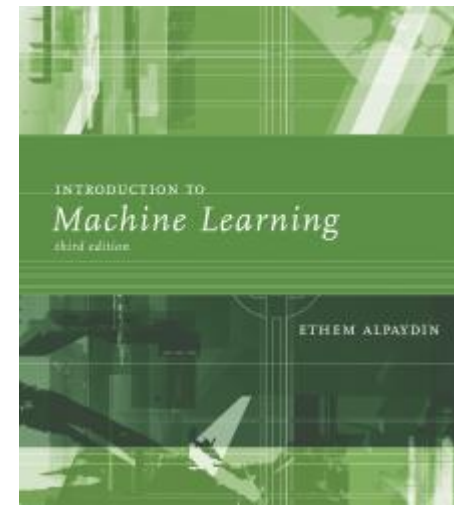


# Advice for applying Machine Learning



Lecture notes by Ethem Alpaydın  
Introduction to Machine Learning (Boğaziçi Üniversitesi)

Lecture notes by Kevyn Collins-Thompson  
Applied Machine Learning (Coursera)

Lecture notes by Andrew NG  
Machine Learning by Stanford University (Coursera)

# Model Selection & Generalization

- How well a model trained on the training set predicts the right output for new generalization instances is called *generalization*.
- **Overfitting:**  $\mathcal{H}$  is more complex than the function
- **Underfitting:**  $\mathcal{H}$  is less complex than the function

# Triple Trade-Off

- In all learning algorithms that are trained from example data, there is a trade-off between three factors:
  - the complexity of the hypothesis we fit to data, namely, the capacity of the hypothesis class,
  - the amount of training data, and
  - the generalization error on new examples.

# Triple Trade-Off

- As the complexity of the model class  $H$  increases, the generalization error decreases first and then starts to increase.
- As the amount of training data increases, the generalization error decreases.

# Bias/Variance Dilemma

- Whenever we discuss model prediction, it's important to understand prediction errors (bias and variance).
- There is a tradeoff between a model's ability to minimize bias and variance.
- Gaining a proper understanding of these errors would help us not only to build accurate models but also to avoid the mistake of overfitting and underfitting.

# Bias/Variance Dilemma

- **Bias** is the difference between the average prediction of our model and the correct value which we are trying to predict.
- Model with *high bias* pays very little attention to the training data and oversimplifies the model.
- It always leads to high error on training and test data.

# Bias/Variance Dilemma

- **Variance** is the variability of model prediction for a given data point or a value which tells us spread of our data.
- Model with *high variance* pays a lot of attention to training data and does not generalize on the data which it hasn't seen before.
- As a result, such models perform very well on training data but has high error rates on test data.

# Bias/Variance Dilemma

- Mathematically

Unknown parameter  $\theta$

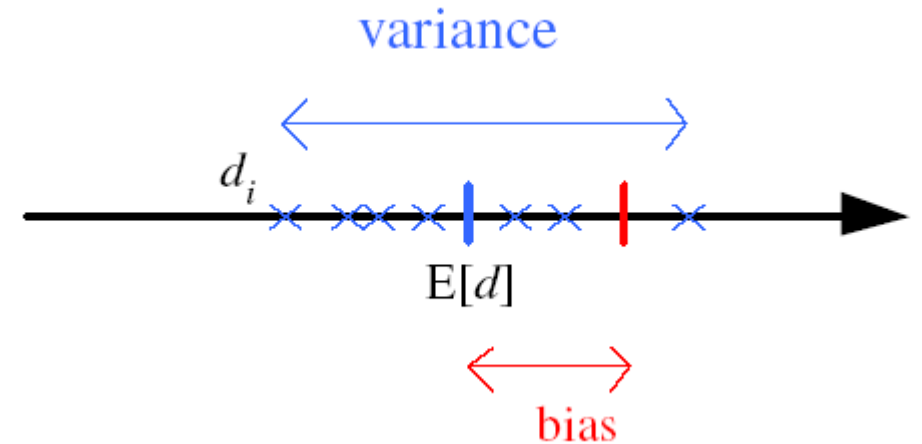
Estimator  $d_i = d(X_i)$  on sample  $X_i$

Bias:  $b_{\theta}(d) = E[d] - \theta$

Variance:  $E[(d - E[d])^2]$

Mean square error:

$$\begin{aligned} r(d, \theta) &= E[(d - \theta)^2] \\ &= (E[d] - \theta)^2 + E[(d - E[d])^2] \\ &= \text{Bias}^2 + \text{Variance} \end{aligned}$$





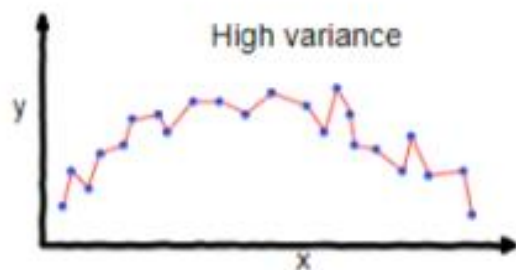
# Bias/Variance Dilemma

- In supervised learning, **underfitting** happens when a model unable to capture the underlying pattern of the data.
- These models usually have high bias and low variance.
- It happens when we have very less amount of data to build an accurate model or when we try to build a linear model with a nonlinear data.
- Also, these kind of models are very simple to capture the complex patterns in data like Linear and logistic regression.

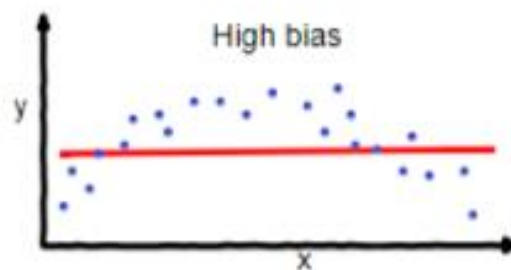
# Bias/Variance Dilemma

- In supervised learning, **overfitting** happens when our model captures the noise along with the underlying pattern in data.
- It happens when we train our model a lot over noisy dataset.
- These models have low bias and high variance.
- These models are very complex like Decision trees which are prone to overfitting.

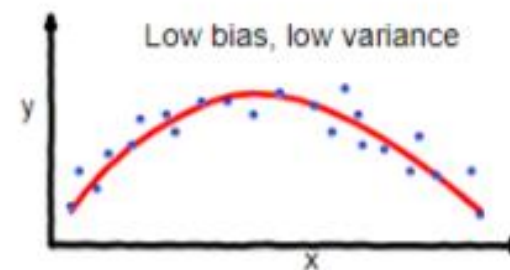
# Bias/Variance Dilemma



**overfitting**



**underfitting**

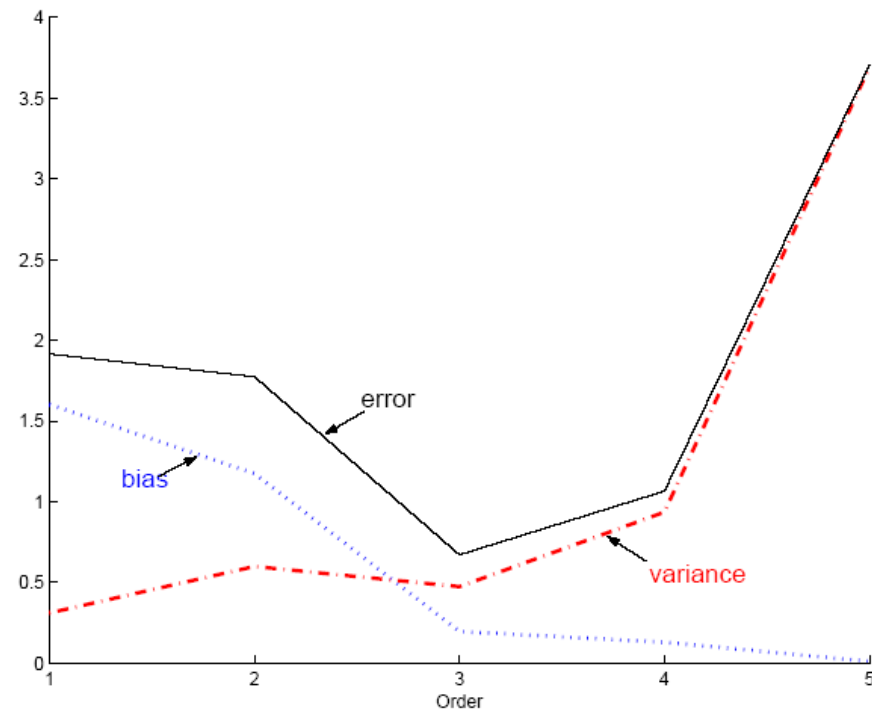


**Good balance**

# Bias/Variance Dilemma

- As the order of the polynomial increases, small changes in the dataset cause a greater change in the fitted polynomials; thus variance increases.
- But a complex model on the average allows a better fit to the underlying function; thus bias decreases
- This is called the *bias/variance dilemma* and is true for any machine learning system and not only for polynomial regression

# Bias/Variance Dilemma



- Order 1 has the smallest variance.
- Order 5 has the smallest bias.
- As the order is increased, bias decreases but variance increases.
- Order 3 has the minimum error.

# Model Selection

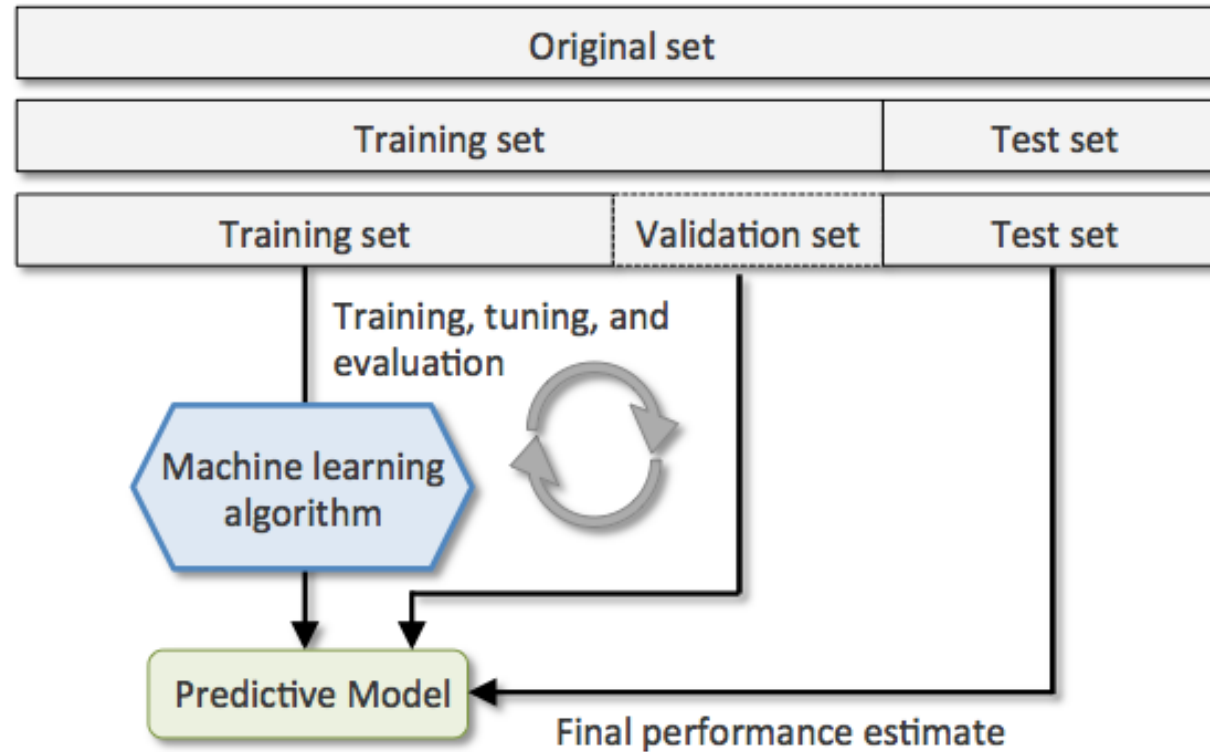
1. Split data
  - Training set (model building)
  - Validation set (model selection)
  - Test set (final evaluation)
2. Learn parameter from training data (minimizing training error)
3. Compute cross validation set error.
4. Select the model with lowest cross validation set error.
5. Estimate generalization error for test set

