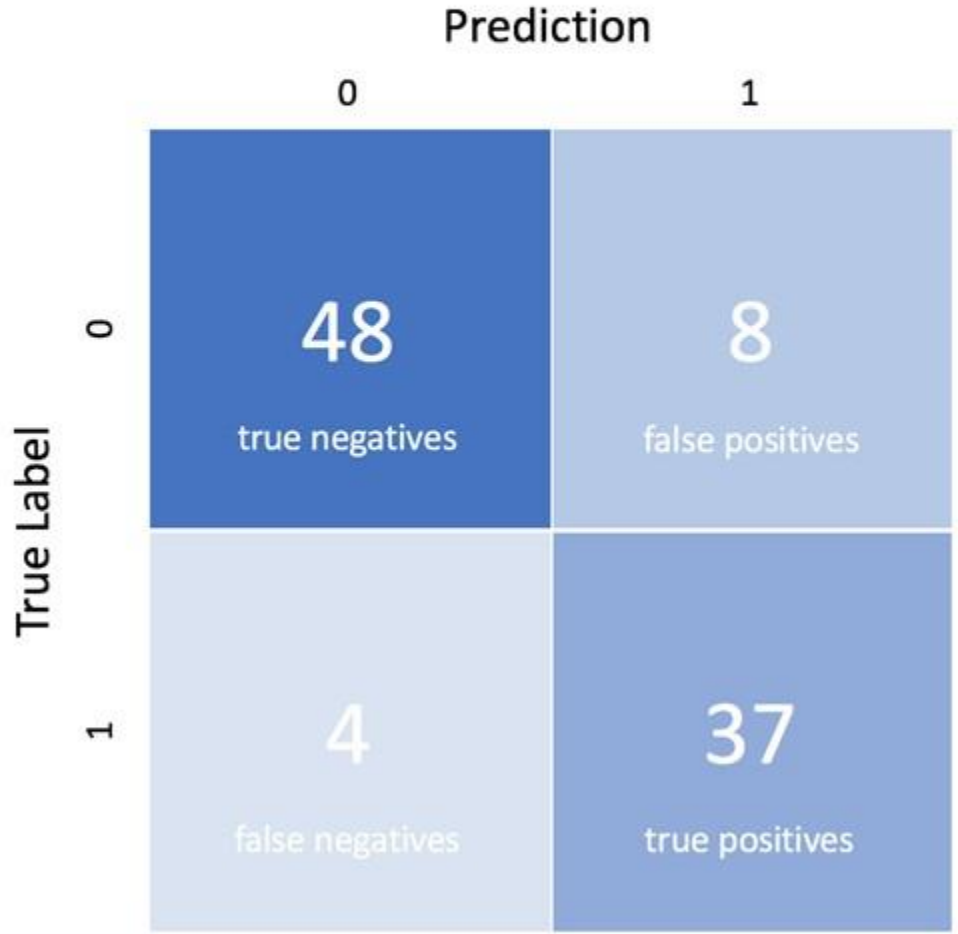
Confusion matrix

- These four outcomes are often plotted on a **confusion matrix**.



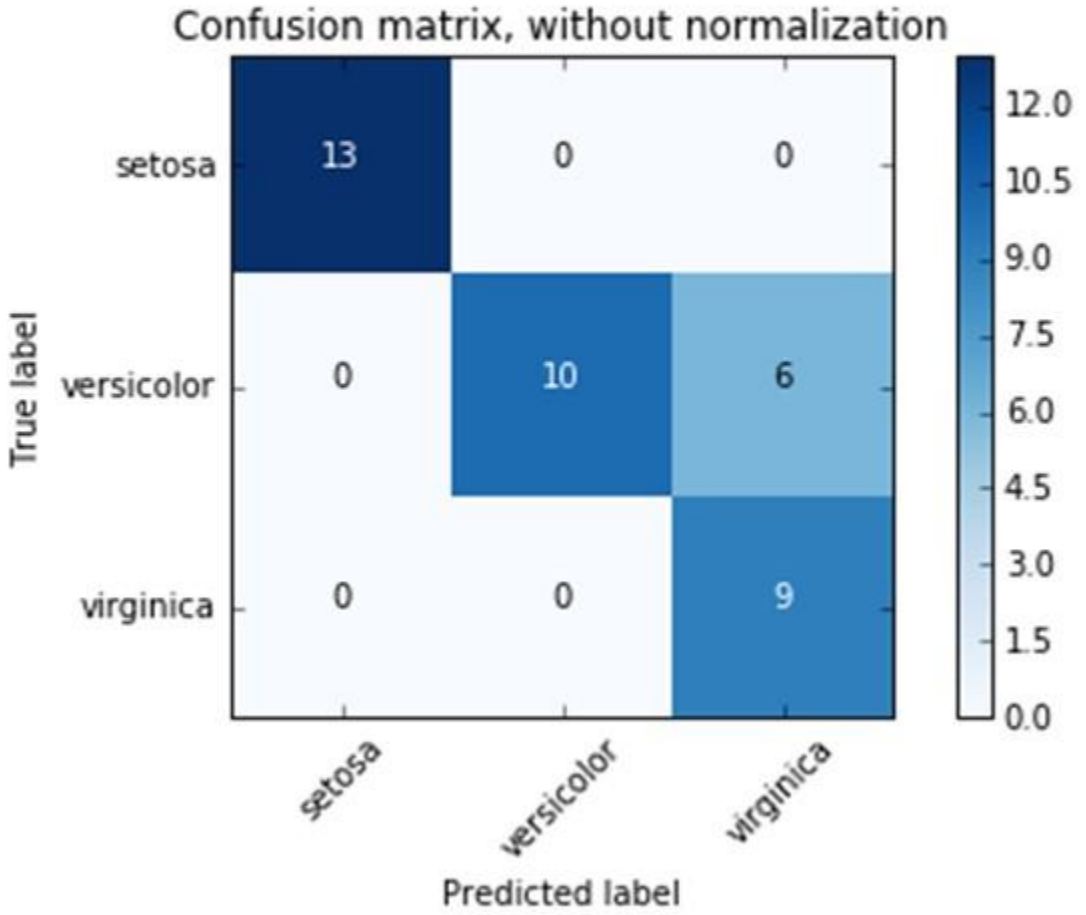
Confusion matrix: Binary classification

- This is a confusion matrix of a binary classification.
- You can generate this matrix after making predictions on your test data and
- then identifying each prediction as one of the four possible outcomes described above.



Confusion matrix: Multi-class classification

- You can also extend this confusion matrix to plot multi-class classification predictions.
- The following is an example confusion matrix
- for classifying observations from the Iris flower dataset.



Classification metrics

- The three main metrics used
  - to evaluate a classification model are:
    - Accuracy,
    - Precision, and
    - Recall.

Accuracy

Accuracy is defined as:

- the percentage of correct predictions for the test data.

It can be calculated easily by: dividing the number of correct predictions by the number of total predictions.

$$\text{accuracy} = \frac{\#\text{ correct predictions}}{\#\text{ all predictions}}$$

$$= \frac{\text{true positives} + \text{true negatives}}{\#\text{ all predictions}}$$

Accuracy

- Accuracy gives misleading results (fails) in case of skewed class distributions.
  - i.e. when (in binary classification) most of samples belongs to one class and other class has very few cases (rare class).

- For example, consider developing a classifier for a disease, where only a small percentage of the population (let's say 1%) has this disease.

Accuracy

- One can design a classifier that always predicts that the person does not have the disease (non-learning algorithm).

- In such case, we would have built a model which is 99% accuracy, but 0% useful.

Accuracy

- Precision and recall are useful (gives better measure) in such cases where classes aren't evenly distributed.

- Note: Rare class (class which has very few samples) is consider as relevant items or positives (or denoted as class 1) by convention.

Precision

Precision is defined as

- The fraction of relevant examples (true positives) among all of the examples, which were predicted to belong in a certain class (rare class).

$$\text{precision} = \frac{\#\text{ true positives}}{\#\text{ predicted positives}}$$

$$= \frac{\#\text{ true positives}}{\#\text{ true positives} + \#\text{ false positives}}$$

Recall

Recall (or True positive rate or sensitivity) is defined as

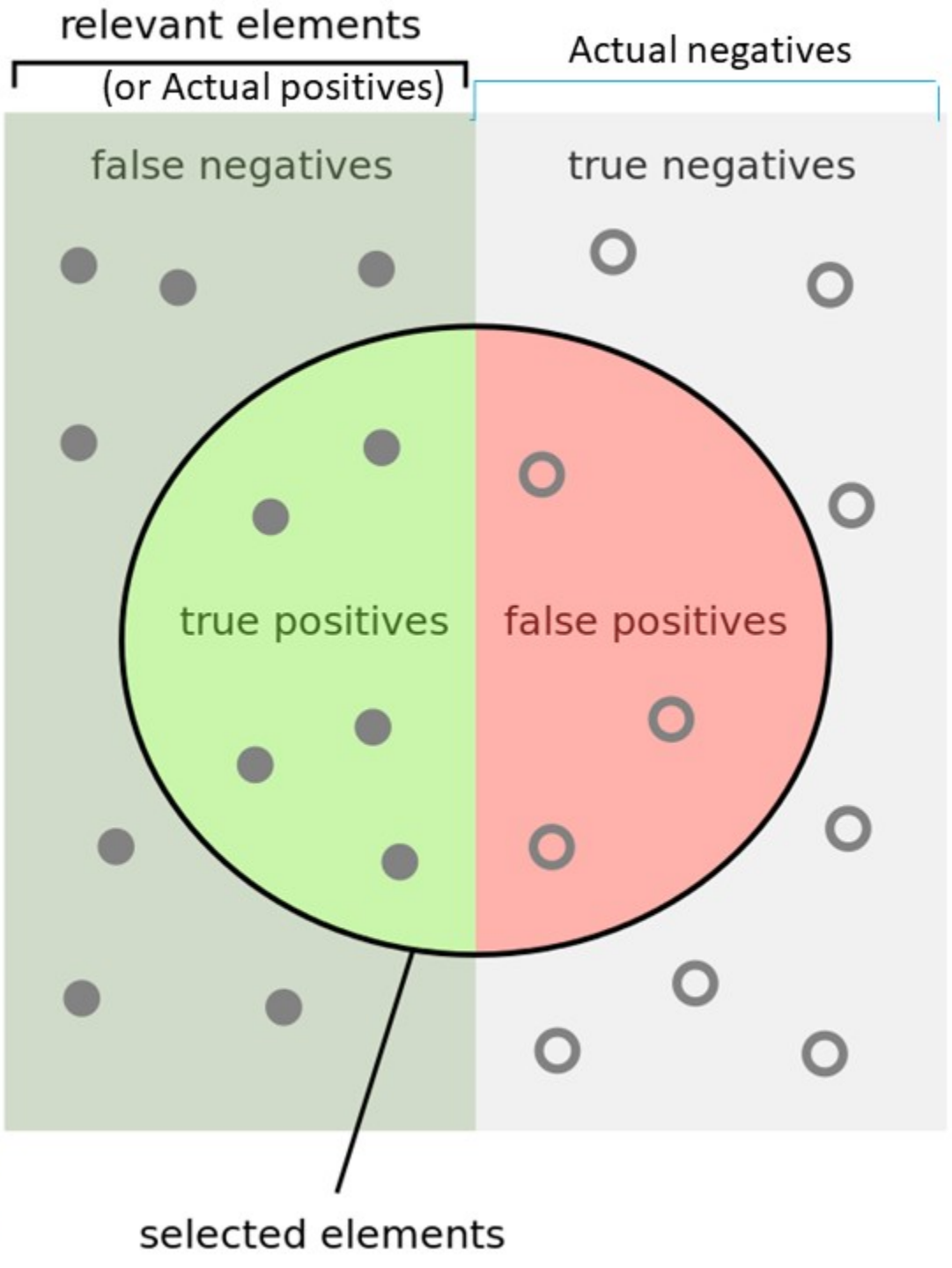
- The fraction of examples which were predicted to belong to a class (rare class) with respect to all of the examples that truly belong in the class (rare class).

$$\text{recall} = \frac{\#\ \text{true positives}}{\#\ \text{actual positives}}$$

$$= \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

precision vs recall

Visualizing the difference between precision and recall.



