Performance Evaluation

Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?
- Methods for Performance Evaluation
  - How to obtain reliable estimates?
- Methods for Model Comparison
  - How to compare the relative performance among competing models?

Metrics for Performance Evaluation

- The results of binary classification: pos, neg
- Confusion matrix: A two-dimensional matrix that allows visualization of the algorithm's performance

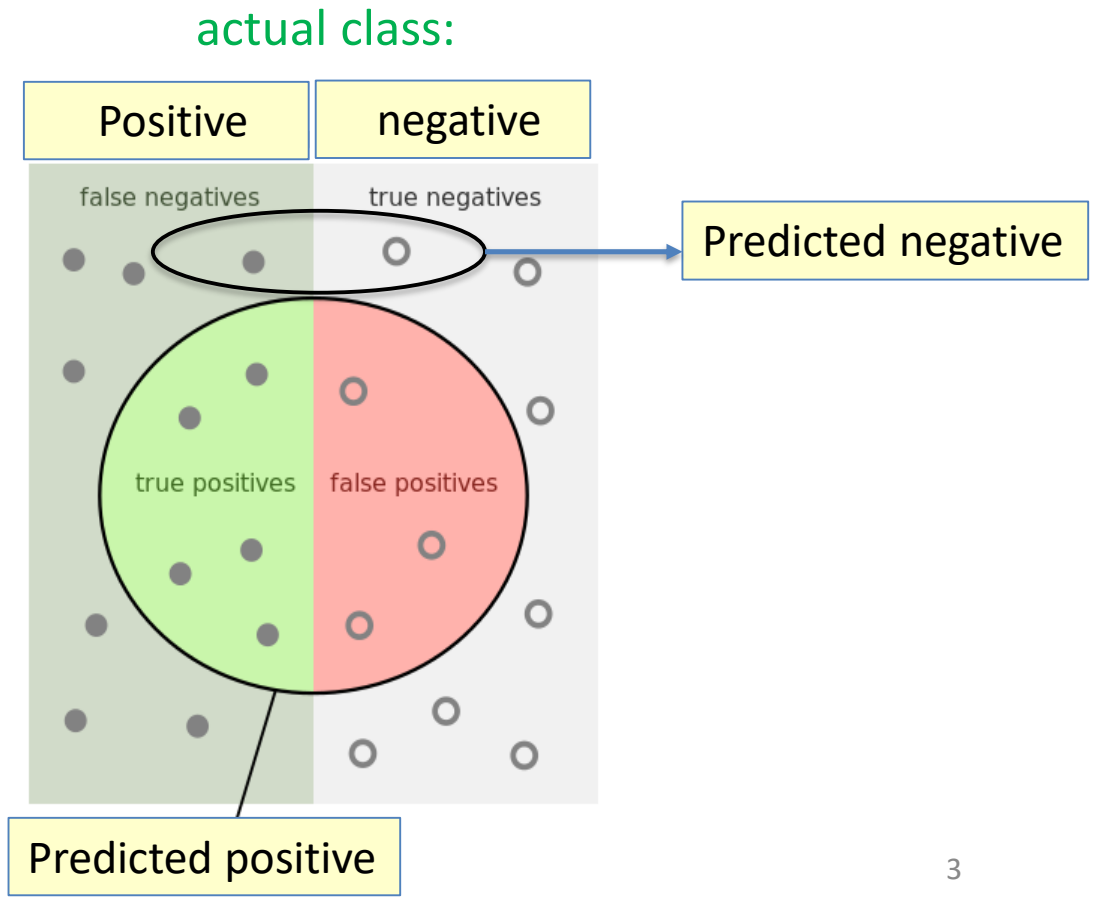
	PREDICTED CLASS		
ACTUAL CLASS		Positive	Negative
	Positive	TP	FN
	Negative	FP	TN

TP (true positive)

FN (false negative)

FP (false positive)

TN (true negative)



Limitations of Accuracy

- Most widely-used metric: accuracy
- Accuracy is defined as the percentage of predictions it got right

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

- When your problem is balanced using accuracy is usually a good start

- However, simple accuracy is not a useful metric.
- Consider a 2-class problem
  - Number of Class negative = 9990
  - Number of Class positive = 10
- If model predicts everything to be class 0, accuracy is 9990/10000 = 99.9 %
  - Accuracy is misleading because model does not detect any class 1 example

- Accuracy is not a good metric when
  1) If the Binary classification problem is biased (most examples are negative or positive)
  2) Or, in multiclass classification, the distribution over labels is non-uniform

Alternative Metrics

- We want to have evaluation metrics that are reflective of the classifier's true performance.
- **Precision:** $\frac{\text{\# (positive identified = true positives)}}{\text{\# (predicted positive)}}$

$$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$$

  - Out of all the examples that predicted as positive, how many are really positive?
- **Recall/sensitivity:** $\frac{\text{\# (positive identified = true positives)}}{\text{\# (all positive)}}$

$$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$$

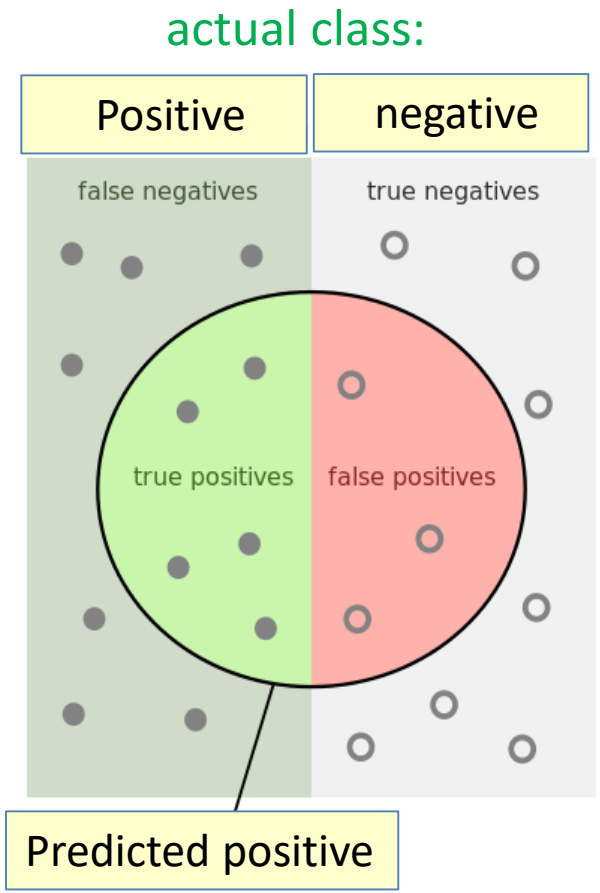
  - Out of all the positive examples, how many are predicted as positive?
- **Specificity/False Positive Rate:** Out of all the negative examples, how many are predicted negative?

$$\frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}$$

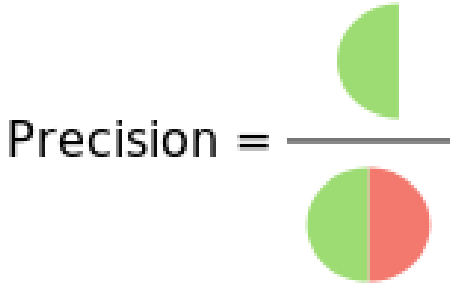
  - Sensitivity and specificity have an inverse relationship

Precision and Recall

- We don’t count true negatives(TN) in precision and recall

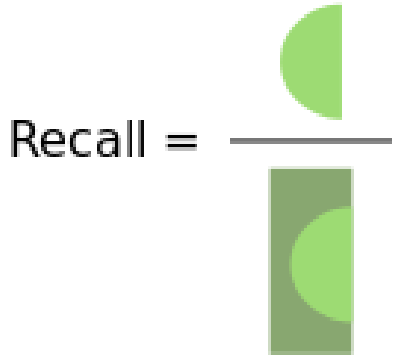


How many selected items are relevant?



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

How many relevant items are selected?



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

Precision

- Precision is a metric that measures how often a machine learning model correctly predicts the positive class.
- Precision answers the question: how often the positive predictions are correct?

Pros:

- It works well for problems with imbalanced classes since it shows the model correctness in identifying the target class.
- Precision is useful when the cost of a **false positive** is high. In this case, you typically want to be confident in identifying the target class, even if you miss out on some (or many) instances.

Cons:

- Precision does not consider **false negatives**. Meaning: it does not account for the cases when we miss our target event!

Recall

- Recall is a metric that measures how often a machine learning model correctly identifies positive instances (true positives) from all the actual positive samples
- Recall answers the question: can an ML model find all instances of the positive class?

Pros:

- It works well for problems with imbalanced classes since it is focused on the model's ability to find objects of the target class.
- Recall is useful when the cost of **false negatives** is high. In this case, you typically want to find all objects of the target class, even if this results in some false positives (predicting a positive when it is actually negative).

Cons:

- Recall is that it does not account for the cost of these **false positives**.