# Train / Validation / Test Data

	<b>Purpose</b>	<b>Yield</b>	<b>Used for Model training</b>	<b>Used for Parameter tuning</b>
<b>Train Data</b>	To learn patterns from the data.	A model that makes near-expected predictions	Yes	Yes
<b>Validation Data</b>	To understand model behaviour and generalizability on unseen data.	Insights on how to tune your model.	No	Yes
<b>Test Data</b>	To understand how the model would perform in real world scenario.	A completely unbiased estimate of model performance.	No	No

<https://medium.com/datadriveninvestor/data-science-essentials-why-train-validation-test-data-b7f7d472dc1f>

# Train / Validation / Test Data



<https://medium.com/datadriveninvestor/data-science-essentials-why-train-validation-test-data-b7f7d472dc1f>



# Example: Linear regression with regularization

Choosing the regularization parameter  $\lambda$ :

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2$$

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_{\theta}(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_{\theta}(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

# Example: Linear regression with regularization

## Choosing the regularization parameter $\lambda$

- Create a list of lambdas  $\lambda \in \{0, 0.01, 0.02, 0.04, 0.08, 0.16, \dots, 5.12, 10.24\}$
- Create a set of models with different degrees or any other variants
- Iterate through the  $\lambda$ s and for each  $\lambda$  go through all the models to learn some  $\theta$
- Compute the cross validation error using the learned  $\theta$  (computed with  $\lambda$ ) on the  $J_{cv}(\theta)$  without regularization ( $\lambda = 0$ )
- select the best combo that produces the lowest error on the cross validation set
- Using the best combo  $\theta$  and  $\lambda$ , apply it on  $J_{test}(\theta)$  to see if it has a good generalization of the problem

# Example: Linear regression with regularization

## Choosing the regularization parameter $\lambda$

Model:  $h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

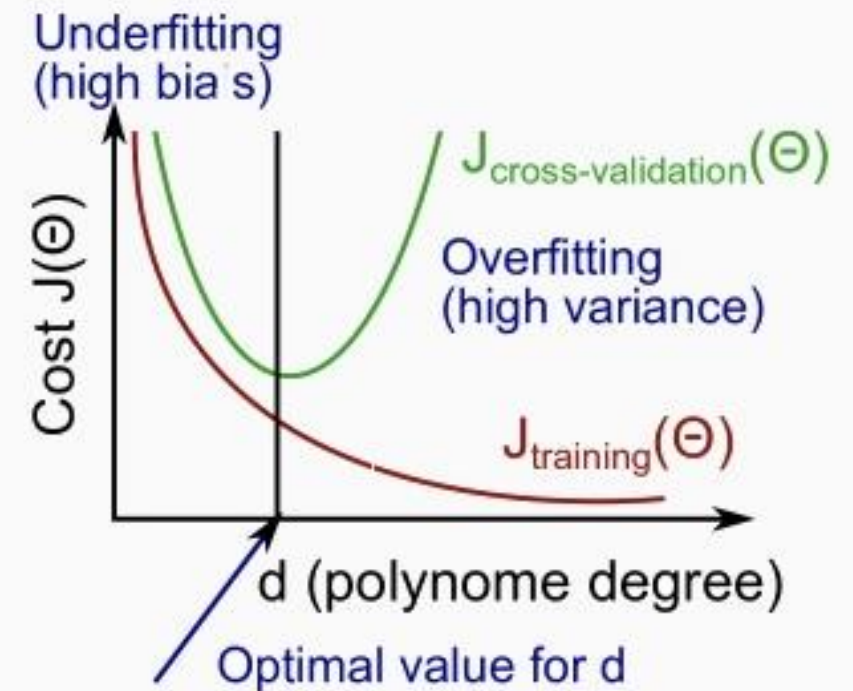
$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2$$

1. Try  $\lambda = 0 \leftarrow \uparrow \rightarrow \min_{\theta} J(\theta) \rightarrow \theta^{(1)} \rightarrow J_w(\theta^{(1)})$   
2. Try  $\lambda = 0.01 \rightarrow \min_{\theta} J(\theta) \rightarrow \theta^{(2)} \rightarrow J_w(\theta^{(2)})$   
3. Try  $\lambda = 0.02 \rightarrow \theta^{(3)} \rightarrow J_w(\theta^{(3)})$   
4. Try  $\lambda = 0.04$   
5. Try  $\lambda = 0.08 \rightarrow \theta^{(5)} \rightarrow J_w(\theta^{(5)})$   
 $\vdots$   
12. Try  $\lambda = 10 \rightarrow \theta^{(12)} \rightarrow J_w(\theta^{(12)})$   
 $\uparrow$  10.24 Pick (say)  $\theta^{(5)}$ . Test error:  $J_{\text{test}}(\theta^{(5)})$

Andrew Ng

# Diagnosis Bias vs. Variance

- The training error will tend to decrease as we increase the degree of the polynomial
- At the same time, the cross validation error will tend to decrease  $d$  up to a point, and then it will **increase** as  $d$  is increased.



# Diagnosis Bias vs. Variance

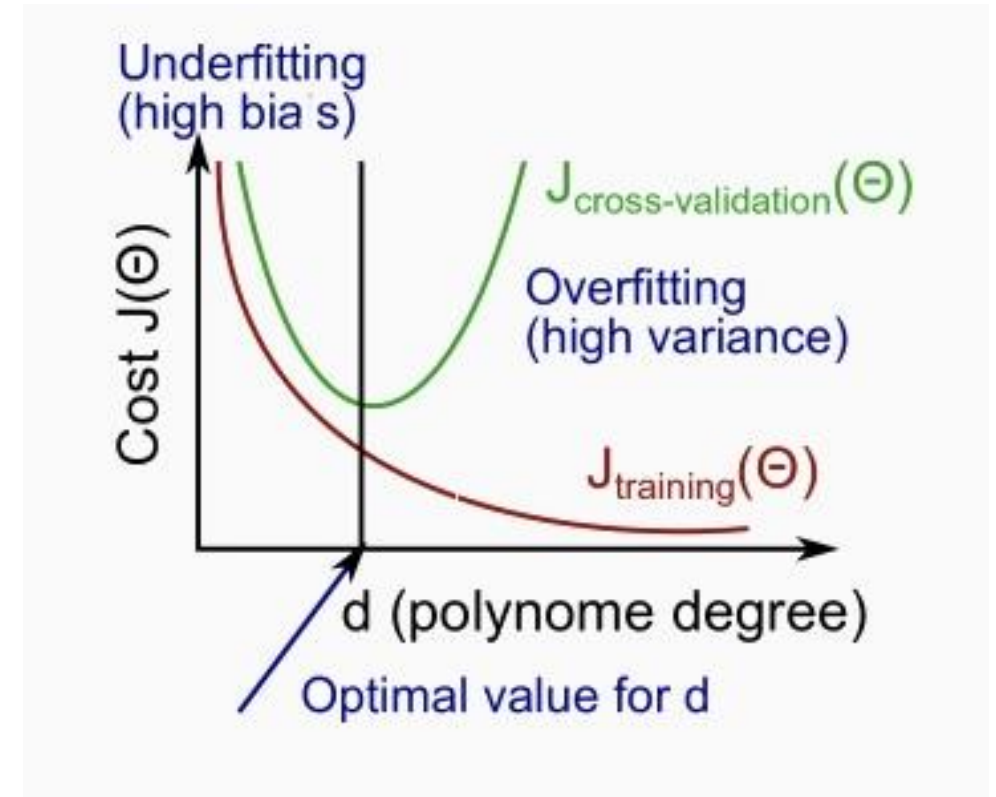
- **High bias (underfitting):**

Both  $J_{train}(\theta)$  and  $J_{CV}(\theta)$  will be high

$$J_{train}(\theta) \approx J_{cv}(\theta)$$

- **High variance (overfitting):**

$J_{train}(\theta)$  will be low and  $J_{CV}(\theta)$  will be much greater than  $J_{train}(\theta)$



# Learning Curve

- Let's say we have some data and split it into a training set and validation set.
- We take one single instance (that's right, one!) from the training set and use it to estimate a model.
- Then we measure the model's error on the validation set and on that single training instance.
- The error on the training instance will be 0, since it's quite easy to perfectly fit a single data point.
- The error on the validation set, however, will be very large.

# Learning Curve

- That's because the model is built around a single instance, and it almost certainly won't be able to generalize accurately on data that hasn't seen before.
- Now let's say that instead of one training instance, we take ten and repeat the error measurements.

# Learning Curve

- Then we take fifty, one hundred, five hundred, until we use our entire training set.
- The error scores will vary more or less as we change the training set.

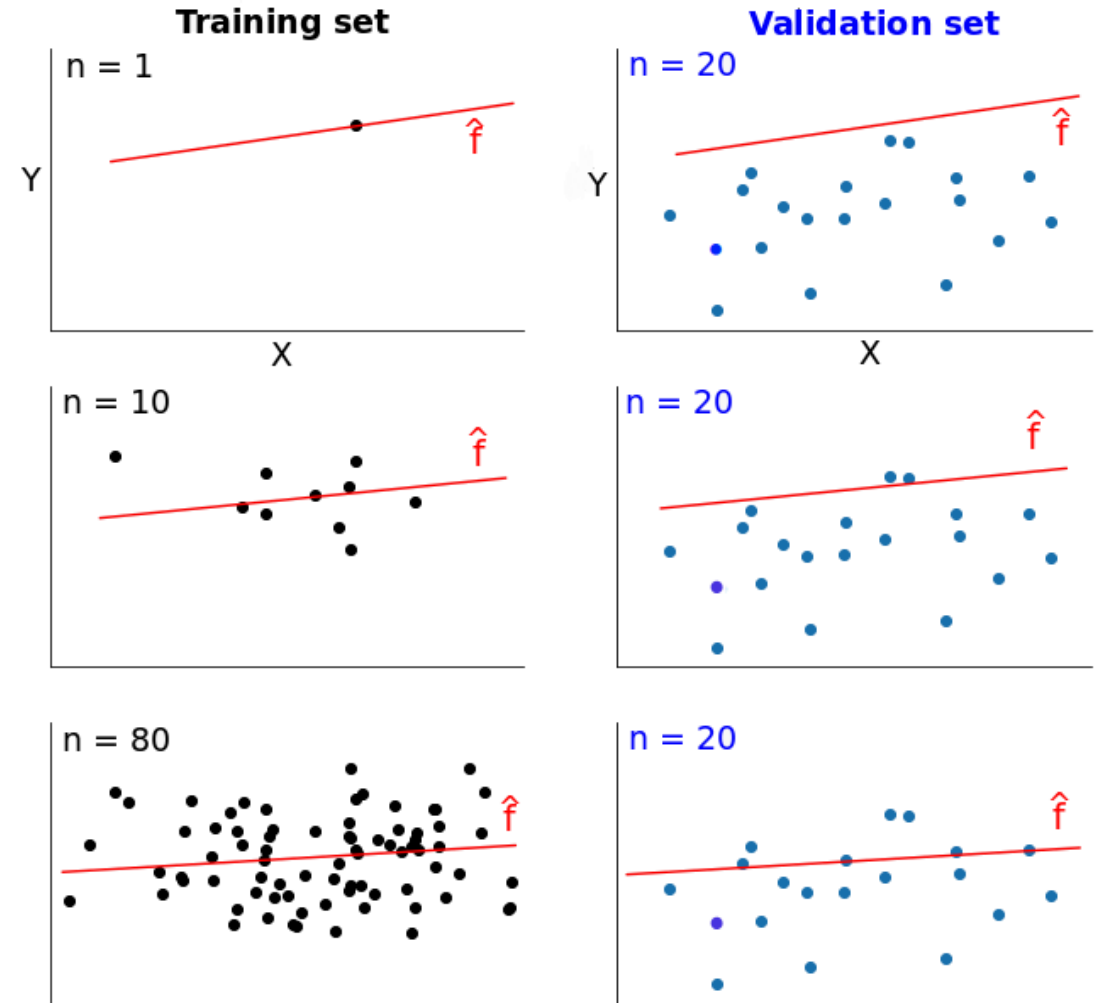


# Learning Curve

- We thus have two error scores to monitor: one for the validation set, and one for the training sets.
- If we plot the evolution of the two error scores as training sets change, we end up with two curves.
- These are called *learning curves*.
- ***In a nutshell, a learning curve shows how error changes as the training set size increases***