precision vs recall

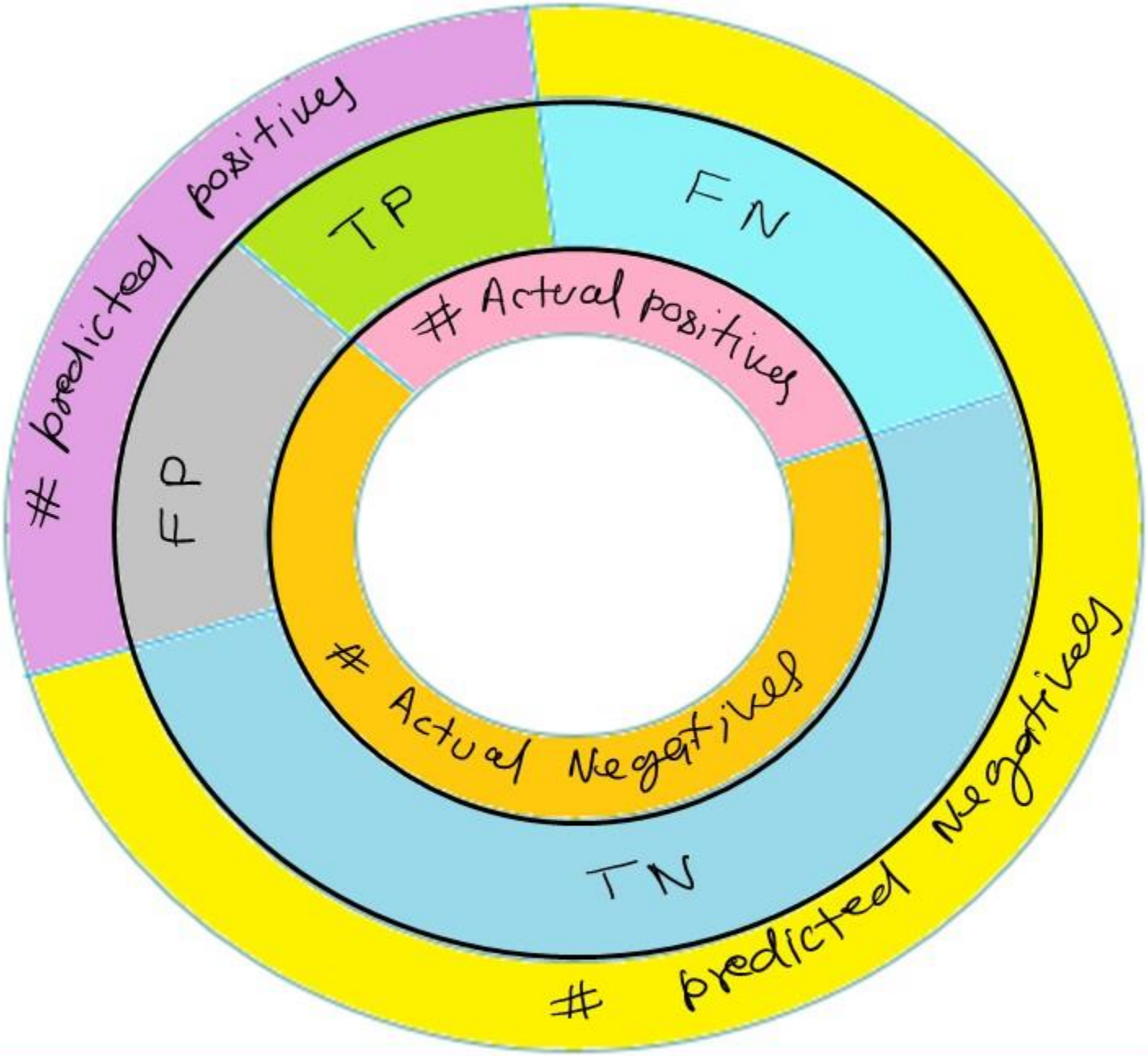
Visualizing the difference between precision and recall.

How many selected items are relevant?

Precision = 

How many relevant items are selected?

Recall = 


predicted positives
TP
FN
Actual positives
FP
Actual Negatives
TN
predicted Negatives

Accuracy Vs Recall

- Coming back to the classifier (non-learning) for skewed class example, where only 1% population has disease (rare class).

- If we measured the recall of this useless predictor,

$$\text{recall} = \frac{\#\text{ true positives}}{\#\text{ actual positives}}$$

- It will give zero value (as it always predict majority class, no prediction for rare class).

Accuracy Vs Recall

- It would indicate that there was something wrong with our model.
- In this example, recall ensures that
  - We're not overlooking the people who have the disease (rare class).
- Where in case of accuracy, rare class was ignored to give 99% accuracy.

Accuracy Vs precision Vs Recall

- Consider the case of a disease G, where 1% of the total population (say 10,000) are having this disease G. Thus,

  - Actual positives = 100 (actually suffering from G i.e. rare class)
  - Actual negatives = 9,900, i.e. (10,000 - 100)

Accuracy Vs precision Vs Recall

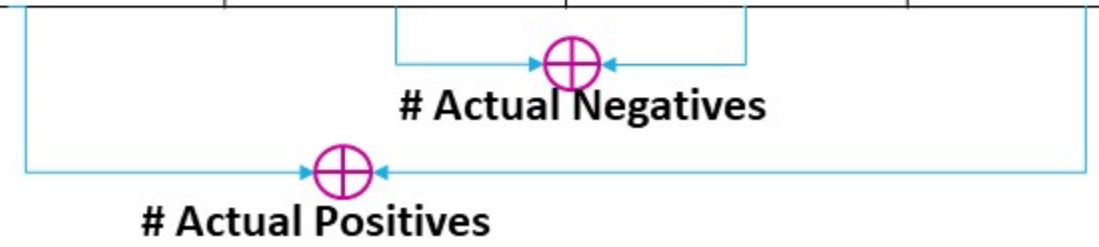
- Let there be four classifiers:
  - Classifier NL-I: Non-learning, which always predict majority class
  - Classifier NL-II: Non-learning, which always predict minority (rare)class
  - Classifier A: predicts 50 samples as positives out of which 2 are positive by mistake
  - Classifier B: predicts 178 samples as positives out of which 80 are positive by mistake

Accuracy Vs Precision Vs Recall

- For these three classifiers:

Actual Positives = 100, Actual Negatives = 9,900

Classifier	# Predicted Positives		# Predicted Negatives	
	TP	FP	TN	FN
NL-I	0	0	9,900	100
NL-II	100	9,900	0	0
A	48	2	9,898	52
B	98	80	9,820	2



Accuracy Vs Precision Vs Recall

- For these three classifiers:

Classifier	Accuracy (%)	Precision	Recall
NL-I	99	-	0
NL-II	1	0.01	1
A	99.46	0.96	0.48
B	99.10	0.55	0.98

Accuracy Vs Precision Vs Recall

- Conclusions from above example:
  - Accuracy is almost same for three (NL-I, A, B) classifiers and fails to evaluate the models properly.
  - Classifier NL-I has undefined precision (0/0), and zero value of recall.
  - This is because it is ignoring all positive cases, thus have TP = 0.

Accuracy Vs Precision Vs Recall

- Conclusions from above example:

  - Classifier NL-II has very low precision *i.e.* 0, but very high recall *i.e.* **1**.
  - This is because it is ignoring all negative cases, thus have FN =0.

  - In case of such classifiers (NL-II), recall may be misleading.

Accuracy Vs Precision Vs Recall

- Classifier A has high value of precision (0.96), and low value of recall (0.48).
- Although classifier A is able to recall only 48% of positives cases,
- but once it classifies sample as positive, it does very precisely (96%) i.e. actual positives have large fraction of predicted positives.
- Such type of classifier (with high precision) may be useful if you want to restrict false positives, and may tolerate false negative a bit.

Accuracy Vs Precision Vs Recall

- Consider the case of a sensitive lab in an organization, where only a few members have access to lab. You want to design a classifier based on various features (may be biometrics), to classify members authorized (to enter) and unauthorized.

- In this case you do not want any unauthorized entry (false positive) to lab, even if authorized member is denied (false negative) entry occasionally.

- Here, you will like to have a classifier with high precision, even if recall is relatively low.

Accuracy Vs Precision Vs Recall

- Classifier B has low value of precision (0.55), and high value of recall (0.98).
- Classifier B is able to recall 98% of positives cases,
- However, in the process of identifying (classifying) positives, it makes more mistakes *i.e.* less precisely (55%) i.e. actual positives have small fraction of predicted positives.
- Such type of classifier (with high recall) may be useful if you want to restrict false negatives, and may tolerate false positives a bit.

Accuracy Vs Precision Vs Recall

- Let the disease G in the previous example be a cancer,
- In this case false negative (*i.e.* a patient with cancer is classified as free of cancer) may be life-threatening.

- Ideally, we want both precision and recall both high
  - *i.e.* minimize both false positives and false negative.
    - While maximizing true positives

Accuracy Vs Precision Vs Recall

- For example consider a forth classifier C
- Classifier C: predicts 101 samples as positives out of which 3 are positive by mistake

Classifier	# Predicted Positives		# Predicted Negatives	
	TP	FP	TN	FN
C	98	3	9,897	2

Classifier	Accuracy (%)	Precision	Recall
C	99.95	0.97	0.98

- Note: both FP and FN are low to make # misclassifications low

Precision-Recall Extremes

- From the definitions of precision & recall:

$$\text{precision} = \frac{\#\text{ true positives}}{\#\text{ predicted positives}} = \frac{\#\text{ true positives}}{\#\text{ true positives} + \#\text{ false positives}}$$

$$\text{recall} = \frac{\#\text{ true positives}}{\#\text{ actual positives}} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

- Worst case: If # true positives is 0 then both precision =0 and recall = 0, (minimum value)

- Best case: If # false positive = 0, then precision = 1 (maximum)

  If # false negative = 0, then recall =1 (maximum)

Precision-Recall Tradeoff

- Thus, $0 \leq \text{precision} \leq 1$ and $0 \leq \text{recall} \leq 1$

- Ideally, one would hope precision = **1** and recall = **1**

  i.e. FP = 0, and FN = 0 (No misclassifications)

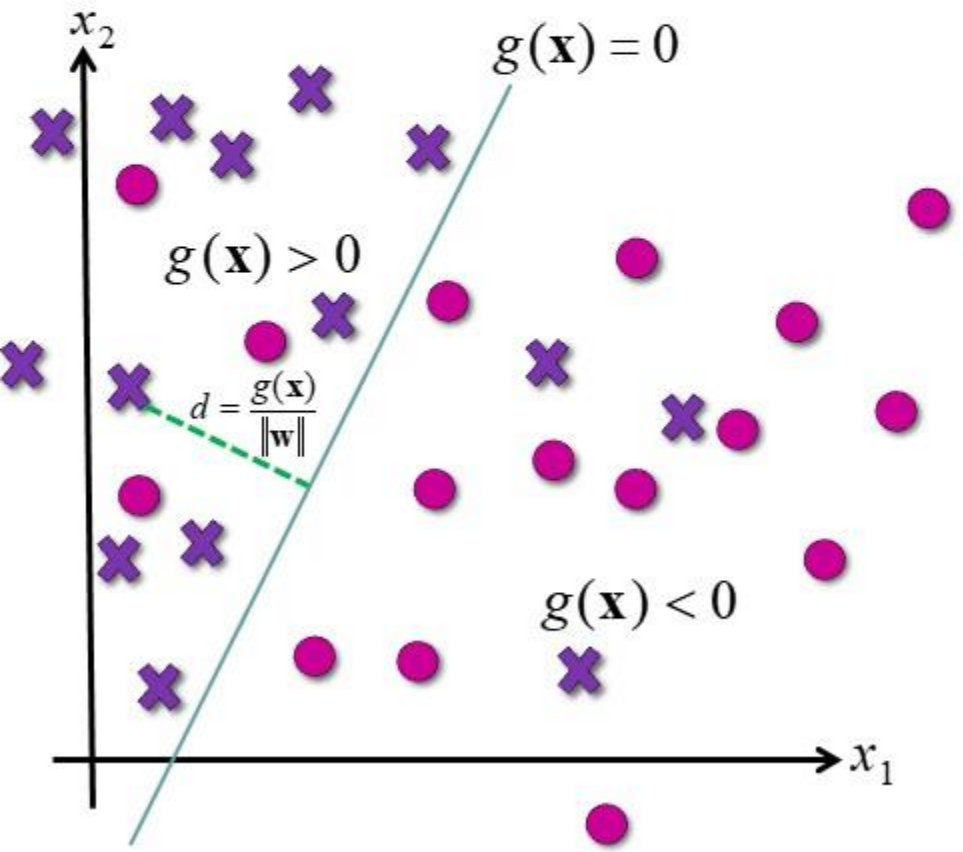
- However, practically there is tradeoff between precision & recall

