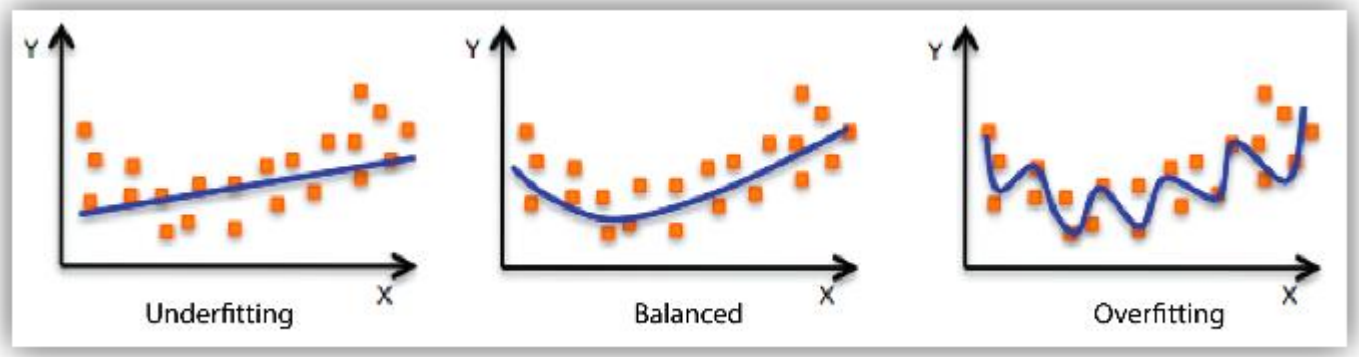
ML Evaluation

[Slides from Fardina Fathmiul Alam]

Overfitting and **Underfitting** are two crucial concepts in machine learning and are the prevalent ***causes for the poor performance*** of a machine learning model.



Overfitting

Overfitting is an undesirable machine learning behavior that occurs when the machine learning model gives accurate predictions for training data but not for new unseen data.

Overfitting is when the model becomes too tuned on the training set, and become unable to **generalize.**

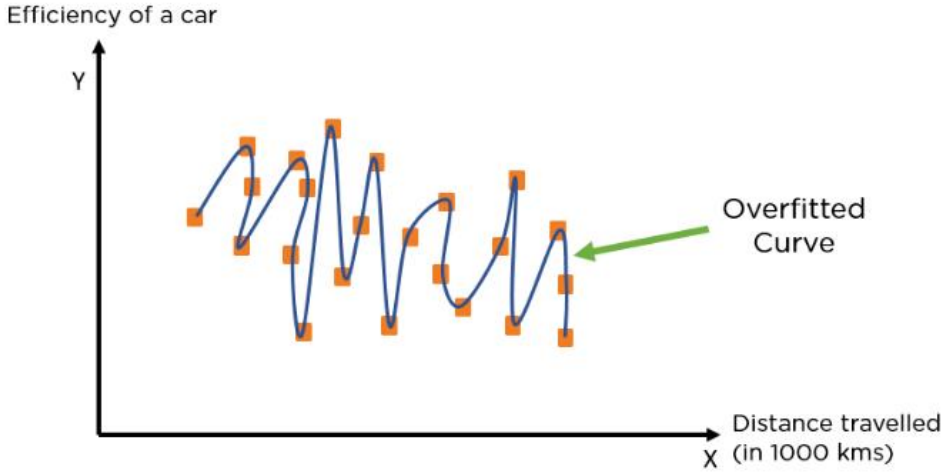
Think of this as taking a practice exam over and over and doing really well because you've already taken it.

What is Overfitting? cont.

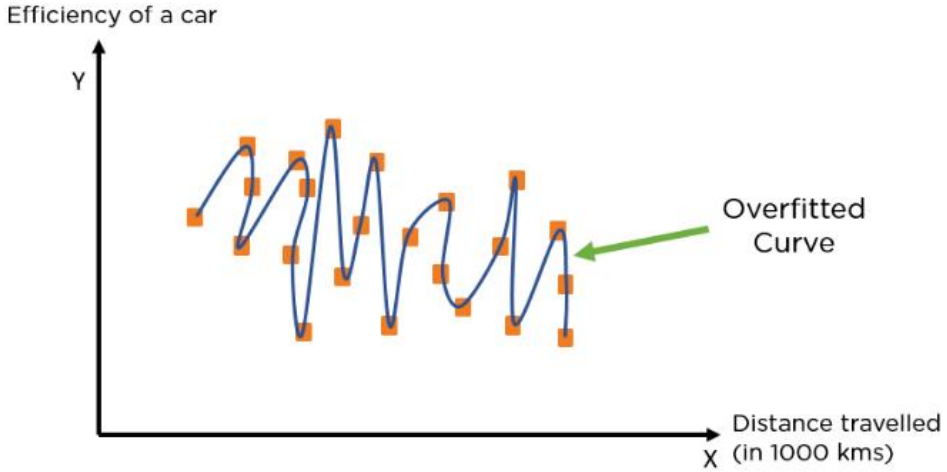
Definition: When a model performs very well for training data but has poor performance with test data (new data), it is known as overfitting.

Why? the machine learning model **learns the details and noise in the training data too well**, capturing noise or random fluctuations in the data rather than the underlying patterns.

- This leads to a model that is overly complex and performs poorly on unseen data.
- Negatively affects the performance of the model on test data.



- The training data contains large amounts of irrelevant information, called noisy data.
- The model is too complex: so it learns the noise within the training data.
- Instead of understanding the real patterns, it **starts memorizing the training data**, including noise and random fluctuations.



- As capturing noise and random variations in the training data, which hinders its ability to generalize to new, unseen data.
- <u>The size of the training dataset used is not enough</u>
  - lacks diversity and leads to models memorizing noise instead of learning true patterns

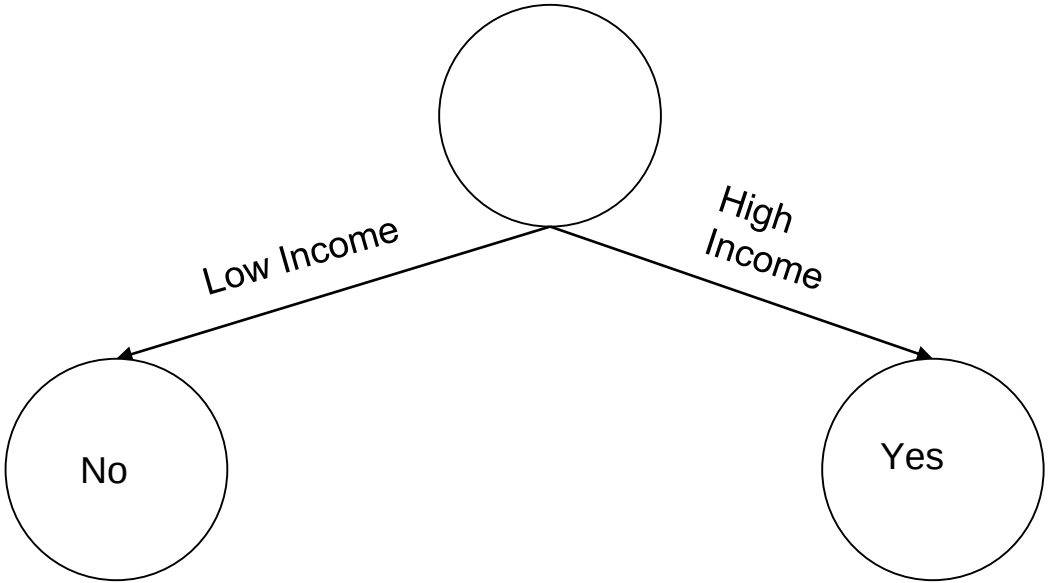
Some ways to Tackle Overfitting

- **Training model with sufficient data**
  - Add more data: More diverse and representative data leads to more robust and generalizable models.
- **Using K-fold cross-validation (splitting train-validation-test)**
  - Helps evaluate model performance and promote generalization to new data.
- **Using Regularization techniques (later topic)**
  - Reduce overfitting by adding penalties to model parameters, making them less sensitive to noise

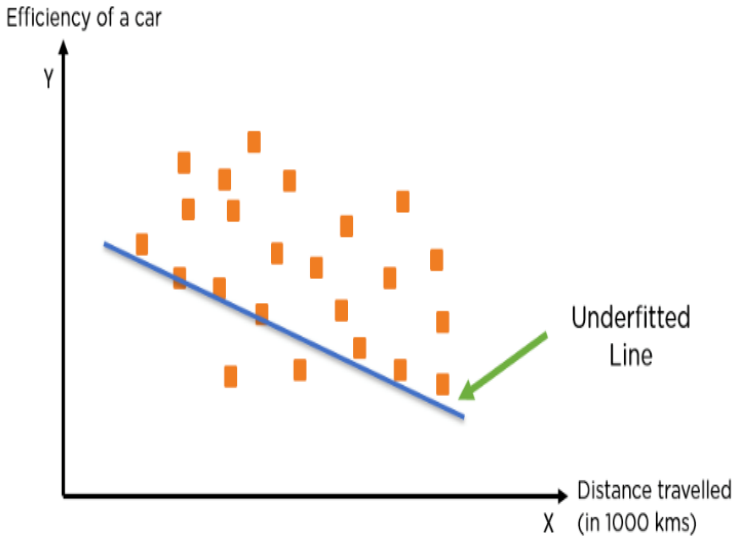
Underfitting

Our model is too general! It memorized one rule and is applying it everywhere.

- Think about college admission process only looking at GPA



What is Underfitting?

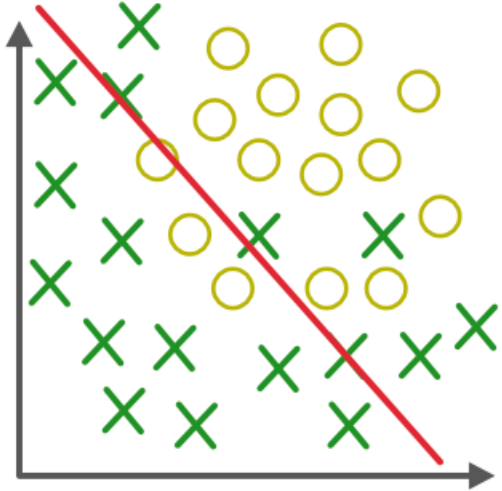


Def: When a model has not learned the patterns in the training data well and is unable to generalize well on the new data, it is known as underfitting.

- happens when the **model is too simple** to represent data complexity and performs poorly on both training and unseen data.
- Fw features miss important details.
- Due to poor performance on the training data, often result in unreliable predictions.

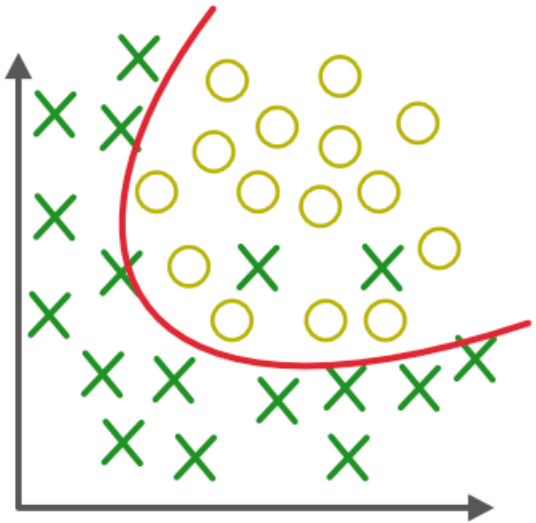
One of the way to tackle underfitting: If the model is too simple to capture the complexity of the data, increasing its complexity can help alleviate underfitting.

- Add More Layers or Neurons in Neural Network
- In polynomial regression, you can increase the degree of the polynomial to allow the model to fit more complex curves to the data.

