Evaluating IR Models and Machine Learning Classifiers

What we will study in this lecture

- How to evaluate the performance of an IR model returning a ranked list of documents relevant to a query
- How to evaluate the performance of a machine learning classifier

Evaluating IR Models

Introduction

- To evaluate an IR system is to measure how well the system meets the information needs of the users
  - This is troublesome, given that a same result set might be interpreted differently by distinct users
  - To deal with this problem, some metrics have been defined that, on average, have a correlation with the preferences of a group of users
- Without proper *retrieval evaluation*, one cannot
  - determine how well the IR system is performing
  - compare the performance of the IR system with that of other systems, objectively
- **Retrieval evaluation** is a critical and integral component of any modern IR system

Introduction

- *Retrieval performance evaluation* consists of associating a quantitative metric to the results produced by an IR system
  - This metric should be directly associated with the relevance of the results to the user
  - Usually, its computation requires comparing the results produced by the system with results suggested by humans for a same set of queries

The Cranfield Paradigm

- Evaluation of IR systems is the result of early experimentation initiated in the 50's by Cyril Cleverdon
- The insights derived from these experiments provide a foundation for the evaluation of IR systems
- Back in 1952, Cleverdon took notice of a new indexing system called **Uniterm**, proposed by Mortimer Taube
  - Cleverdon thought it appealing and with Bob Thorne, a colleague, did a small test
  - He manually indexed 200 documents using Uniterm and asked Thorne to run some queries
  - This experiment put Cleverdon on a life trajectory of reliance on experimentation for evaluating indexing systems

The Cranfield Paradigm

- Cleverdon obtained a grant from the National Science Foundation to compare distinct indexing systems
- These experiments provided interesting insights, that culminated in the modern metrics of precision and recall
  - **Recall ratio:** the fraction of relevant documents retrieved
  - **Precision ration:** the fraction of documents retrieved that are relevant
- For instance, it became clear that, in practical situations, the majority of searches does not require high recall
- Instead, the vast majority of the users require just a few relevant answers

The Cranfield Paradigm

- The next step was to devise a set of experiments that would allow evaluating each indexing system in isolation more thoroughly
- The result was a **test reference collection** composed of documents, queries, and relevance judgements
  - It became known as the *Cranfield-2* collection
- The reference collection allows using the same set of documents and queries to evaluate different ranking systems
- The uniformity of this setup allows quick evaluation of new ranking functions

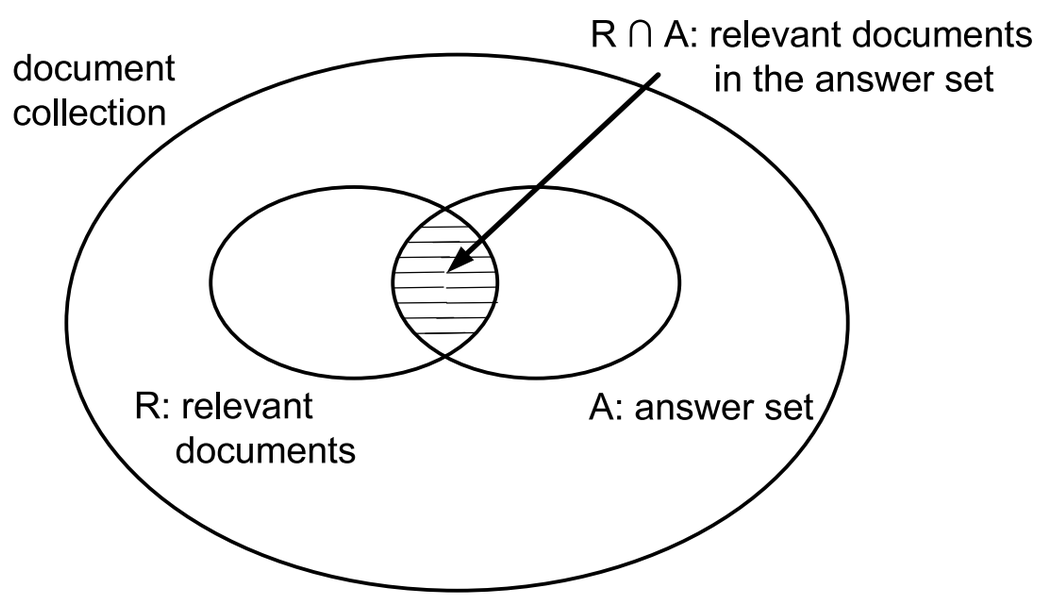
Reference Collections

- Reference collections, which are based on the foundations established by the Cranfield experiments, constitute the most used evaluation method in IR
- A reference collection is composed of:
  - A set $\mathcal{D}$ of pre-selected documents
  - A set I of information need descriptions used for testing
  - A set of relevance judgements associated with each pair $[i_m, d_j]$, $i_m \in \mathcal{I}$ and $d_j \in \mathcal{D}$
- The relevance judgement has a value of 0 if document d_j is non-relevant to i_m, and 1 otherwise
- These judgements are produced by human specialists