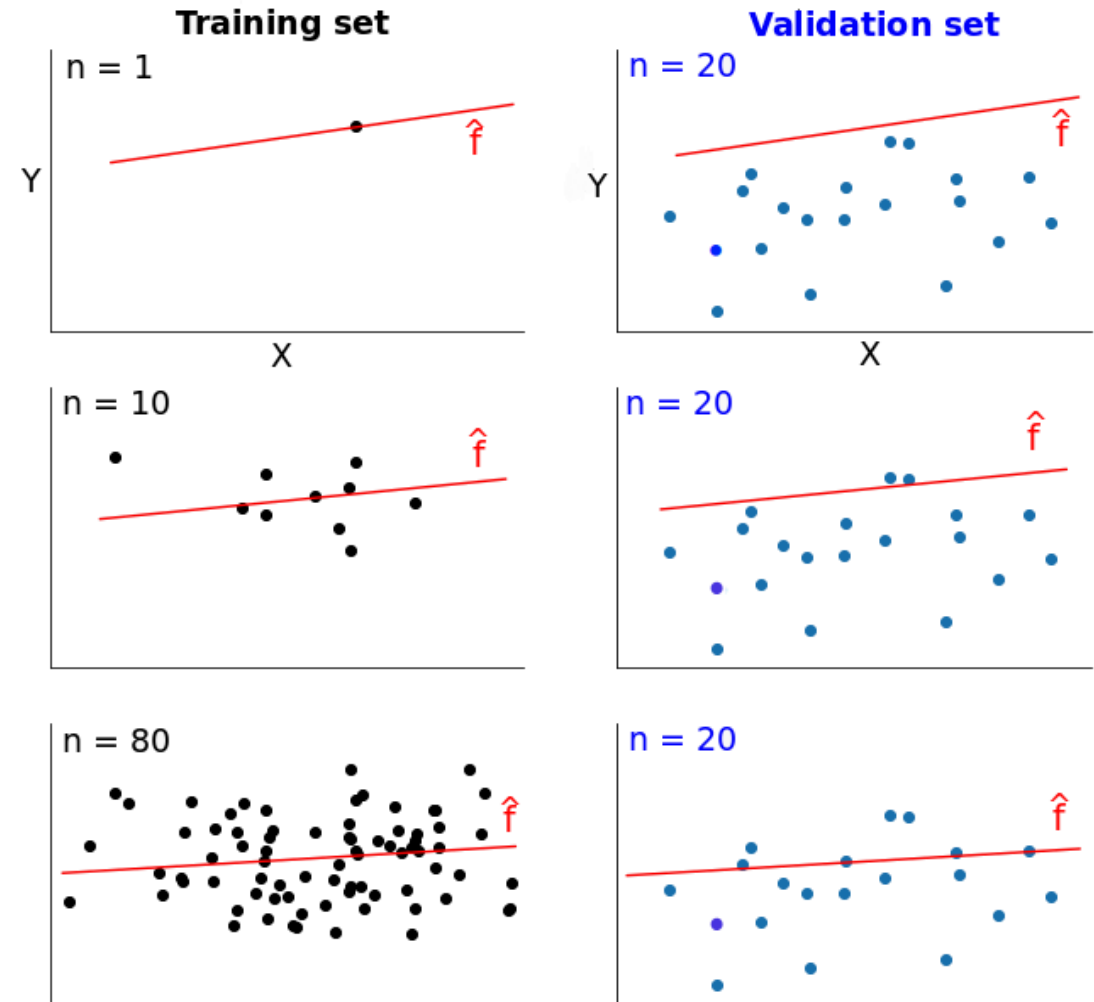
# Learning Curve

- As we increase the training set size, the model cannot fit perfectly anymore the training set.
- So the training error becomes larger.



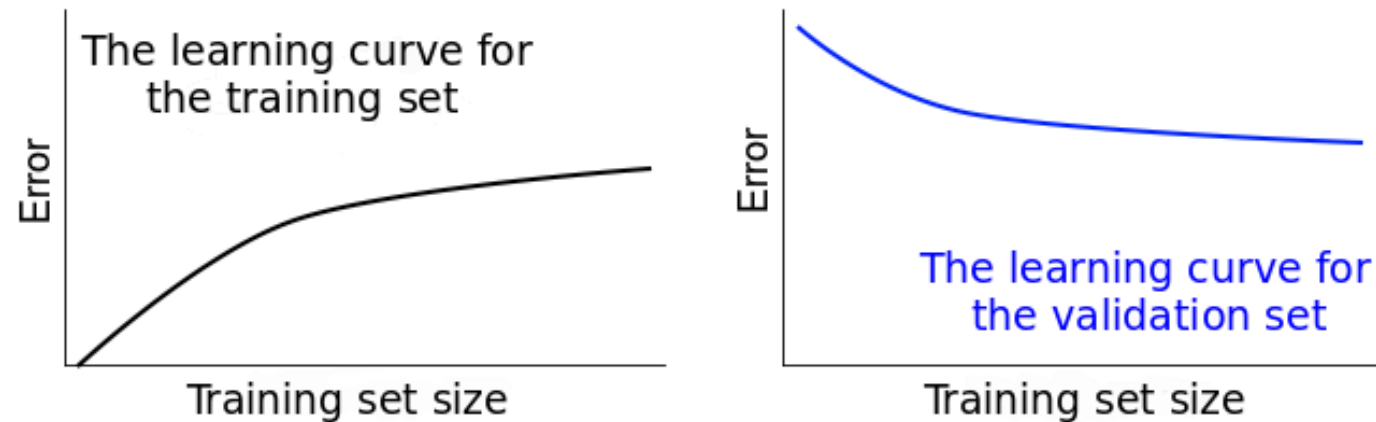
# Learning Curve

- However, the model is trained on more data, so it manages to fit better the validation set.
- Thus, the validation error decreases.
- To remind you, the validation set stays the same across all three cases



# Learning Curve

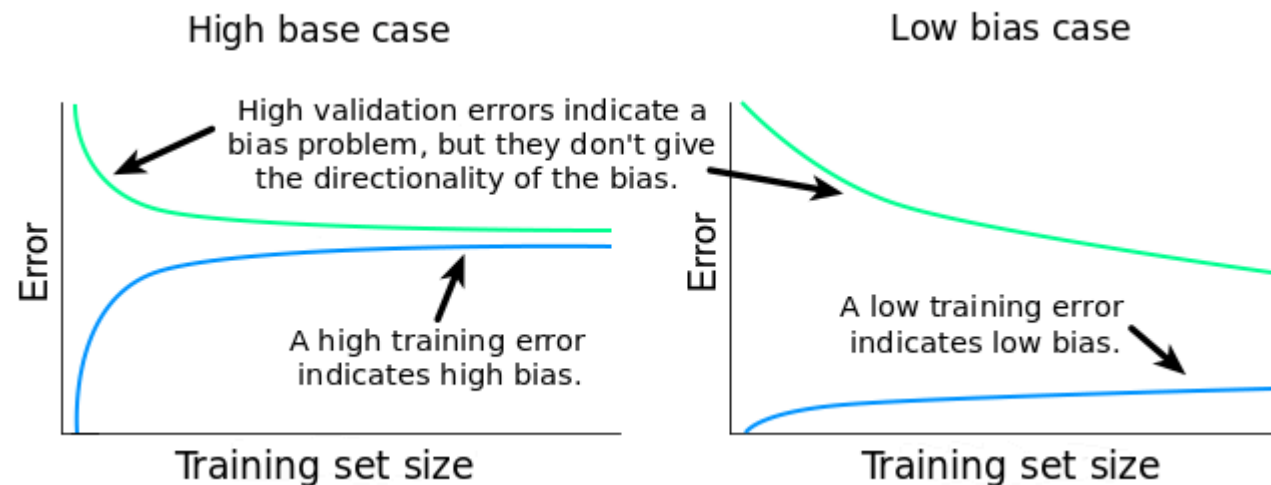
Plot the error scores for each training size, we'd get two learning curves looking similarly to



# Learning Curve

If the training error is very low, it means that the training data is fitted very well by the estimated model.

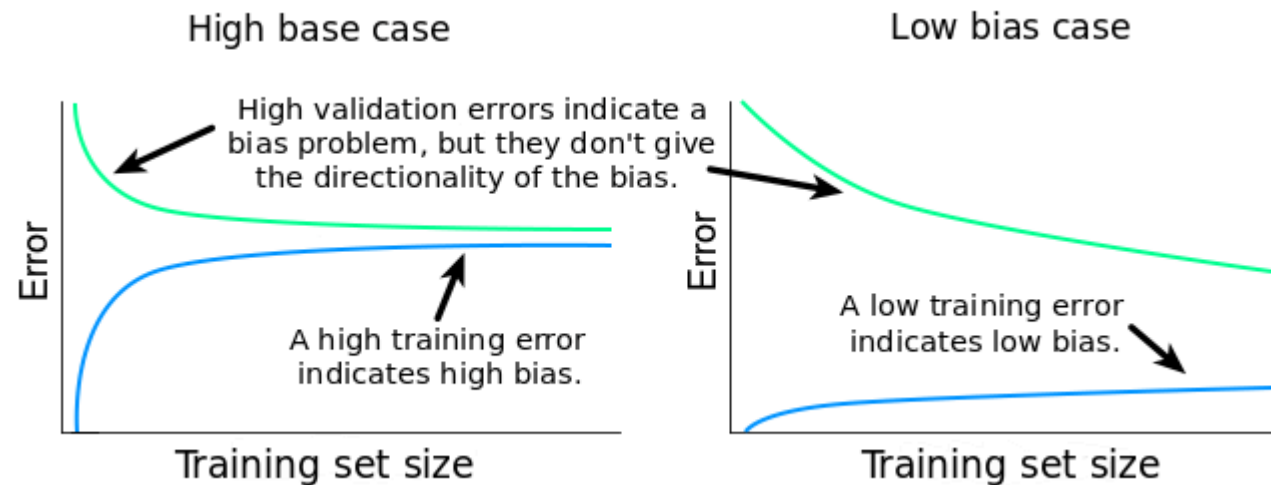
If the model fits the training data very well, it means it has *low* bias with respect to that set of data.



# Learning Curve

If the training error is high, it means that the training data is not fitted well enough by the estimated model.

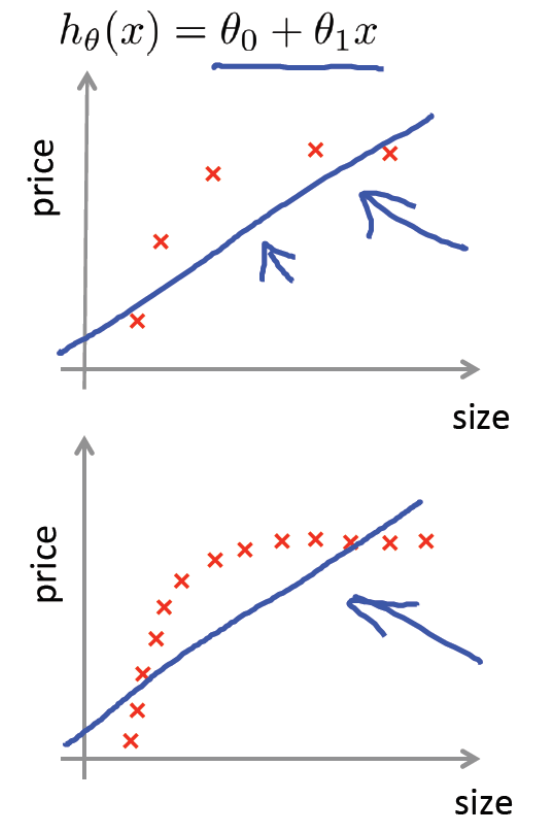
If the model fails to fit the training data well, it means it has *high* bias with respect to that set of data.