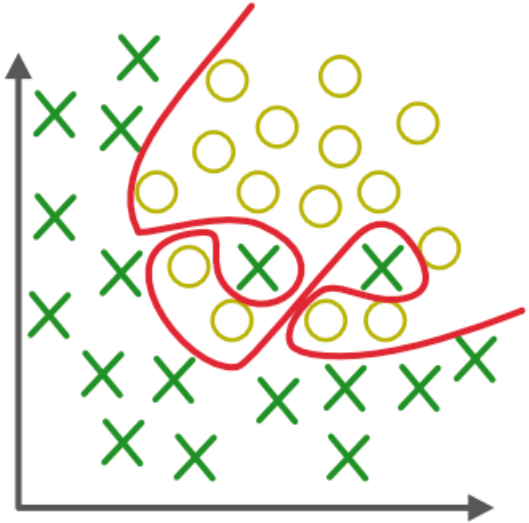


Under-fitting

(too simple to explain the variance)



Appropirate-fitting



Over-fitting

(forcefitting--too good to be true)

Overfitting vs Underfitting

	Overfitting	Underfitting
Definition	Model captures noise, fails to generalize.	Model is too simplistic, lacks generalization.
Causes	Too many parameters, limited data, excessive complexity.	Simple model, insufficient features, under-training.
Training Performance	Performs exceptionally well on training data.	Performs poorly on both training and new data.
Generalization	Poor generalization to new, unseen data.	Limited ability to capture underlying patterns.
Typical Solutions	Reduce model complexity, more training data, regularization.	Increase model complexity, feature engineering, more data.

Combatting Training Failures

Start with “Testing and Training”

Often in Machine Learning Models

The point of having models is that, **once they are trained, they will be able to classify *new* data.**

In our bank example, no two loan applicants are unique. What we're hoping is that the model uncovered the *underlying rules* about who repays loans are not. Like "higher income good".

Mitigating Overfitting: The Importance of Data Splitting in Model Training

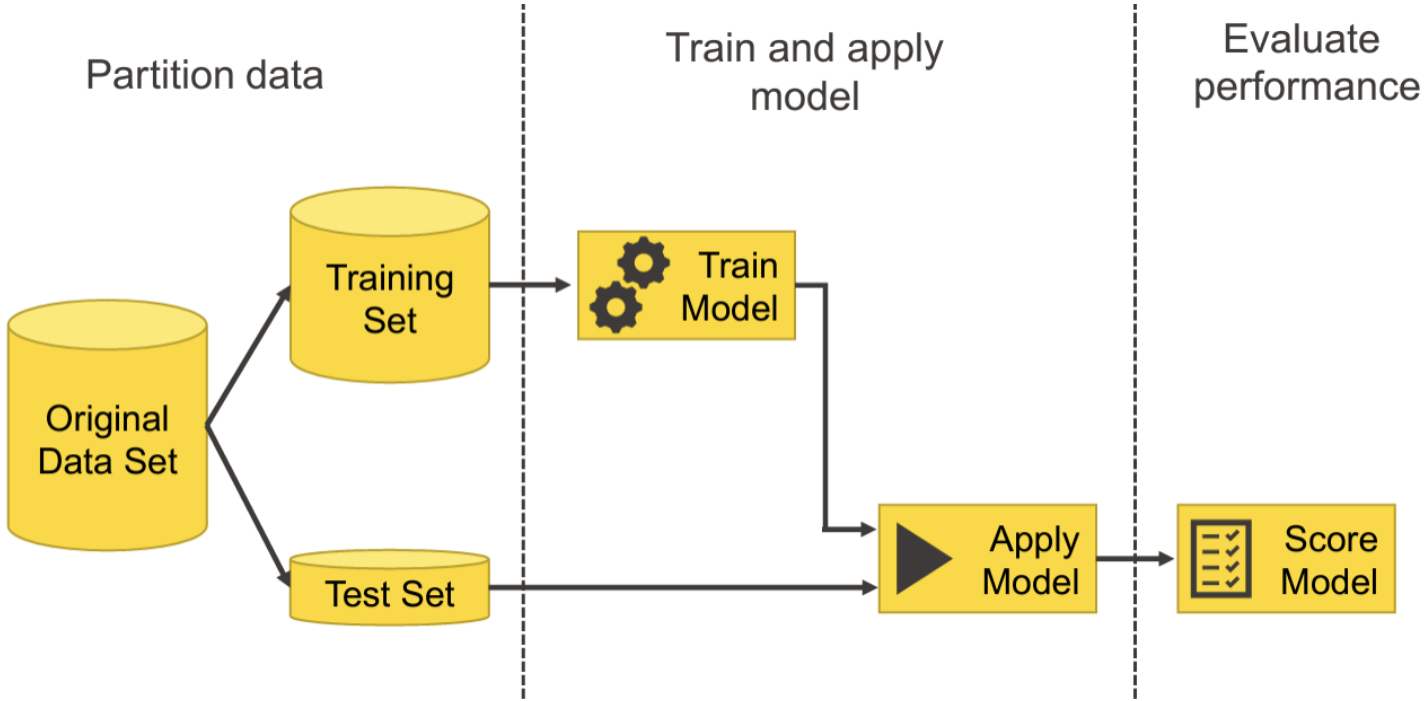
Issue: Models trained on an entire set of sample data carry the risk of overfitting the sample data, leading to poor performance when predicting beyond the sample. To compensate, sample data is often split into three subsets:

- Training data is used to fit a model.
- Validation data is used to evaluate model performance while adjusting parameter estimates and conducting feature selection.
- Test data is used to evaluate final model performance and compare different models.

The data distribution of each of the three subsets should be similar, and most of the data should be part of the training set (60-80% training data, 10-20% validation data, and 10-20% test data).

How do we know if it worked?

We Test Our Model!



Hide Some Data From Our Algorithm

Name	Income	Wealth	Repaid
Alice	100,000	300,000	Yes
Bob	150,000	10,000	No
Eve	500,000	1,500,000	Yes
Max	100,000	-100,000	No
Joan	100,000	350,000	Yes
Zhi	150,000	350,000	Yes
Tad	0	1400	No

Training

Name	Income	Wealth	Repaid
Alice	100,000	300,000	Yes
Bob	150,000	10,000	No
Eve	500,000	1,500,000	Yes
Max	100,000	-100,000	No
Joan	100,000	350,000	Yes

Testing

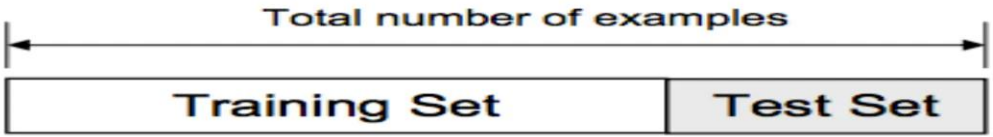
Name	Income	Wealth	Repaid
Zhi	150,000	350,000	Yes
Tad	0	1400	No

We already know their target label, use it to evaluate in on unseen data data

Training and Testing Data

Training Data: Subset of the dataset used to teach/train a machine learning model.

Testing Data: Subset of the dataset used to evaluate the model's performance.