Precision and Recall

Precision and Recall

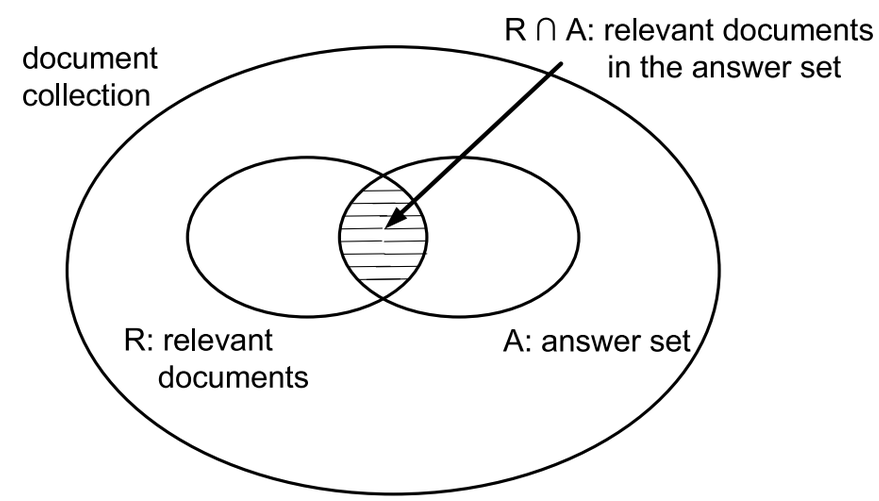
- Consider,
  - I: an information request
  - R: the set of relevant documents for I
  - A: the answer set for I, generated by an IR system
  - $R \cap A$: the intersection of the sets R and A



Precision and Recall

- The recall and precision measures are defined as follows
  - **Recall** is the fraction of the relevant documents (the set R) which has been retrieved i.e.,

$$Recall = \frac{|R \cap A|}{|R|}$$



  - **Precision** is the fraction of the retrieved documents (the set A) which is relevant i.e.,

$$Precision = \frac{|R \cap A|}{|A|}$$

Precision and Recall

- Consider a reference collection and a set of test queries
- Let R_{q_1} be the set of relevant docs for a query q_1:
  - $R_{q_1} = \{d_3, d_5, d_9, d_{25}, d_{39}, d_{44}, d_{56}, d_{71}, d_{89}, d_{123}\}$
- Consider a new IR algorithm that yields the following answer to q_1 (relevant docs are marked with a bullet):

01. d_{123} •	06. d_9 •	11. d_{38}
02. d_{84}	07. d_{511}	12. d_{48}
03. d_{56} •	08. d_{129}	13. d_{250}
04. d_6	09. d_{187}	14. d_{113}
05. d_8	10. d_{25} •	15. d_3 •

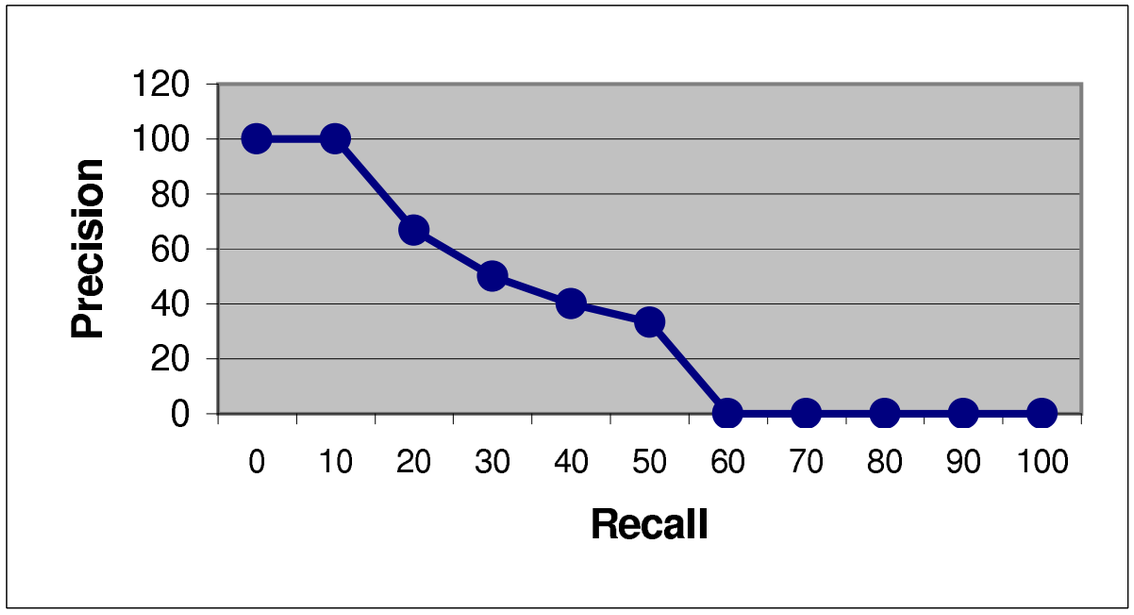
Precision and Recall

- If we examine this ranking, we observe that
  - The document d_{123}, ranked as number 1, is relevant
    - This document corresponds to 10% of all relevant documents
    - Thus, we say that we have a precision of 100% at 10% recall
  - The document d_{56}, ranked as number 3, is the next relevant
    - At this point, two documents out of three are relevant, and two of the ten relevant documents have been seen
    - Thus, we say that we have a precision of 66.6% at 20% recall

01. d_{123} •	06. d_9 •	11. d_{38}
02. d_{84}	07. d_{511}	12. d_{48}
03. d_{56} •	08. d_{129}	13. d_{250}
04. d_6	09. d_{187}	14. d_{113}
05. d_8	10. d_{25} •	15. d_3 •

Precision and Recall

- If we proceed with our examination of the ranking generated, we can plot a curve of precision versus recall as follows:



Recall	Precision
0	100
10	100
20	66.6
30	50
40	40
50	33.3
60	0
70	0
80	0
90	0
100	0