# Learning Curve

## High Bias – Underfitting

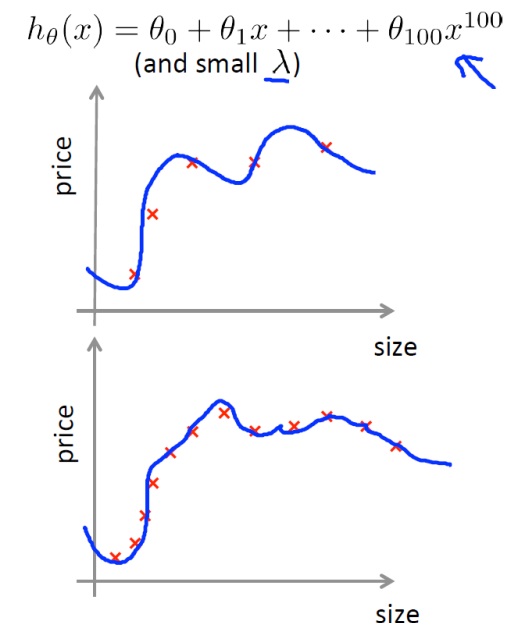
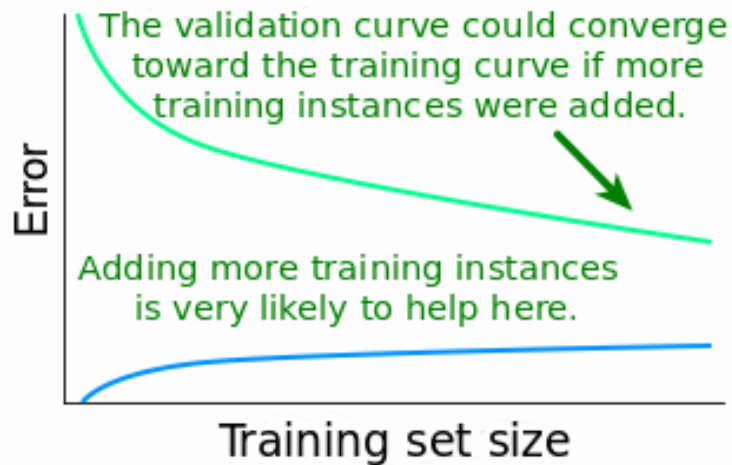
If a learning algorithm is suffering from high bias, getting more training data **will not** (by itself) help much.



# Learning Curve

## High Variance – Overfitting:

If a learning algorithm is suffering from high variance, getting more training data is likely to help.



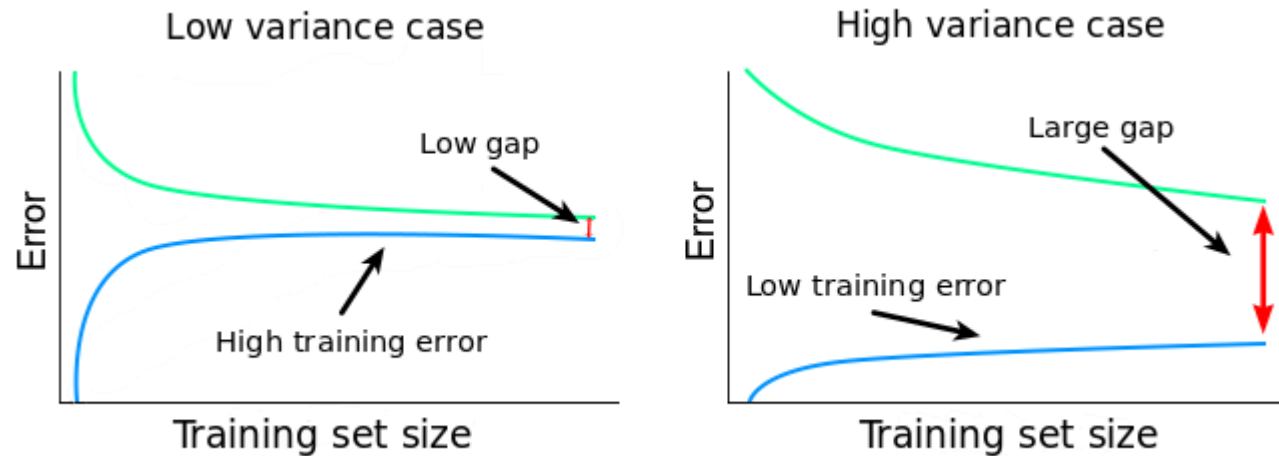
Andrew Ng



# Learning Curve

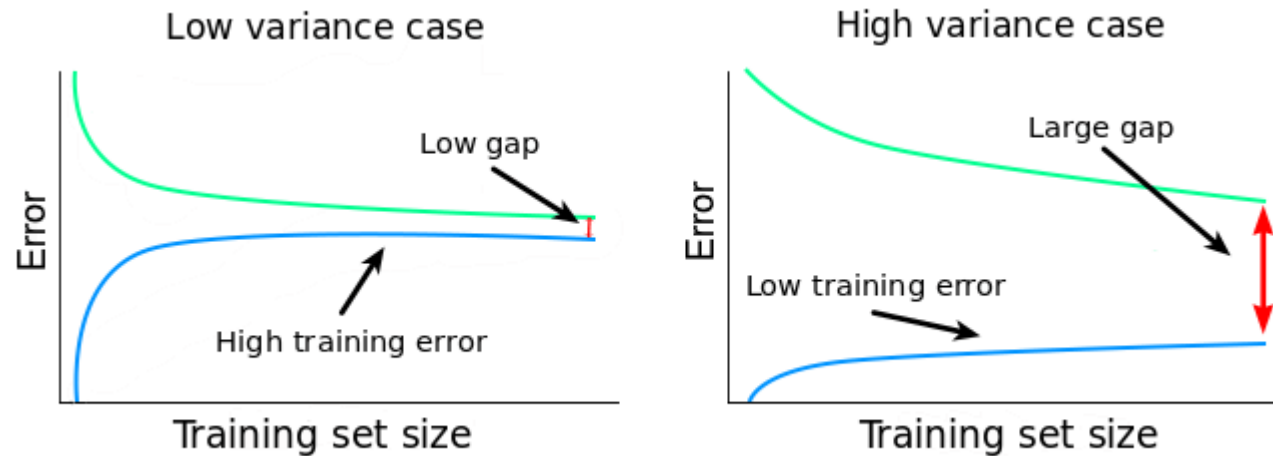
The relationship between the training and validation error, and the gap can be summarized this way:

$$gap = validationError - trainingError$$



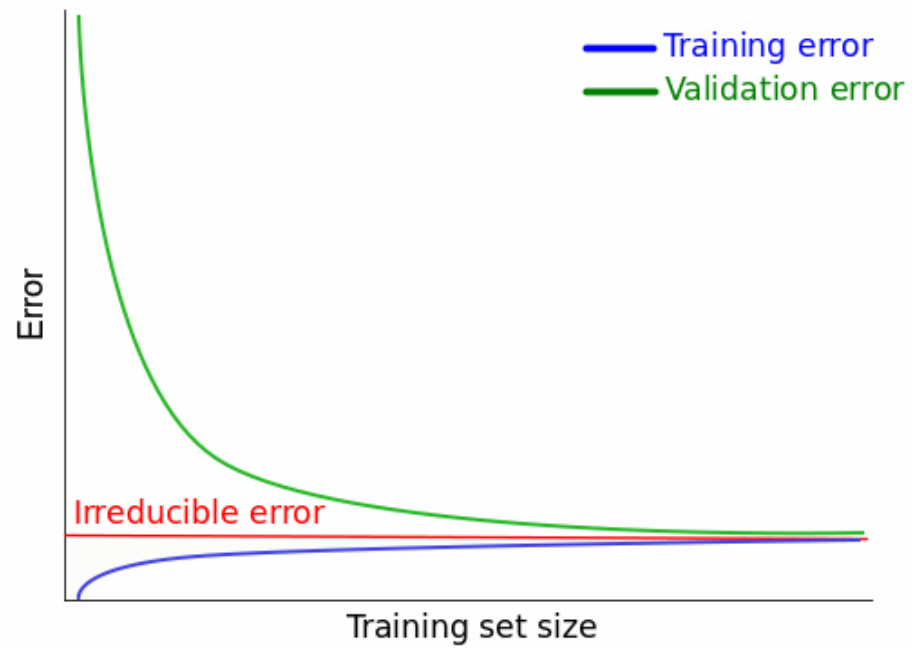
# Learning Curve

- The bigger the difference between the two errors, the bigger the gap.
- The bigger the gap, the bigger the variance (overfitting).
- If the gap is very narrow, we can safely conclude that the variance is low (underfitting)



# Learning Curve

- Ideal case



# Debugging a learning algorithm

Suppose you have implemented regularized linear regression to predict housing price. When you test your hypothesis on a new set of houses, you find that it make unacceptable large errors with its predictions.

What should you try next?

- Give more training examples
- Try smaller set of features
- Try getting additional features
- Try adding polynomial features
- Try decreasing  $\lambda$
- Try increasing  $\lambda$

# Debugging a learning algorithm

## **Fix high bias**

- Try getting additional features
- Try adding polynomial features
- Try decreasing  $\lambda$

## **Fix high variance**

- Give more training examples
- Try smaller set of features
- Try increasing  $\lambda$

# Cross-Validation

- *Cross-validation* is a statistical method of evaluating generalization performance that is more stable and thorough than using a split into a training and a test set.
- In cross-validation, the data is instead split repeatedly and multiple models are trained.

# K-fold cross-validation

- The most commonly used version of cross-validation is *k-fold cross-validation*, where  $k$  is a user-specified number, usually 5 or 10.

# K-fold cross-validation

- The need for multiple training/validation sets  $\{X_i, V_i\}_i$ :

Training/validation sets of fold  $i$

- K-fold cross-validation: Divide  $X$  into  $k$ ,  $X_i, i=1, \dots, K$

$$\mathcal{V}_1 = \mathcal{X}_1 \quad \mathcal{T}_1 = \mathcal{X}_2 \cup \mathcal{X}_3 \cup \dots \cup \mathcal{X}_K$$

$$\mathcal{V}_2 = \mathcal{X}_2 \quad \mathcal{T}_2 = \mathcal{X}_1 \cup \mathcal{X}_3 \cup \dots \cup \mathcal{X}_K$$

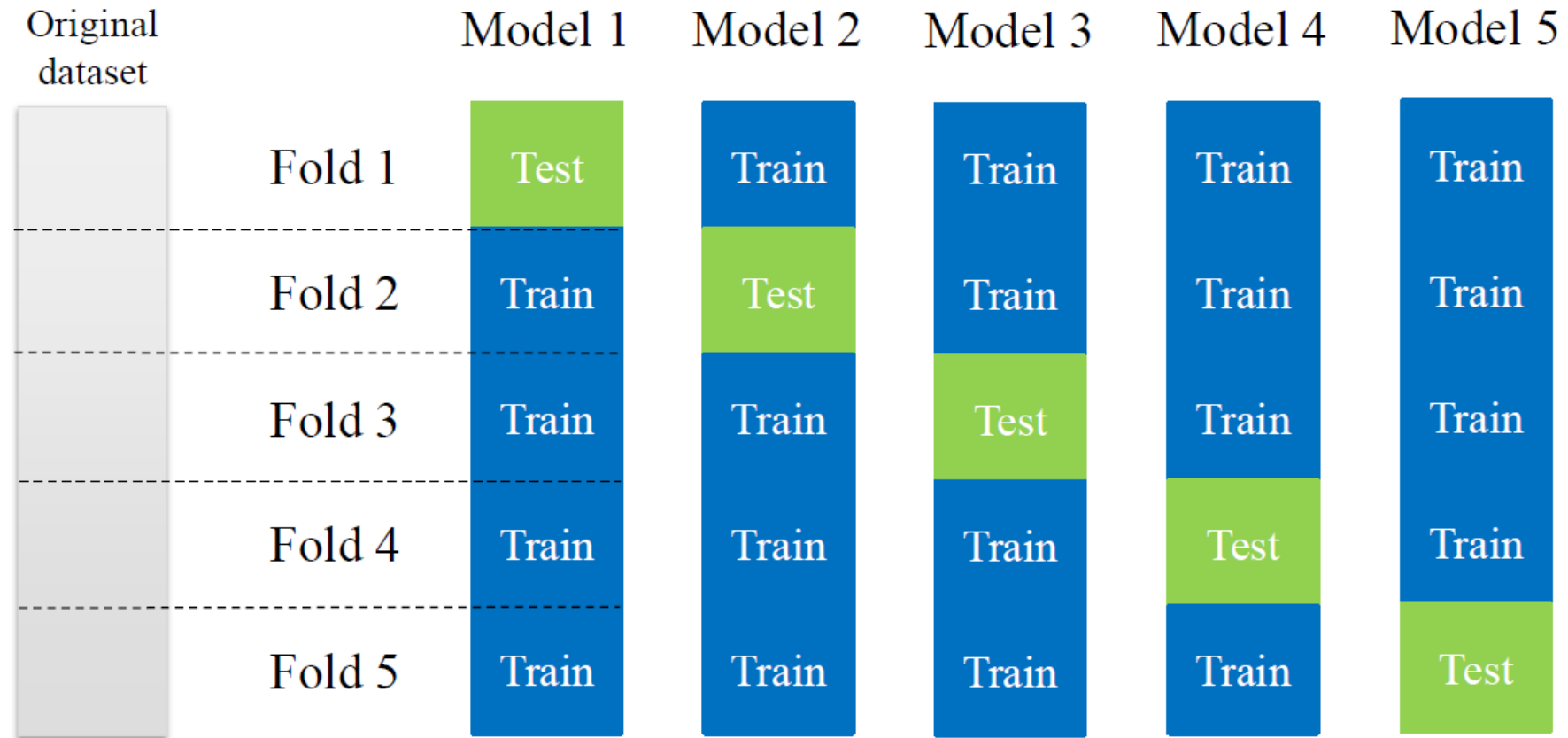
$\vdots$

$$\mathcal{V}_K = \mathcal{X}_K \quad \mathcal{T}_K = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \dots \cup \mathcal{X}_{K-1}$$