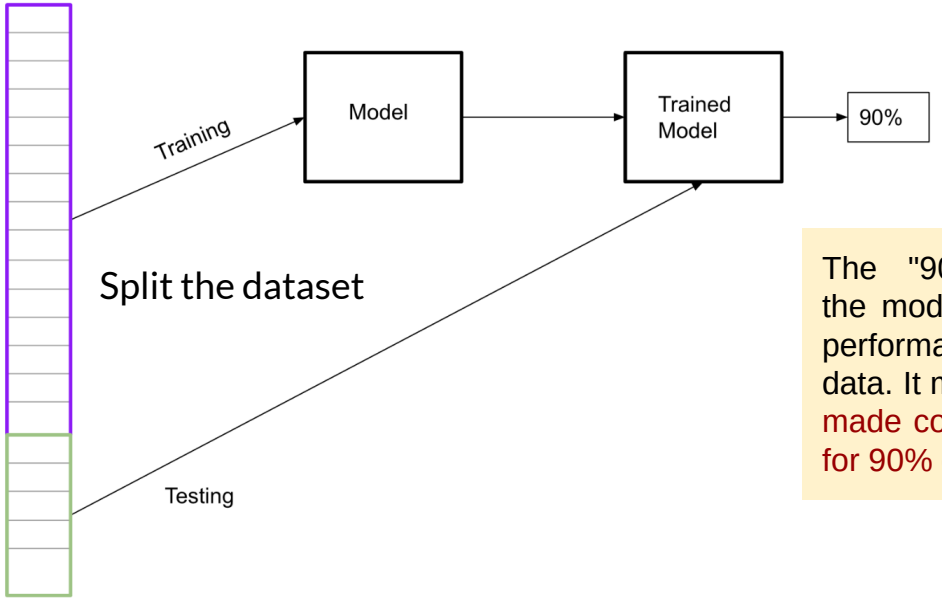



```
from sklearn.model_selection import train_test_split

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

Testing

Loaded the dataset



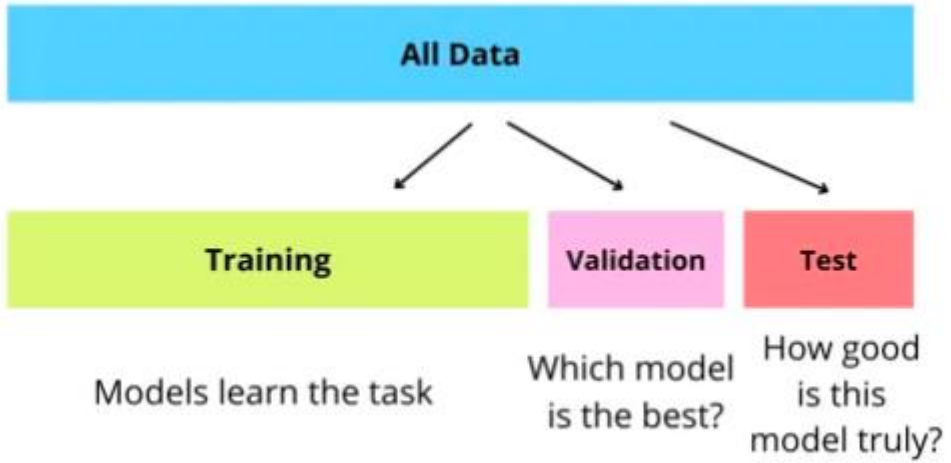
The "90%" represents the model's accuracy or performance on the test data. It means the model made correct predictions for 90% of the test data.

More Split : Train and Validation and Test

The train set → train the model,

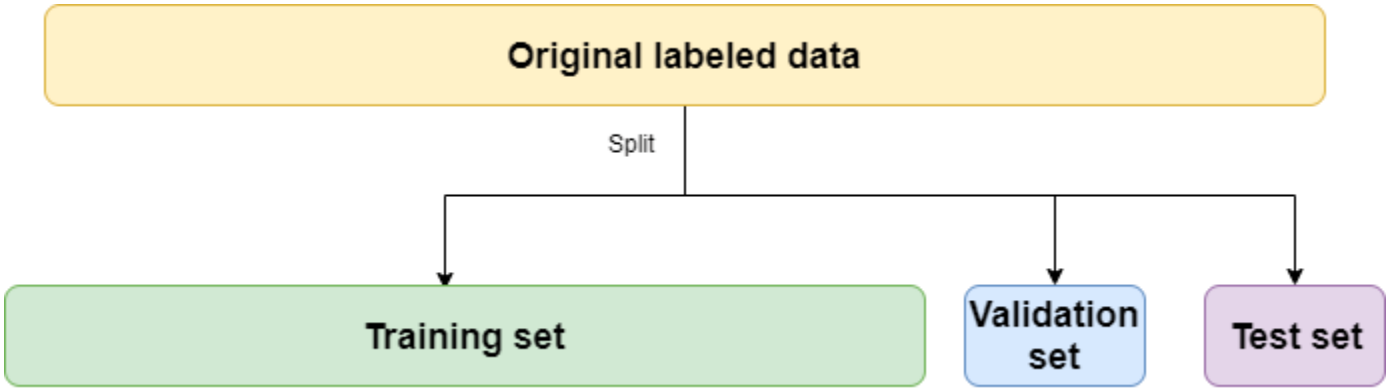
the test set → evaluate the model ,on data that was not used for training

The validation set → tune the hyperparameters of the model to improve the performance.



In machine learning, both the test and validation datasets are used to evaluate the performance of a trained model but purpose are different!

Validation Sets



How to split a dataset

TRAINING SET

The subset of data used to train a machine learning model

TEST SET

The subset of data used to evaluate the performance of a trained machine learning model on unseen examples, simulating real-world data

VALIDATION SET

The intermediary subset of data used during the model development process to fine-tune hyperparameters

Validation Sets

Unlike the test dataset, the validation dataset is typically **a subset of the training data** and is used iteratively during model development

- helps evaluate the model's performance during the training phase.
- **The model is not trained on this data**; instead, it is used to assess how well the model generalizes to unseen examples and helps make decisions about fine-tune (adjusting) hyperparameters or model architecture.

Test vs Validation Split

The test set helps us to **evaluate the model's performance on new data**,

and the validation set helps us to tune the hyperparameters of the model so that it is not overfitting to the training data.

- **Overfitting (later topic):** Overfitting occurs when the model learns the training data too well, and it becomes unable to generalize to new data.

Ensuring Robust Model Evaluation

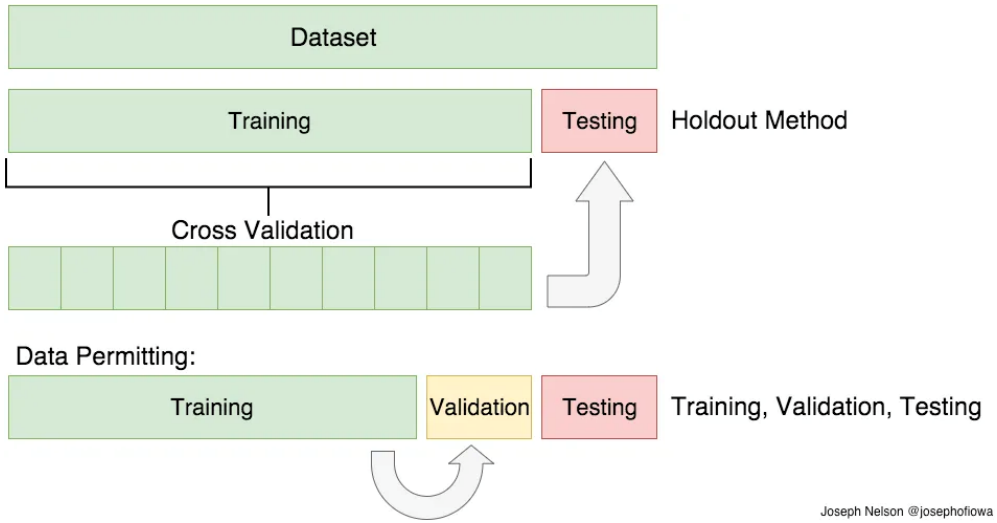
“Train/Test split" method in machine learning can be risky!

If the split isn't random

E.g. one part of the data only includes people from a specific state, employees with a certain income level, or only women, it can make the model learn too much about these specific cases. (Overfitting problem NEXT!)

To prevent this issue, we use "cross-validation"

Cross Validation is a technique that makes sure the model gets to learn from different parts of the data in a more balanced way.



Dataset Splitting: Data is divided into subsets, called "folds".

Training and Testing: Model trained and tested on different folds. One fold used as validation set, others for training.

Repetition: Process repeats with each fold as validation set.

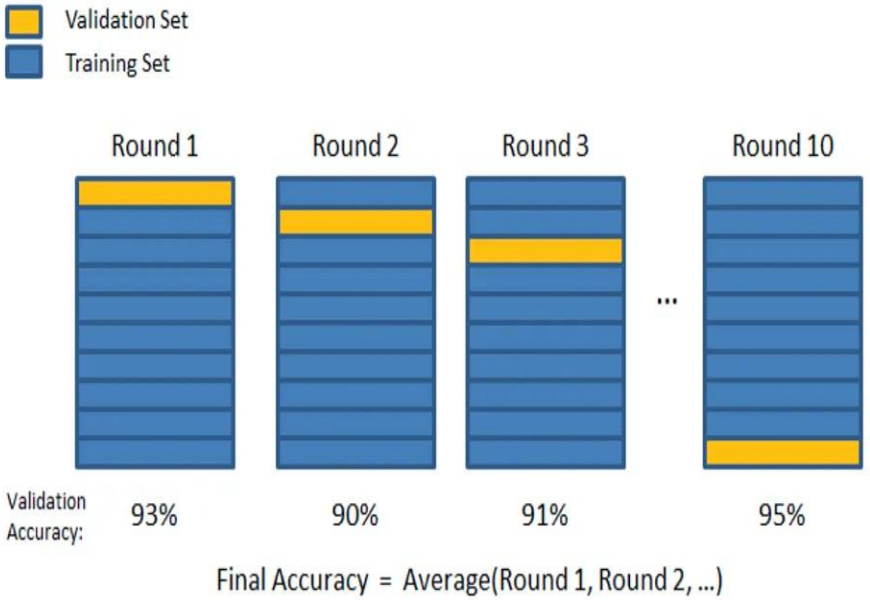
Performance Assessment: Evaluates model across all folds. Ensures model generalizes well to new data (avoid overfitting).

Types of Cross Validation

- K-Fold Cross-Validation
- Leave-One-Out Cross-Validation (LOOCV)
- Many more…..

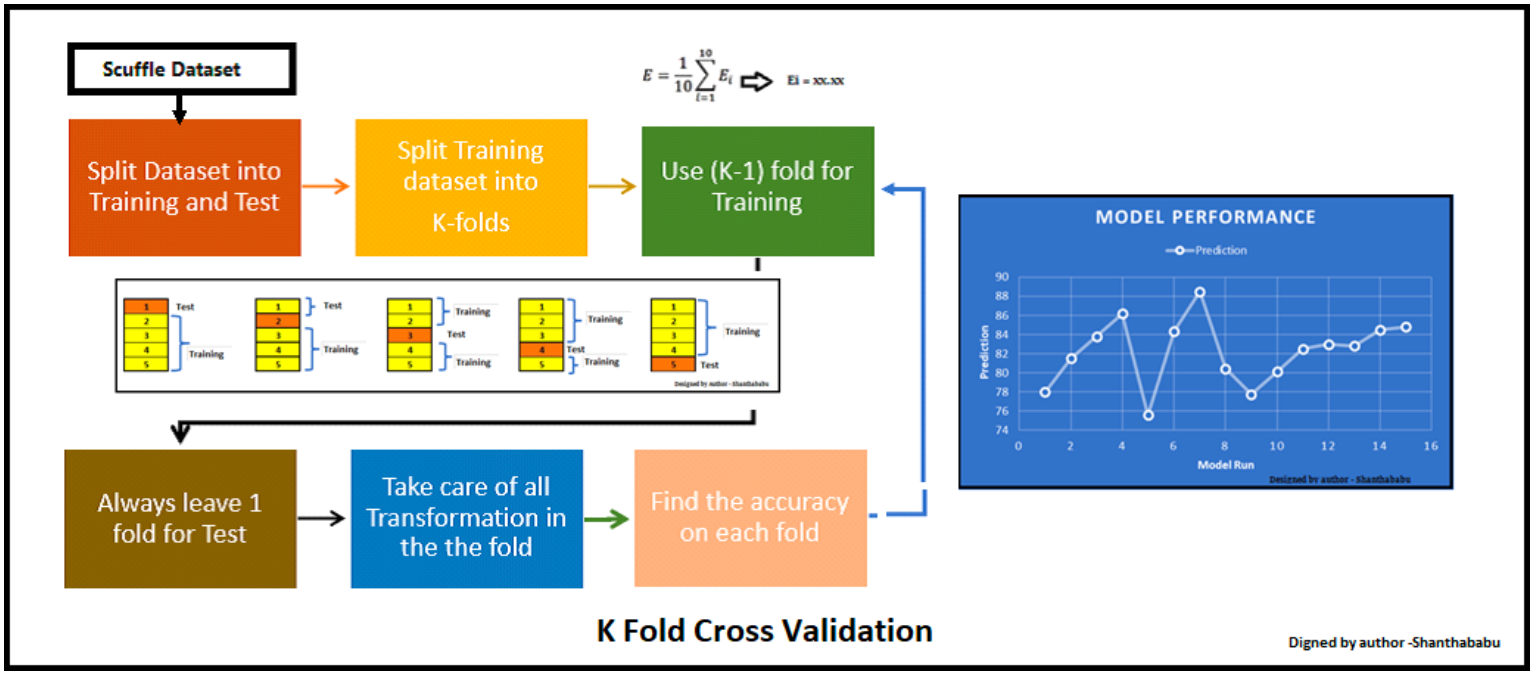
(K)Ten Fold Cross Validation