Precision and Recall

- Consider now a second query q_2 whose set of relevant answers is given by

$$R_{q_2} = \{d_3, d_{56}, d_{129}\}$$

- The previous IR algorithm processes the query q_2 and returns a ranking, as follows

01. d_{425}	06. d_{615}	11. d_{193}
02. d_{87}	07. d_{512}	12. d_{715}
03. d_{56} •	08. d_{129} •	13. d_{810}
04. d_{32}	09. d_4	14. d_5
05. d_{124}	10. d_{130}	15. d_3 •

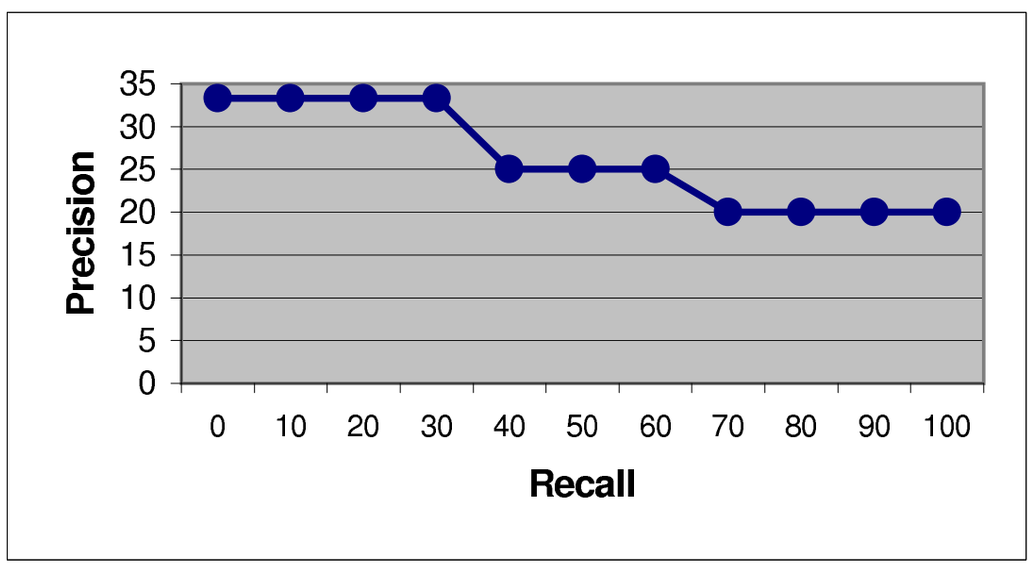
Precision and Recall

- If we examine this ranking, we observe
  - The first relevant document is d_{56}
    - It provides a recall and precision levels equal to 33.3%
  - The second relevant document is d_{129}
    - It provides a recall level of 66.6% (with precision equal to 25%)
  - The third relevant document is d_3
    - It provides a recall level of 100% (with precision equal to 20%)

01. d_{425}	06. d_{615}	11. d_{193}
02. d_{87}	07. d_{512}	12. d_{715}
03. d_{56} •	08. d_{129} •	13. d_{810}
04. d_{32}	09. d_4	14. d_5
05. d_{124}	10. d_{130}	15. d_3 •

Precision and Recall

- The precision figures at the 11 standard recall levels are interpolated as follows
- Let r_j, $j \in \{0, 1, 2, \ldots, 10\}$, be a reference to the j-th standard recall level
- Then, $\boxed{P(r_j) = max_{\forall r \mid r_j \leq r} \; P(r)}$
- In our last example, this interpolation rule yields the precision and recall figures illustrated below



Recall	Precision
0	33.3
10	33.3
20	33.3
30	33.3
40	25
50	25
60	25
70	20
80	20
90	20
100	20

Precision and Recall

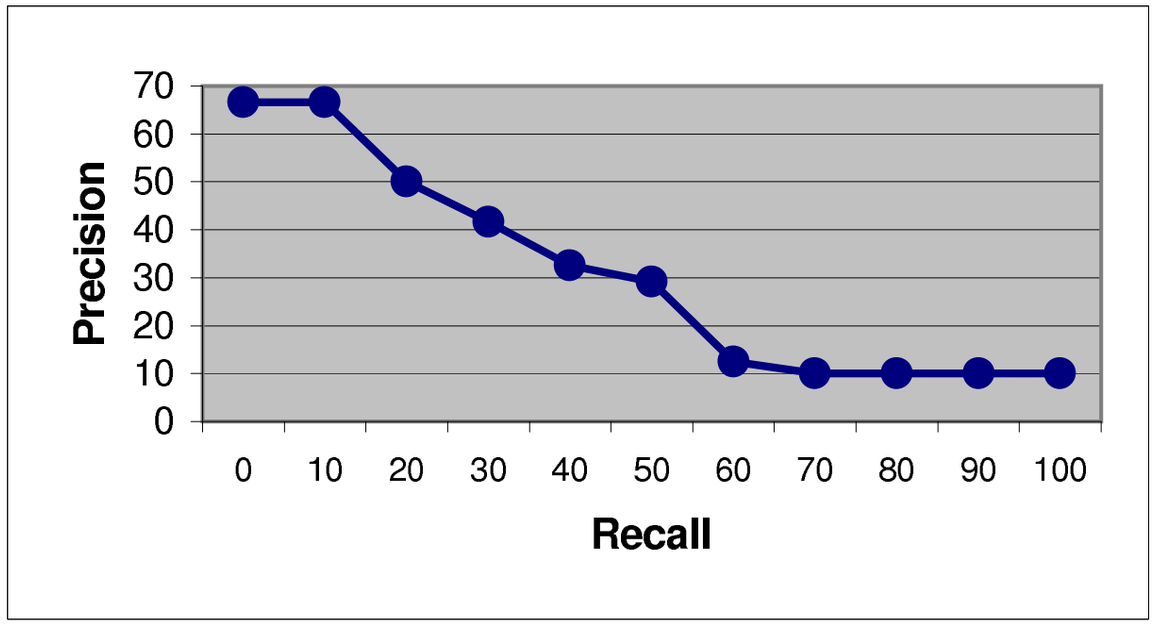
- In the examples above, the precision and recall figures have been computed for single queries
- Usually, however, retrieval algorithms are evaluated by running them for several distinct test queries
- To evaluate the retrieval performance for N_q queries, we average the precision at each recall level as follows