  i.e. # false positives vs # false negatives

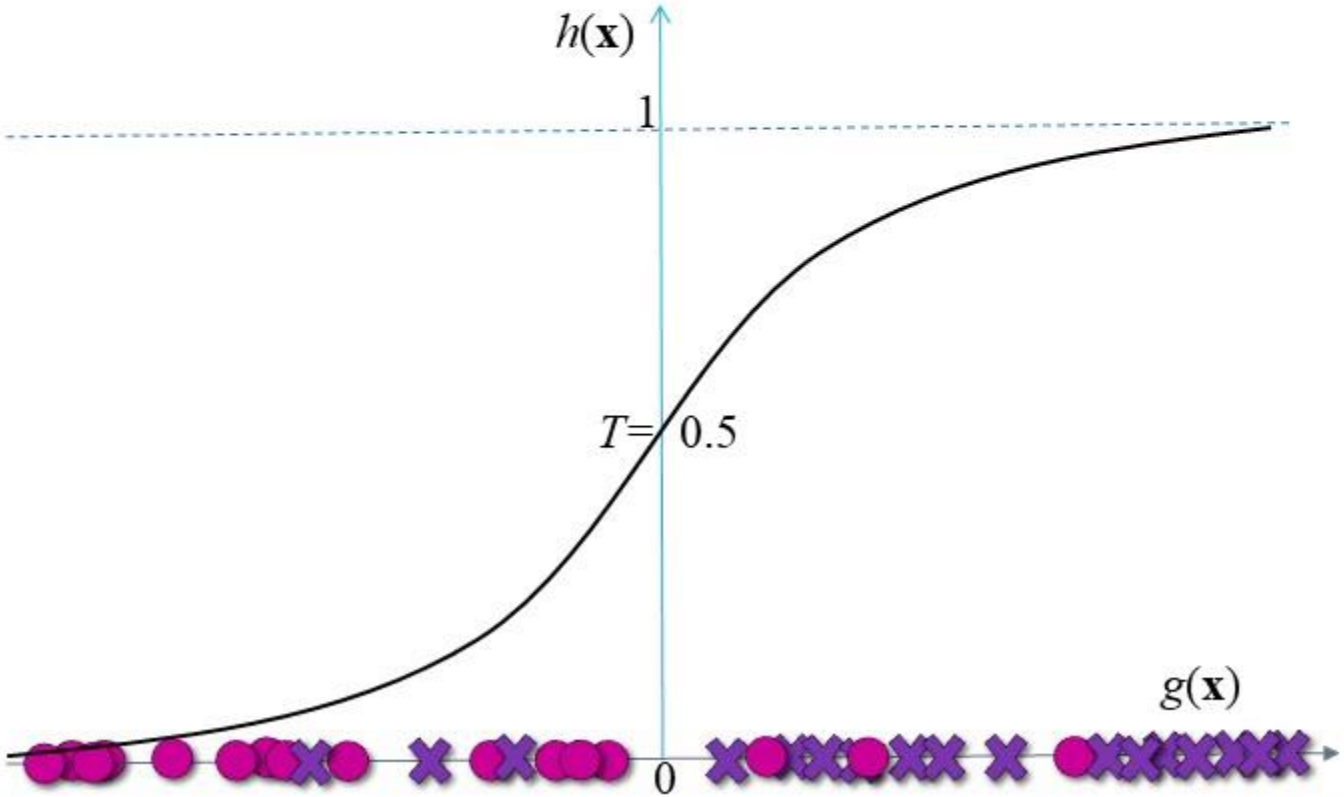
Positive class (actual);
Negative class (actual)

Precision-Recall Tradeoff

Samples in feature space



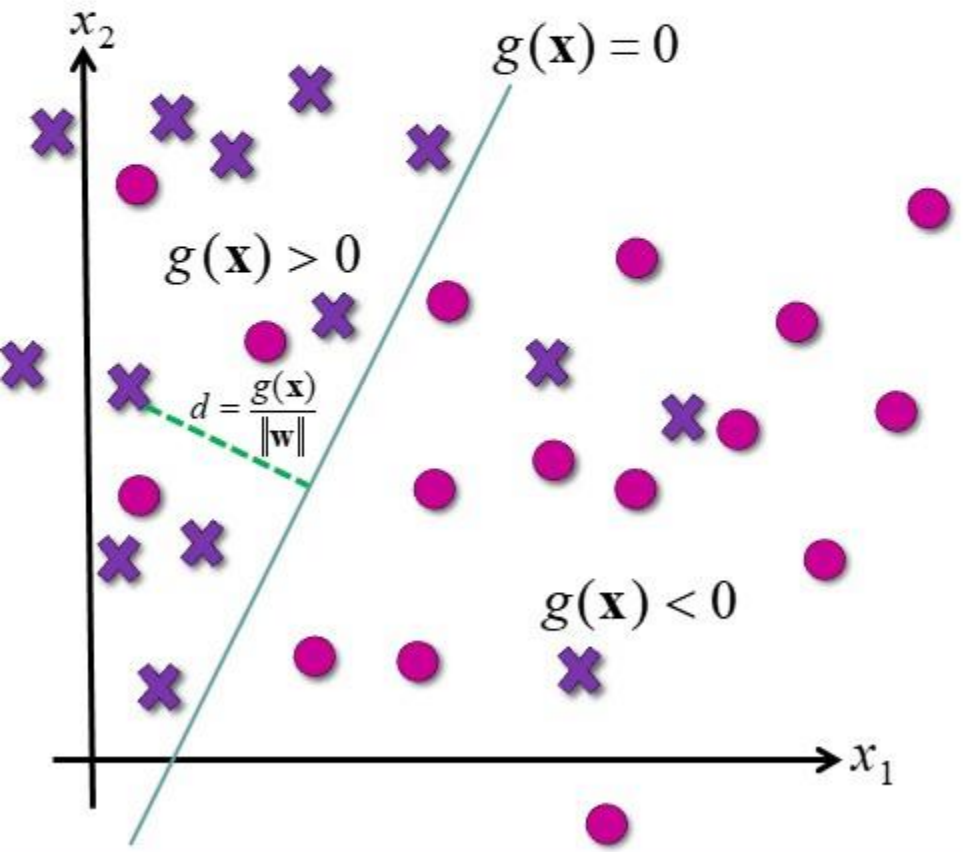
Samples shown in horizontal axis as per distance from the plane $g(\mathbf{x}) = 0$



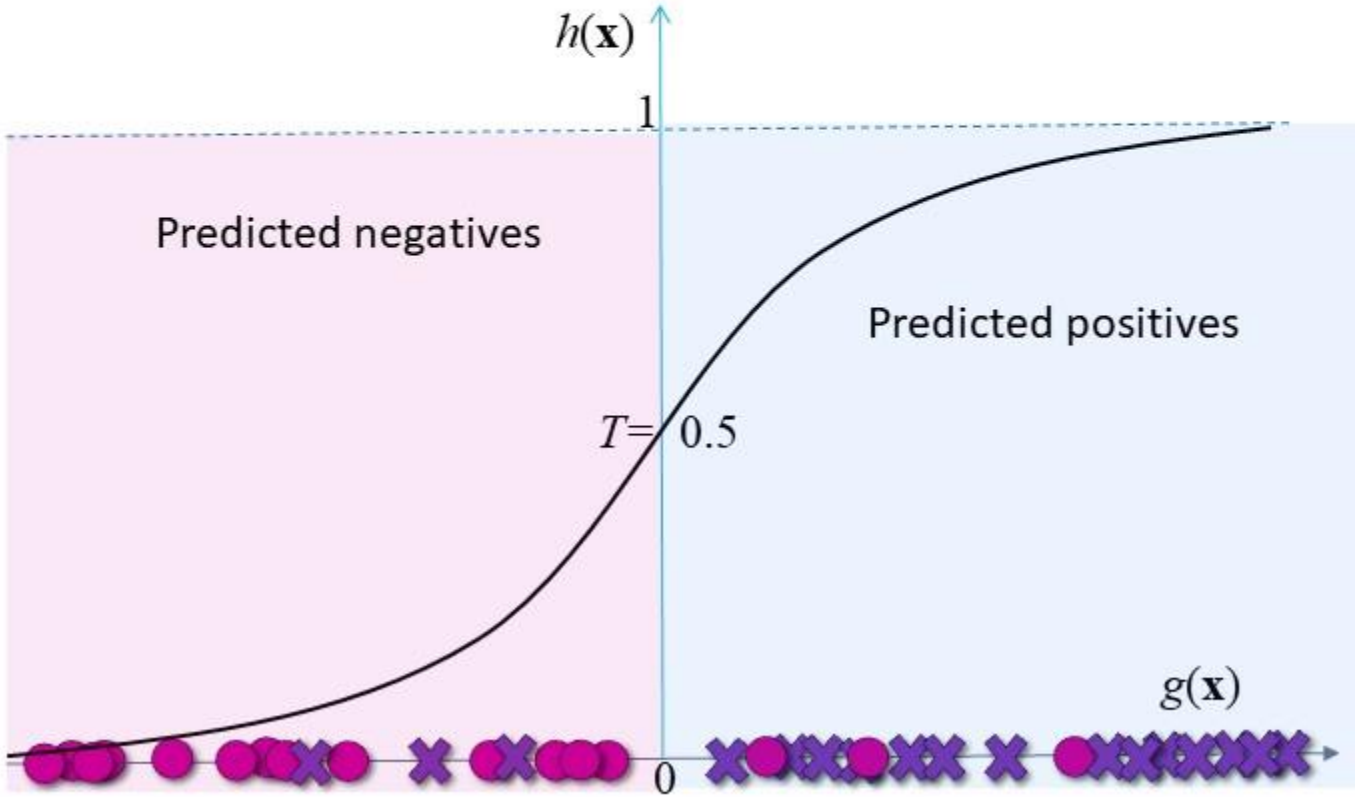
Positive class (actual);
Negative class (actual)

Precision-Recall Tradeoff

Samples in feature space



Samples shown in horizontal axis as per distance from the plane $g(\mathbf{x}) = 0$



Precision-Recall Tradeoff

- Remember $h(\mathbf{x})$ is probability of belonging to $y = 1$ (positive class)

If $h(\mathbf{x}) > T$ then

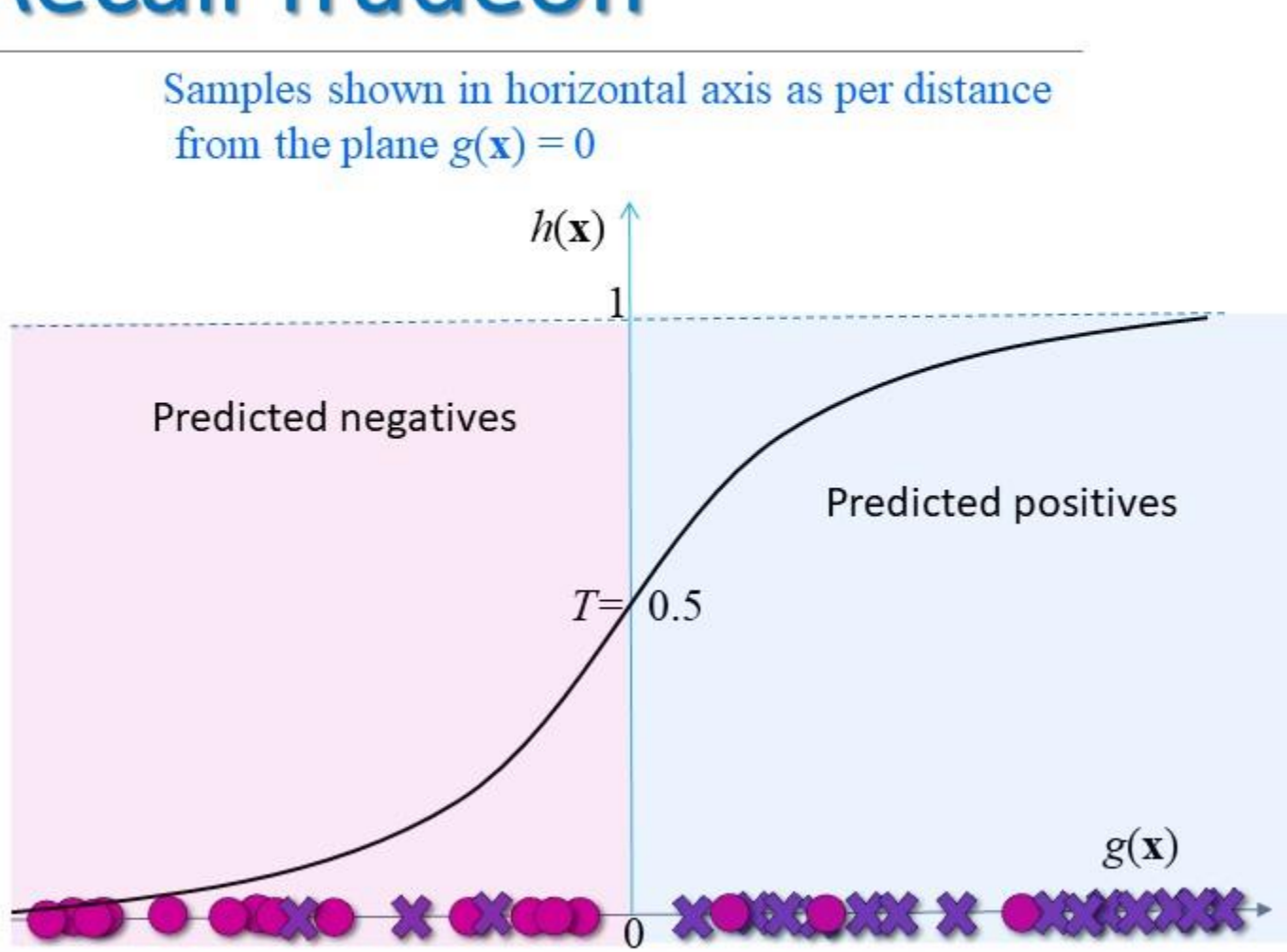
$\mathbf{x}$ belongs to positive class ($y = 1$)

and,

If $h(\mathbf{x}) < T$ then

$\mathbf{x}$ belongs to negative class ($y = 0$).