- Divide the dataset into k number of roughly equal-sized subsets (known as folds)
- Then, the model is trained on 'k-1' of these folds and tested on the remaining one.
- This process is **repeated 'k' times**, with each fold serving as the test set exactly once.

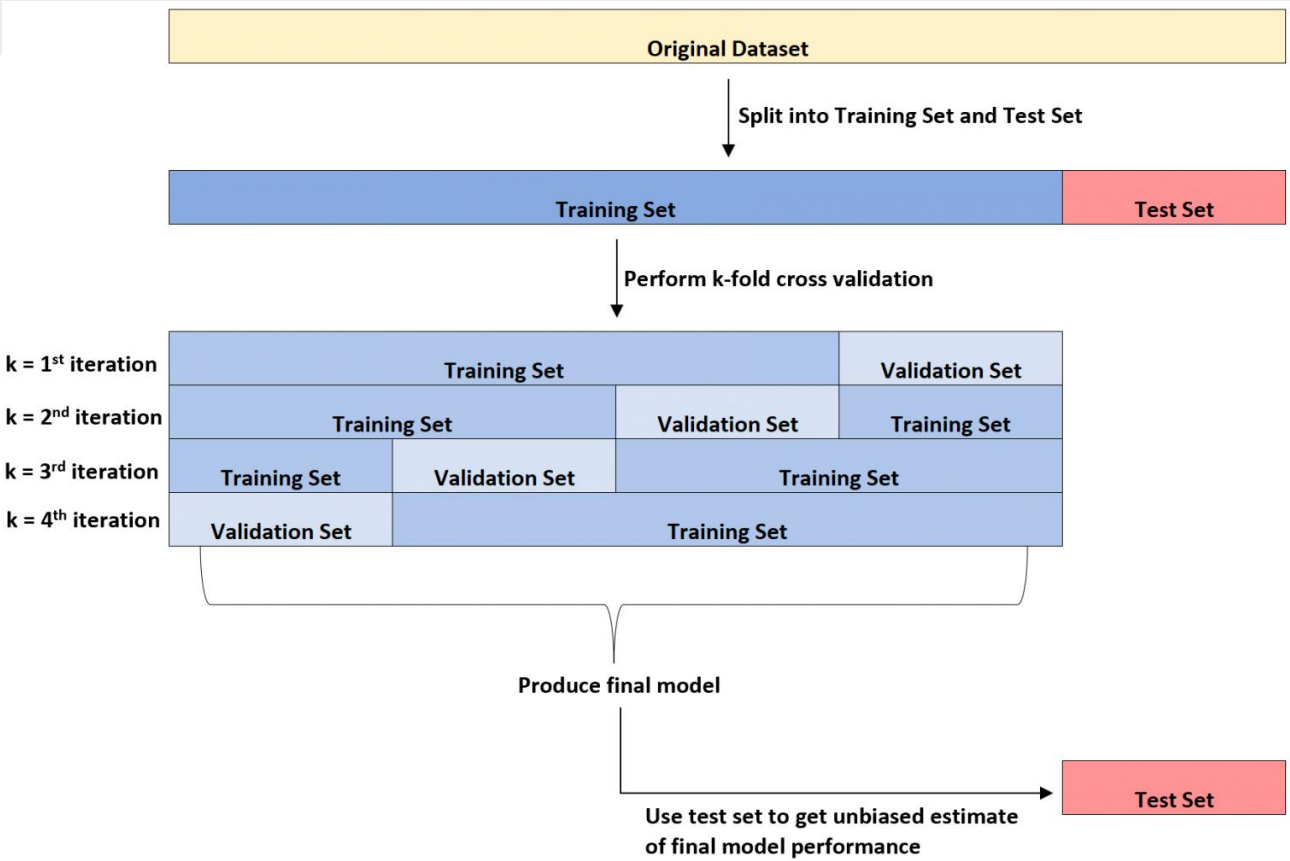


Example: Ten Fold Cross Validation

- Break the data into ten train/test splits
- Train on those, report test accuracy
- Performance metrics like test accuracy are averaged across all iterations to obtain a final estimate.

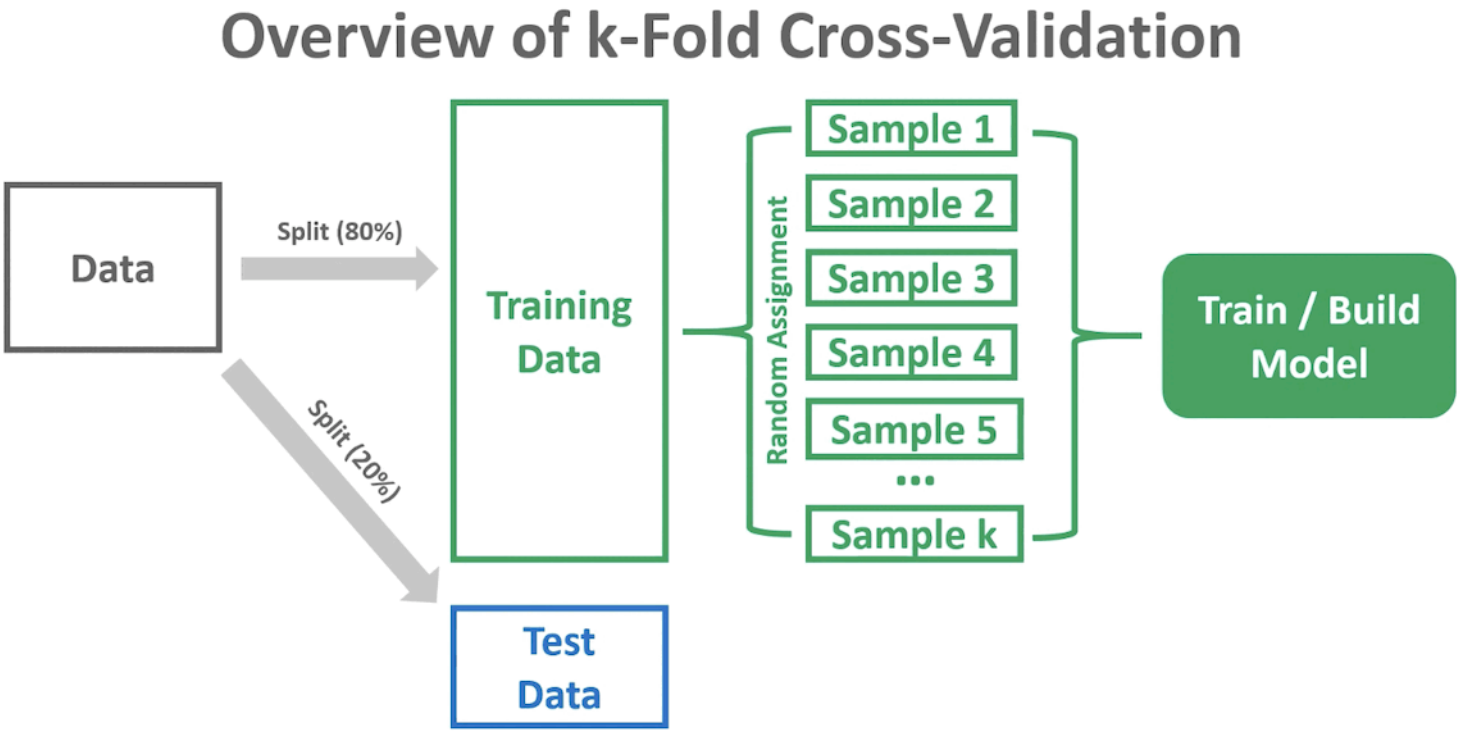

Scuffle Dataset
$E = \frac{1}{10}\sum_{i=1}^{10} E_i$ ⇨ Ei = xx.xx
Split Dataset into Training and Test
Split Training dataset into K-folds
Use (K-1) fold for Training
Test
Training
Always leave 1 fold for Test
Take care of all Transformation in the the fold
Find the accuracy on each fold
MODEL PERFORMANCE
Prediction
Model Run
Designed by author - Shanthababu
K Fold Cross Validation
Digned by author -Shanthababu

Ref: https://www.statology.org/validation-set-vs-test-set/



Example of Training and Validation (Cross Validation K=4 Fold in this case) and Testing Dataset

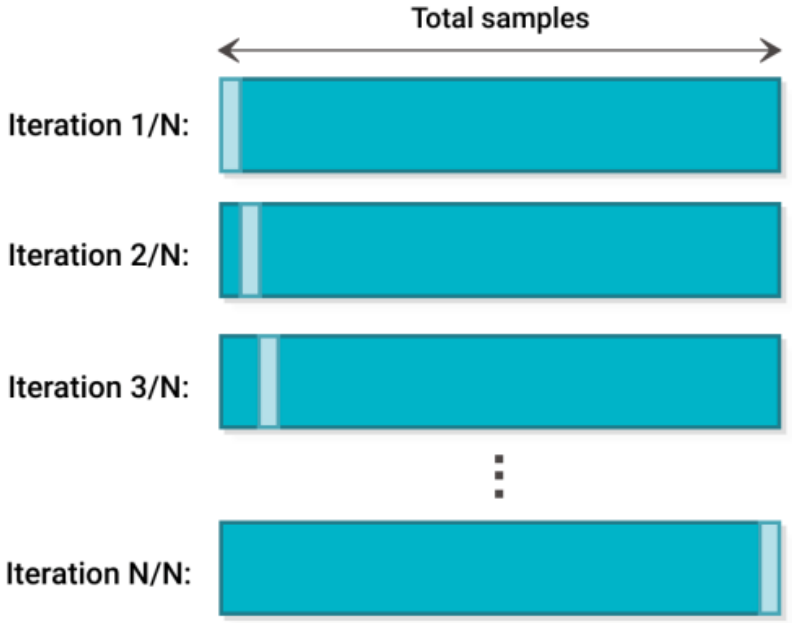
Example: K-Fold Cross Validation



Leave One Out Cross Validation (LOOCV)

When every ounce of training data counts

- Each data point is held out as the validation set while the model is trained on the remaining data.
- This process is repeated for each data point.
    - Particularly useful for small datasets but can be computationally expensive for larger ones and can be slow, but provides a very reliable estimate of a model's performance.



Leave One Out Cross Validation (LOOCV)

- **Split a dataset into a training set and a testing set**, using all but one observation as part of the training set.
  - The model is trained on the remaining 'n-1' data points, where 'n' is the total number of data points.
- Only leave one observation “out” from the training set. This is where the method gets the name “leave-one-out” cross-validation.
- This process is repeated 'n' times, with each data point being used as the test set once.

Evaluation

Using some evaluation metrics

Accuracy

The simplest way to check how well our model is doing is to look at it's **accuracy.**

$$accuracy = \frac{correct}{total}$$

100% accuracy is perfect, 0% accuracy is completely wrong.

Example: Accuracy

Question: "If 90 out of 100 predictions are correct for spam detection in an email classification model, what is the model's accuracy?"

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

= 90/100 * 100% = 0.9 * 100= 90%

However, 'Accuracy' cannot always be used as the sole metric to evaluate model performance

Consider this scenario:

Class Imbalance

Imagine a dataset with 90% of the negative class and 10% of the positive class--is 80% accuracy impressive here? What about 90%?

Class Imbalance: A serious problem

Class imbalance is a serious issue. Class imbalance happens when an overwhelming majority if your data is a single class.

Situations With Class Imbalance

Class Imbalance Issue: Occurs when one class dominates the data, making one group significantly larger than others.