$$\overline{P}(r_j) = \sum_{i=1}^{N_q} \frac{P_i(r_j)}{N_q}$$

- where
  - $\overline{P}(r_j)$ is the average precision at the recall level r_j
  - $P_i(r_j)$ is the precision at recall level r_j for the i-th query

Precision and Recall

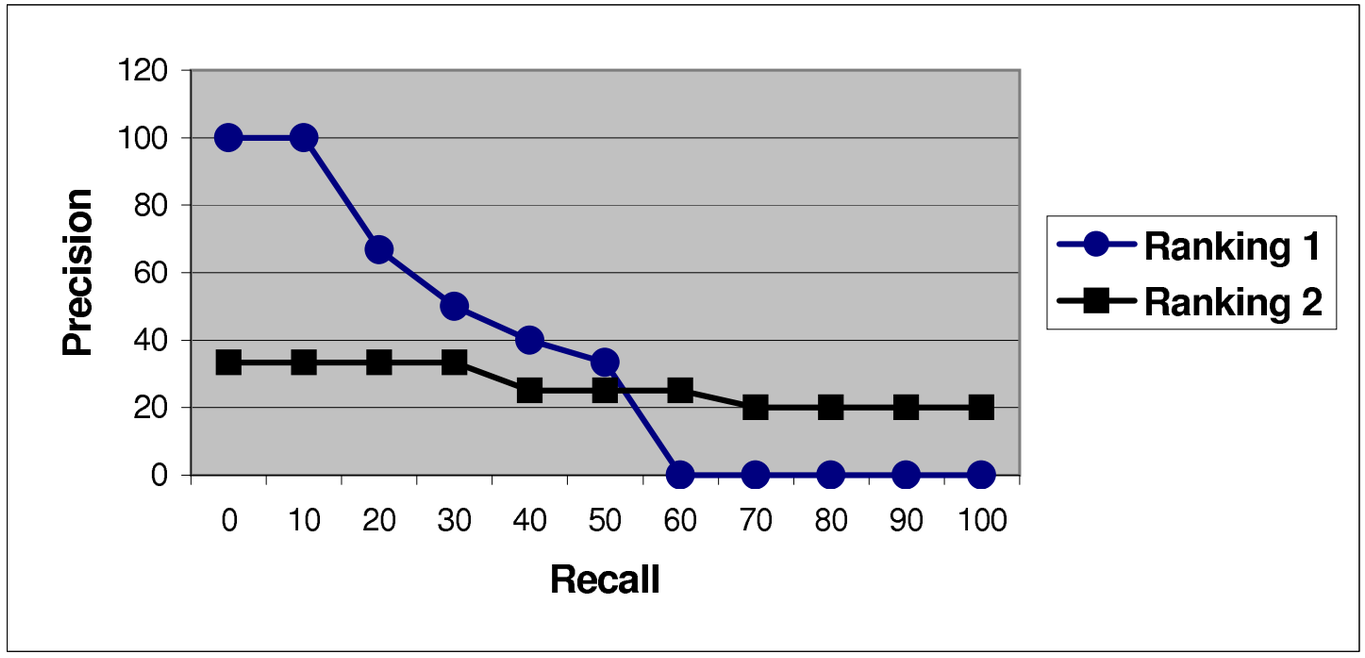
- To illustrate, the figure below illustrates precision-recall figures averaged over queries q_1 and q_2



Recall	Precision
0	66.6
10	66.6
20	49.9
30	41.6
40	32.5
50	29.1
60	12.5
70	10
80	10
90	10
100	10

Precision and Recall

- Average precision-recall curves are normally used to compare the performance of distinct IR algorithms
- The figure below illustrates average precision-recall curves for two distinct retrieval algorithms



Precision-Recall Appropriateness

- Precision and recall have been extensively used to evaluate the retrieval performance of IR algorithms
- However, a more careful reflection reveals problems with these two measures:
  - First, the proper estimation of maximum recall for a query requires detailed knowledge of all the documents in the collection
  - Second, in many situations the use of a single measure could be more appropriate
  - Third, recall and precision measure the effectiveness over a set of queries processed in batch mode
  - Fourth, for systems which require a weak ordering though, recall and precision might be inadequate

Single Value Summaries

- Average precision-recall curves constitute standard evaluation metrics for information retrieval systems
- However, there are situations in which we would like to evaluate retrieval performance over individual queries
- The reasons are twofold:
  - First, averaging precision over many queries might disguise important anomalies in the retrieval algorithms under study
  - Second, we might be interested in investigating whether a algorithm outperforms the other for each query
- In these situations, a single precision value can be used

P@5 and *P@10*

- In the case of Web search engines, the majority of searches does not require high recall
- Higher the number of relevant documents at the top of the ranking, more positive is the impression of the users
- Precision at 5 (*P@5*) and at 10 (*P@10*) measure the precision when 5 or 10 documents have been seen
- These metrics assess whether the users are getting relevant documents at the top of the ranking or not