Trading off Precision and Recall

- In logistic regression, the output is given as $f(X)$ (aka prediction score)
  - Classify 1(positive) if $f(X) \geq \theta$ (usually $\theta = 0.5$)
  - Classify 0(negative) if $f(X) < \theta$

- Therefore, at different threshold θ, we have a different confusion matrix
- Increase threshold → Higher precision, lower recall
  - we want to classify y = 1 (e.g., disease) only if we are very confident
- Decrease threshold → Higher recall, lower precision
  - We want to avoid missing too many cases of disease (avoid false negatives)

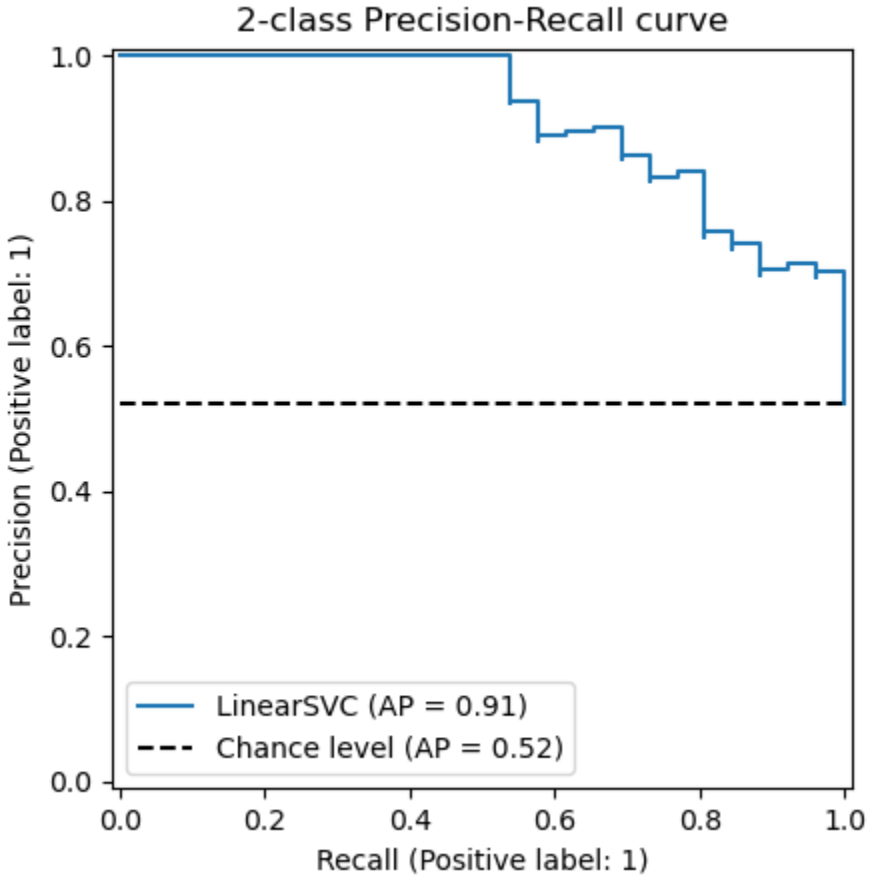
- A system with high recall but low precision returns many results, but most of its predicted labels are incorrect when compared to the training labels.
- A system with high precision but low recall is just the opposite, returning very few results, but most of its predicted labels are correct when compared to the training labels.
- An ideal system with high precision and high recall will return many results, with all results labeled correctly.

Precision and Recall Curve

- A **precision-recall curve** is a plot of the Precision (y-axis) and the Recall (x-axis) for different thresholds, much like the ROC curve.
  - It is a curve that combines Precision and Recall in a single visualization.
  - For every threshold, you calculate Precision and Recall and plot it.

- You can use this curve to make an educated decision when it comes to the classic precision/recall dilemma.

- Obviously, the higher the recall the lower the precision.
- Knowing at which recall your precision starts to fall fast can help you choose the threshold and deliver a better model



Precision and Recall Curve

- We can summarize the information in a precision-recall curve with a single value.
- This summary metric is the AUC-PR (Area Under the Precision-Recall) score
- Generally, the higher the AUC-PR score, the better a classifier performs for the given task.
- A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate.
- High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).

F-measure

- It makes sense to consider Recall and Precision together or combine them into a single metric.
- F-measure (F-score): a measure that combines precision and recall is the harmonic mean of precision and recall.
- The F-score is commonly used for evaluating information retrieval systems such as search engines, and also for many kinds of machine learning models, in particular in natural language processing.
- It is possible to adjust the F-score to give more importance to precision over recall, or vice-versa.
- The more you care about recall over precision, the higher beta you should choose. For example, with F1 score we care equally about recall and precision with F2 score, recall is twice as important to us.

$$F_\beta = (1 + \beta^2) \frac{precision * recall}{\beta^2 * precision + recall}$$

F1-score

- The $\mathbf{F1-score}$ can be used to combine precision and recall

	Precision(P)	Recall (R)	Average	F_1 Score
Algorithm 1	0.5	0.4	0.45	0.444
Algorithm 2	0.7	0.1	0.4	0.175
Algorithm 3	0.02	1.0	0.51	0.0392

The best is Algorithm 1

Algorithm 3 classifies always 1

Average says not correctly that Algorithm 3 is the best

$$\text{Average} = \frac{P + R}{2} \qquad F_1\text{score} = 2\frac{P \cdot R}{P + R}$$

- $P = 0 \text{ or } R = 0 \Rightarrow F_1\text{score} = 0$
- $P = 1 \text{ and } R = 1 \Rightarrow F_1\text{score} = 1$

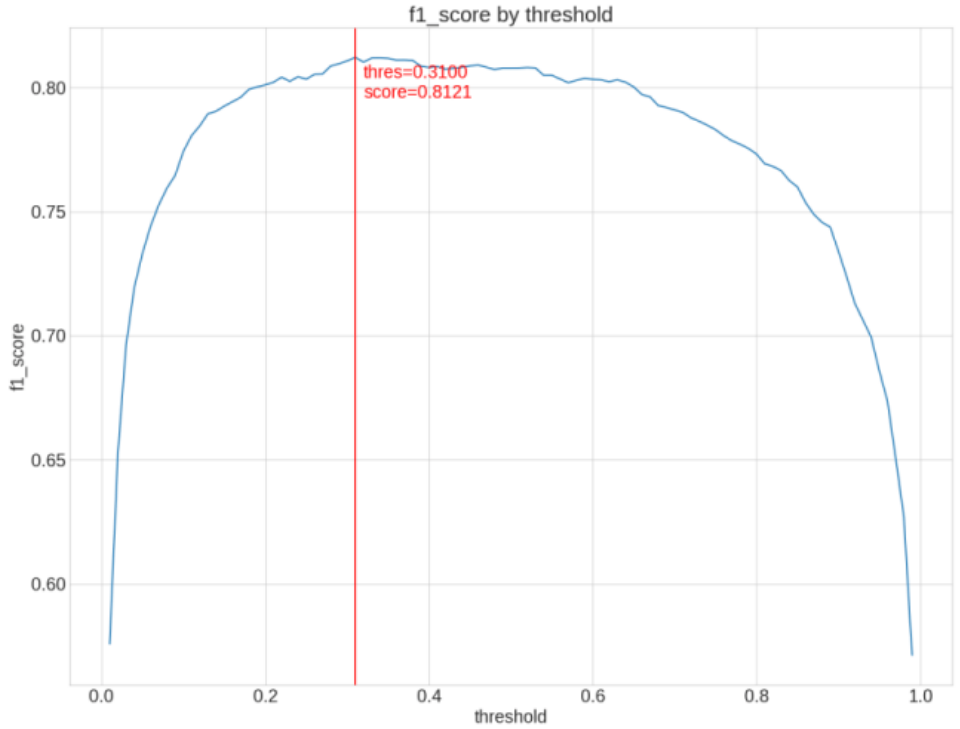
F1-score

- We can use F-score pretty much in every binary classification problem where you care more about the positive class.

- It is important to remember that F1 score is calculated from Precision and Recall which, in turn, are calculated on the predicted classes (not prediction scores).

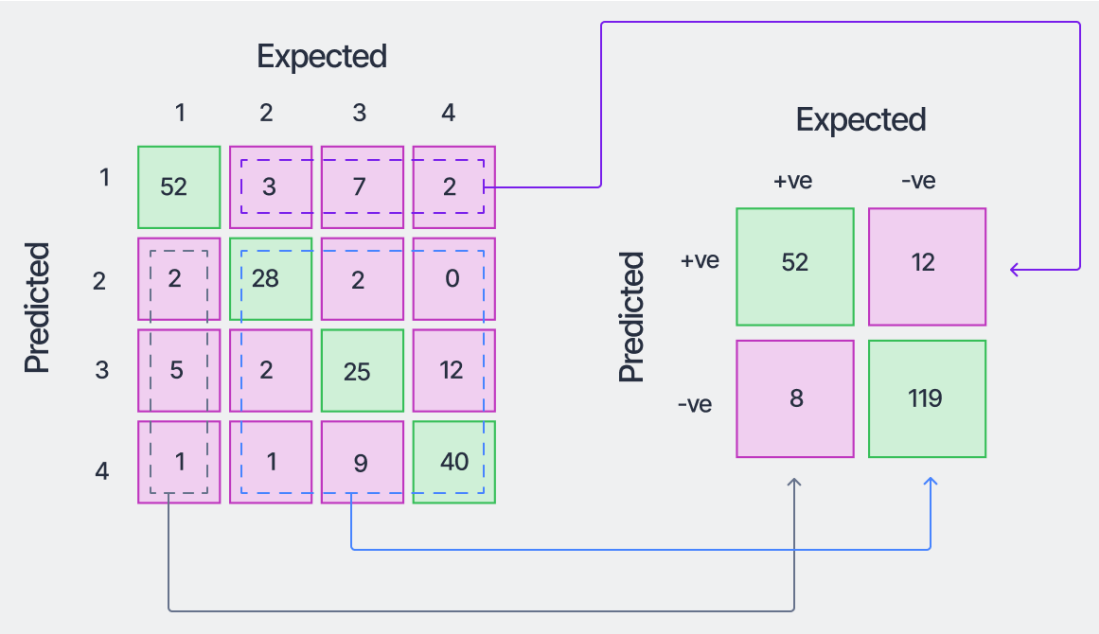
- How should we choose an optimal threshold? Let's plot F1 score over all possible thresholds:

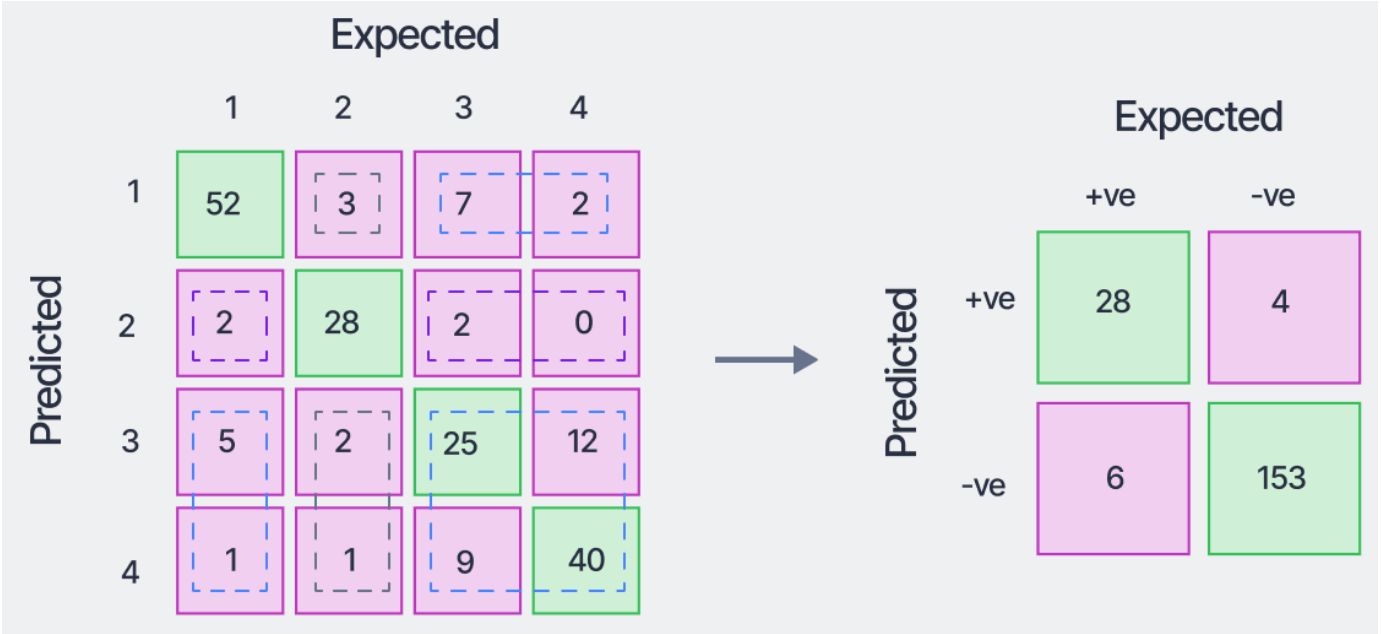
- We can adjust the threshold to optimize the F1 score.
- With $0 < \beta < 1$ we care more about precision and so the higher the threshold the higher the F beta score.
- When $\beta > 1$ our optimal threshold moves toward lower thresholds and with $\beta = 1$ it is somewhere in the middle.



Multiclass Confusion Matrix

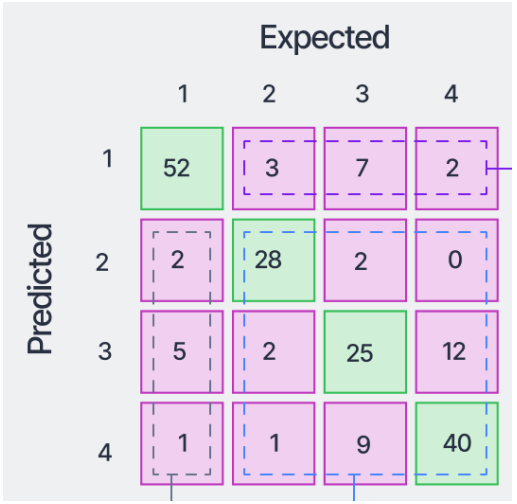
- So far we have seen binary classification problems.
- For multiclass problems, the confusion matrix can be converted into a one-vs-all type matrix (binary-class confusion matrix)
- For each class value, we calculate class-wise metrics like accuracy, precision, recall, etc.





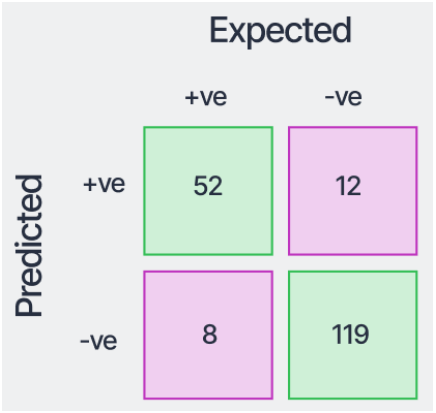
Multiclass Confusion Matrix

- We compute precision and recall (and others) for each class value



Class	Precision (%)	Recall (%)	F1-Score (%)
1	81.25	86.67	83.87
2	87.50	82.35	84.85
3	56.82	58.14	57.47
4	78.43	74.07	76.19

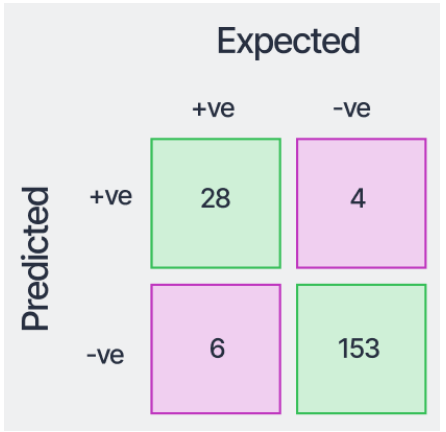
Class=1



precision= $\frac{TP}{TP+FP}$ = 52/(52+12)

recall= $\frac{TP}{TP+FN}$= 52/(52+8)

Class=2



precision= $\frac{TP}{TP+FP}$ = 28/(28+4)

recall= $\frac{TP}{TP+FN}$= 28/(28+6)

Class=3 .....

Confusion Matrix in sklearn

```
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score,
from sklearn.metrics import roc_auc,  average_precision_score

y_pred_class = y_pred_pos > threshold
tn, fp, fn, tp = confusion_matrix(y_true, y_pred_class).ravel()

# accuracy
accuracy = (tp + tn) / (tp + fp + fn + tn)
# f1-score
f1_score(y_true, y_pred_class)
# ROC AUC
roc_auc = roc_auc_score(y_true, y_pred_pos)
# PR AUC
average_precision_score(y_true, y_pred_pos)

from sklearn.metrics import classification_report

y_true = [0, 1, 2, 2, 2]
y_pred = [0, 0, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report(y_true, y_pred, target_names=target_names))
```

	precision	recall	f1-score	support
class 0	0.50	1.00	0.67	1
class 1	0.00	0.00	0.00	1
class 2	1.00	0.67	0.80	3
accuracy			0.60	5
macro avg	0.50	0.56	0.49	5
weighted avg	0.70	0.60	0.61	5

Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?

- Methods for Performance Evaluation
  - How to obtain reliable estimates?

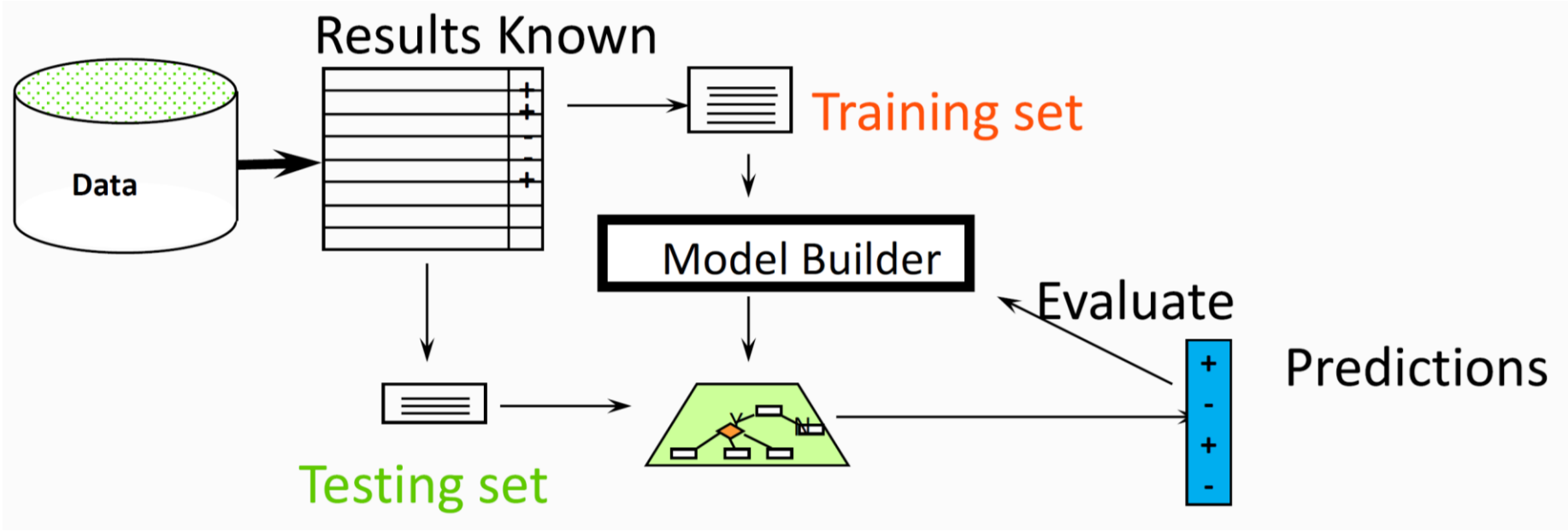
- Methods for Model Comparison
  - How to compare the relative performance among competing models?