Examples of Imbalance:

- **Fraud:** Most transactions are not fraudulent.
- **Disease:** The majority don't have cancer.
- **Purchases:** Most people don't buy a specific product.

Common Scenario: Typically involves **a rare positive signal amidst negatives**, which is a prevalent occurrence.

Limitation of “Accuracy” Metric

Accuracy may not show the full model performance, especially when classes are imbalanced

Issue with Imbalanced Data: Accuracy can mislead in imbalanced datasets where one class dominates.

- *A high accuracy might be achieved* by simply predicting the majority class, even if the model is performing poorly on the minority class

“Accuracy is utterly useless if the class distribution in your data set is skewed”

Limitation of “Accuracy” Metric

Neglecting Error Types: doesn't differentiate between false positives and false negatives, treating all errors or misclassification equally.

So more alternative evaluation metrics are needed. NEXT: Confusion Matrix

Confusion Matrix

A confusion matrix is a matrix that **summarizes the performance of a machine learning model on a set of test data.**

- shows true positives, true negatives, false positives, and false negatives, offering insights into the model's accuracy and prediction errors across classes.

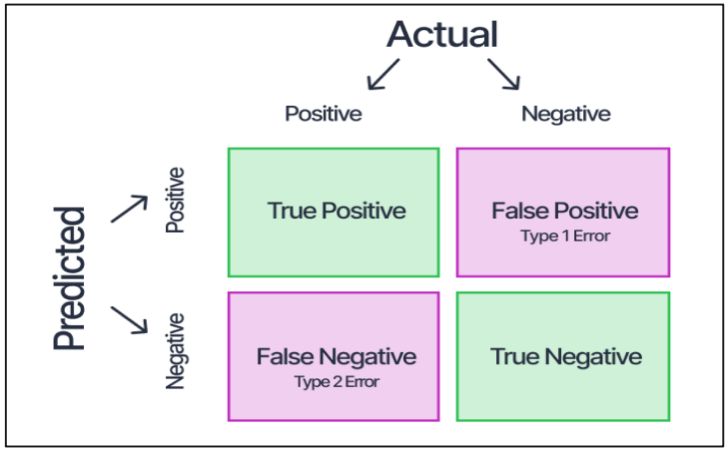
Confusion Matrix

Instead of looking at our model accuracy, let's look at our true positive, false positive, true negative, and false negative rate:

	Positive	Negative
Classified Positive	80%	0%
Classified Negative	10%	10%

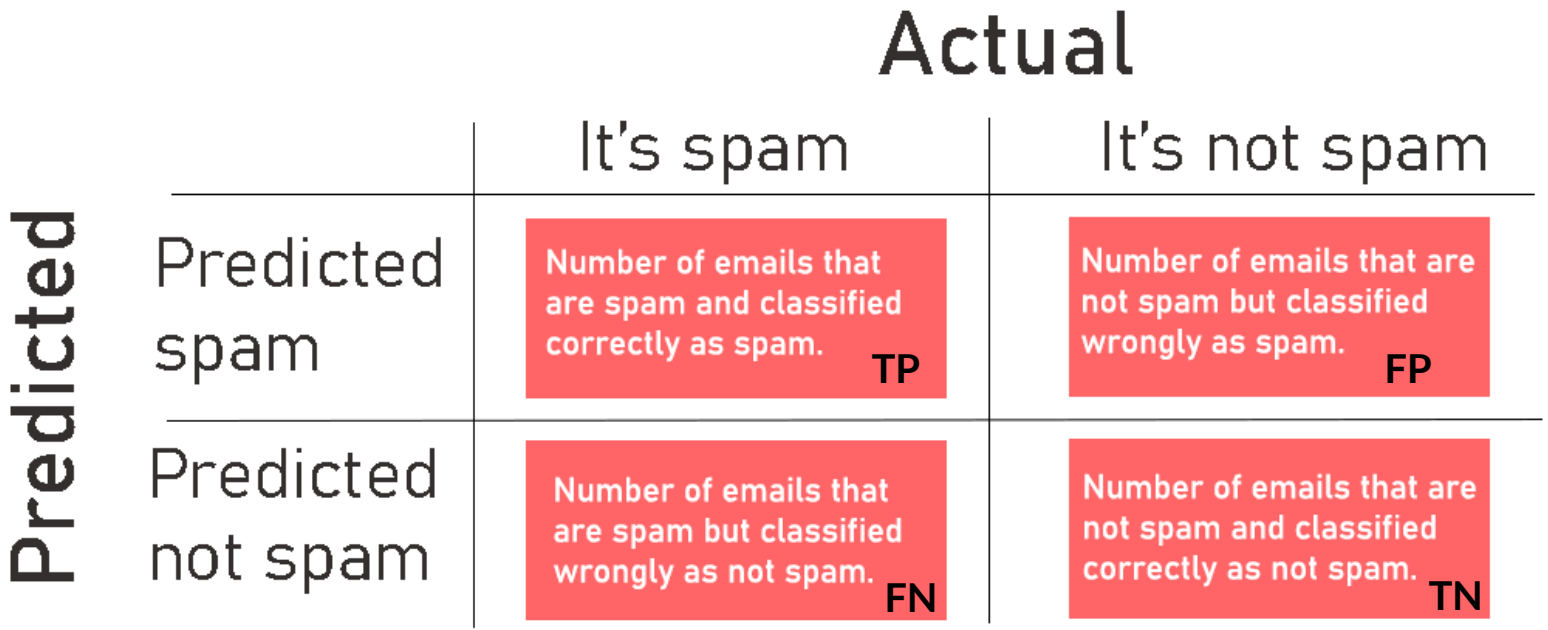
Terminologies used in Confusion Matrix

Instead of looking at our model accuracy, let's look at our true positive, false positive, true negative, and false negative rate: (**TP,FP,FN,TN**)



	Positive	Negative
Classified Positive	80%	0%
Classified Negative	10%	10%

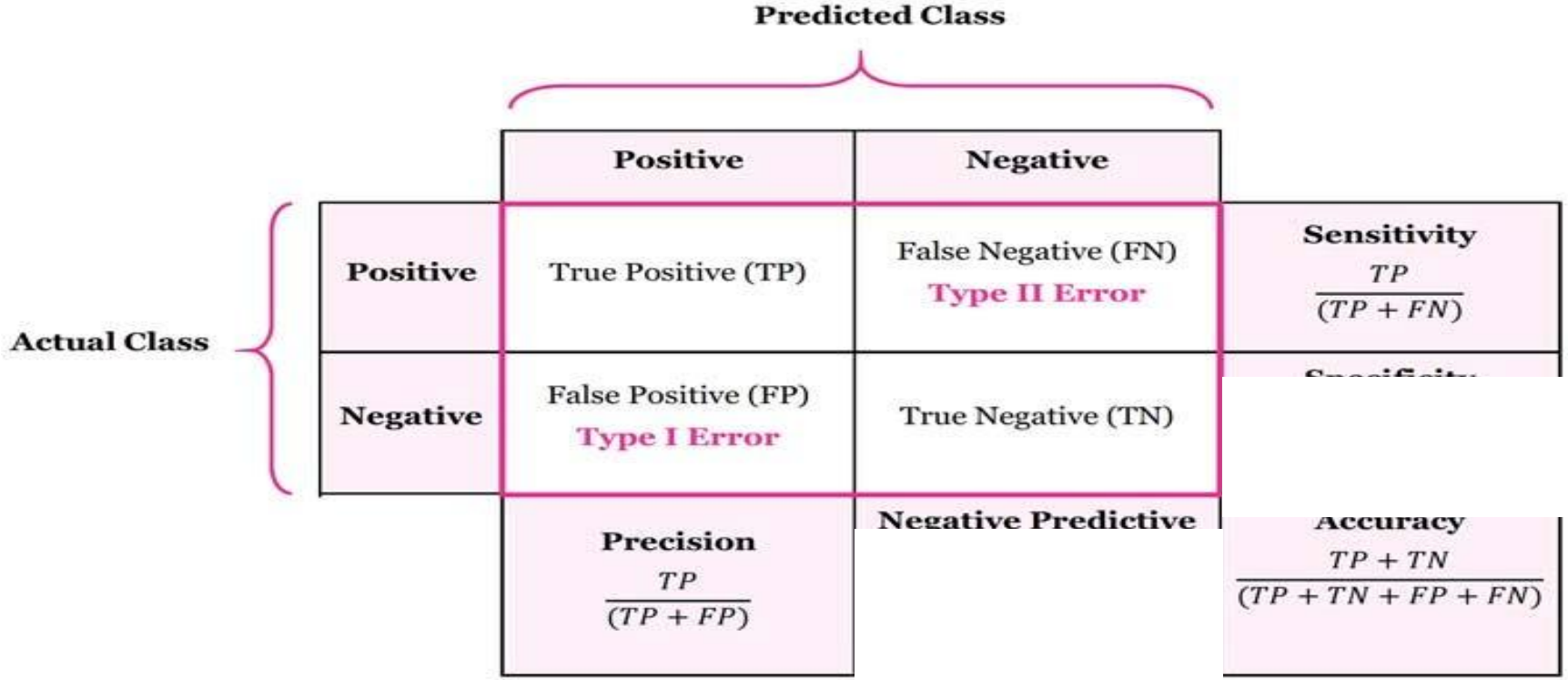
Example: Confusion Matrix



The confusion matrix allows to calculate various evaluation metrics

- Accuracy
- Precision
- Recall (Sensitivity or True Positive Rate)
- F1-Score

The confusion matrix allows to calculate various evaluation metrics

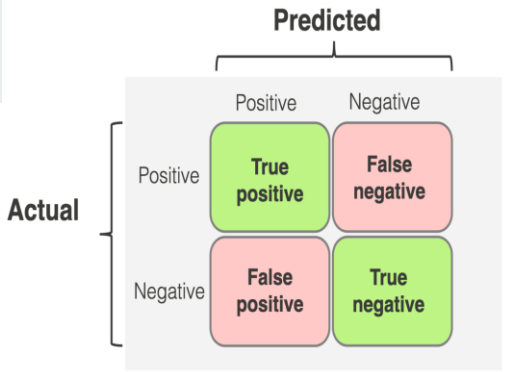


Precision

How much we are **precise (accurate)** in identifying positive cases

Evaluation Metric: **Precision**

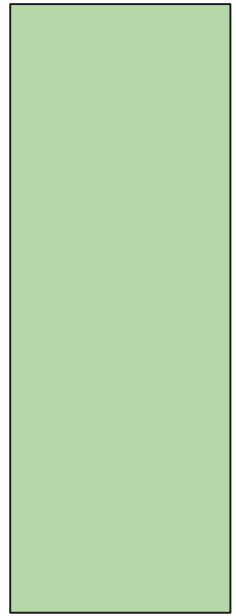
How much we are **precise (accurate)** in identifying positive cases



Out of all positives been predicted, how many are actually positive?

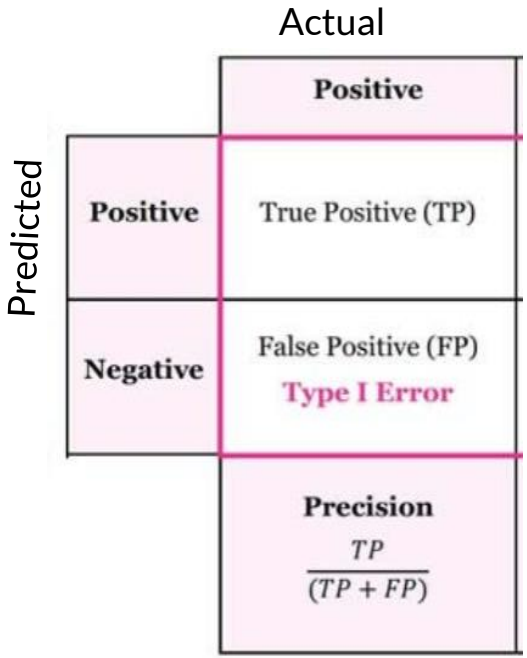
$$precision = \frac{tp}{tp + fp}$$

Everything we said was positive.

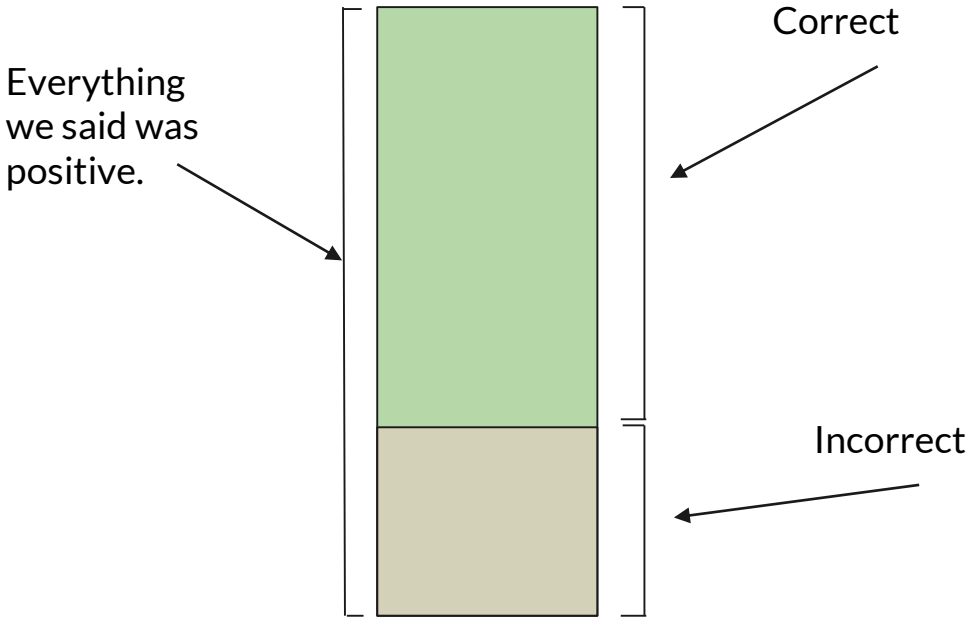


Precision:

Precision measures **how many of the things that we classified as positive were actually positive.**

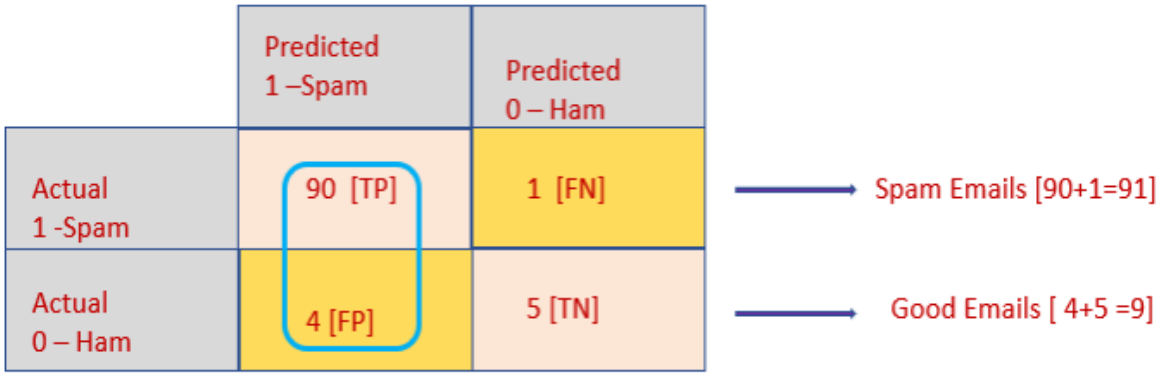


$$precision = \frac{tp}{tp + fp}$$



"Out of all the instances the model predicted as positive, **how many were actually correct?**"

Example: Precision



$$\text{Precision} = \frac{90}{90+4} = 0.95$$

Precision = 95%

- Measured on a scale of 0 to 1 or as a percentage.
- **Higher precision is better**.