- $\mathcal{T}_i$  share  $K-2$  parts



# K-fold cross-validation



# Cross-Validation in scikit-learn

- Cross-validation is implemented in scikit-learn using the `cross_val_score` function from the `model_selection` module.
- The parameters of the `cross_val_score` function are the model we want to evaluate, the training data, and the ground-truth labels

# Cross-Validation in scikit-learn

Let's evaluate LogisticRegression on the iris dataset

```
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
iris = load_iris()
logreg = LogisticRegression()
scores = cross_val_score(logreg, iris.data, iris.target)
print("Cross-validation scores: {}".format(scores))
```

# Cross-Validation in scikit-learn

- By default, `cross_val_score` performs three-fold cross-validation, returning three accuracy values

## Output:

Cross-validation scores: [ 0.961 0.922 0.958]

# Cross-Validation in scikit-learn

We can change the number of folds used by changing the `cv` parameter:

```
scores = cross_val_score(logreg, iris.data, iris.target, cv=5)
print("Cross-validation scores: {}".format(scores))
```

***Output:***

Cross-validation scores: [ 1. 0.967 0.933 0.9 1. ]

# Cross-Validation in scikit-learn

A common way to summarize the cross-validation accuracy is to compute the mean:

```
print("Average cross-validation score: {:.2f}".format(scores.mean()))
```

***Output:***

Average cross-validation score: 0.96

# Benefits of Cross-Validation

- When using cross-validation, each example will be in the training set exactly once: each example is in one of the folds, and each fold is the validation set once.
- Therefore, the model needs to generalize well to all of the samples in the dataset for all of the cross-validation scores (and their mean) to be high
- Having multiple splits of the data also provides some information about how sensitive our model is to the selection of the training dataset.
- Another benefit of cross-validation as compared to using a single split of the data is that we use our data more effectively.

# Disadvantage of cross-validation

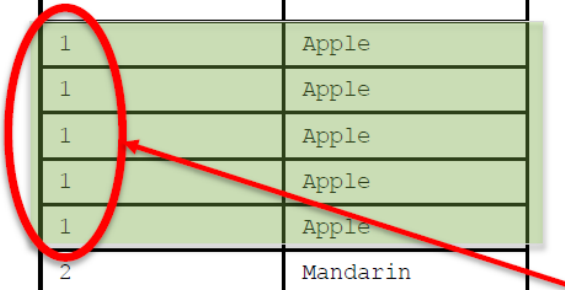
- The main disadvantage of cross-validation is increased computational cost.
  - As we are now training  $k$  models instead of a single model, cross-validation will be roughly  $k$  times slower than doing a single split of the data.



# Stratified Cross-validation

- Splitting the dataset into  $k$  folds by starting with the first one- $k$ -th part of the data might not always be a good idea

# Stratified Cross-validation



fruit_label	fruit_name
1	Apple
1	Apple
1	Apple
1	Apple
1	Apple
2	Mandarin
...	...
3	Orange
...	...
4	Lemon
4	Lemon
4	Lemon
4	Lemon
4	Lemon

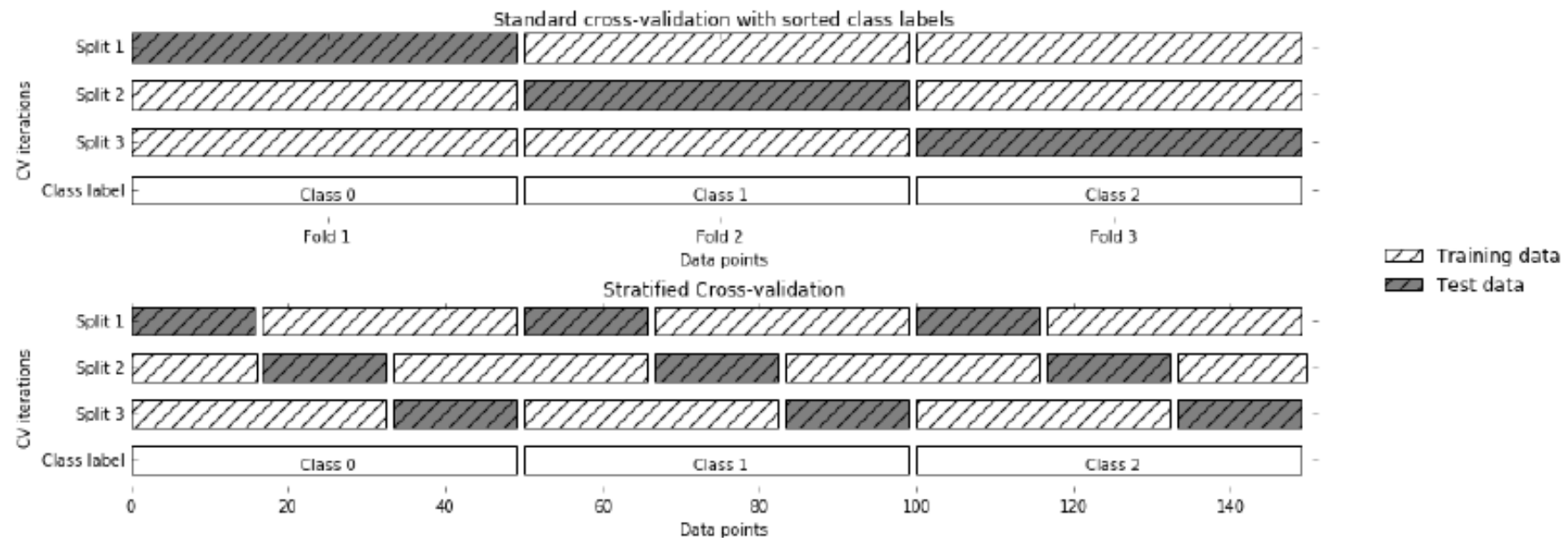
Example has 20 data samples = 4 classes with 5 samples each.

5-fold CV: 5 folds of 4 samples each.

Fold 1 uses the first 20% of the dataset as the test set, which only contains samples from class 1.

Classes 2, 3, 4 are missing entirely from test set and so will be missing from the evaluation.

# Stratified Cross-validation

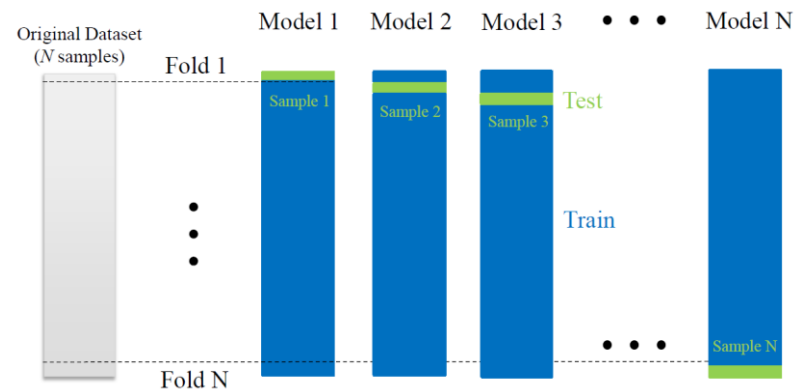


# Stratified Cross-validation

- For example, if 90% of your samples belong to class A and 10% of your samples belong to class B, then stratified cross-validation ensures that in each fold, 90% of samples belong to class A and 10% of samples belong to class B.

# Leave-one-out

- Each fold is a single sample.
- This can be very time consuming, particularly for large datasets, but sometimes provides better estimates on small datasets
- This is typically used in applications such as medical diagnosis, where labeled data is hard to find



# Leave-one-out in scikit-learn

```
from sklearn.model_selection import LeaveOneOut
loo = LeaveOneOut()
scores = cross_val_score(logreg, iris.data, iris.target, cv=loo)
print("Number of cv iterations: ", len(scores))
print("Mean accuracy: {:.2f}".format(scores.mean()))
```

## Output:

Number of cv iterations: 150

Mean accuracy: 0.95

# 5×2 Cross-Validation

- 5 times 2 fold cross-validation

$$\mathcal{T}_1 = \mathcal{X}_1^{(1)} \quad \mathcal{V}_1 = \mathcal{X}_1^{(2)}$$

$$\mathcal{T}_2 = \mathcal{X}_1^{(2)} \quad \mathcal{V}_2 = \mathcal{X}_1^{(1)}$$

$$\mathcal{T}_3 = \mathcal{X}_2^{(1)} \quad \mathcal{V}_3 = \mathcal{X}_2^{(2)}$$

$$\mathcal{T}_4 = \mathcal{X}_2^{(2)} \quad \mathcal{V}_4 = \mathcal{X}_2^{(1)}$$

⋮

$$\mathcal{T}_9 = \mathcal{X}_5^{(1)} \quad \mathcal{V}_9 = \mathcal{X}_5^{(2)}$$

$$\mathcal{T}_{10} = \mathcal{X}_5^{(2)} \quad \mathcal{V}_{10} = \mathcal{X}_5^{(1)}$$

# Bootstrapping

- To generate multiple samples from a single sample, an alternative to cross-validation is the *bootstrap* that generates new samples by drawing instances from the original sample *with* replacement.
- The bootstrap samples may overlap more than cross-validation samples and hence their estimates are more dependent; but is considered the best way to do resampling for very small datasets.



# Bootstrapping

- In the bootstrap, we sample  $N$  instances from a dataset of size  $N$  with replacement.
- The original dataset is used as the validation set.
- The probability that we pick an instance is  $1/N$ ; the probability that we do not pick it is  $1 - 1/N$
- Prob that we do not pick an instance after  $N$  draws

$$\left(1 - \frac{1}{N}\right)^N \approx e^{-1} = 0.368$$

that is, only 36.8% is new!

# Model Selection, revisit

1. Split data
  - Training set (model building)
  - Validation set (model selection)
  - Test set (final evaluation)
2. Learn parameter from training data (minimizing training error)
3. Compute cross validation set error.
4. Select the model with lowest cross validation set error.
5. Estimate generalization error for test set

# Model Selection

- Now that we know how to evaluate how well a model generalizes, we can take the next step and improve the model's generalization performance by tuning its parameters.
- The most commonly used method is *grid search*, which basically means trying all possible combinations of the parameters of interest.

# Grid Search: Example

Consider the case of a kernel SVM with an RBF (radial basis function) kernel, as implemented in the SVC class.