Methods of Estimation

- Holdout
  - Reserve ~2/3 for training and ~1/3 for testing
- Random subsampling
  - Repeated holdout
- Cross validation
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one
  - Leave-one-out: k=n

Holdout

- Split data into train and test sets
- Build a model on a training set
- Evaluate on test set



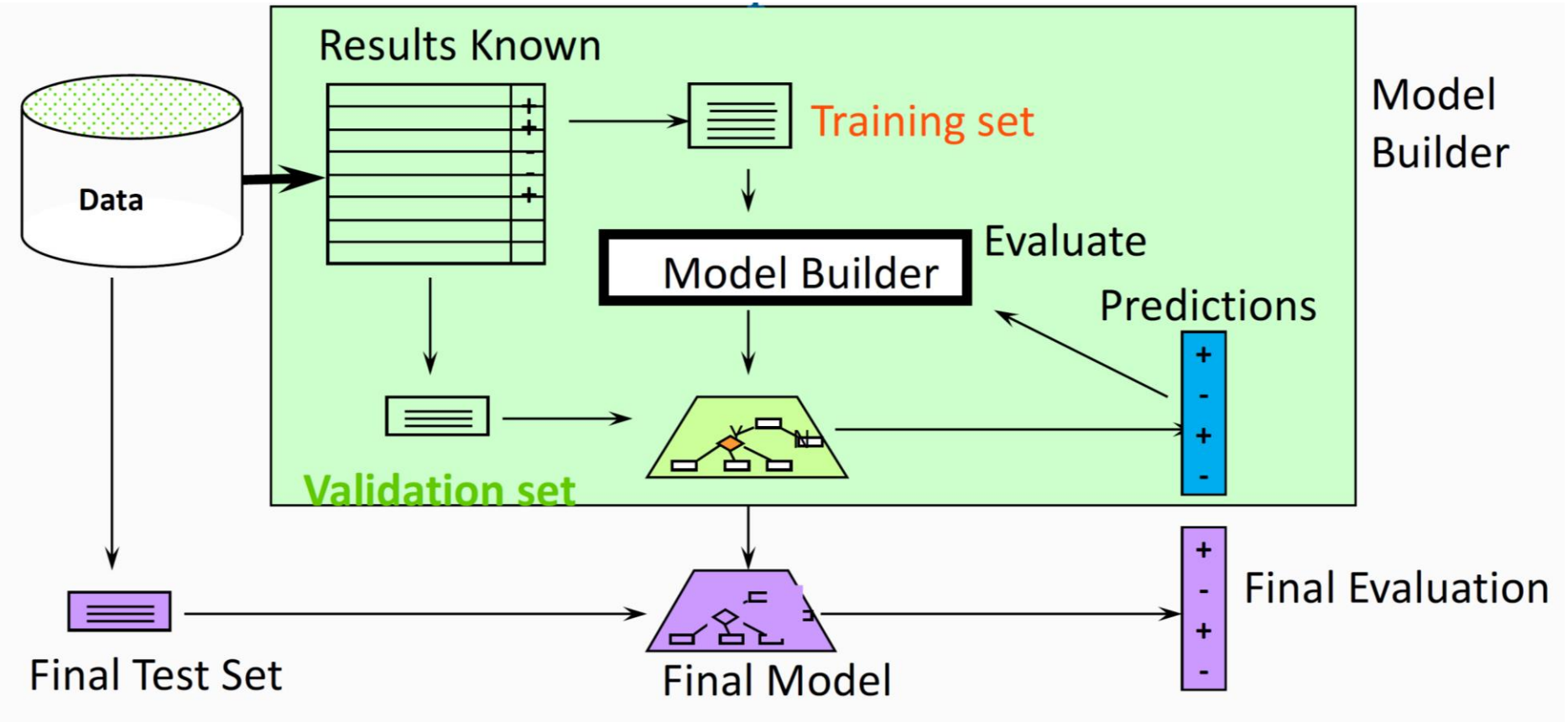
Validation Data for Hyper-parameter Tuning

- It is important that the test data is not used in any way to create the classifier

- Some learning schemes operate in two stages:
  - Stage 1: builds the basic structure
  - Stage 2: optimizes parameter settings (hyper-parameter tuning)

- The test data can’t be used for parameter tuning
- Proper procedure uses three sets: training data, validation data, and test data
  - Validation data is used to optimize parameters

Holdout with Validation Set



Evaluation on “small” data

- The holdout method reserves a certain amount for testing and uses the remainder for training
  - Usually: one third for testing, the rest for training

- For “unbalanced” datasets, samples might not be representative
  - Few or none instances of some classes

- What if we have a small data set?
  - The chosen 2/3 for training may not be representative.
  - The chosen 1/3 for testing may not be representative.

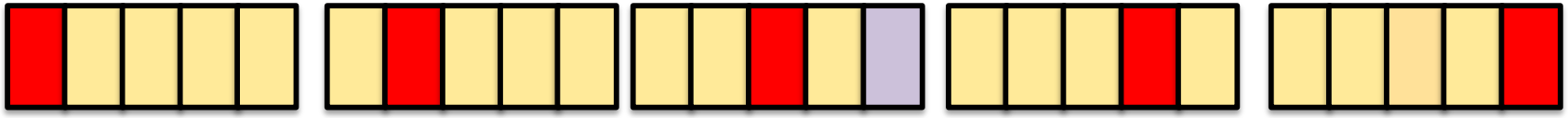
Cross-validation

- Cross-validation avoids overlapping test sets
  - First step: data is split into k subsets of equal size
  - Second step: each subset in turn is used for testing and the remainder for training
- This is called k-fold cross-validation
- Often the subsets are stratified before the cross-validation is performed
- The error estimates are averaged to yield an overall error estimate

k-fold cross validation

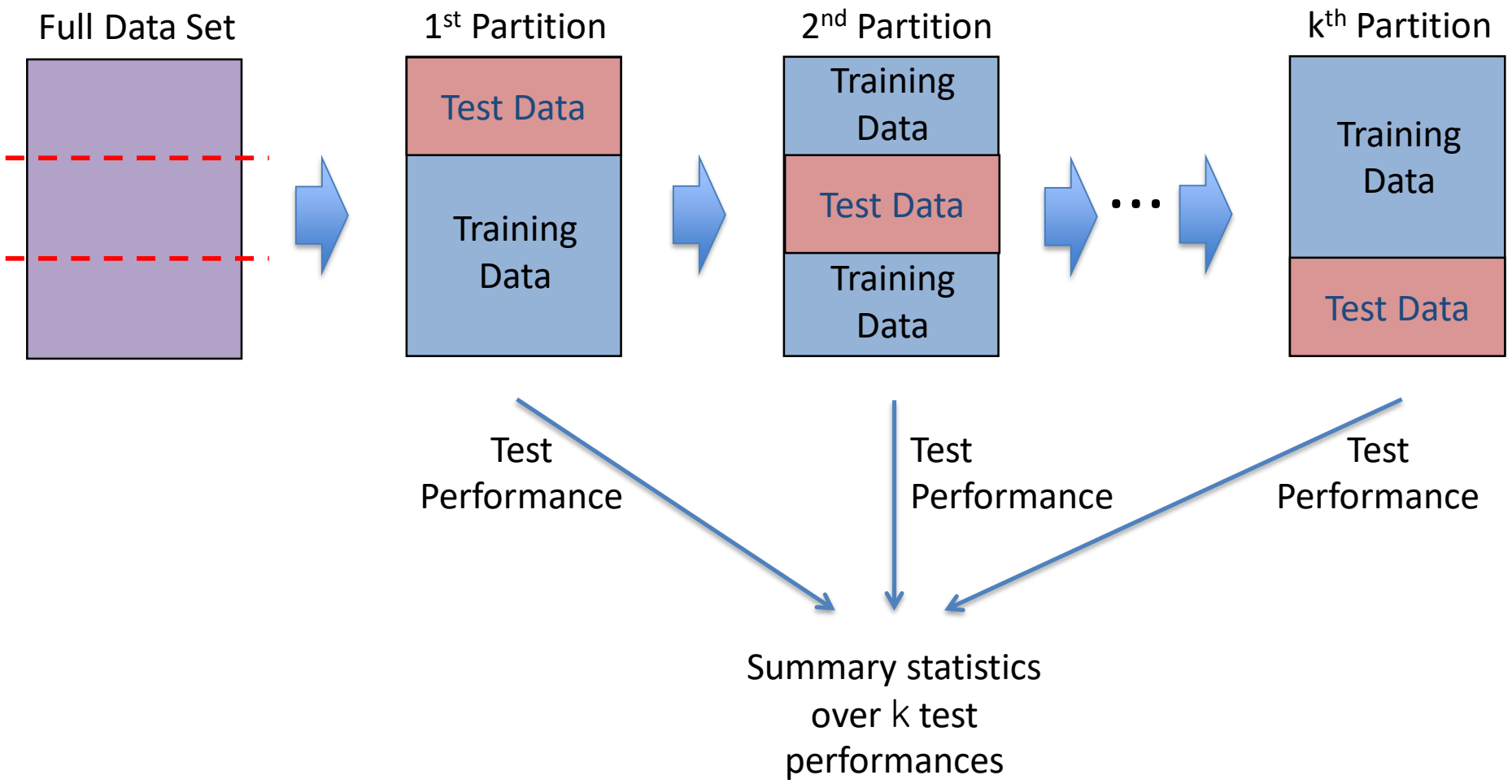
- Instead of a single test-training split:

- Split data into k equal-sized parts



- Train and test k different classifiers
- Report average accuracy and standard deviation of the accuracy