P@5 and *P@10*

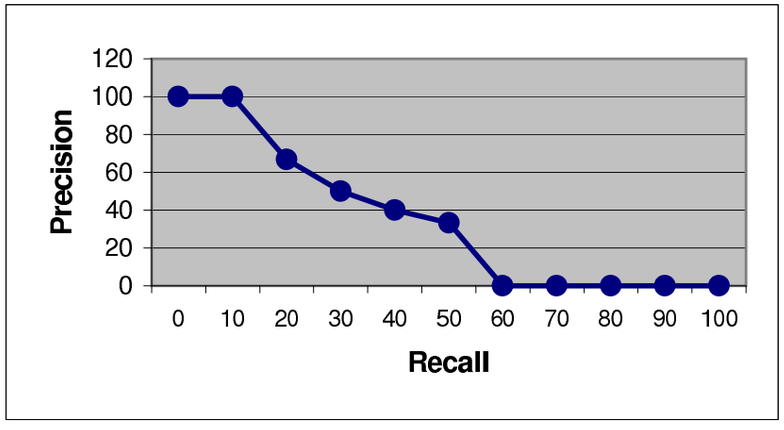
- To exemplify, consider again the ranking for the example query q_1 we have been using:

01. d_{123} •	06. d_9 •	11. d_{38}
02. d_{84}	07. d_{511}	12. d_{48}
03. d_{56} •	08. d_{129}	13. d_{250}
04. d_6	09. d_{187}	14. d_{113}
05. d_8	10. d_{25} •	15. d_3 •

- For this query, we have *P@5* $= 40\%$ and *P@10* $= 40\%$
- Further, we can compute *P@5* and *P@10* averaged over a sample of 100 queries, for instance
- These metrics provide an early assessment of which algorithm might be preferable in the eyes of the users

MAP: Mean Average Precision

- The idea here is to average the precision figures obtained after each new relevant document is observed
  - For relevant documents not retrieved, the precision is set to 0
- To illustrate, consider again the precision-recall curve for the example query q_1



Recall	Precision
0	100
10	100
20	66.6
30	50
40	40
50	33.3
60	0
70	0
80	0
90	0
100	0

- The mean average precision (MAP) for q_1 is given by

$$MAP_1 = \frac{1 + 0.66 + 0.5 + 0.4 + 0.33 + 0 + 0 + 0 + 0 + 0}{10} = 0.28$$

MRR: Mean Reciprocal Rank

- MRR is a good metric for those cases in which we are interested in the first correct answer such as
  - Question-Answering (QA) systems
  - Search engine queries that look for specific sites
    - URL queries
    - Homepage queries

MRR: Mean Reciprocal Rank

- Let,
  - $\mathcal{R}_i$: ranking relative to a query q_i
  - $S_{correct}(\mathcal{R}_i)$: position of the first correct answer in $\mathcal{R}_i$
  - S_h: threshold for ranking position
- Then, the reciprocal rank $RR(\mathcal{R}_i)$ for query q_i is given by

$$RR(\mathcal{R}_i) = \begin{cases} \frac{1}{S_{correct}(\mathcal{R}_i)} & \text{if } S_{correct}(\mathcal{R}_i) \leq S_h \\ 0 & \text{otherwise} \end{cases}$$

- The mean reciprocal rank (MRR) for a set Q of N_q queries is given by

$$\boxed{MRR(Q) = \sum_i^{N_q} RR(\mathcal{R}_i)}$$

MRR: example

q_1

01. d_{123} •
02. d_{84}
03. d_{56} •
04. d_6
05. d_8
06. d_9 •
07. d_{511}
08. d_{129}
09. d_{187}
10. d_{25} •
11. d_{38}
12. d_{48}
13. d_{250}
14. d_{113}
15. d_3 •

q_2

01. d_{425}
02. d_{87}
03. d_{56} •
04. d_{32}
05. d_{124}
06. d_{615}
07. d_{512}
08. d_{129} •
09. d_4
10. d_{130}
11. d_{193}
12. d_{715}
13. d_{810}
14. d_5
15. d_3 •

- For q_1, the first useful document is in position 1
- For q_2, the first useful document is in position 3
- Assuming S_h=5, $MRR = \frac{1}{2} \cdot \left(\frac{1}{1} + \frac{1}{3} \right) = 0.66$
- Assuming S_h=2, $MRR = \frac{1}{2} \cdot \left(\frac{1}{1} + 0 \right) = 0.5$

Evaluating Machine Learning Classifiers

On using accuracy

- So far we have evaluated classifiers using accuracy
- Number of correct prediction / Total number of predictions

Is this sufficient?

Example

- Assume a classifier correctly predicts 99990 out of 100000 newborn babies with a potential genetic defect
- The accuracy is 99.99%
- The error rate is 0.01%
- Is this good?