- There are two important parameters: the kernel bandwidth, gamma, and the regularization parameter, C.
- Say we want to try the values 0.001, 0.01, 0.1, 1, 10, and 100 for the parameter C, and the same for gamma.
- Because we have six different settings for C and gamma that we want to try, we have 36 combinations of parameters in total

# Grid Search: Example, cont.

Split the data (three sets): the training set to build the model, the validation (or development) set to select the parameters of the model, and the test set to evaluate the performance of the selected parameters

```
from sklearn.svm import SVC
# split data into train+validation set and test set
X_trainval, X_test, y_trainval, y_test = train_test_split(iris.data, iris.target, random_state=0)
# split train+validation set into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(X_trainval, y_trainval, random_state=1)
print("Size of training set: {} size of validation set: {} size of test set:"
      " {} \n".format(X_train.shape[0], X_valid.shape[0], X_test.shape[0]))
```

# Grid Search: Example, cont.

```
best_score = 0
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters, train an SVC
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # evaluate the SVC on the test set
        score = svm.score(X_valid, y_valid)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}
```

Learn parameter from training data (minimizing training error)  
Compute cross validation set error.  
Select the model with lowest cross validation set error.

# Grid Search: Example, cont.

Estimate generalization error for test set

```
# rebuild a model on the combined training and validation set,  
# and evaluate it on the test set  
svm = SVC(**best_parameters)  
svm.fit(X_trainval, y_trainval)  
test_score = svm.score(X_test, y_test)  
print("Best score on validation set: {:.2f}".format(best_score))  
print("Best parameters: ", best_parameters)  
print("Test set score with best parameters: {:.2f}".format(test_score))
```

# Grid Search: Example, cont.

## ***Output:***

Size of training set: 84 size of validation set: 28 size of test set: 38

Best score on validation set: 0.96

Best parameters: {'C': 10, 'gamma': 0.001}

Test set score with best parameters: 0.92



# Grid Search with Cross-Validation

- For a better estimate of the generalization performance, instead of using a single split into a training and a validation set, we can use cross-validation to evaluate the performance of each parameter combination.

# Grid Search with Cross-Validation

```
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:  
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:  
        # for each combination of parameters, train an SVC  
        svm = SVC(gamma=gamma, C=C)  
        # perform cross-validation  
        scores = cross_val_score(svm, X_trainval, y_trainval, cv=5)  
        # compute mean cross-validation accuracy  
        score = np.mean(scores)  
        # if we got a better score, store the score and parameters  
        if score > best_score:  
            best_score = score  
            best_parameters = {'C': C, 'gamma': gamma}  
  
    # rebuild a model on the combined training and validation set  
    svm = SVC(**best_parameters)  
    svm.fit(X_trainval, y_trainval)
```

# Grid Search in scikit-learn

Because grid search with cross-validation is such a commonly used method to adjust parameters, scikit-learn provides the `GridSearchCV` class, which implements it in the form of an estimator.

# Grid Search in scikit-learn

```
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}

from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

grid_search = GridSearchCV(SVC(), param_grid, cv=5)
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, random_state=0)
grid_search.fit(X_train, y_train)

print("Test set score: {:.2f}".format(grid_search.score(X_test, y_test)))
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
print("Best estimator:\n{}".format(grid_search.best_estimator_))
```

# Performance Metrics

# Performance Metrics in Machine Learning

- Different performance metrics are used to evaluate different Machine Learning Algorithms.
- Important to choose the right metric when selecting between models and adjusting parameters.

# Evaluation Metrics for Classification Problems

- Understand why accuracy only gives a partial picture of a classifier's performance.
- Understand the motivation and definition of important evaluation metrics in machine learning.
- Learn how to use a variety of evaluation metrics to evaluate supervised machine learning models.
- Learn about choosing the right metric for selecting between models or for doing parameter tuning.

# Confusion Matrix

- It is one of the most intuitive and easiest metrics used for finding the correctness and accuracy of the model.
- It is used for Classification problem where the output can be of two or more types of classes.

<https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b>



# Confusion Matrix

- Binary Classification

True Class	Predicted class	
	Yes	No
Yes	TP: True Positive	FN: False Negative
No	FP: False Positive	TN: True Negative

# Terms associated with Confusion matrix

- **True Positives (TP):** True positives are the cases when the actual class of the data point was 1 (True) and the predicted is also 1 (True)
- **True Negatives (TN):** True negatives are the cases when the actual class of the data point was 0 (False) and the predicted is also 0 (False)

<https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b>

# Terms associated with Confusion matrix

- **False Positives (FP):** False positives are the cases when the actual class of the data point was 0 (False) and the predicted is 1 (True). False is because the model has predicted incorrectly and positive because the class predicted was a positive one. (1)
- **False Negatives (FN):** False negatives are the cases when the actual class of the data point was 1 (True) and the predicted is 0 (False). False is because the model has predicted incorrectly and negative because the class predicted was a negative one. (0)

<https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b>

# When to minimise what?

- We know that there will be some error associated with every model that we use for predicting the true class of the target variable.
- This will result in False Positives and False Negatives
- There's no hard rule that says what should be minimised in all the situations.
- It purely depends on the business needs and the context of the problem you are trying to solve.

<https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b>

# When to minimise what?

## **Minimising False Negatives:** Cancer detection problem

Let's say in our cancer detection problem example, out of 100 people, only 5 people have cancer. In this case, we want to correctly classify all the cancerous patients as even a very BAD model(Predicting everyone as NON-Cancerous) will give us a 95% accuracy(will come to what accuracy is). But, in order to capture all cancer cases, we might end up making a classification when the person actually NOT having cancer is classified as Cancerous. This might be okay as it is less dangerous than NOT identifying/capturing a cancerous patient since we will anyway send the cancer cases for further examination and reports. But missing a cancer patient will be a huge mistake as no further examination will be done on them.

<https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b>

# When to minimise what?

- **Minimising False Positives: Email Spam Detection**

Let's say that you are expecting an important email like hearing back from a recruiter or awaiting an admit letter from a university. Let's assign a label to the target variable and say, **1**: "Email is a spam" and **0**: "Email is not a spam"

Suppose the Model classifies that important email that you are desperately waiting for, as Spam (case of False positive). Now, in this situation, this is pretty bad than classifying a spam email as important or not spam since in that case, we can still go ahead and manually delete it and it's not a pain if it happens once a while. So in case of Spam email classification, minimising False positives is more important than False Negatives

<https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b>

# Confusion Matrix in scikit learn

	Predicted class	
True Class	Yes	No
Yes	TP: True Positive	FN: False Negative
No	FP: False Positive	TN: True Negative

```
from sklearn.metrics import confusion_matrix  
confusion = confusion_matrix(y_test, pred_logreg)  
print("Confusion matrix:\n{}".format(confusion))
```

# Confusion Matrix

- Always look at the confusion matrix for your classifier.

True negative	TN = 400	FP = 7	
	FN = 17	TP = 26	
	Predicted negative	Predicted positive	$N = 450$



# Accuracy

- Accuracy in classification problems is the number of correct predictions made by the model over all kinds predictions made
- Accuracy is the number of correct predictions (TP and TN) divided by the number of all samples

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP + TN}{N}$$

# Accuracy

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
	Predicted negative	Predicted positive	$N = 450$

$$Accuracy = \frac{TP + TN}{N} = \frac{26 + 400}{450} = 0,946$$

# Accuracy

**When to use Accuracy:** Accuracy is a good measure when the target variable classes in the data are nearly balanced.

**When NOT to use Accuracy:** Accuracy should NEVER be used as a measure when the target variable classes in the data are a majority of one class.

<https://medium.com/thalus-ai/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b>

# Classification Error

$$Error = \frac{FP + FN}{TP + TN + FP + FN}$$

$$Error = \frac{7 + 17}{400 + 26 + 17 + 7} = 0.060$$

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
	Predicted negative	Predicted positive	$N = 450$

# Precision

*Precision* measures how many of the samples predicted as positive are actually positive:

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

Precision is used as a performance metric when the goal is to limit the number of false positives

Known as positive predictive value

# Precision

True negative	TN = 400	FP = 7	
True positive	FN = 17	TP = 26	
	Predicted negative	Predicted positive	$N = 450$

$$Precision = \frac{TP}{TP + FP} = \frac{26}{26 + 7} = 0.787$$

# Recall

- Recall measures how many of the positive samples are captured by the positive predictions:

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

- Recall is used as performance metric when we need to identify all positive samples; that is, when it is important to avoid false negatives.
- Known as *sensitivity*, *hit rate*, *true positive rate (TPR)*

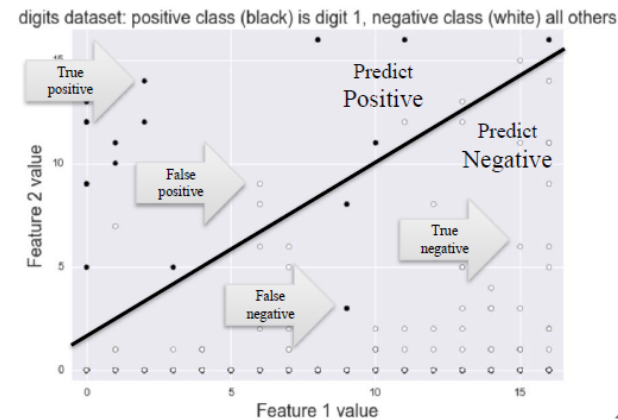
# Tradeoff between precision and recall

Low Precision, High Recall

High Precision, Lower Recall

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

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



TN = 429	FP = 6
FN = 2	TP = 13

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{13}{19} = 0.68$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{13}{15} = 0.87$$



# When to use Precision and When to use Recall?

It is clear that recall gives us information about a classifier's performance with respect to false negatives (how many did we miss), while precision gives us information about its performance with respect to false positives (how many did we caught).

# Tradeoff between precision and recall

## Recall-oriented machine learning tasks:

- Search and information extraction in legal discovery
- Tumor detection
- Often paired with a human expert to filter out false positives

## Precision-oriented machine learning tasks:

- Search engine ranking, query suggestion
- Document classification
- Many customer-facing tasks (users remember failures!)

# Tradeoff between precision and recall

While precision and recall are very important measures, looking at only one of them will not provide you with the full picture.

One way to summarize them is the *f-score* or *f-measure*, which is with the harmonic mean of precision and recall:

$$F = 2 \cdot \frac{\textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}}$$

# Tradeoff between precision and recall

This particular variant is also known as the  $f_1$ -score.

As it takes precision and recall into account, it can be a better measure than accuracy on imbalanced binary classification datasets.

$$f_1 = 2 \cdot \frac{\textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}} = \frac{2 \cdot TP}{2 \cdot TP + FN + FP}$$

# False Positive Rate

What fraction of all negative instances does the classifier incorrectly identify as positive?

$$FPR = \frac{FP}{TN + FP}$$

known as *false alarm rate*

# Sensitivity and Specificity

- **Sensitivity** is the same as true positive rate and recall

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

- **Specificity** is how well we detect the negatives, which is the number of true negatives divided by the total number of negatives; this is equal to 1 minus the false alarm rate

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

# Summary

	Predicted class		
True Class	Positive	Negative	Total
Positive	$tp$ : true positive	$fn$ : false negative	$p$
Negative	$fp$ : false positive	$tn$ : true negative	$n$
Total	$p'$	$n'$	$N$

Name	Formula
error	$(fp + fn) / N$
accuracy	$(tp + tn) / N = 1 - \text{error}$
tp-rate	$tp / p$
fp-rate	$fp / n$
precision	$tp / p'$
recall	$tp / p = \text{tp-rate}$
sensitivity	$tp / p = \text{tp-rate}$
specificity	$tn / n = 1 - \text{fp-rate}$