Example 3-Fold CV

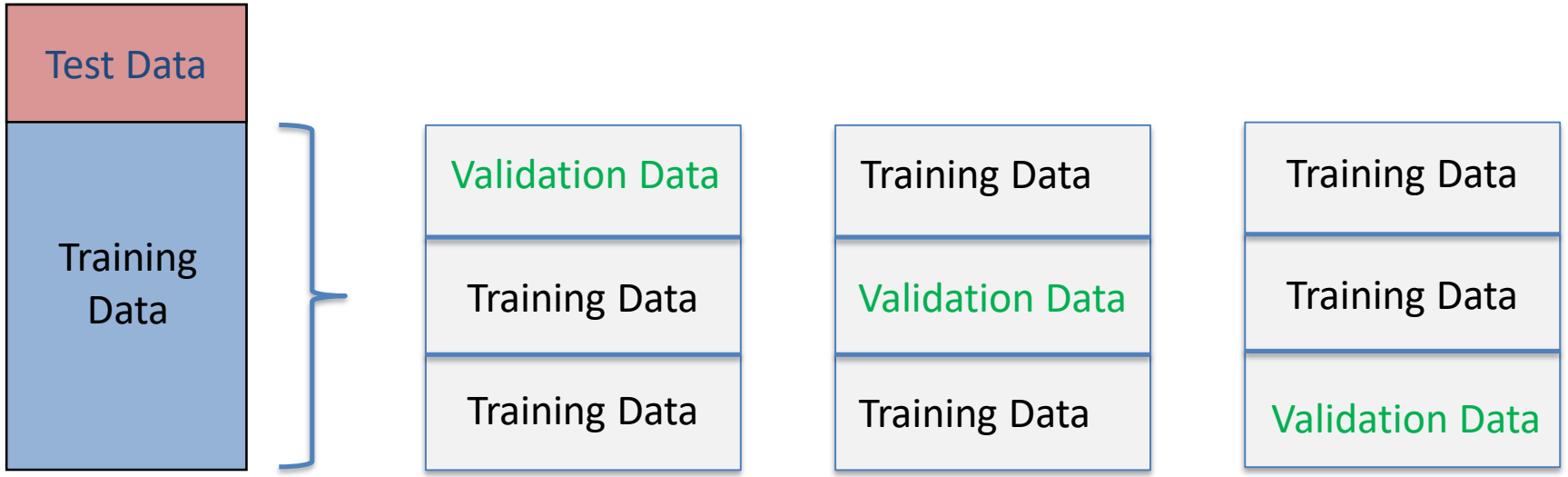


More on Cross-Validation

- Cross-validation(CV) generates an approximate estimate of how well the classifier will do on "unseen" data
  - As $k \to n$, the model becomes more accurate (more training data)
  - but, CV becomes more computationally expensive
  - Choosing $k < n$ is a compromise. k=5 is often used.
  - $k = n$ is called "leave-one-out";

- Averaging over different partitions is more robust than just a single train/validate partition of the data
- It is an even better idea to do CV repeatedly!

Cross-Validation with Validation Set



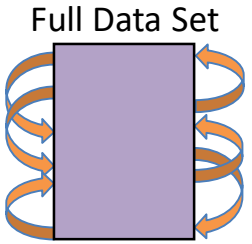
- For ONE training data bucket of k-fold CV, apply CV again and create k sub-buckets (k=3 in above example)
- Train the model using k-1 sub-buckets (training data in above example) with different hyper-parameter values and evaluate with validation data
- Repeat this process for every bucket in k-fold data

Multiple Trials of k-Fold CV

- Repeat k-fold CV t times

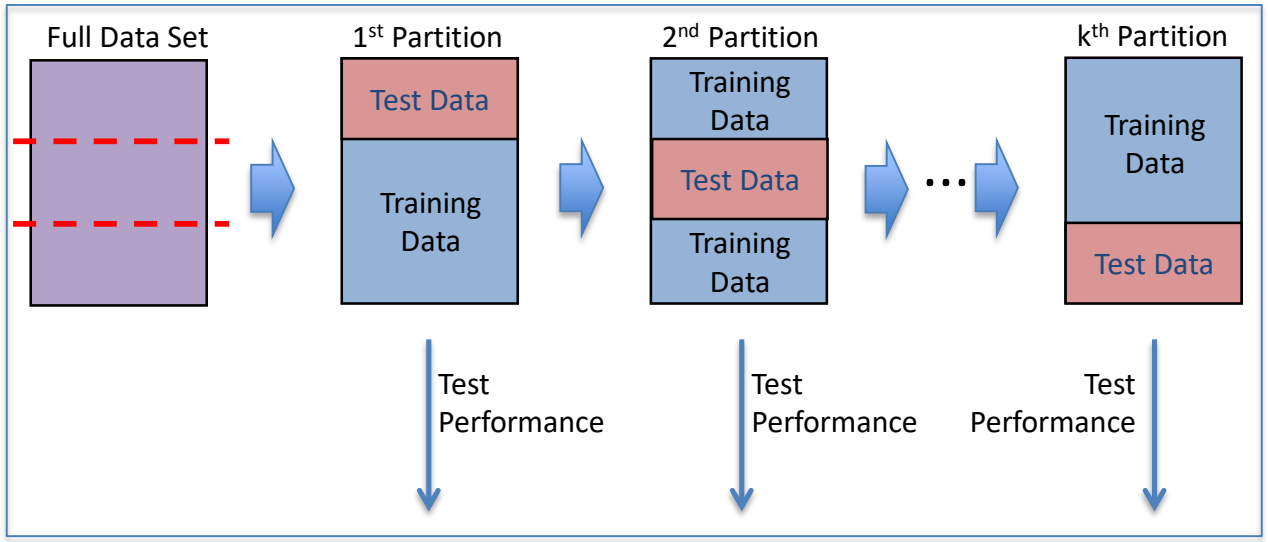
1) Loop for t trials:

   a) Randomize dataset



Shuffle

   b) Perform k-fold CV



2) Compute statistics over $t \times k$ test performances

Leave-One-Out Cross-Validation

- Leave-One-Out: a particular form of cross-validation:
  - Set number of folds to number of training instances
  - I.e., for n training instances, build classifier n times
- Makes best use of the data
- Involves no random subsampling
- Very computationally expensive

Cross-Validation in sklearn

```
from sklearn.model_selection import cross_val_score, RepeatedKFold, LeaveOneOut

clf = svm.SVC(kernel='linear', C=1, random_state=42)
scores = cross_val_score(clf, X, y, cv=5)

# compute accuracies
print(scores)
```

```
array([0.96..., 1. , 0.96..., 0.96..., 1. ])
```

```
X = numpy.array([[1, 2], [3, 4], [1, 2], [3, 4]])
random_state = 12883823
rkf = RepeatedKFold(n_splits=2, n_repeats=2, random_state=random_state)
for train, test in rkf.split(X):
    print("%s %s" % (train, test))
```

```
[2 3] [0 1]
[0 1] [2 3]
[0 2] [1 3]
[1 3] [0 2]
```

```
X = [1, 2, 3, 4]
loo = LeaveOneOut()
for train, test in loo.split(X):
    print("%s %s" % (train, test))
```

```
[1 2 3] [0]
[0 2 3] [1]
[0 1 3] [2]
[0 1 2] [3]
```

Model Evaluation

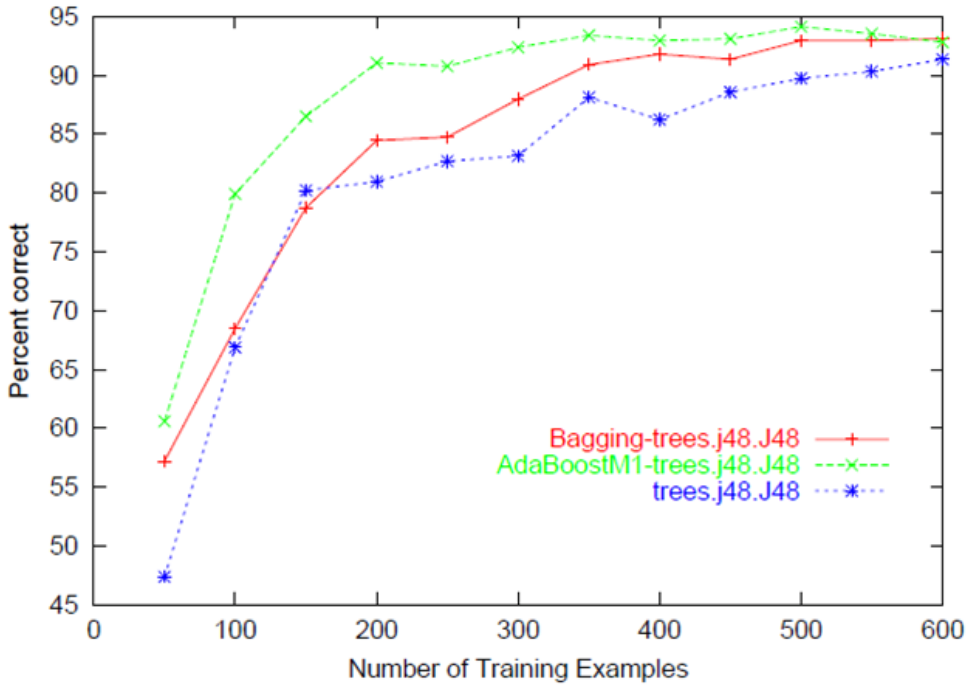
- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?