Not better than a constant classifier

- What if the genetic defect is very rare, and found in only 10 out of 100000 babies?
- A constant classifier that always predicts no defect will have an accuracy of 99.99%!!
- Therefore, the classifier is accurate, but not useful in predicting birth defects

Class imbalance problem

- Associate with data where the distribution of samples across categories is not uniform
- Creates problems in reporting performance
- Also, it creates problems in building an effective model (we'll deal with this later)

Data at our disposal

- Actual class values
- Predicted class values
- Estimated probability of the prediction (when available)

Confidence

- If two models make the same number of mistake, but one is able to assess its uncertainty, then it's a better model

- How to obtain it:

  - instead of the `predict` function, use the predict_proba function

Confidence

From the Spam classification example

```
predicted = clf.predict_proba(X_new_tfidf)

print(predicted)
```

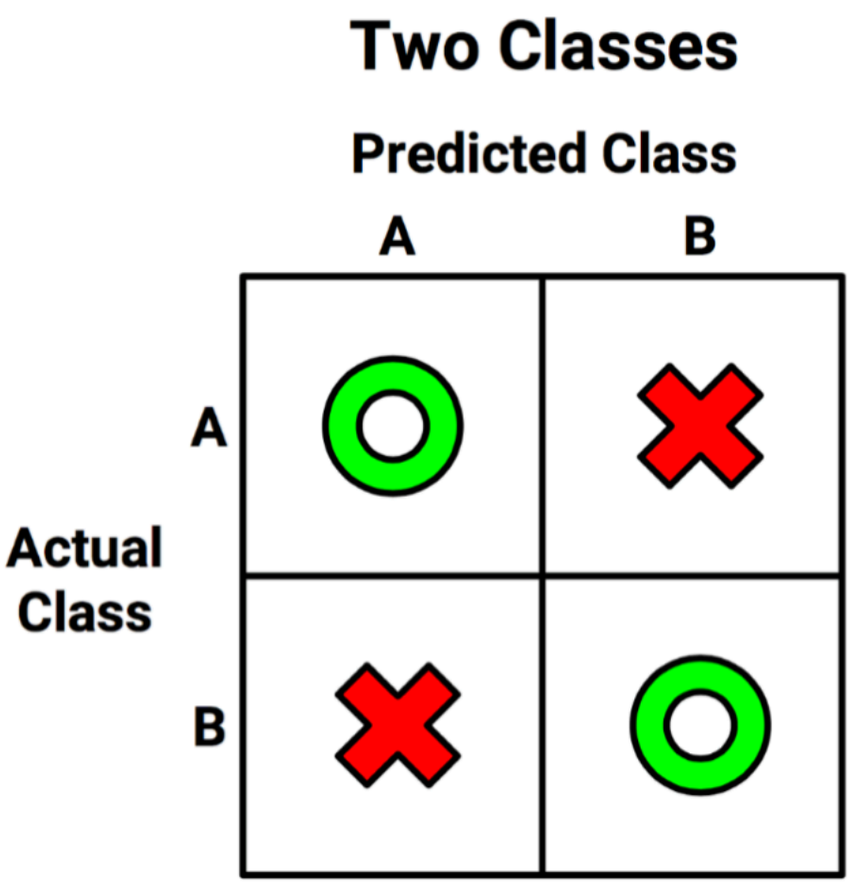
Result (for ham and spam classification)

```
[[0.99584819 0.00415181]
 [0.99175885 0.00824115]
 [0.98112721 0.01887279]
 ...
 [0.58990179 0.41009821]
 [0.9855819  0.0144181 ]
 [0.20580844 0.79419156]]
```

Confusion matrix

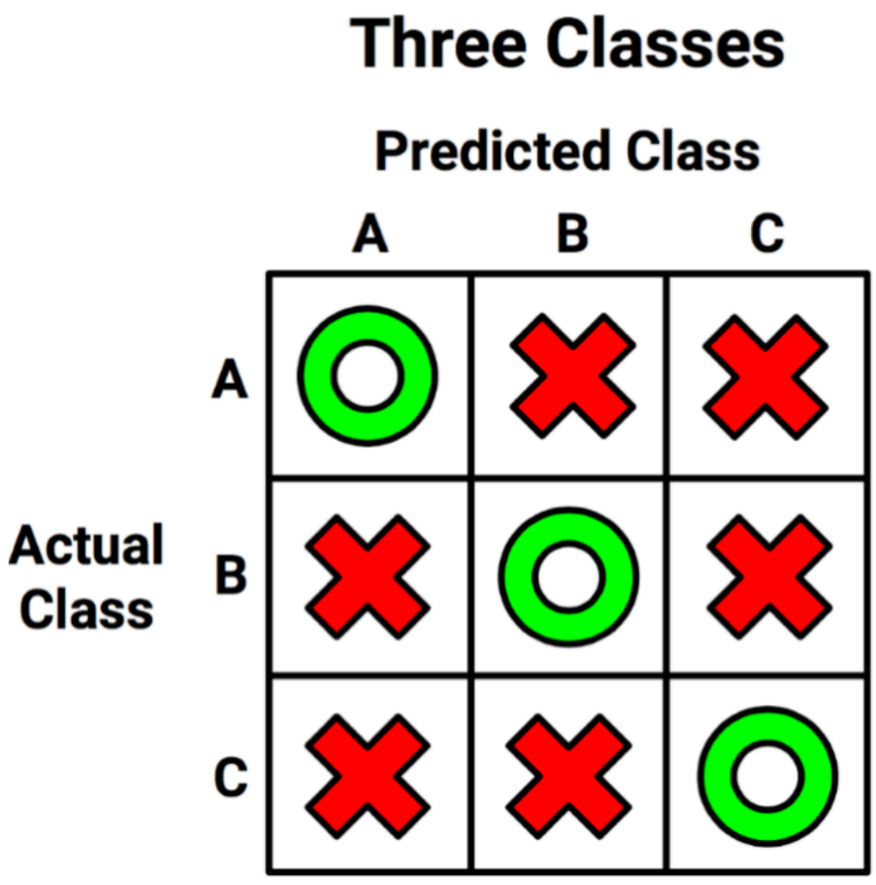
- Table that categorizes prediction according to whether they match actual values
- for a 2-class problem, they are 2x2, but when the classification is across multiple (N) classes, the matrix is NxN