- Note: Here, threshold $T = 0.5$

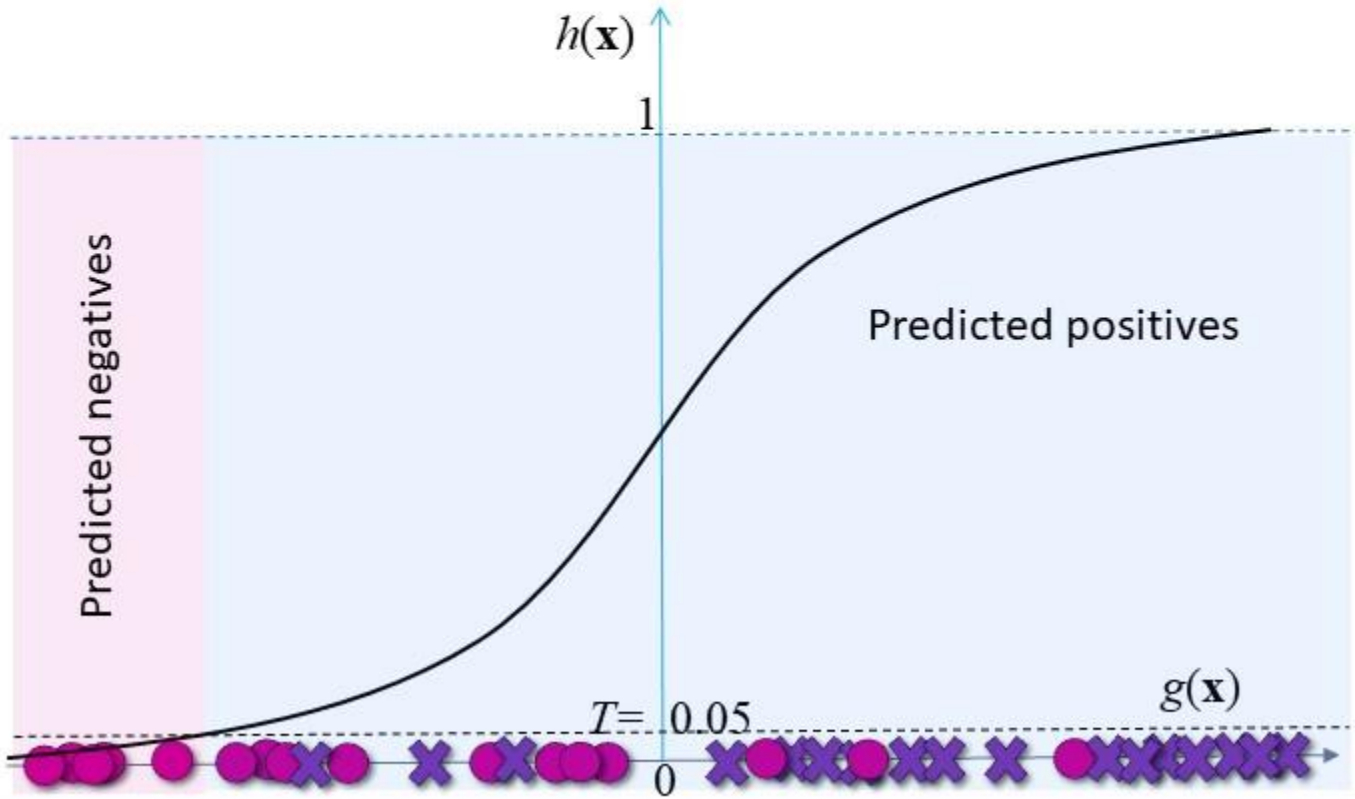


Positive class (actual);
Negative class (actual)

Precision-Recall Tradeoff

- If we reduce threshold T to very low value *i.e.* $T \approx 0$.
- For example $T = 0.05$
- Thus, If probability of $\mathbf{x}$ belonging to class y =1 (*i.e.* $h(\mathbf{x})$) is greater than 0.05
- Predicted class is Positive Class

Samples shown in horizontal axis as per distance from the plane $g(\mathbf{x}) = 0$



Precision-Recall Tradeoff

- Also, TP ≈ # actual positives

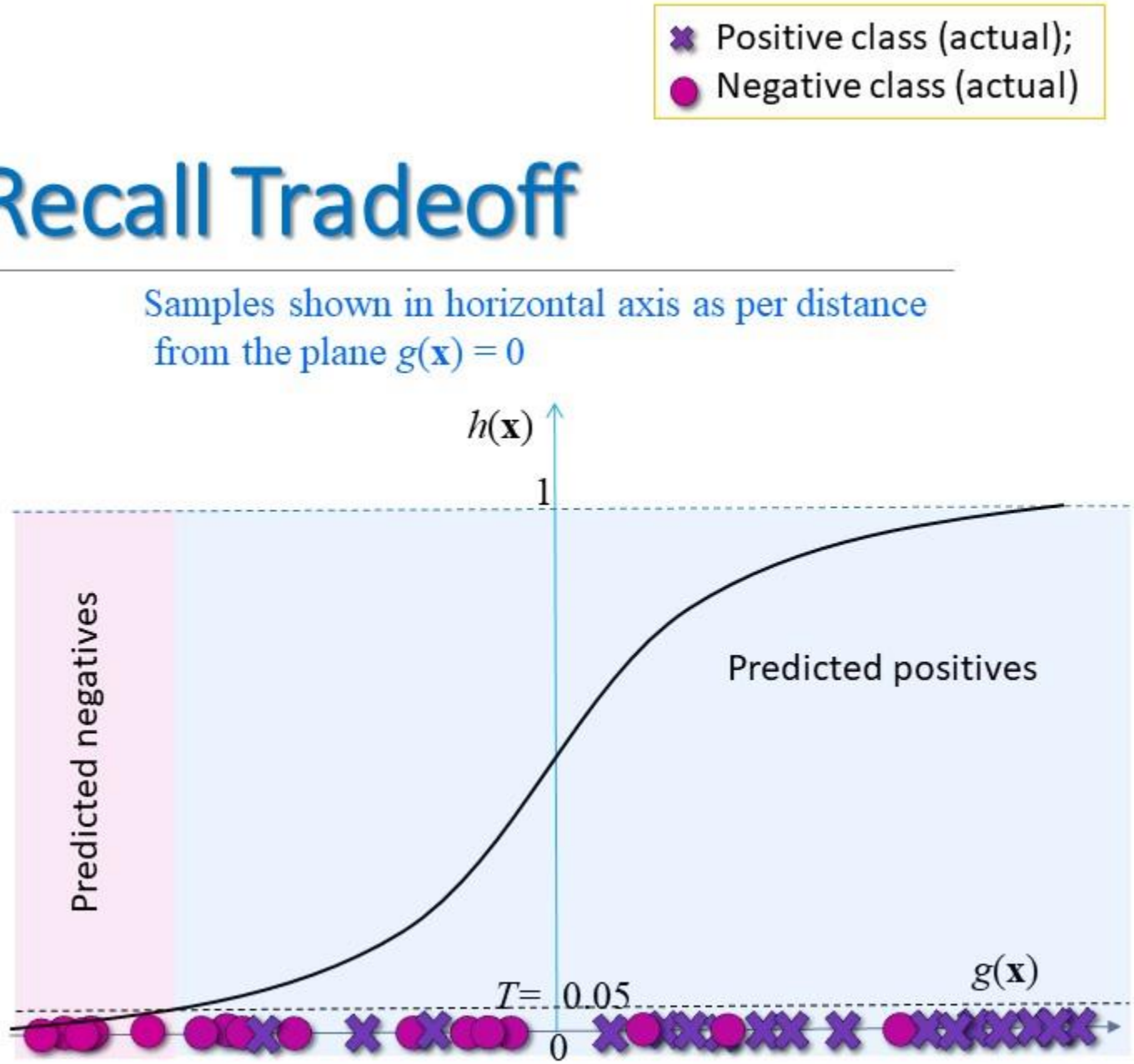
  Thus, Recall ≈ 1 (very high)

  FP ≈ Total sample (*i.e.* very large)

  # predicted positives = TP + FP

  = very large

  Thus, Precision is very low

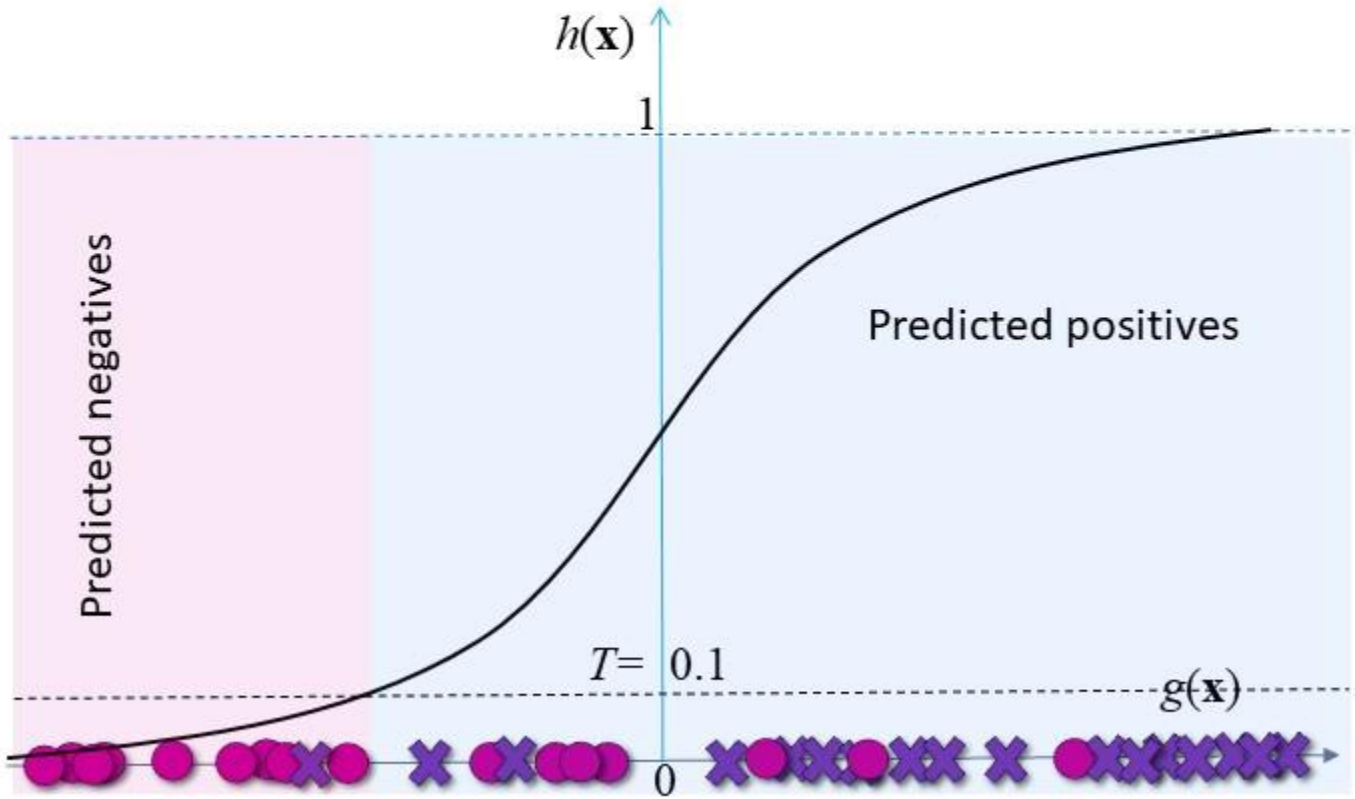


Positive class (actual);
Negative class (actual)

Precision-Recall Tradeoff

- As T increases ↑:
- # FP will decrease
- # predicted positives will decrease ↓
- Generally, # TP ↓, but in lesser rate.