- value 1.0 indicates the model is always correct when predicting the target class and *never makes a mistake*.

When to Use Precision

We use precision when **we only care about being correct about the things we identify as positive.**

- Google does not care if it turns away 1000 good engineers; Google just wants to make sure the ones it DOES hire are good

precision answers the question: how often the positive predictions are correct?

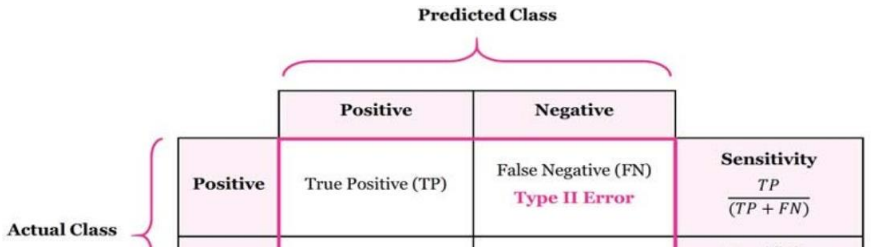
Recall

How much we are good at detecting/ identifying all actual positive cases?

Recall (Sensitivity or True Positive Rate)

Recall is a measure of h**ow many positives your model is able to recall from the data.**

$$Recall = \frac{tp}{tp + fn}$$



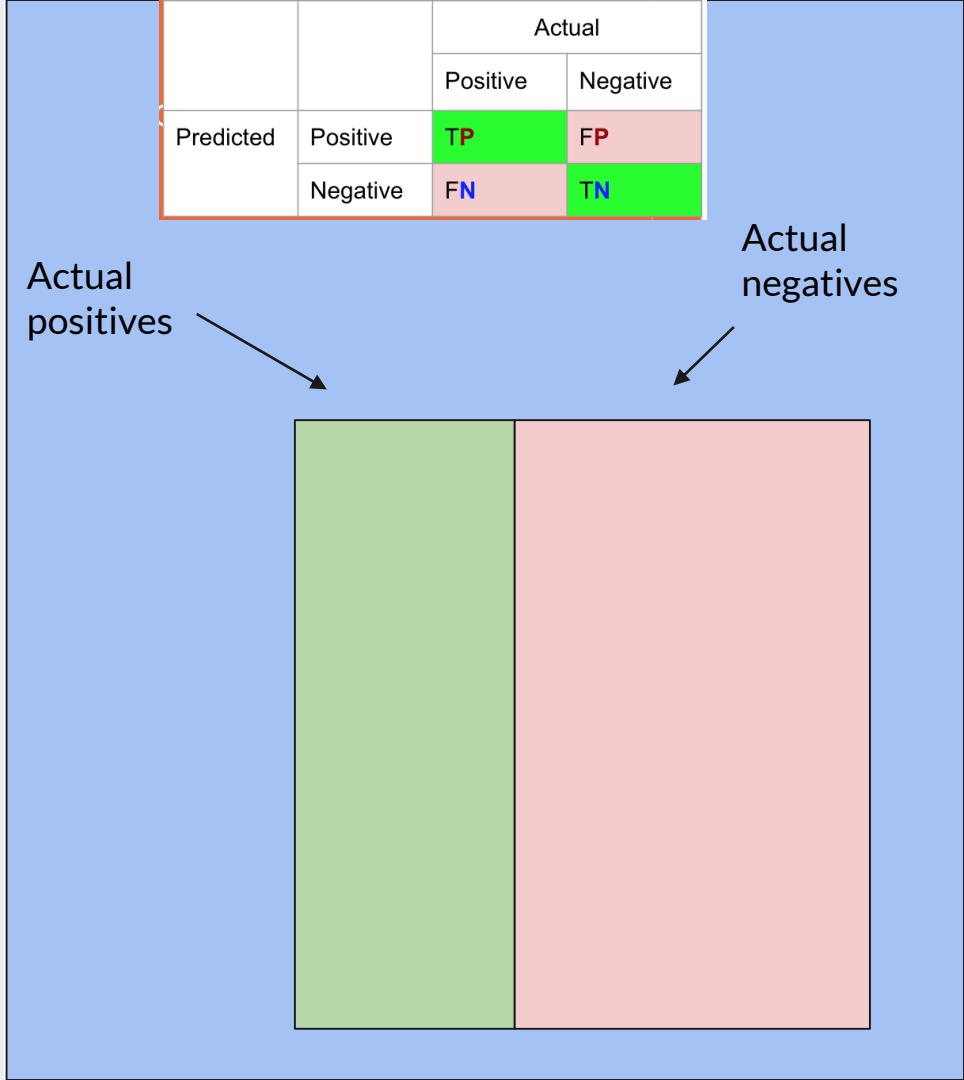
Recall: The proportion of true positives to the total actual positives (TP / (TP + FN)). It measures the model's ability to ***identify all positive instances***.

Recall

Recall is a measure of how many of our positive class that we ***missed (actually positive but we predicted as negative)***.

$$Recall = \frac{tp}{tp + fn}$$

		Actual	
		Positive	Negative
Predicted	Positive	TP	FP
	Negative	FN	TN



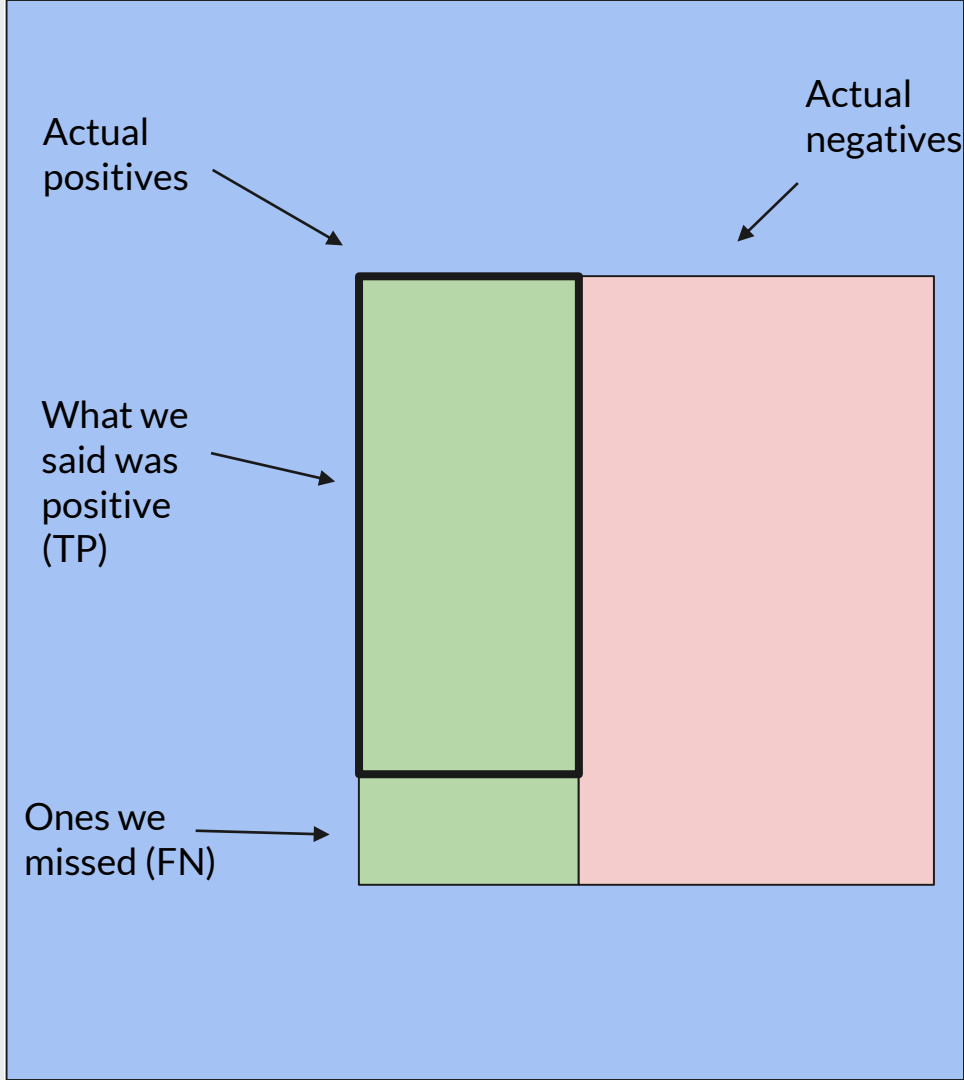
Recall

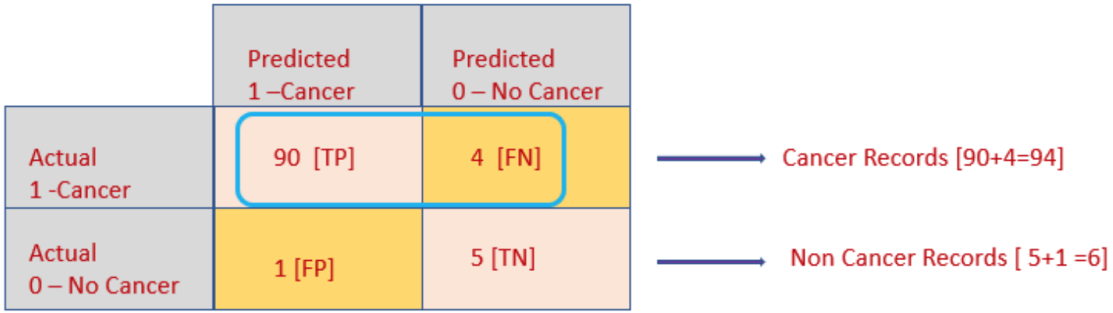
		Actual	
		Positive	Negative
Predicted	Positive	TP	FP
	Negative	FN	TN

Recall is a measure of how many of our positive class that we ***missed (actually positive but we predicted as negative)***.

$$Recall = \frac{tp}{tp + fn}$$

“Out of all the actual positive cases, how many did the model correctly identify?”





$$Recall = \frac{90}{90 + 4} = 0.95$$

Recall = 95%

- Measured on a scale of 0 to 1 or as a percentage.
- **Higher recall is preferable**.
- value of 1.0 means the model captures all instances of the target class and never misses it in predictions.

When to use Recall:

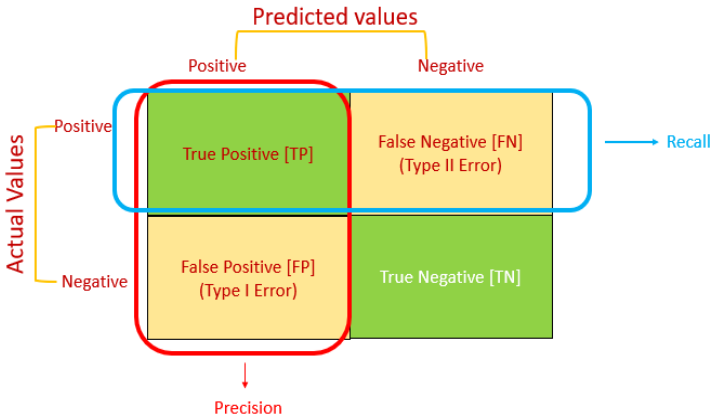
We'd use recall when we want to make sure ***we don't miss anything***.

- For example identifying people contagious with a deadly super plague

In-class Discussion

In the given scenarios, which metric (precision or recall) holds more practical importance or application?

Precision vs. Recall



- Testing for cancer
- Our legal system
- Fraud alerts
- Loans

Think about:
Precision: Aims to be right when it says something is positive (minimize false positives).
Recall: Aims to not miss anything that's actually positive (minimize false negatives).

In-class Discussion

Precision vs. Recall

- Testing for cancer
  - Recall: Don't want to be wrong (all our cancer records should be predicted correctly), and we can always do further tests
- Our legal system
  - Depends (precision if aim to reduce false accusations or arrests; recall if priority is to capture all crimes, even if it requires extensive investigations).
- Fraud alerts
  - Recall: Better to deny some good transactions than pay for the fraudulent ones
- Loans
  - Precision: We only want to loan money to people who will pay it back

What about our confidence?

Confidence is how sure a model is about its predictions.

- Most ML algorithms can tell you how confident they are in an answer