Confusion Matrix



- O indicates the number of correct prediction cases (main diagonal)
- X indicates the number of wrong cases (off-diagonal)

More than 2 categories



- The elements on the main diagonal are called **positives**
- The remaining elements are called **negatives**

Note

- The use of the term positive and negative is not intended to imply any value judgment (good versus bad)
- The choice of the positive outcome depend on the problem, i.e., what are we interested to identify
- Otherwise, if both (or all) classes matter, it can just be arbitrary

Positives and Negatives

- True Positive (TP): correctly classified as the class of interest
- True negative (TN): correctly classified as not the class of interest
- False positive (FP): incorrectly classified as the class of interest
- False negative (FN): incorrectly classified as not the class of interest

For the spam detector

		Predicted to be Spam: no	Predicted to be Spam: yes
Actually Spam	no	TN True Negative	FP False Positive
	yes	FN False Negative	TP True Positive

In the following…

We will use the values from the confusion matrix to define various performance measures

Accuracy

Also called success rate, is the proportion of correctly classified samples:

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

Error rate

Is the proportion of incorrectly classified samples

$$error\ rate = \frac{FP + FN}{TP + TN + FP + FN} = 1 - accuracy$$

How to get the confusion matrix - I

Using sklearn, as a list:

```
from sklearn import metrics
print(metrics.confusion_matrix(y_test,predicted))
```

```
[[1582    1]
 [  71  181]]
```

How to get the confusion matrix - II

Using pandas, as a labeled dataframe

```
import pandas as pd
print(pd.crosstab(y_test,predicted))
```

```
col_0   ham  spam
type
ham    1582     1
spam     71   181
```

Accuracy and error rate

Accuracy:

```
from sklearn import metrics
print(metrics.accuracy_score(y_test,predicted))
```

```
0.9607629427792915
```

Error rate = 1 - Accuracy

```
print(1-metrics.accuracy_score(y_test,predicted))
```

```
0.03923705722070847
```

Sensitivity and Specificity

Sensitivity vs. specificity

- As we have noticed, classifiers may be more or less aggressive in terms of reducing false positives or false negatives, but not both
- To better analyze this tradeoff, we will introduce two new measures, sensitivity and specificity

Sensitivity

- Also called true positive rate
- Proportion of correctly classified positive examples

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

Specificity

- Also called true negative rate
- Proportion of correctly classified negative examples:

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

Computing them from our confusion matrix

pred	ham	spam
actual		
ham	1582	1
spam	71	181

```
sens = 181 / (181 + 71)
```

```
0.718254
```

```
spec = 1582 / (1582 + 1)
```

```
0.9993683
```

How to use sensitivity and specificity

- Change the model and see how sensitivity and specificity change
- Typically, you may set thresholds for them
- We will also see some visualizations for sensitivity and specificity

Precision and Recall

Precision and Recall

- Closely related to sensitivity and specificity
- Widely used in Information Retrieval
- Used to assess the output of a search in an information retrieval system (e.g., a search engine)

Precision

- Known as the positive predictive value
- Proportion of positive examples that are truly positive
- I.e., when a model predicts the positive class, how often is it correct?

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

Precision - discussion

- What happens for a search engine like Google.com if most of the results are incorrect?
- The reader will have to go through a list of many irrelevant links to find a few useful ones
- In other words, the system will not be usable

Recall

- Is a measure of how complete the results are
- The formula is exactly the same as sensitivity

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

For spam: computing from the confusion matrix

pred	ham	spam
actual		
ham	1582	1
spam	71	181

```
prec = 181 / (181 + 1)
```

```
0.9945055
```

```
rec = 181 / (181 + 71)
```

```
0.718254
```

How to compute the precision and recall for the two classes?

pred	ham	spam
actual		
ham	1582	1
spam	71	181

- Prec_ham=1582/(1582+71)=0.9570478
- Prec_spam= 181 / (181 + 1) = 0.9945055

How to aggregate them?

pred	ham	spam
actual		
ham	1582	1
spam	71	181

- Prec_ham=1582/(1582+71)=0.9570478
- Prec_spam= 181 / (181 + 1) = 0.9945055
- Macro-averaged precision=(0.9570478+0.9945055)/2=0.9757767

How to aggregate them?

pred	ham	spam
actual		
ham	1582	1
spam	71	181

- Prec_ham=1582/(1582+71)=0.9570478
- Prec_spam= 181 / (181 + 1) = 0.9945055
- Micro-averaged precision=(1582+181)/(1582+181+71+1)=0.9607629

Macro vs. Micro averaged precision and recall

- Macro-averaged precision and recall is the mean of precision/recall across categories
- Micro-averaged precision and recall are the same, and correspond to accuracy

Using scikitlearn - No averaging

```
print(metrics.recall_score(y_test, predicted,average=None))
```

```
[0.99936829 0.71825397]
```

```
print(metrics.precision_score(y_test,predicted,average=None))
```

```
[0.95704779 0.99450549]
```

Using scikitlearn - Averaging

```
print(metrics.recall_score(y_test, predicted,average='micro'))
print(metrics.precision_score(y_test,predicted,average='micro'))
```

```
0.9607629427792915
0.9607629427792915
```

```
print(metrics.recall_score(y_test, predicted,average='macro'))
print(metrics.precision_score(y_test,predicted,average='macro'))
```

```
0.8588111281573063
0.9757766431995107
```