# Scikit learn

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
# Accuracy = TP + TN / (TP + TN + FP + FN)
# Precision = TP / (TP + FP)
# Recall = TP / (TP + FN) Also known as sensitivity, or True Positive Rate
# F1 = 2 * Precision * Recall / (Precision + Recall)
print('Accuracy: {:.2f}'.format(accuracy_score(y_test, tree_predicted)))
print('Precision: {:.2f}'.format(precision_score(y_test, tree_predicted)))
print('Recall: {:.2f}'.format(recall_score(y_test, tree_predicted)))
print('F1: {:.2f}'.format(f1_score(y_test, tree_predicted)))
```

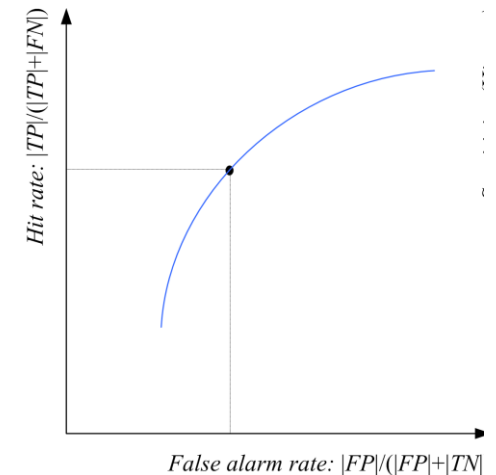


# ROC curve

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:

$$\text{True Positive Rate } TPR = \frac{TP}{TP+FN}$$

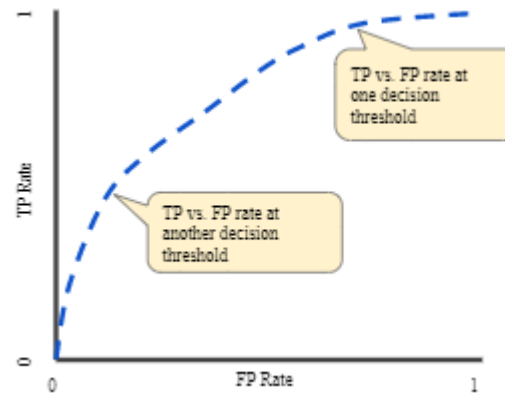
$$\text{False Positive Rate } FPR = \frac{FP}{TN+FP}$$



<https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>

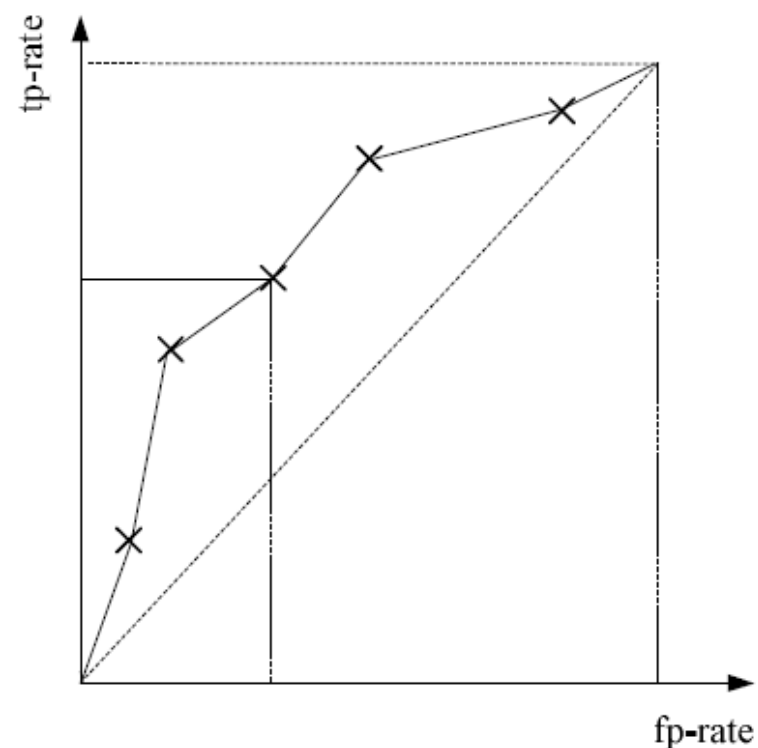
# ROC curve

- 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.
- The following figure shows a typical ROC curve.



# ROC curve

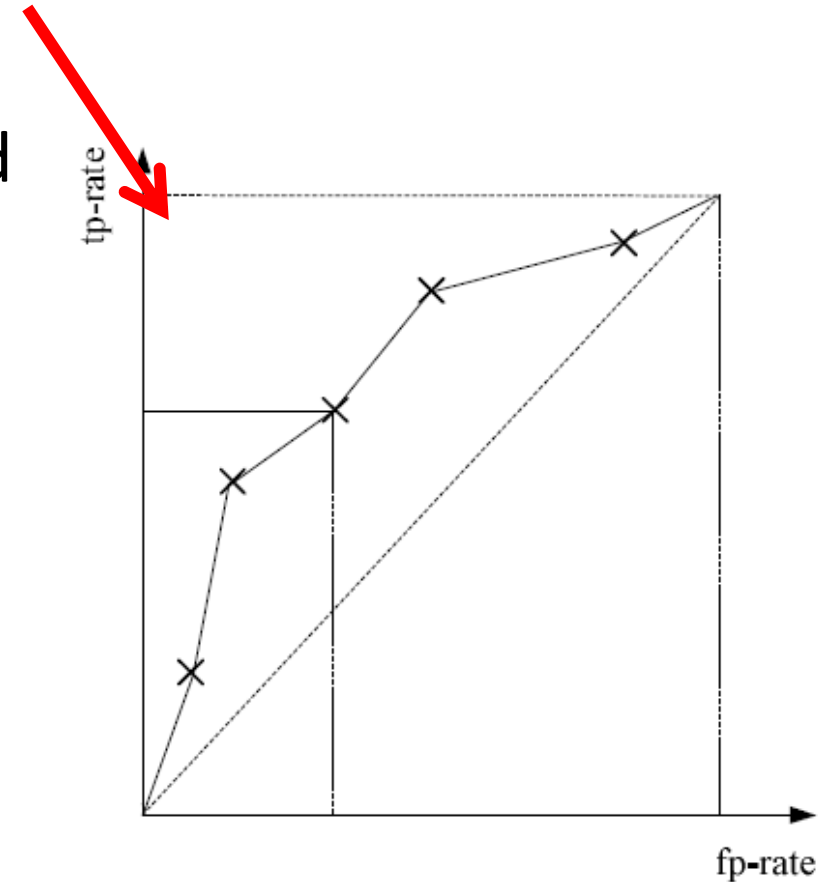
Each classifier has a threshold that allows us to move over this curve, and we decide on a point, based on the relative importance of hits versus false alarms, namely, true positives and false positives.



(a) Example ROC curve

# ROC curve

- Ideally, a classifier has a tp-rate of 1 and a fp-rate of 0, and hence a classifier is better the more it gets closer to the upper-left corner



(a) Example ROC curve

# Area Under the Curve (AUC)

- ROC is a probability curve and AUC represents degree or measure of separability.
- It tells how much model is capable of distinguishing between classes.
- Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s

<https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5>

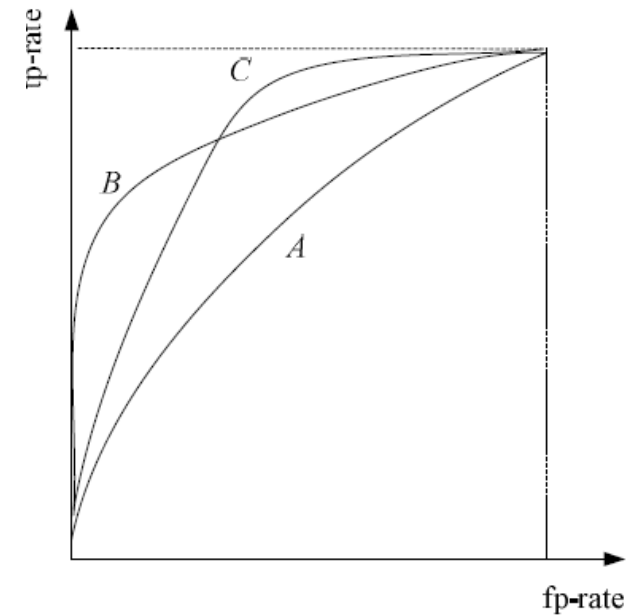
# Area Under the Curve (AUC)

Summarizing an ROC curve in one number: Area Under the Curve (AUC)

A classifier ideally has an AUC of 1 and AUC values of different classifiers can be compared to give us a general performance averaged over different loss conditions.

AUC = 0 (worst) AUC = 1 (best)

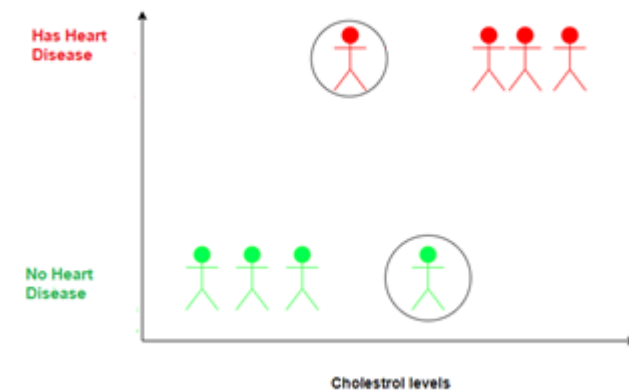
*B and C are preferred over A*



(b) Different ROC curves for different classifiers

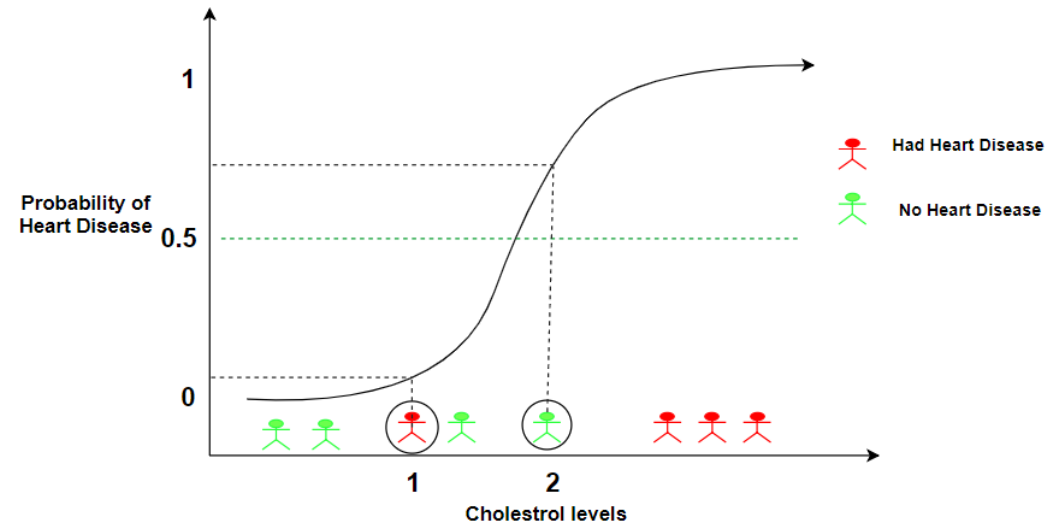
# ROC Curve - example

- Consider a **hypothetical example** containing a group of people.
- The y-axis has two categories i.e **Has Heart Disease** represented by red people and **does not have Heart Disease** represented by green circles.
- Along the x-axis, we have **cholesterol** levels and the classifier tries to classify people into two categories depending upon their cholesterol levels



# ROC Curve - example

- Let us now evaluate the effectiveness of logistic regression with the classification threshold set to 0.5, with some new people about whom we already know if they have heart disease or not.





# ROC Curve - example

- Let's create a Confusion Matrix to summarize the classifications

		Actual	
		Has Heart Disease	Doesnot have Heart Disease
Predicted	Has Heart Disease	3	1
	Doesnot have Heart Disease	1	3

- Once the confusion matrix is filled in, we can calculate the True Positive rate and the False Positive Rate to evaluate this logistic regression at 0.5 threshold.

# ROC Curve - example

$$\text{True Positive Rate } TPR = \frac{TP}{TP+FN} = \frac{3}{4} = 0.75$$

$$\text{False Positive Rate } FPR = \frac{FP}{TN+FP} = \frac{1}{1+3} = 0.25$$

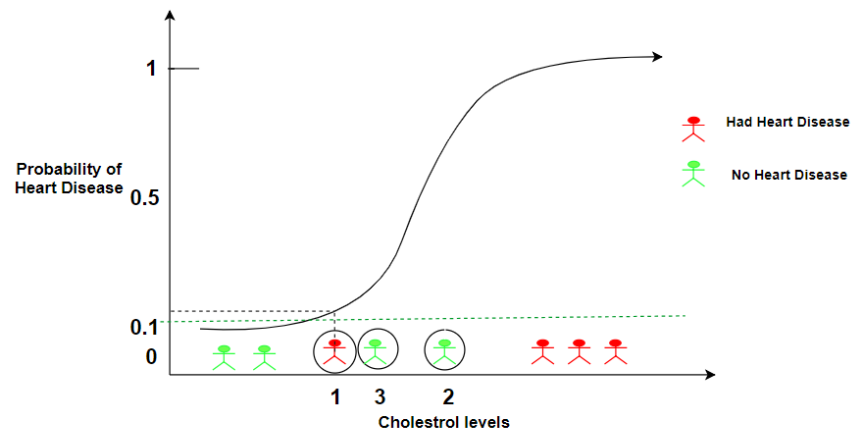
		Actual	
		Has Heart Disease	Doesnot have Heart Disease
Predicted	Has Heart Disease	3	1
	Doesnot have Heart Disease	1	3

# ROC Curve - example

Let's talk about what happens when we use a different threshold for deciding if a person has heart disease or not

# ROC Curve - example

## Setting the Threshold to 0.1



**Predicted**

	Actual	
	Has Heart Disease	Doesnot have Heart Disease
Has Heart Disease	4	2
Doesnot have Heart Disease	0	2

a lower threshold:  
Increases the number of False Positives  
Decreases the number of False Negatives

# ROC Curve - example

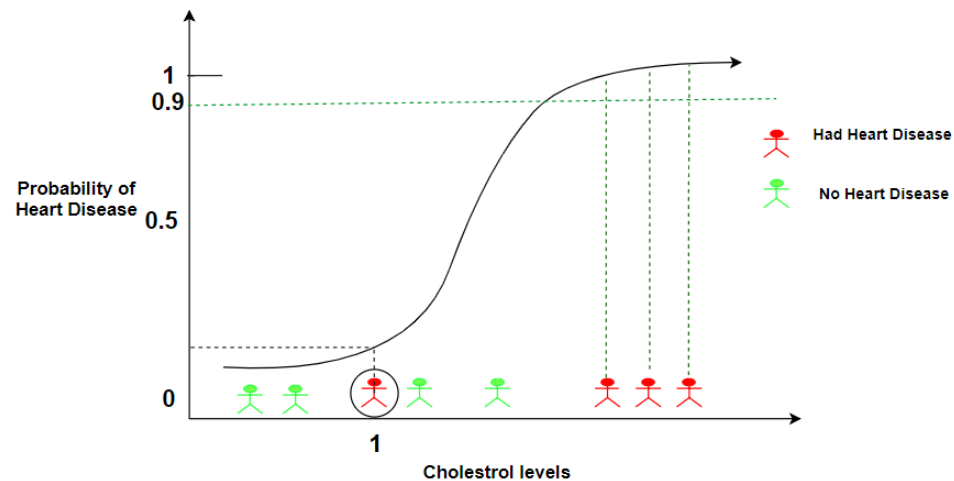
$$\text{True Positive Rate } TPR = \frac{TP}{TP+FN} = \frac{4}{4+0} = 1$$

$$\text{False Positive Rate } FPR = \frac{FP}{TN+FP} = \frac{2}{2+2} = 0.5$$

		Actual	
		Has Heart Disease	Doesnot have Heart Disease
Predicted	Has Heart Disease	4	2
	Doesnot have Heart Disease	0	2

# ROC Curve - example

## Setting the Threshold to 0.9



Actual		
	Has Heart Disease	Doesnot have Heart Disease
Has Heart Disease	3	0
Doesnot have Heart Disease	1	4

a higher threshold:  
Decreases the number of False Positives  
Increases the number of False Negatives

# ROC Curve - example

$$\text{True Positive Rate } TPR = \frac{TP}{TP+FN} = \frac{3}{3+1} = 0.75$$

$$\text{False Positive Rate } FPR = \frac{FP}{TN+FP} = \frac{0}{4+0} = 0$$

		Actual	
		Has Heart Disease	Doesnot have Heart Disease
Has Heart Disease	3	0	
Doesnot have Heart Disease	1	4	

# ROC Curve - example

Thresholds

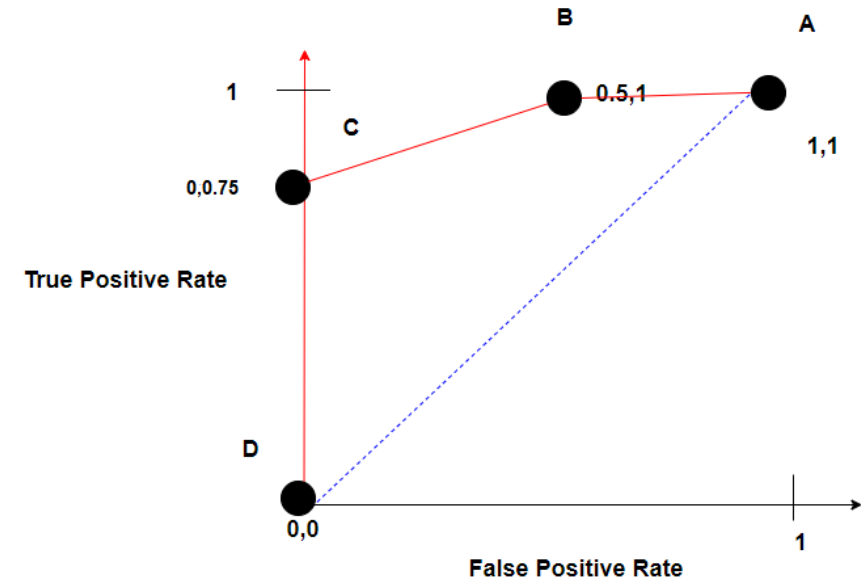
A=0

B=0.1

C=0.9

D=1

$A < B < C < D$





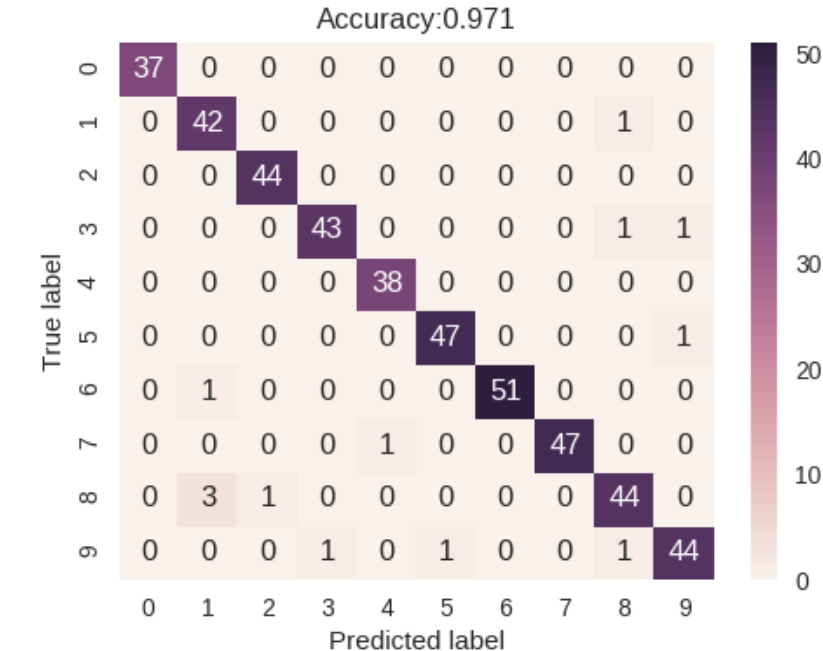
# Metrics for Multiclass Classification

Multi-class evaluation is an extension of the binary case.

- A collection of true vs predicted binary outcomes, one per class
  - Confusion matrices are especially useful
  - Classification report
- 
- Accuracy for multiclass classification is again defined as the fraction of correctly classified examples.

# Metrics for Multiclass Classification

Confusion matrix for the 10-digit classification task



$$Accuracy = \frac{\sum_i diag_i}{N}$$

# Metrics - example

Confusion matrix

	a_B	a_P	a_S	a_T	a_H
p_B	17	3	2	3	2
p_P	1	12	0	1	2
p_S	0	1	16	0	1
p_T	3	3	1	13	1
p_H	0	0	0	0	16

	a_B	a_P	a_S	a_T	a_H
p_B				fn	
p_P				fn	
p_S				fn	
p_T	fp	fp	fp	tp	fp
p_H				fn	

$$Precision_X = \frac{TP_X}{TP_X + FP_X} = \frac{TP_x}{TotalPredicted_x} \rightarrow Precision_T = \frac{13}{13 + 3 + 3 + 1 + 1} = 0.62$$

$$Recall_X = \frac{TP_X}{TP_X + FN_x} = \frac{TP_x}{TotalClass_x} \rightarrow Recall_T = \frac{13}{13 + 3 + 1} = 0.76$$

# Metrics - example

	GoldLabel_A	GoldLabel_B	GoldLabel_C	
Predicted_A	30	20	10	TotalPredicted_A=60
Predicted_B	50	60	10	TotalPredicted_B=120
Predicted_C	20	20	80	TotalPredicted_C=120
	TotalGoldLabel_A=100	TotalGoldLabel_B=100	TotalGoldLabel_C=100	

$$Precision_X = \frac{TP_X}{TP_X + FP_X} \rightarrow Precision_A = \frac{30}{30 + 20 + 10} = 0.5$$

$$Recall_X = \frac{TP_X}{TP_X + FN_X} \rightarrow Recall_A = \frac{30}{30 + 50 + 20} = 0.3$$

# Scikit learn

In[65]:

```
print(classification_report(y_test, pred))
```

Out[65]:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	37
1	0.89	0.91	0.90	43
2	0.95	0.93	0.94	44
3	0.90	0.96	0.92	45
4	0.97	1.00	0.99	38
5	0.98	0.98	0.98	48
6	0.96	1.00	0.98	52
7	1.00	0.94	0.97	48
8	0.93	0.90	0.91	48
9	0.96	0.94	0.95	47
avg / total	0.95	0.95	0.95	450

# Micro vs Macro Average

- "macro" averaging computes the unweighted per-class  $f$ -scores. This gives equal weight to all classes, no matter what their size is.
- "micro" averaging computes the total number of false positives, false negatives, and true positives over all classes, and then computes precision, recall, and  $f$ -score using these counts.

# Micro vs Macro Average

Class	Predicted Class	Correct?
orange	lemon	0
orange	lemon	0
orange	apple	0
orange	orange	1
orange	apple	0
lemon	lemon	1
lemon	apple	0
apple	apple	1
apple	apple	1

## Macro-average:

- Each class has equal weight.

1. Compute metric within each class
2. Average resulting metrics across classes

<u>Class</u>	<u>Recall</u>
orange	$1/5 = 0.20$
lemon	$1/2 = 0.50$
apple	$2/2 = 1.00$
Macro-average $(0.20 + 0.50 + 1.00) / 3 = \mathbf{0.57}$	

# Micro vs Macro Average

Class	Predicted Class	Correct?
orange	lemon	0
orange	lemon	0
orange	apple	0
orange	orange	1
orange	apple	0
lemon	lemon	1
lemon	apple	0
apple	apple	1
apple	apple	1

## Micro-average:

- Each instance has equal weight.
  - Largest classes have most influence
1. Aggregate outcomes across all classes
  2. Compute metric with aggregate outcomes

Micro-average precision:  
 $4 / 9 = \mathbf{0.44}$



# Regression Metrics

- Typically  $R^2$  score is enough
  - Compute how well future instances will be predicted
  - Best possible score is 1.0

*Relative square error (RSE):* 
$$E_{RSE} = \frac{\sum_t (r^t - g(x^t | \theta))^2}{\sum_t (r^t - \bar{r})^2}$$

*Coefficient of determination:* 
$$R^2 = 1 - E_{RSE}$$

# Regression Metrics

Alternative metrics include:

- mean\_absolute\_error
  - absolute difference of target & predicted values
  - $MAE = \frac{1}{N} \sum_t |r^t - g(x^t | \theta)|$
- mean\_squared\_error
  - squared difference of target & predicted values
  - $MSE = \frac{1}{N} \sum_t (r^t - g(x^t | \theta))^2$