- You may want to treat 51% confident of the positive class and 99% confident differently.

What if Confidence Matters?

Log loss measures how accurate a model's predictions are, especially its confidence in those predictions.

- N =total number of data points
- M=outcomes or classes (usually 0 and 1).
- Xij→ whether the actual outcome j is the correct one for data point i.
- Pij→ predicted probability that data point i belongs to class j.

Log loss: A measure of accuracy that penalizes overconfidence.

$$LogLoss = \frac{-1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} x_{ij} * \ log(p_{ij})$$

A lower log loss indicates better model performance, with values closer to 0 being ideal.

Log loss helps us understand how well the model's confidence matches reality.

Sometimes, we may wish for a model to have as few FPs and FNs as possible → want to maximise both precision and recall.

- In practice, it is not possible to maximise both at the same time because of the trade-off between precision and recall.

  “Increasing precision will decrease recall, and vice versa.”

We can use “F1- Score” - combines precision and recall into a single number

In some cases, we aim for a model to minimize both false positives (FPs) and false negatives (FNs). This means we want to maximize both precision and recall, giving equal importance to both metrics.

- E.g: When identifying harmful content, the goal is to ensure that flagged content is **both accurate (high precision)** and that no inappropriate **content is overlooked (high recall)**.

- In practice, it is not possible to maximise both at the same time because of the trade-off between precision and recall.

  “Increasing precision will decrease recall, and vice versa.”

We can use “F1- Score” - combines precision and recall into a single number

F1 Score

F1 score metric is used when you seek a balance between precision and recall.

$$\text{F1 Score} = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Harmonic mean of precision and recall

- Used when both are important

F1-score ranges between 0 and 1. The clos[illegible] to 1, the better the model.

Summary

Precision: Focuses on minimizing false positives in predictions.

Recall: Focuses on minimizing false negatives in predictions.

Log Loss: Evaluates the alignment between predicted probabilities and true class probabilities, aiming for lower values closer to 0.

F1 Score: Balances both precision and recall in a single metric, useful for scenarios where false positives and false negatives are equally important.

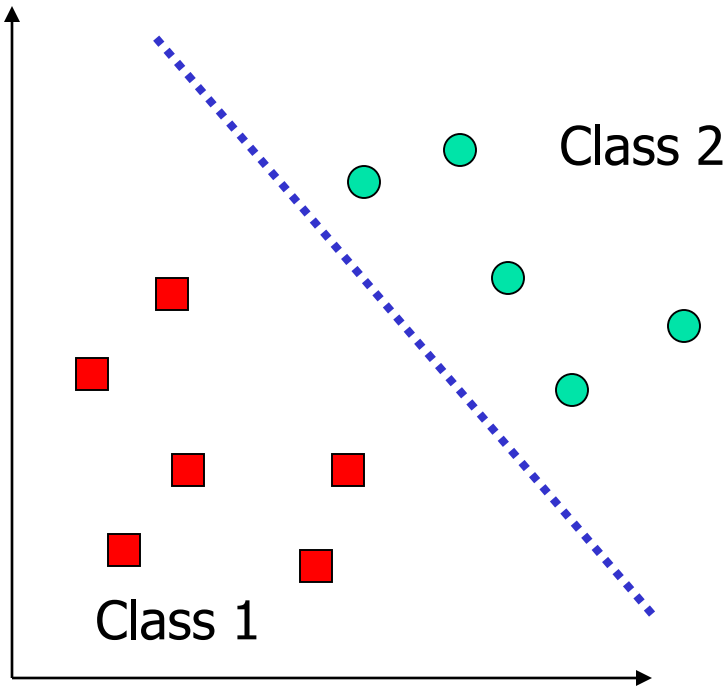
History of SVM

- SVM is related to statistical learning theory [3]
- SVM was first introduced in 1992 [1]
- SVM becomes popular because of its success in handwritten digit recognition
  - 1.1% test error rate for SVM. This is the same as the error rates of a carefully constructed neural network
- SVM is now regarded as an important example of "kernel methods", one of the key area in machine learning

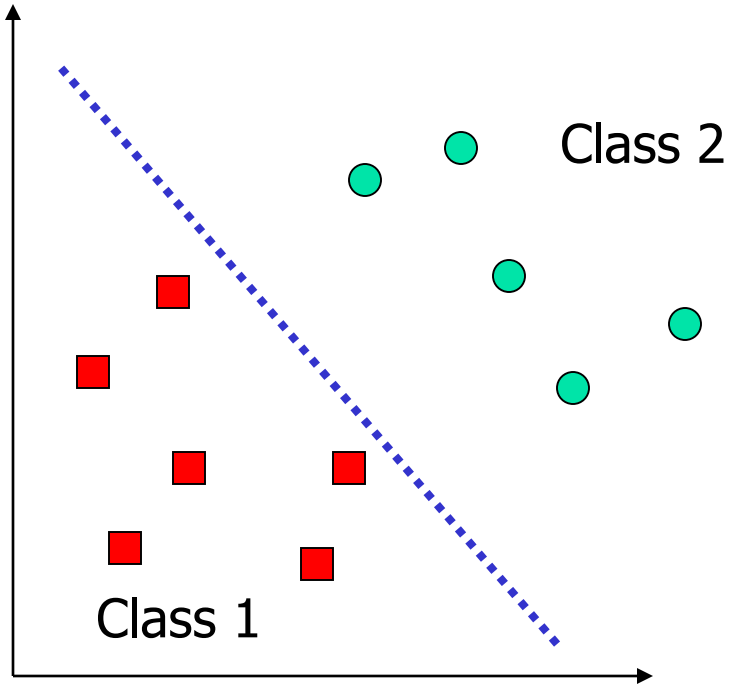
[1] B.E. Boser *et al*. A Training Algorithm for Optimal Margin Classifiers. Proceedings of the Fifth Annual Workshop on Computational Learning Theory 5 144-152, Pittsburgh, 1992.