Samples shown in horizontal axis as per distance from the plane $g(\mathbf{x}) = 0$

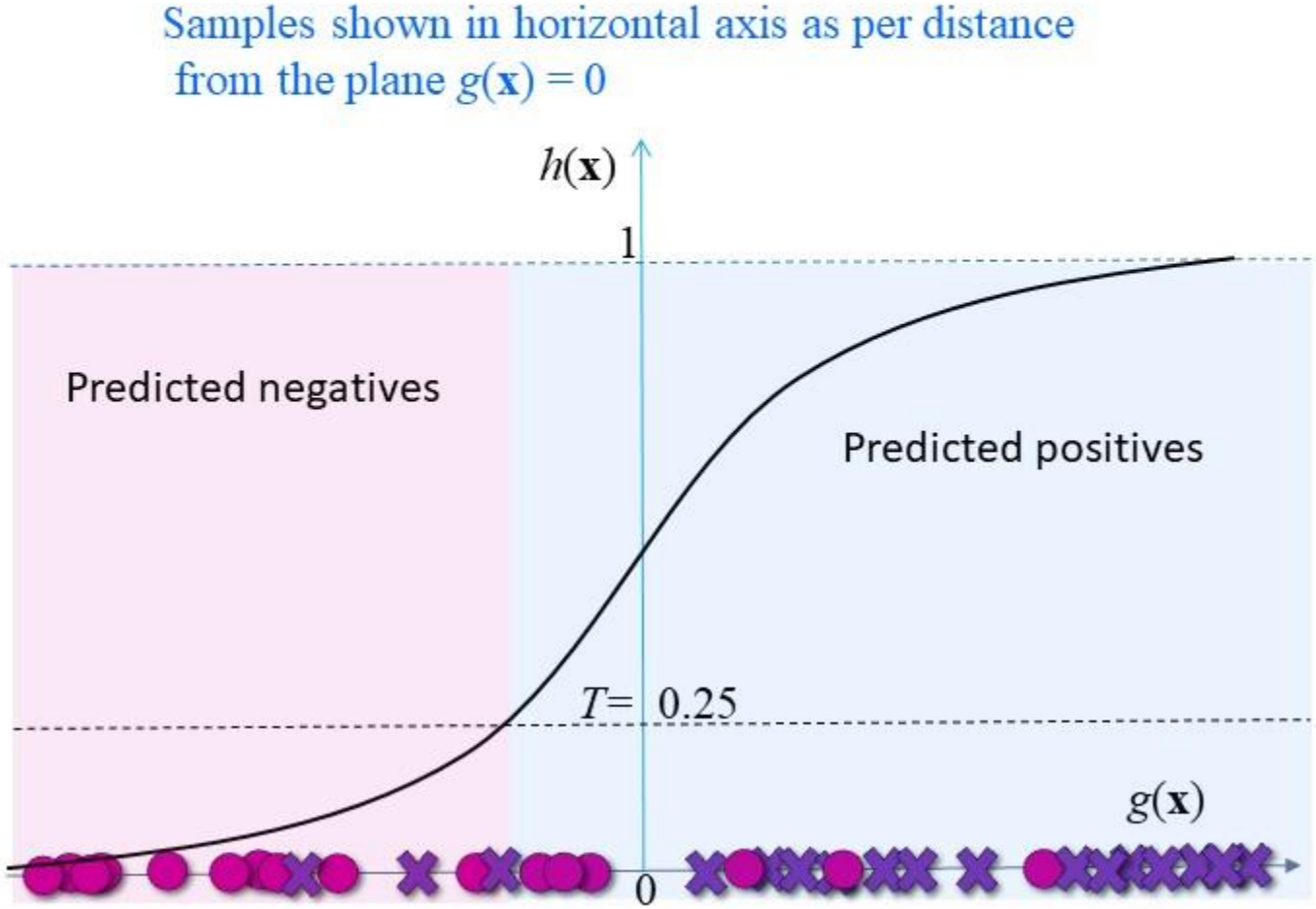


Precision-Recall Tradeoff

- As T increases ↑:
- # FP will decrease
- # predicted positives

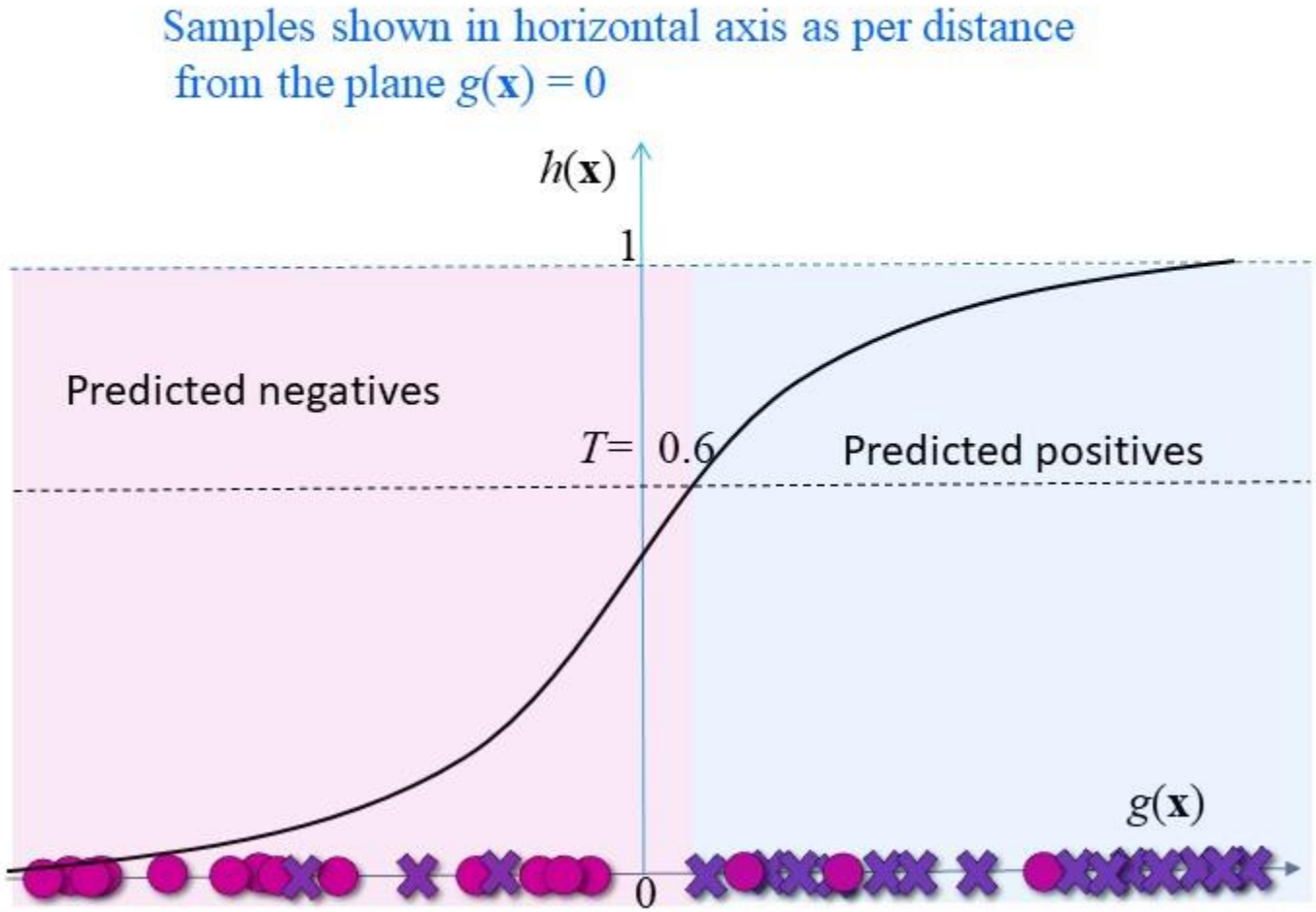
  will decrease ↓
- Generally, # TP ↓,

  but in lesser rate.



Precision-Recall Tradeoff

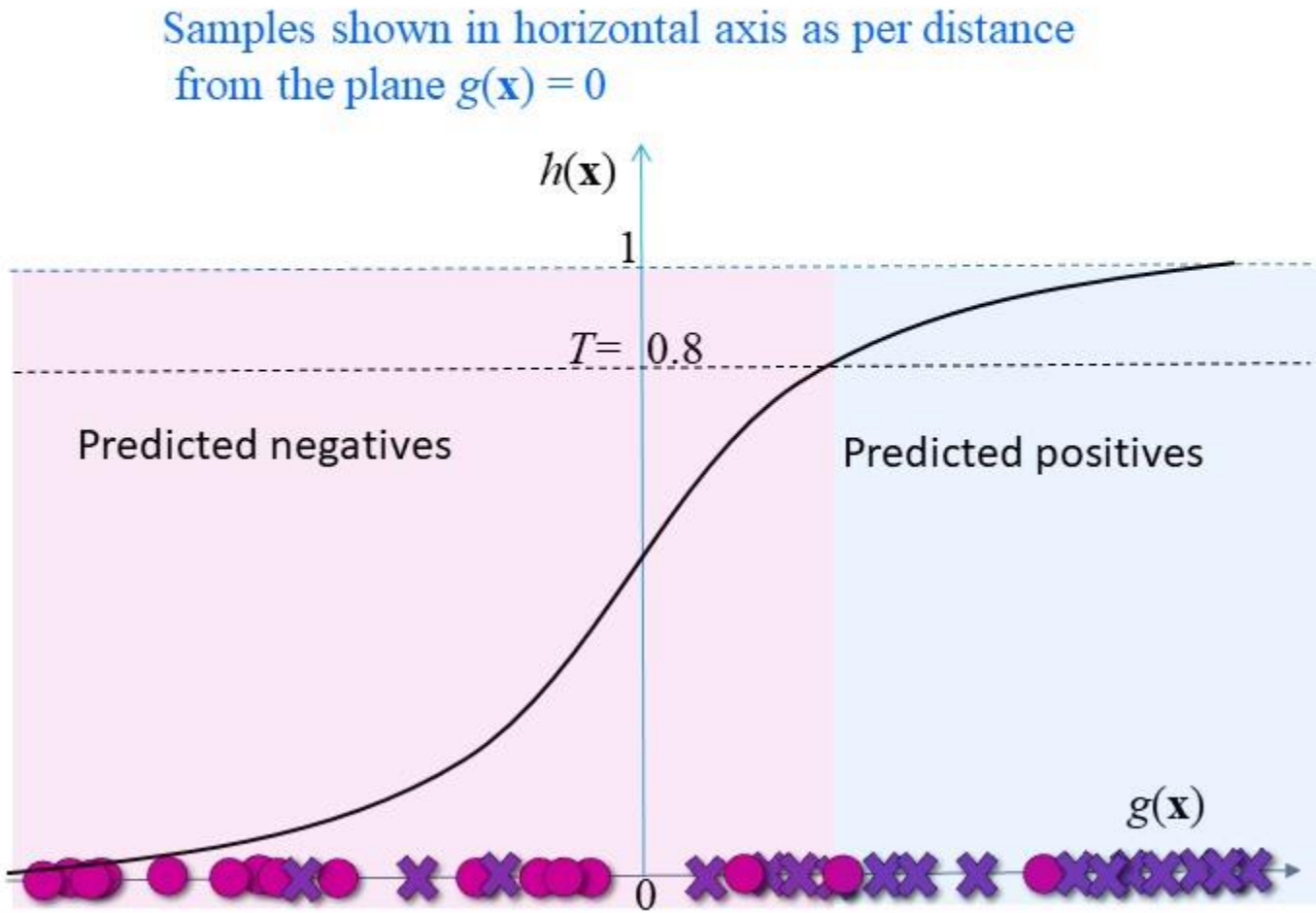
- As T increases ↑:
- # FP will decrease
- # predicted positives will decrease ↓
- Generally, # TP ↓, but in lesser rate.