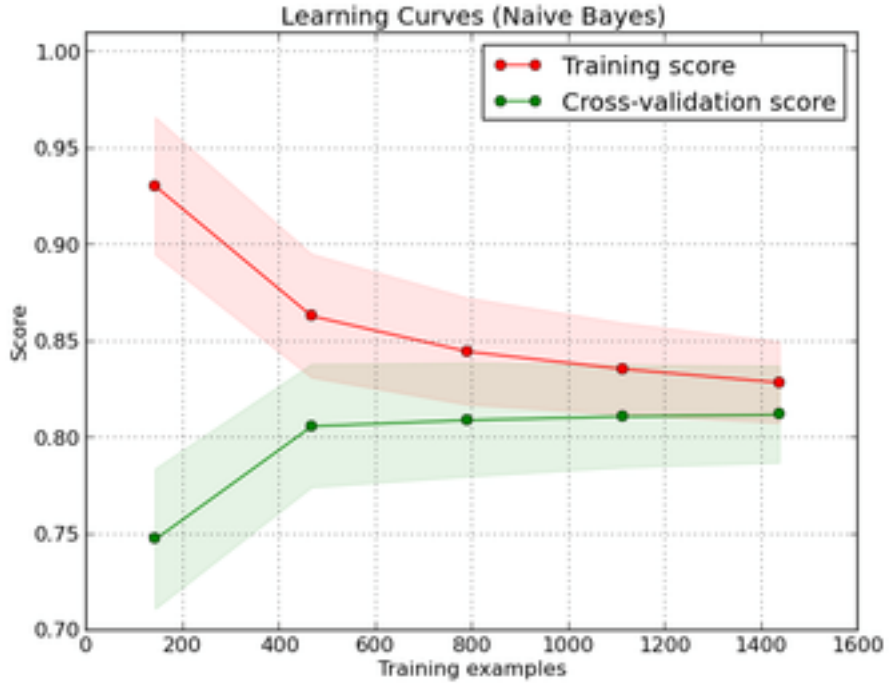
- Methods for Performance Evaluation
  - How to obtain reliable estimates?

- Methods for Model Comparison
  - How to compare the relative performance among competing models?

Learning Curve

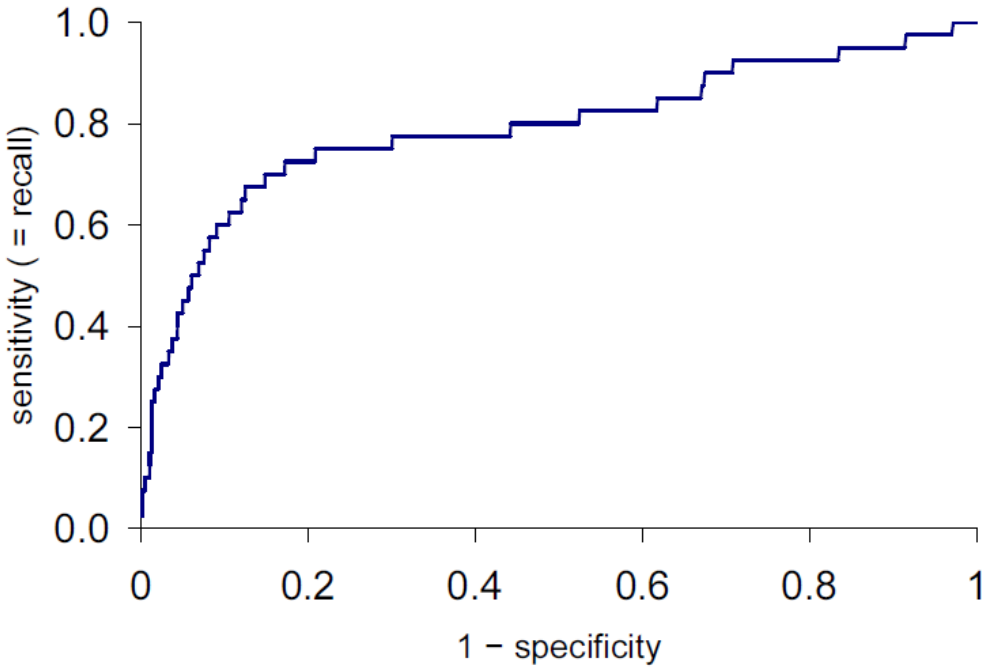
- Shows performance (metric) versus the # training examples
  - Compute over a single training/testing split
  - Then, average across multiple trials of CV





ROC (Receiver Operating Characteristic)

- ROC is Receiver-Operating Characteristic.
- Developed in 1950s for signal detection theory to analyze noisy signals
- 1-dimensional data set containing 2 classes (positive and negative)
- ROC curve plots TP (on the y-axis) against FP (on the x-axis)
- Any points located at x > t is classified as positive
- Performance of each classifier represented as a point on the ROC curve
- Changing the threshold of algorithm, sample distribution or cost matrix changes the location of the point

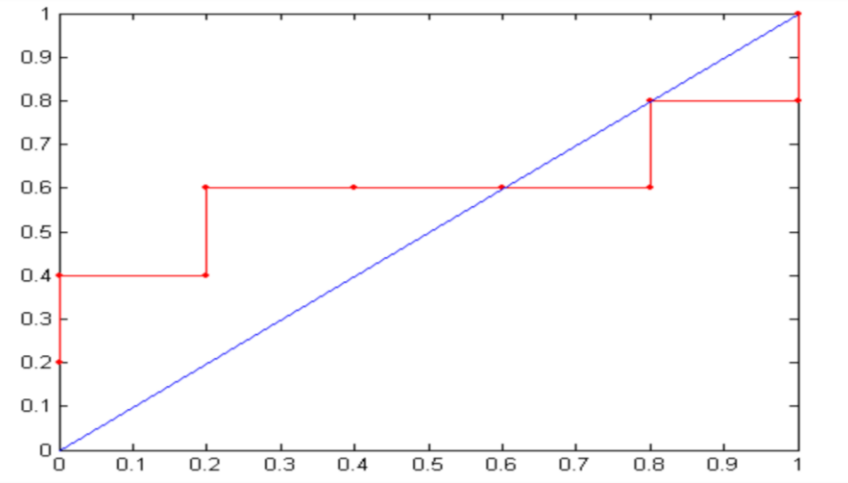


How to Construct an ROC curve

- Use classifier that produces posterior probability (prediction score) for each test instance P(+|A)
- Sort the instances according to P(+|A) in decreasing order
- Apply threshold at each unique value of P(+|A)
- Count the number of TP, FP, TN, FN at each threshold
  - TPR = TP/(TP+FN)
  - FPR = FP/(FP + TN)

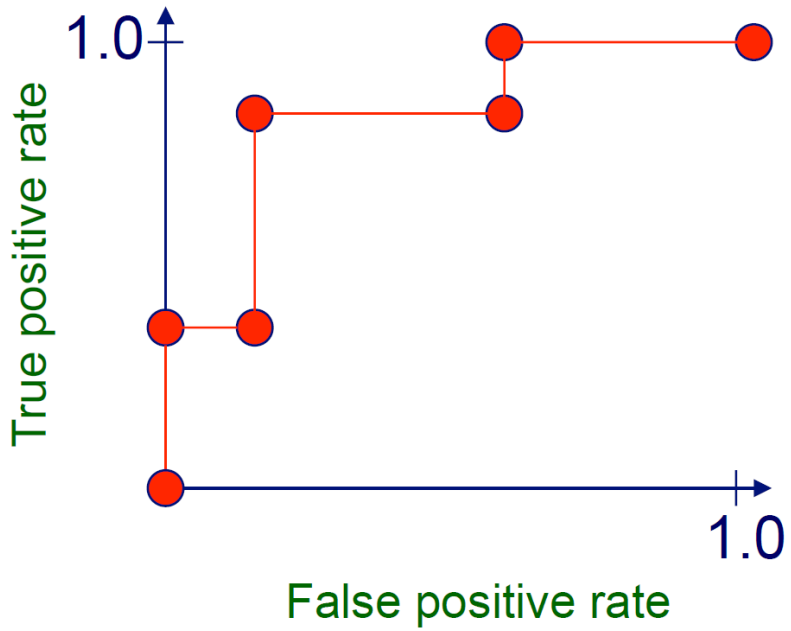
	Class	+	-	+	-	-	-	+	-	+	+	
Threshold >=		0.25	0.43	0.53	0.76	0.85	0.85	0.85	0.87	0.93	0.95	1.00
	TP	5	4	4	3	3	3	3	2	2	1	0
	FP	5	5	4	4	3	2	1	1	0	0	0
	TN	0	0	1	1	2	3	4	4	5	5	5
	FN	0	1	1	2	2	2	2	3	3	4	5
→	TPR	1	0.8	0.8	0.6	0.6	0.6	0.6	0.4	0.4	0.2	0
→	FPR	1	1	0.8	0.8	0.6	0.4	0.2	0.2	0	0	0

ROC Curve:



ROC Example

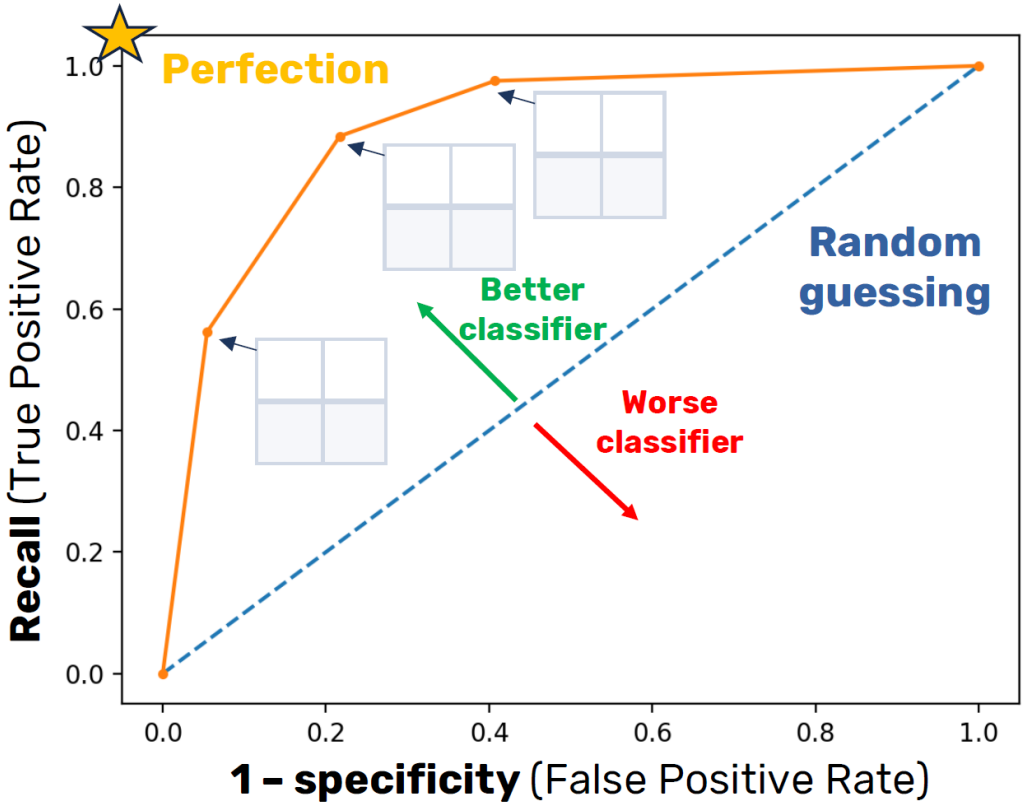
instance	confidence positive		correct class
Ex 9	.99		+
Ex 7	.98	TPR= 2/5, FPR= 0/5	+
Ex 1	.72	TPR= 2/5, FPR= 1/5	-
Ex 2	.70		+
Ex 6	.65	TPR= 4/5, FPR= 1/5	+
Ex 10	.51		-
Ex 3	.39	TPR= 4/5, FPR= 3/5	-
Ex 5	.24	TPR= 5/5, FPR= 3/5	+
Ex 4	.11		-
Ex 8	.01	TPR= 5/5, FPR= 5/5	-



https://pages.cs.wisc.edu/~dpage/cs760/

ROC Curve

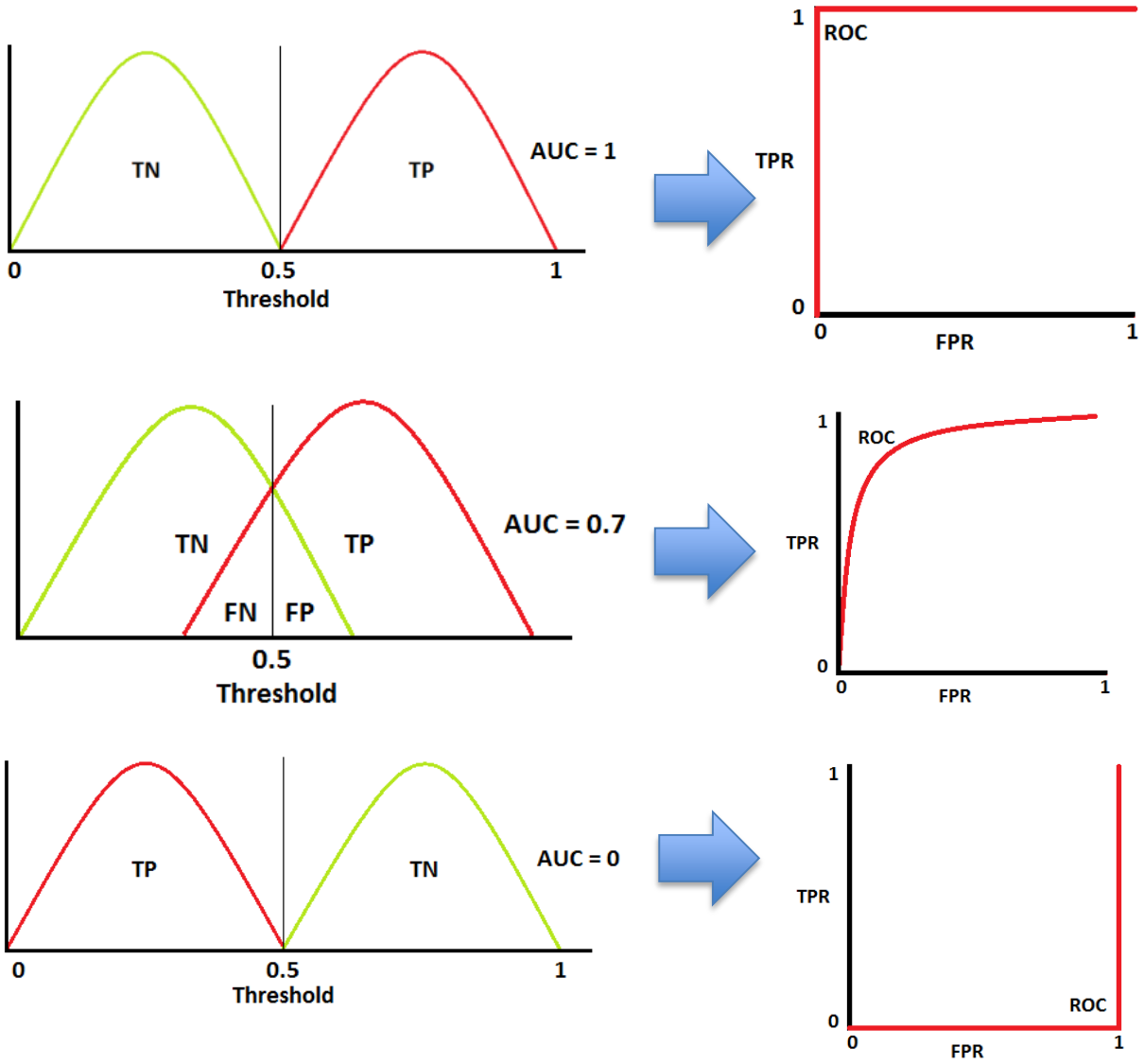
- ROC curves are a very general way to represent and compare the performance of different models (on a binary classification task)



Observations

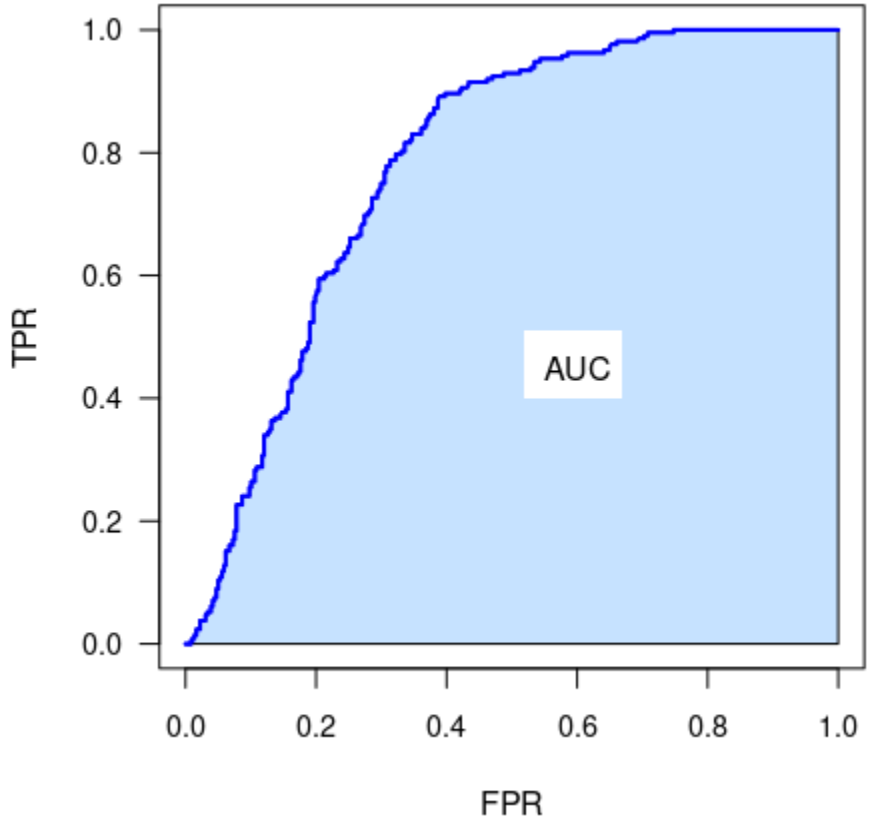
- 0,0 : classify always negative
- 1,1 : classify always positive
- Diagonal line: random classifier
- Below diagonal line: worse than random classifier
- Different classifiers can be compared