[2] L. Bottou *et al*. Comparison of classifier methods: a case study in handwritten digit recognition. Proceedings of the 12th IAPR International Conference on Pattern Recognition, vol. 2, pp. 77-82.
[3] V. Vapnik. The Nature of Statistical Learning Theory. 2nd edition, Springer, 1999.

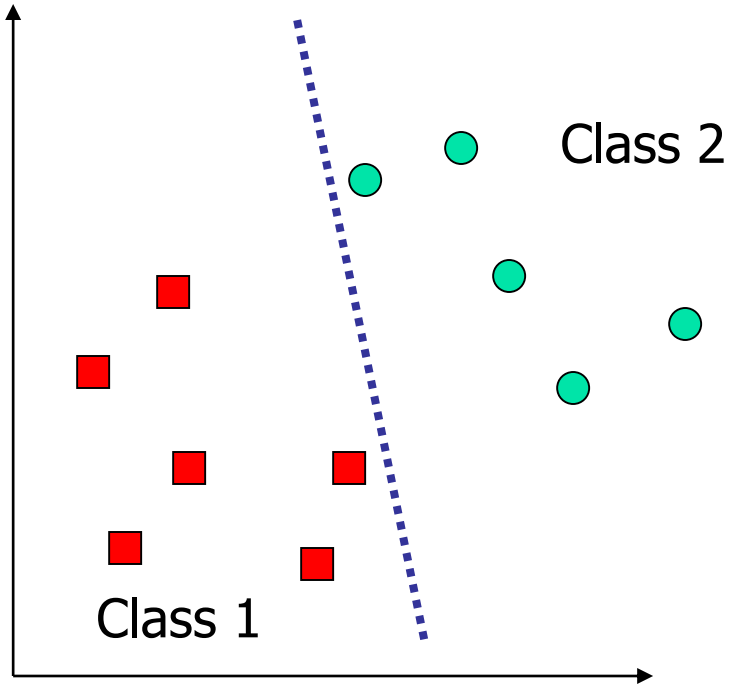
What is a good Decision Boundary?

- Consider a two-class, linearly separable classification problem
- Many decision boundaries!
  - The Perceptron algorithm can be used to find such a boundary
  - Different algorithms have been proposed
- Are all decision boundaries equally good?



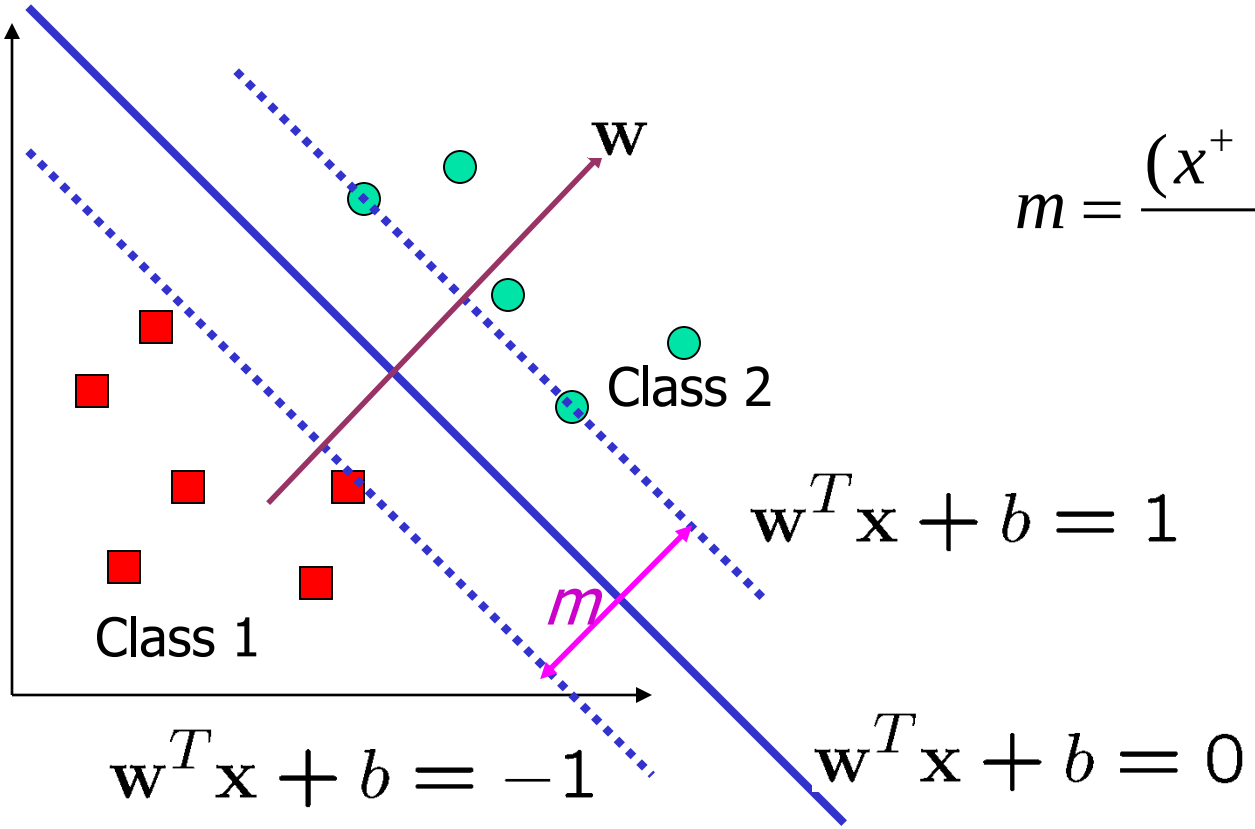
Examples of Bad Decision Boundaries





Large-margin Decision Boundary

- The decision boundary should be as far away from the data of both classes as possible
  - We should maximize the margin, *m*
  - Distance between the origin and the line $\mathbf{w}^t\mathbf{x}=k$ is $k/\|\mathbf{w}\|$



$$m = \frac{(x^+ - x^-)\cdot w}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$

Finding the Decision Boundary

- Let $\{x_1, ..., x_n\}$ be our data set and let $y_i \in \{1,-1\}$ be the class label of x_i
- The decision boundary should classify all points correctly

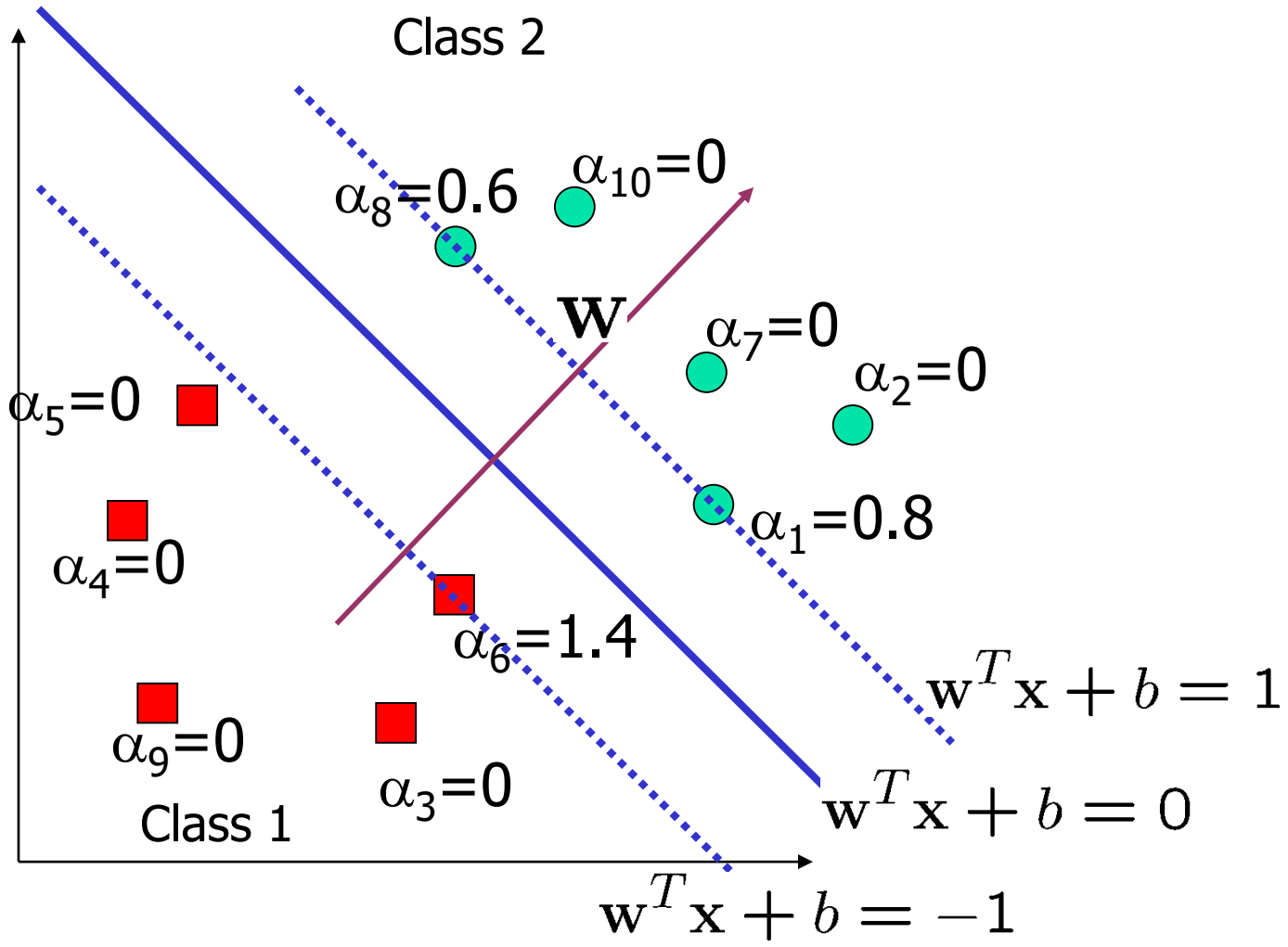
$$\Rightarrow \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \qquad \forall i$$

- The decision boundary can be found by solving the following constrained optimization problem

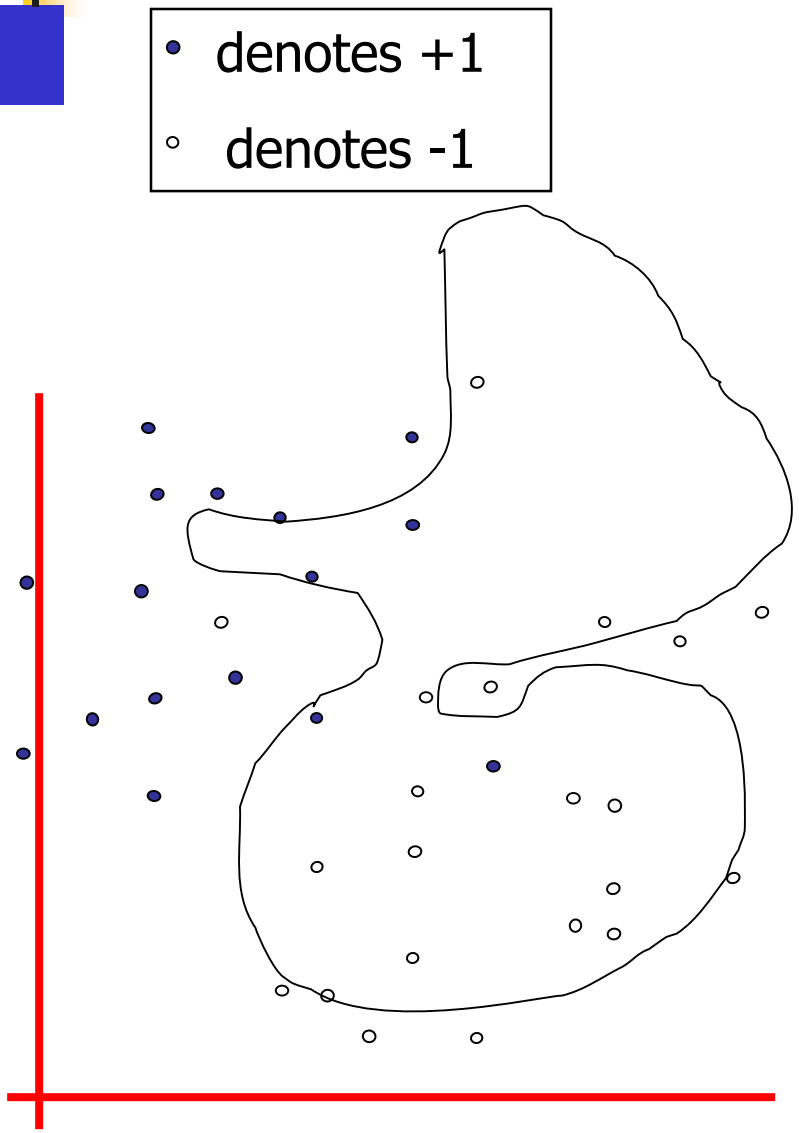
$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$
$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \forall i$$

- This is a constrained optimization problem. Solving it requires some new tools
  - Feel free to ignore the following several slides; what is important is the constrained optimization problem above

A Geometrical Interpretation



Dataset with Noise



- Hard Margin: So far we require all data points be classified correctly

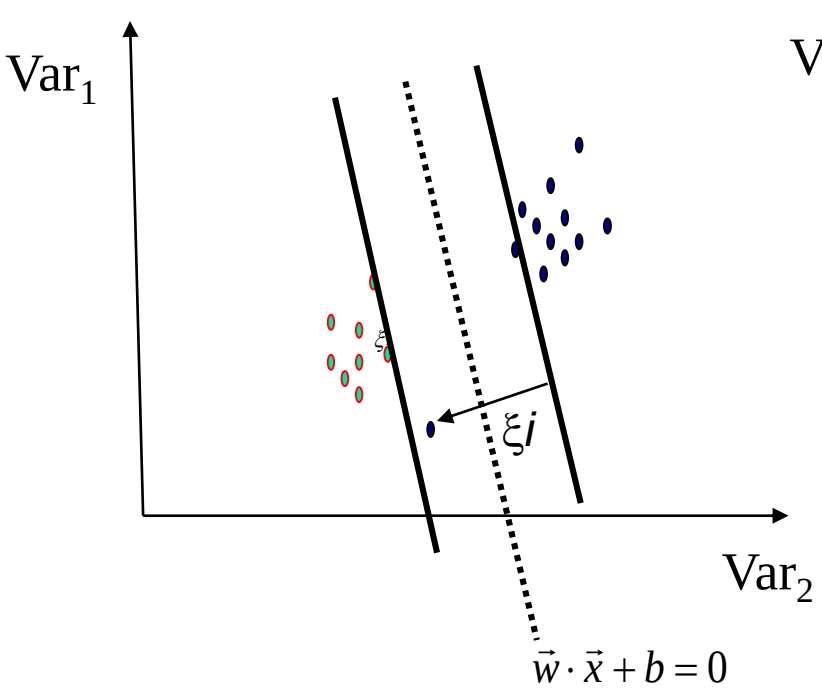
  - No training error

- What if the training set is noisy?

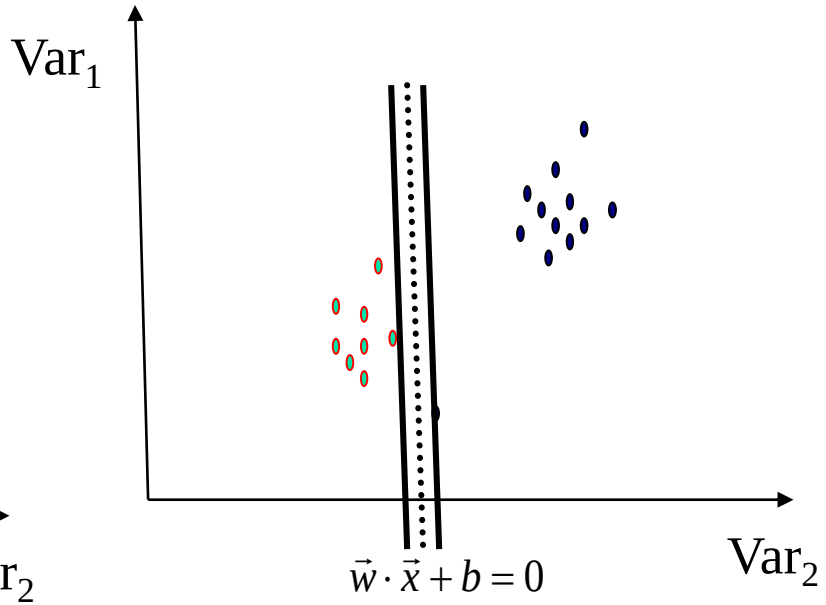
  - Solution 1: use very powerful kernels

OVERFITTING!

Robustness of Soft vs Hard Margin SVMs



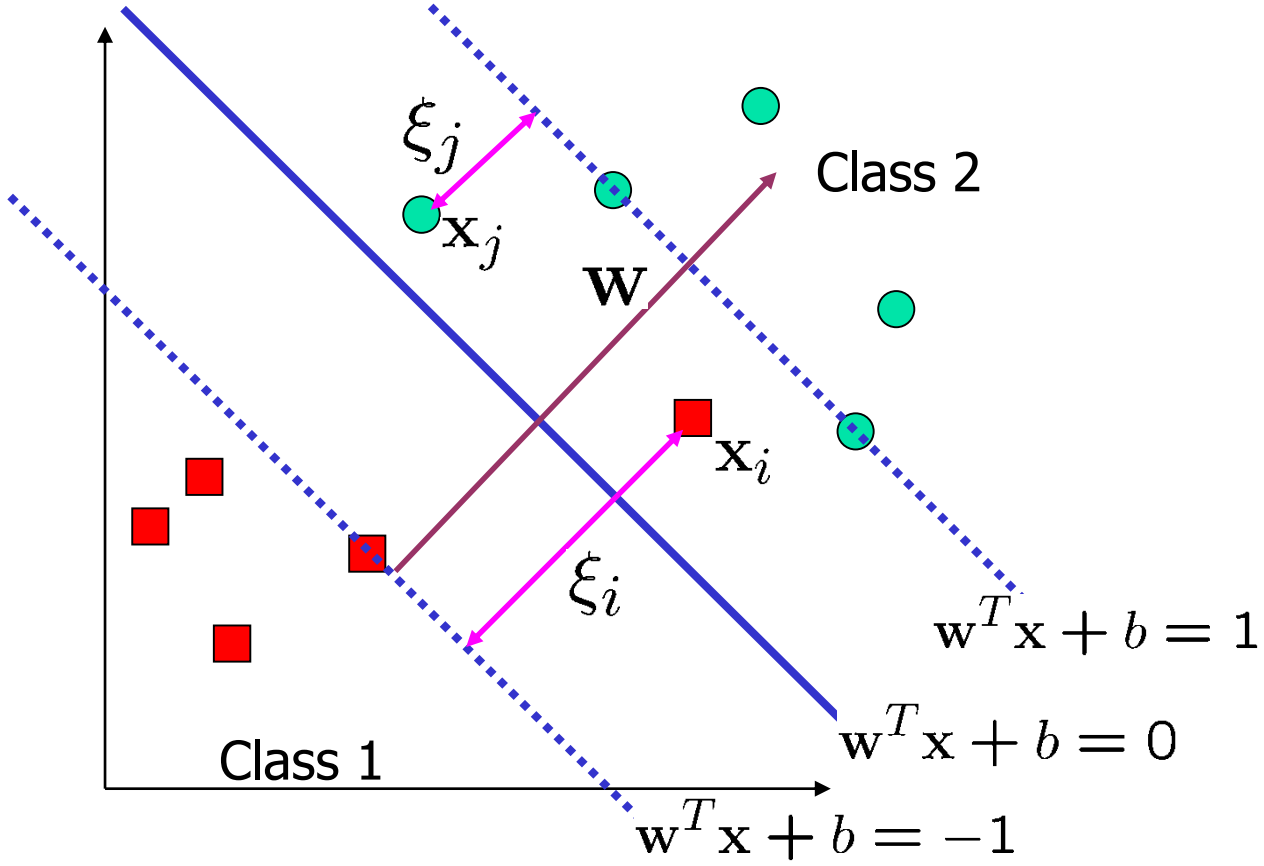
Soft Margin SVM



Hard Margin SVM

Non-linearly Separable Problems

- We allow "error" ξ_i in classification; it is based on the output of the discriminant function $\mathbf{w}^T\mathbf{x}+b$
- ξ_i approximates the number of misclassified samples



Soft Margin Hyperplane

- If we minimize $\sum_i \xi_i$, ξ_i can be computed by

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq 1 - \xi_i & y_i = 1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 & \forall i \end{cases}$$

  - ξ_i are "slack variables" in optimization
  - Note that ξ_i=0 if there is no error for $\mathbf{x}_i$
  - ξ_i is an upper bound of the number of errors
- We want to minimize $\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$
  - C : tradeoff parameter between error and margin
- The optimization problem becomes

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$$
$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$