Positive class (actual);
Negative class (actual)

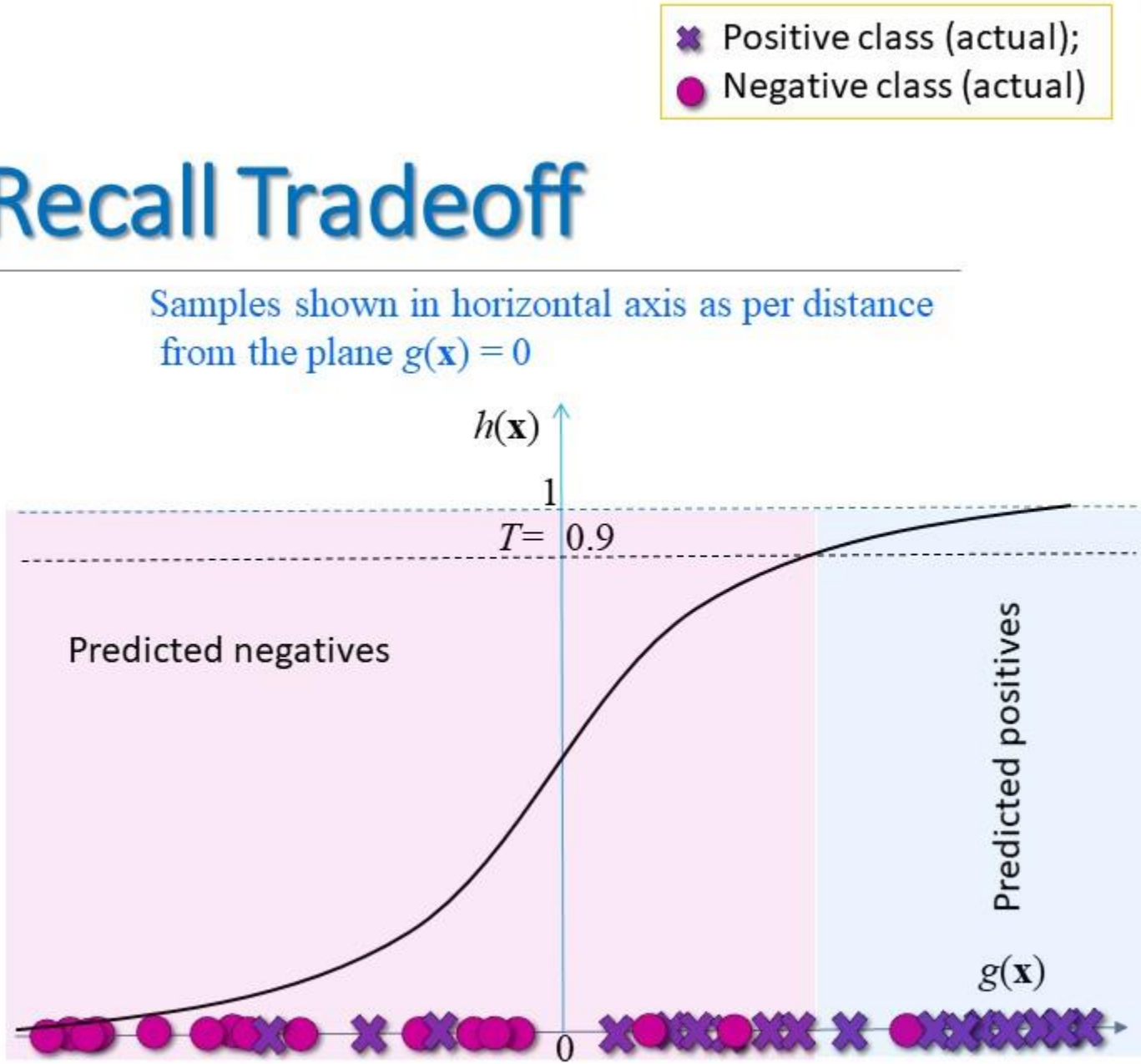
Precision-Recall Tradeoff

- As T increases ↑: Thus,
- Recall decreases ↓
- Since, there is decrease in # FP and
- TP ↓ relatively slowly
- Precision increases ↑ with T



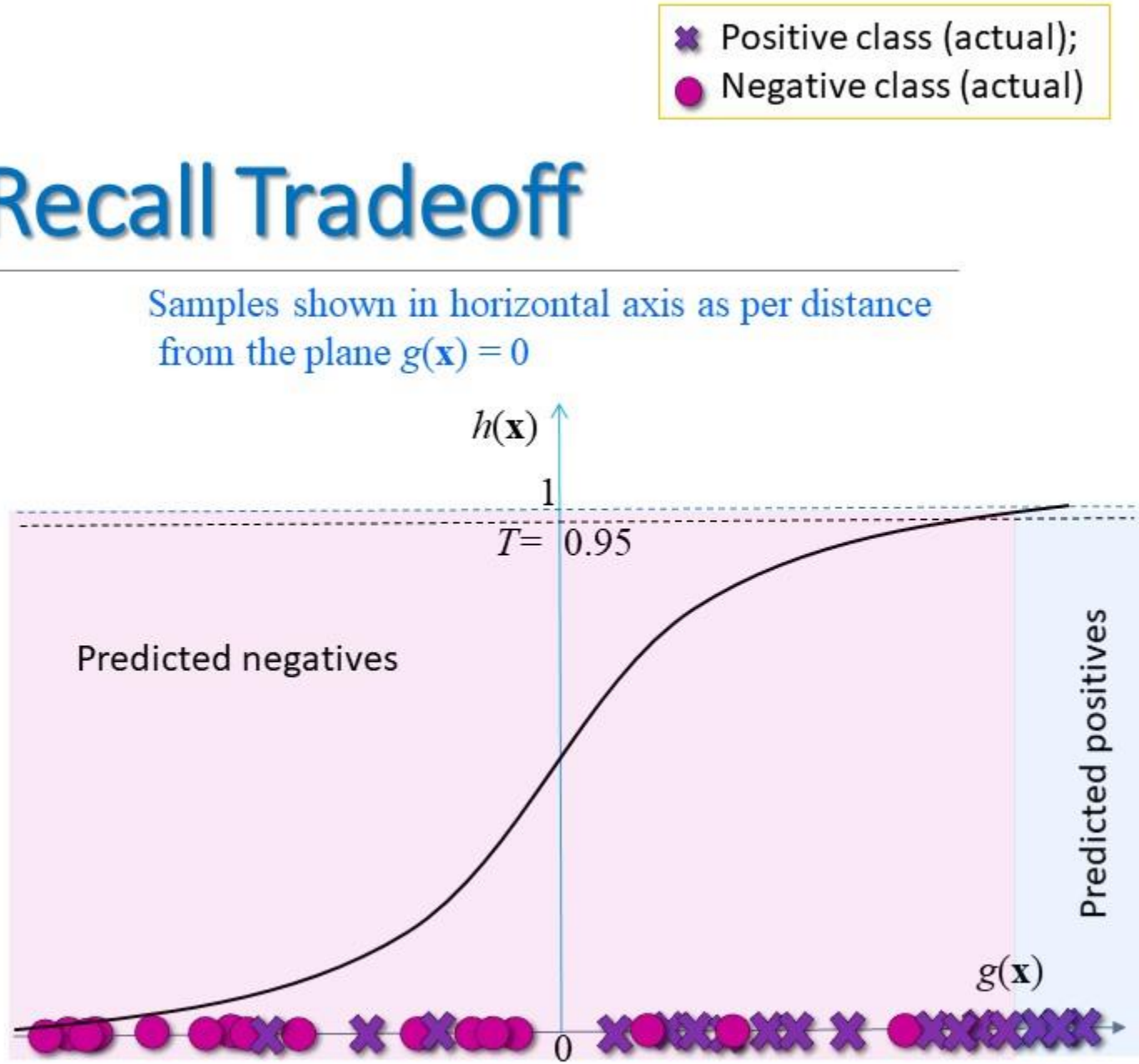
Precision-Recall Tradeoff

- In other words, as there are lesser # predicted positives,
- Higher fraction of it will be TP
- Thus, Precision ↑
- However, #TP decreases, as #actual positives is independent of T
- Recall ↓



Precision-Recall Tradeoff

- As T increases ↑ to other extreme *i.e.* *near* 1:
- #predicted positives ≈ TP
- Precision ≈ 1 (very high)
- However, TP very small
- Recall is very small



Precision-Recall Tradeoff

- Thus, depending on the application we may decide the threshold of a classifier.
- For example in case of entry to sensitive lab,
- we have high threshold and can afford false negatives, but not false positives.
- *i.e.* High precision and Low recall

Precision-Recall Tradeoff

- In other application, say in case of COVID-19 detection
- We want low threshold as we can not afford outbreak due to false negatives
- and can afford false positives, but not false negative.
- *i.e.* High recall and Low precision

Precision-Recall Curve (PR Curve)

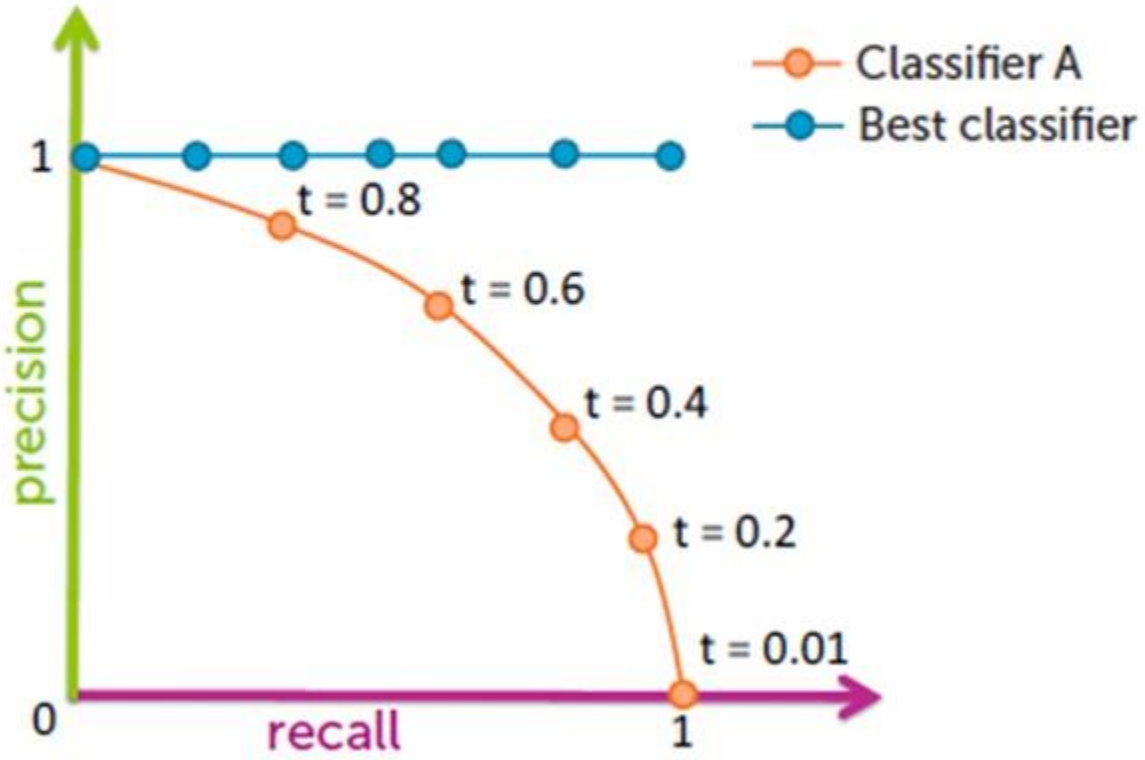
- We can plot a curve between recall (x-axis) vs precision (y-axis) for different probability thresholds.

- This is Precision-Recall curve (or PR curve)

- Note: Both the precision and the recall are focused on the positive class (the minority class) and are unconcerned with the true negatives (majority class).