Interpretation of ROC



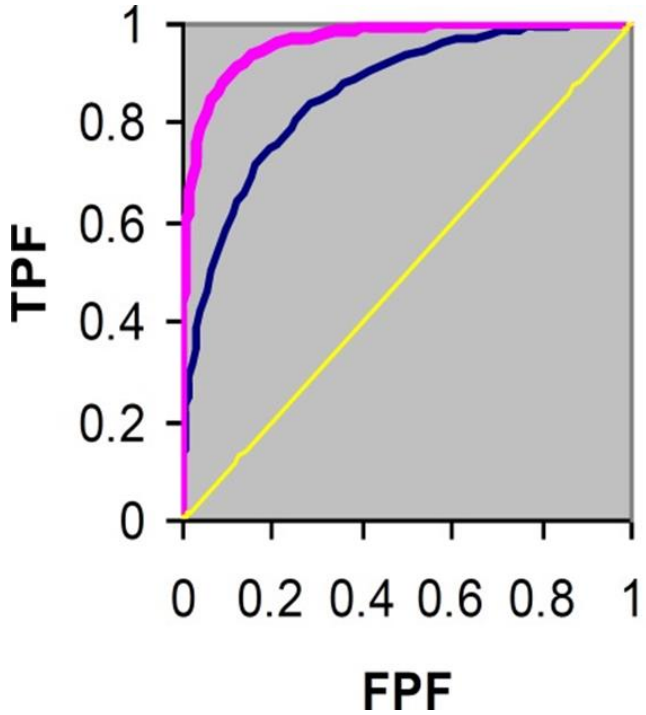
ROC Area Under Curve

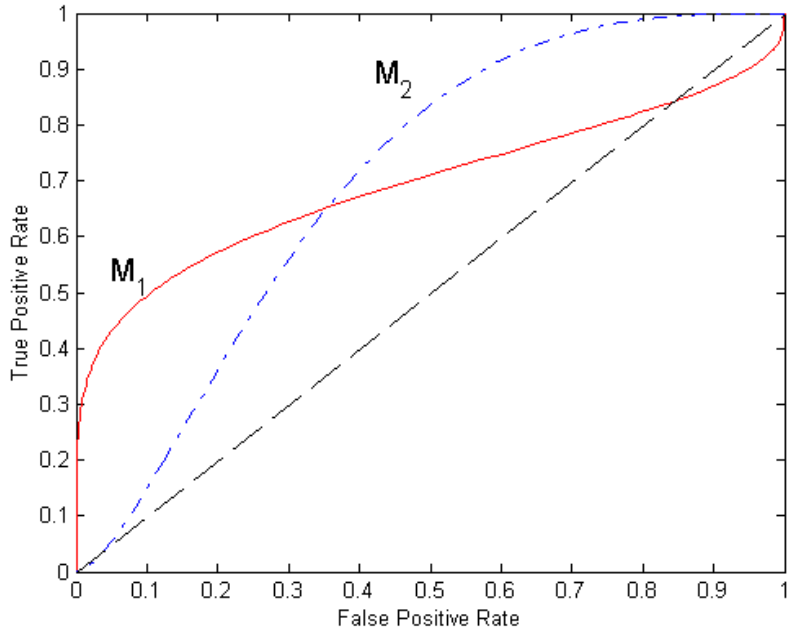
- We can see a healthy ROC curve, pushed towards the top-left side both for positive and negative classes.
- In order to get one number that tells us how good our curve is, we can calculate the Area Under the ROC Curve, or ROC AUC score. The more top-left your curve is the higher the area and hence higher ROC AUC score.
- ROC AUC is a single number that captures the overall quality of the classifier.
- Use it when you care equally about positive and negative classes.
- Do not use it when your data is heavily imbalanced.



Using ROC for Model Comparison

- If two ROC curves do not cross each other, then one of your classifiers is clearly performing better
- For all possible FPR values, you get a higher TPR. Obviously the area under the ROC will be greater.
- if they do cross each other, no model consistently outperform the other
  - M_1 is better for small FPR and M_2 is better for large FPR





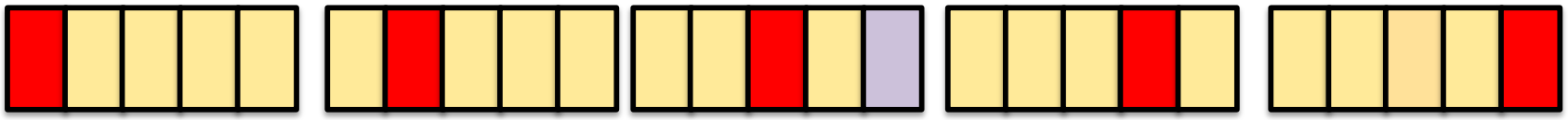
ROC in sklearn

```
import numpy as np
from sklearn import metrics

y = np.array([1, 1, 2, 2])
scores = np.array([0.1, 0.4, 0.35, 0.8])

fpr, tpr, thresholds = metrics.roc_curve(y, scores, pos_label=2)
print(fpr)
print(tpr)
print(thresholds)
```

```
array([0. , 0. , 0.5, 0.5, 1. ])
array([0. , 0.5, 0.5, 1. , 1. ])
array([ inf, 0.8 , 0.4 , 0.35, 0.1 ])
```

Evaluation: Significance tests

- You have two different classifiers, A and B
- You train and test them on the same data set using N-fold cross-validation
- want to determine whether there is a statistically significant difference between two classifiers such that you can actually use only the best one.
- Is the difference between A and B's accuracies significant?



Paired t test

- We can use the paired t-test to find out if there is a real difference between the two classifiers A and B.

- We start by calculating the difference of the accuracies between the two models. (assuming we are interested in the accuracy)
- The accuracy of A on test set *i* is paired with the accuracy of B on test set *i*.

	Test 1	Test 2	Test 3	Test 4
A	85%	81%	89%	78%
B	87%	84%	88%	80%
|A-B|	2%	3%	1%	2%

Paired t test

- Assumption: Accuracies are drawn from a normal distribution (with unknown variance)
- Null hypothesis: The accuracies of A and B are drawn from the same distribution.
  - Hence, the difference in the accuracies has an expected value equal to 0 (E[diff]=0). Basically, there is no difference between the two models.
- Alternative hypothesis: The accuracies are drawn from two different distributions, E[diff] ≠ 0.
  - Basically, the models are actually different, so one is better than the other.

Paired t test

- Take a sample of N observations (given from the k-fold cv). The assumption is that these results come from a normal distribution with fixed mean and variance.
- Compute the sample mean and sample variance for these observations.
- Compute the t-statistic.

- If t-statistic is greater than the critical t-value $t_{N-1,\alpha}$, reject null hypothesis meaning the accuracies of A and B are actually different.

Paired t test

- Now just follow the above steps using the following simple formulas.

$$m = \frac{1}{N}\sum_{i=1}^{N} diff_i$$

$$sd = \sqrt{\frac{\sum_{i=1}^{N}(diff_i - m)^2}{N-1}}$$

$$\text{t} - \text{statistic} = \frac{\left(\sqrt{N} * m\right)}{sd}$$

	Test 1	Test 2	Test 3	Test 4
A	85%	81%	89%	78%
B	87%	84%	88%	80%
|A-B|	2%	3%	1%	2%

$$N = 4$$

$$m = (2 + 3 + 1 + 2)/N = 2$$

$$sd = [((2-2)^2 + (3-2)^2 + (1-2)^2 + (2-2)^2) / (N-1)]\text{^}(1/2) = 0.471$$

$$\text{t} - \text{statistic} = \frac{\left(\sqrt{N} * m\right)}{sd} = 0.816$$

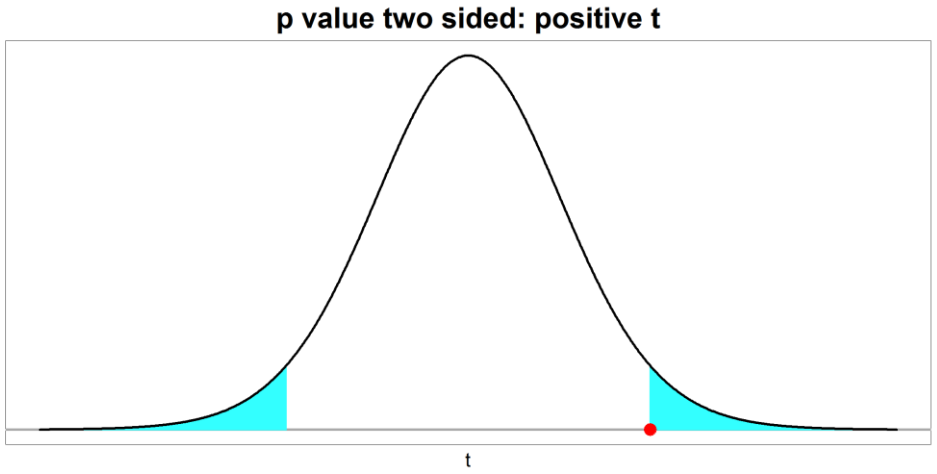
Paired t test

- If we set $\alpha = 0.05$ and a degree of freedom n-1 = 3 , we can check the score on the t-table you can find here.

$$t_{3,0.05} = 3.182$$

- 1) Since t = 0.816 is less than 3.182, we cannot reject the null hypothesis and there is no significant difference between our 2 classifiers.
- 2) p value is the probability of finding the observed t value, given that the null hypothesis is true.

  If p value is less than α, reject the null hypothesis



Paired t test in Python – 1/2

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from mlxtend.data import iris_data
from sklearn.model_selection import train_test_split

x, y = iris_data()
lr = LogisticRegression(max_iter=150)
dt = DecisionTreeClassifier(max_depth = 1)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
lr.fit(x_train, y_train)
dt.fit(x_train, y_train)

# score: Predictions for X_test are compared with y_test and either accuracy (for classifiers) or
# R² score (for regression) estimators is returned.
s1 = lr.score(x_test, y_test)
s2 = dt.score(x_test, y_test)

print('Model A accuracy: %.2f%%' % (s1*100))
print('Model B accuracy: %.2f%%' % (s2*100))
```

Model A accuracy: 95.56%
Model B accuracy: 64.44%

Paired t test in Python – 2/2

```
from mlxtend.evaluate import paired_ttest_5x2cv

t, p = paired_ttest_5x2cv(estimator1=lr, estimator2=dt, X=x, y=y)
alpha = 0.05

print('t statistic: %.3f' % t)
print('alpha: ', alpha)
print('p value: %.3f' % p)

if p > alpha:
  print("Fail to reject null hypotesis")
else:
  print("Reject null hypotesis")
```

```
t statistic: 7.936
alpha:  0.05
p value: 0.001

Reject null hypotesis
```