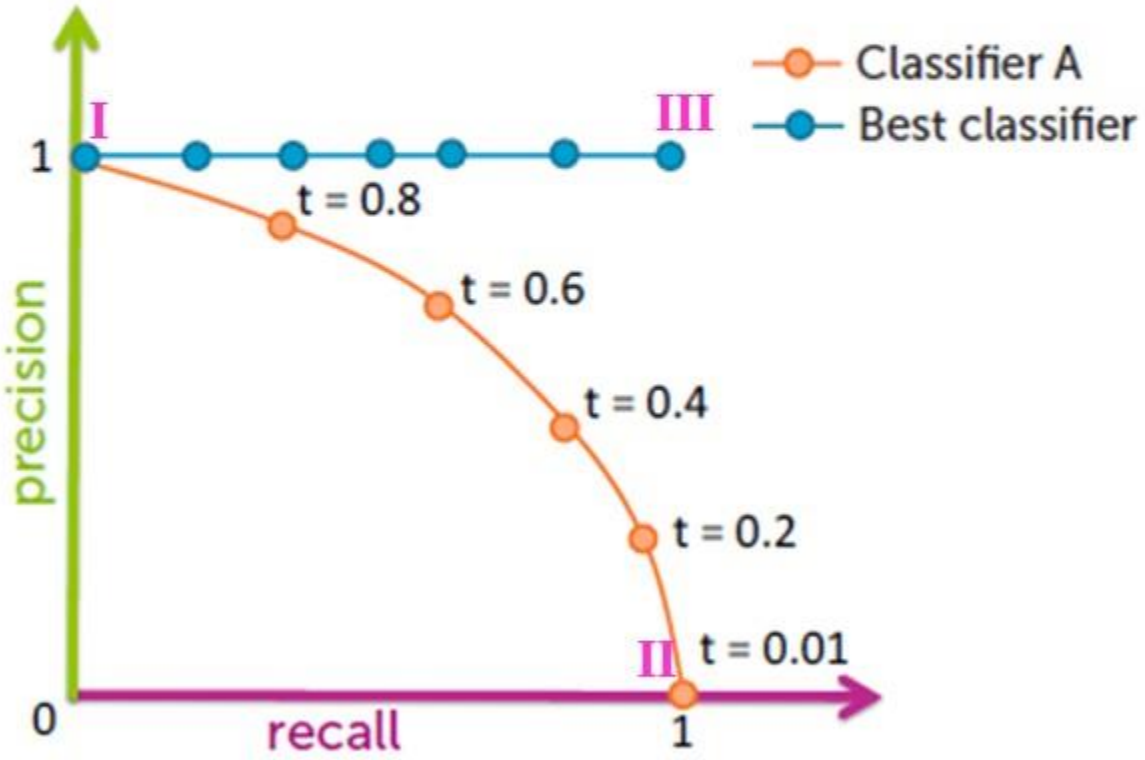
Precision-Recall Curve (PR Curve)

- Here t is threshold value of a classifier.
- On the curves, each point corresponds to a different threshold
- and its location corresponds to the resulting Precision and Recall when we choose that threshold.



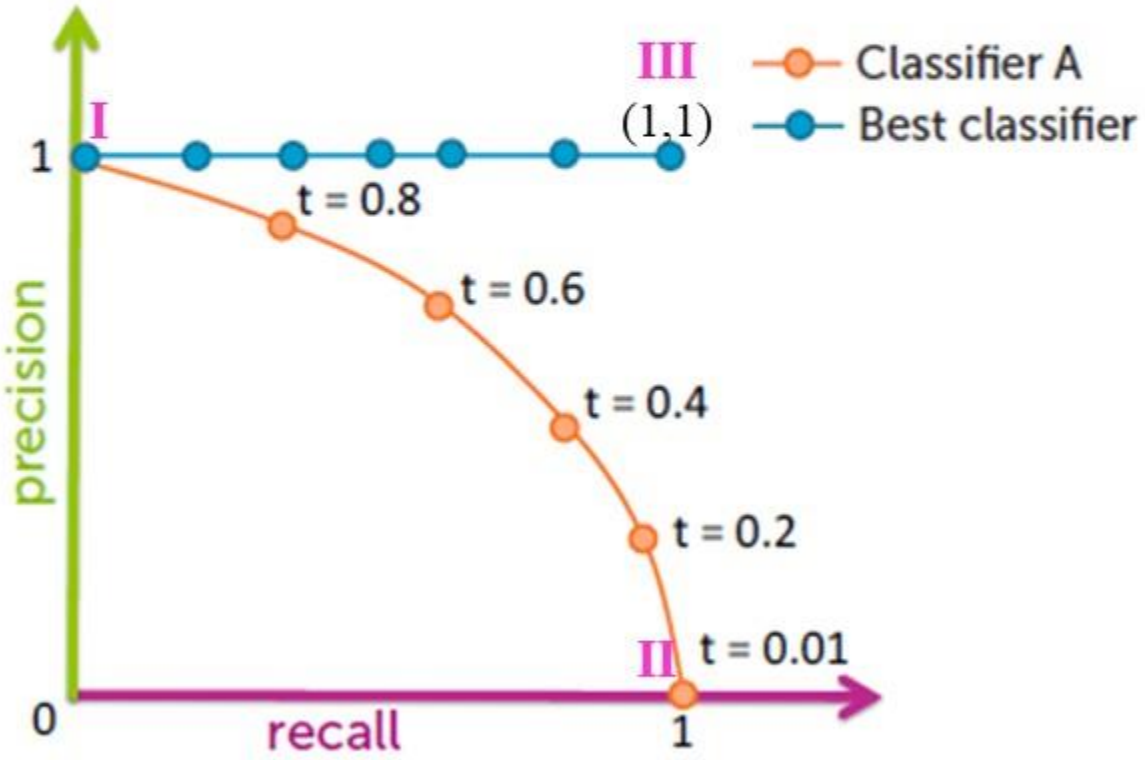
Precision-Recall Curve (PR Curve)

- Point I: represents recall ≈ 0 and precision ≈ 1 ($t \approx 1$).
- Pont II: represents recall ≈ 1 and precision ≈ 0 ($t \approx 0$).
- Point III: represent recall =1 and precision = 1 (Best point)



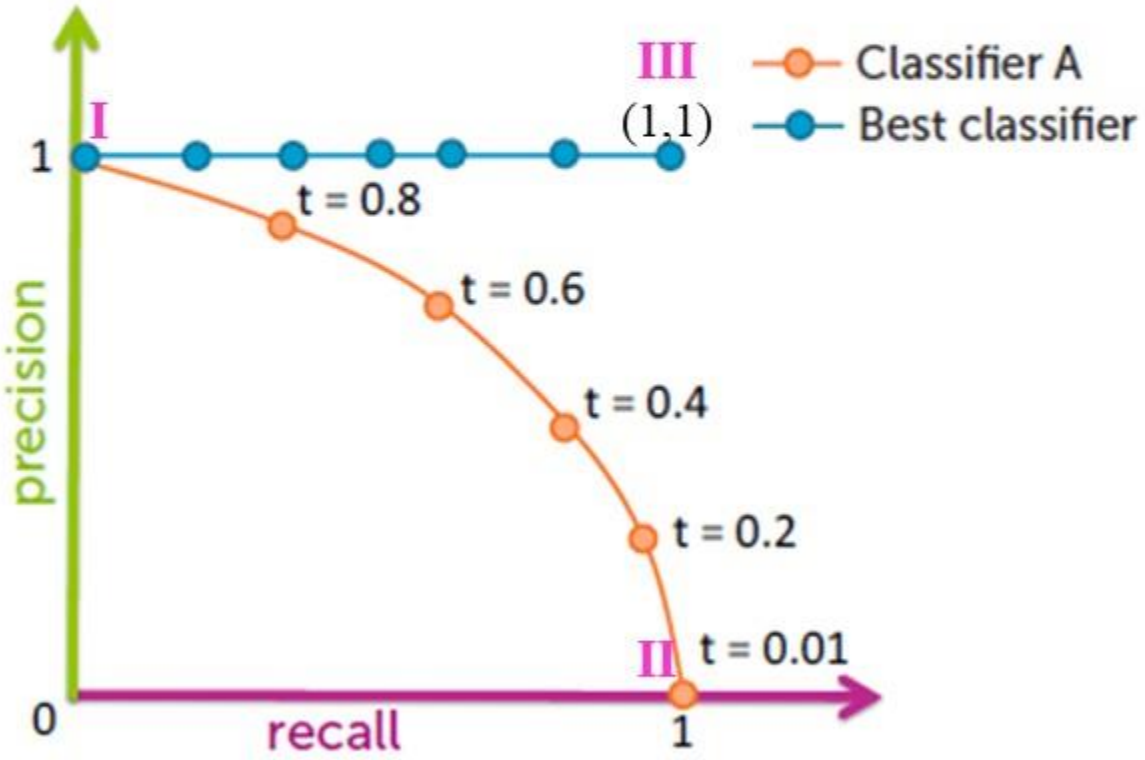
Precision-Recall Curve (PR Curve)

- Curve for a classifier A is shown in orange color.
- Blue curve represent a classifier which always have perfect precision *i.e.* 1, for different values of recall (as we wish to vary.)



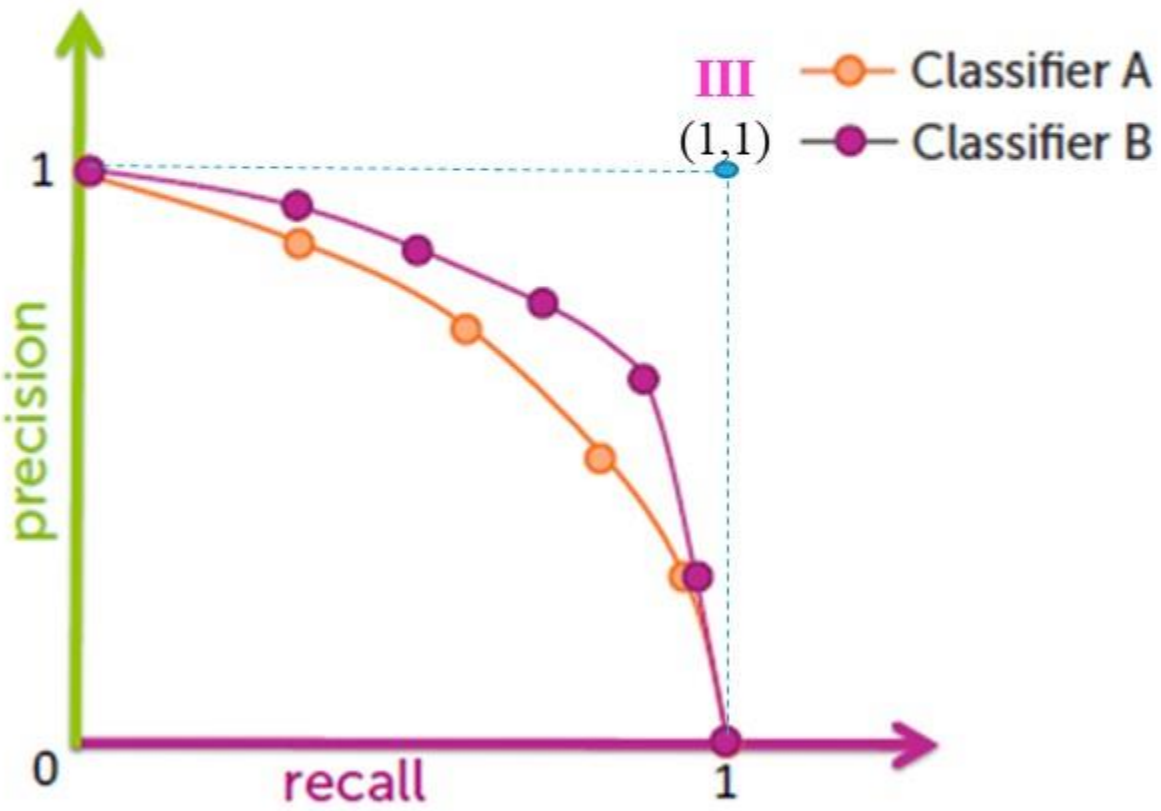
Precision-Recall Curve (PR Curve)

- Note: if want to maximize both precision and recall.
  - Select point nearest to III (1,1)
- If class distribution changes (in data), PR curve for same classifier will change.

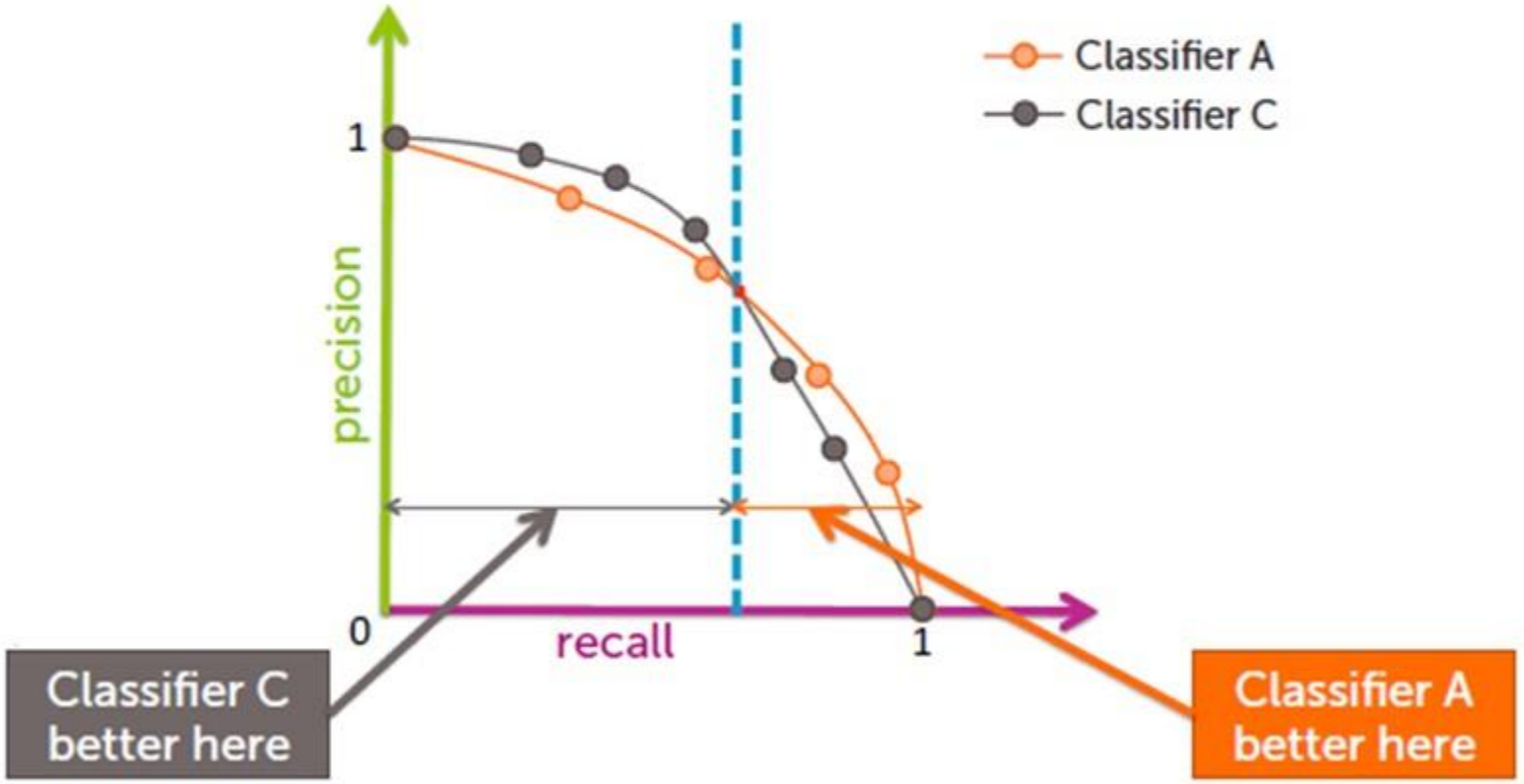


Precision-Recall Curve (PR Curve)

- Point III represent best performance by any classifier with no misclassification.
- Thus, any PR curve pulled towards III shows better performance.
- Classifier B is better than Classifier A



Precision-Recall Curve (PR Curve)



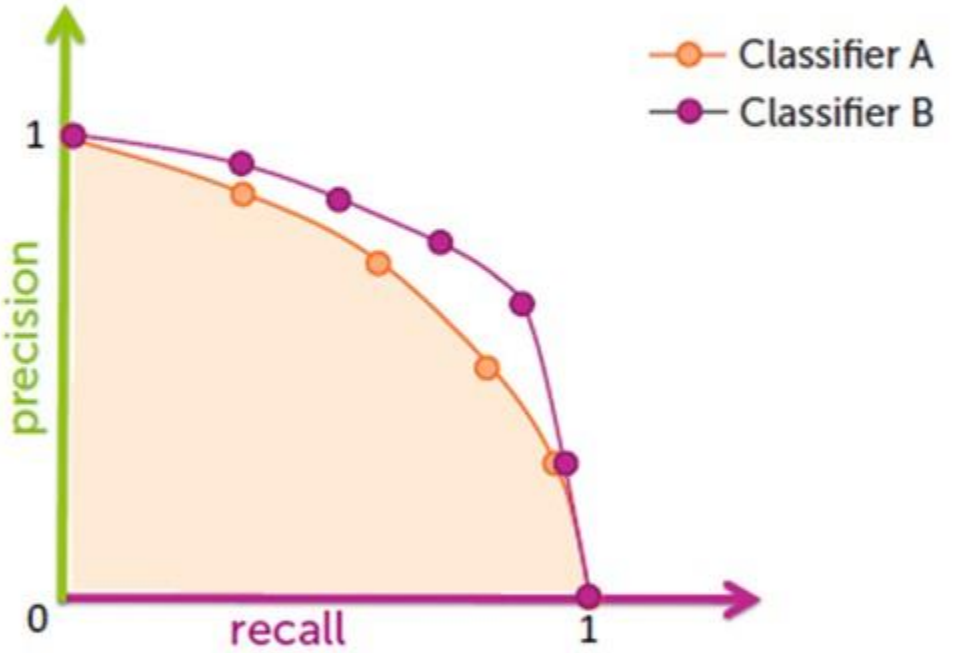
AUC for PR Curve

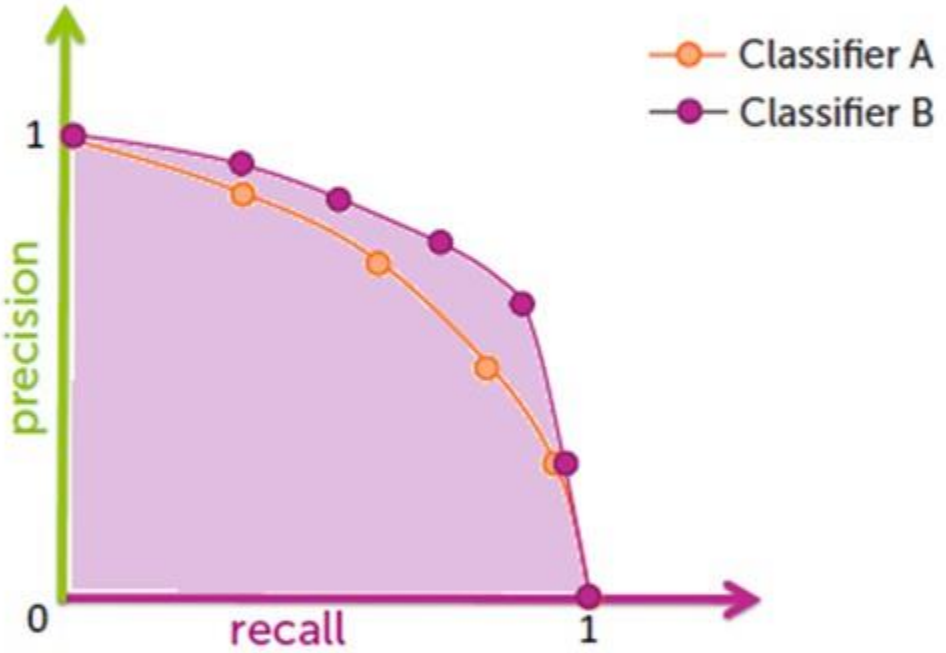
- Overall performance (irrespective of threshold) of two (or more) classifiers can be compared,
  - By comparing area under curve (AUC) of PR curves.

- The Precision-Recall AUC summarizes the curve with a range of threshold values as a single score.

- The score can then be used as a point of comparison between different models.

AUC for PR Curve





- Classifier B is better than Classifier A

F1 Score (F score)

- Rather than having two metric to evaluate a model, It is better to have one
- F score is a way to combine (harmonic mean) both precision and recall in one metric.

$$F = 2\frac{\text{precision x recall}}{\text{precision + recall}}$$

ROC Curve
(Receiver Operator Characteristics Curve)

- An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds.

- This curve plots two parameters: TPR & FPR

- True Positive Rate (TPR)= $\frac{\#\text{ true positives}}{\#\text{ actual positives}} = \frac{\#\text{ true positives}}{\#\text{ true positives} + \#\text{ false negatives}}$

- False Positive Rate (FPR) = $\frac{\#\text{ false positives}}{\#\text{ actual negatives}} = \frac{\#\text{ false positives}}{\#\text{ ture negatives} + \#\text{ false positives}}$

ROC Curve

- Note:

$$\text{Recall} = \text{True Positive Rate (TPR)} = \frac{\#\text{ true positives}}{\#\text{ actual positives}} = \frac{\#\text{ true positives}}{\#\text{ true positives} + \#\text{ false negatives}}$$

- TPR is also called Sensitivity

ROC Curve

- False Positive Rate (FPR) $= \frac{\#\text{ false positives}}{\#\text{ actual negatives}} = \frac{\#\text{ false positives}}{\#\text{ ture negatives}+\#\text{ false positives}}$
- Specificity $= \frac{\#\text{ ture negatives}}{\#\text{ ture negatives}+\#\text{ false positives}}$
- Thus, FPR = 1 – Specificity

- FPR tells us the fraction of healthy persons (# actual negatives) labelled as patient (positive class) *i.e.* false positives.

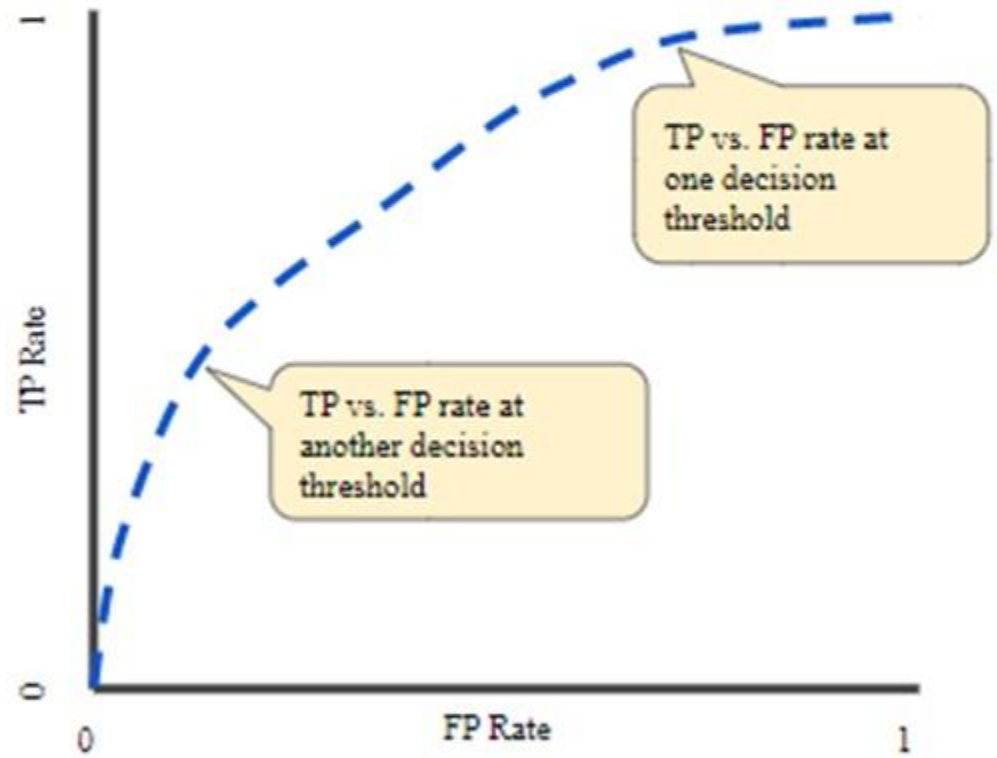
ROC Curve

- An ROC curve plots TPR vs. FPR at different classification thresholds.

- Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.

ROC Curve

- The following figure shows a typical ROC curve.



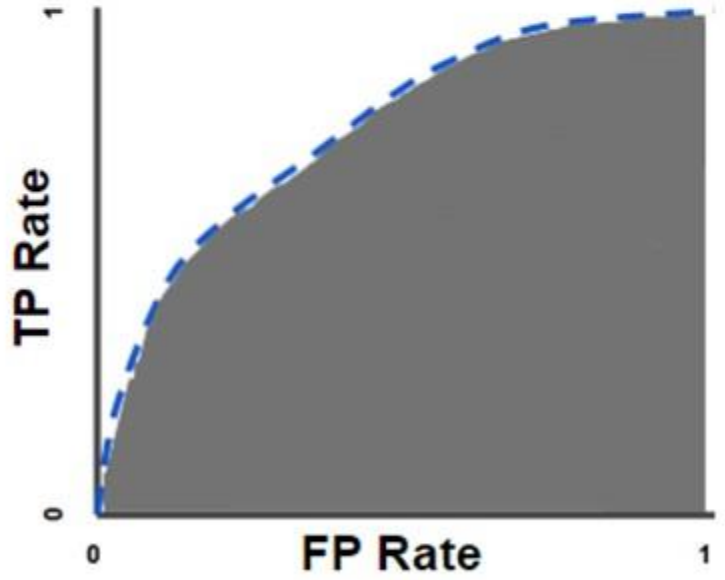
ROC Curve

- To compute the points in an ROC curve,
- we could evaluate a classifier (*e.g.* logistic regression model) many times with different classification thresholds, but this would be inefficient.

- Fortunately, there's an efficient, sorting-based algorithm that can provide this information for us, called AUC.

Area Under the ROC Curve: AUC

- AUC measures the entire two-dimensional area underneath the entire ROC curve from (0,0) to (1,1).



Area Under the ROC Curve: AUC

- AUC provides an aggregate measure of performance across all possible classification thresholds.

- One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example.