Complete report

```
print(metrics.classification_report(y_test, predicted))
```

```
Training Sample Size: 3724   Test Sample Size: 1835
              precision    recall  f1-score   support

         ham       0.96      1.00      0.98      1583
        spam       0.99      0.72      0.83       252

    accuracy                           0.96      1835
   macro avg       0.98      0.86      0.91      1835
weighted avg       0.96      0.96      0.96      1835
```

Discussion

- Depending on the application, we may prefer to have a high precision or high recall
- Let’s discuss this!
- Very challenging to achieve both high precision and high recall

The F-Measure

F-Measure

- A single number combining precision and recall
- Sometimes called as F_1-score or F-score
- It is defined as the harmonic mean of precision and recall
- Harmonic mean works better than arithmetic mean because we are aggregating proportions

F-Measure: Definition

$$F - measure = \frac{2 \times precision \times recall}{precision + recall} = \frac{2 \times TP}{2 \times TP + FP + FN}$$

Computing F-Measure

```
print(metrics.f1_score(y_test, predicted,average=None))
```

```
[0.97775031 0.83410138]
```

Be careful!

- While F-measure may seem convenient because it's a single number, avoid abusing it
- Always report performance measures showing strengths and weaknesses of your model

Visualizing performance tradeoffs: The ROC curve

Motivations

- Sensitivity, specificity, precision and recall assess the model using a single number
- With the following visualization, we will see how the model performs across a wide range of conditions

Receiver Operating Characteristics (ROC) curve

- It is possible that two models with similar accuracy may achieve it differently, e.g. minimizing false positives or false negatives
- The Receiver Operating Characteristic (ROC) curve examines the tradeoff between true positive detection and avoidance of false positives

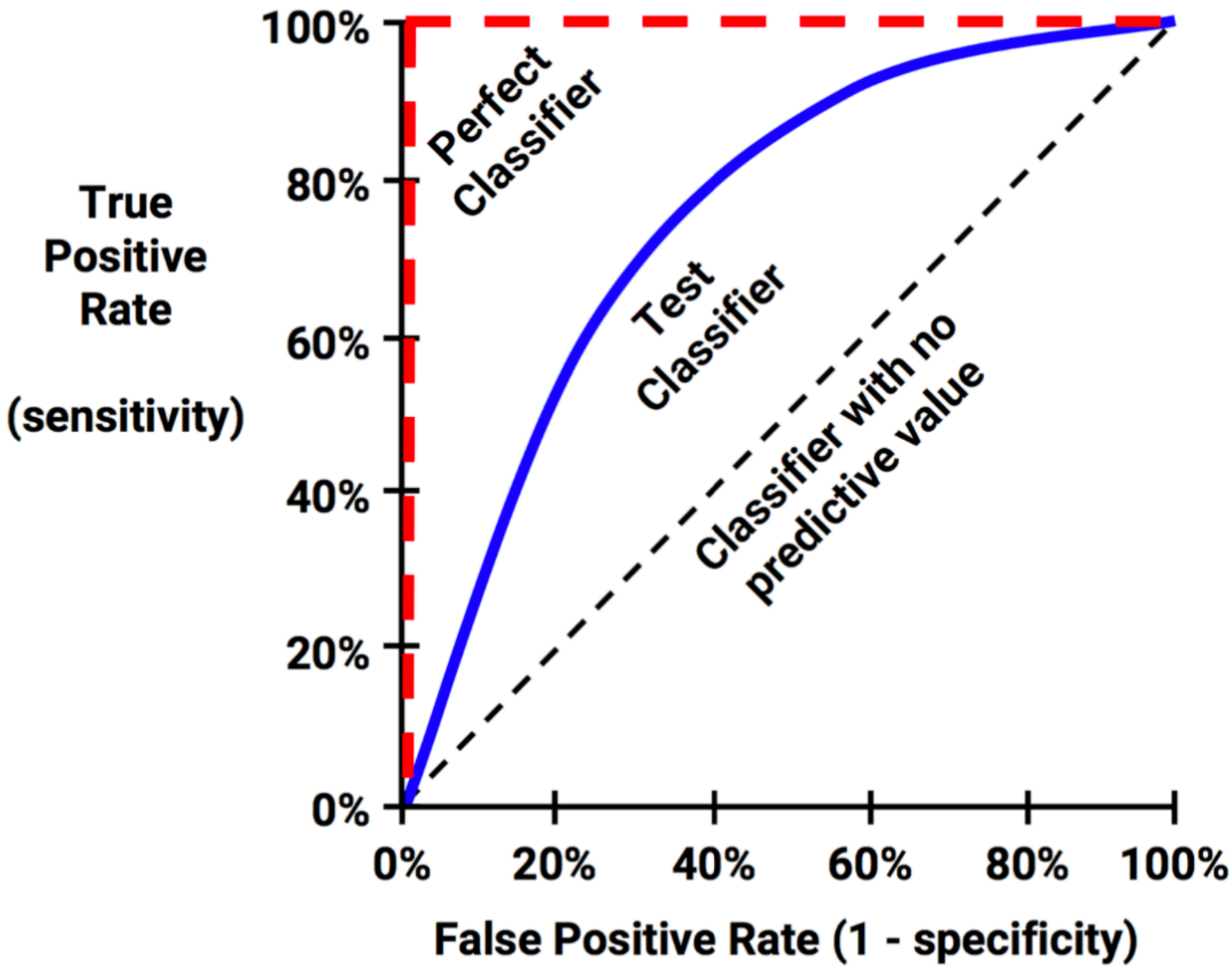
Some history

- Used to assess the ability of radars and radios to discriminate between true signals and false alarms
- Widely used today to assess the efficacy of machine learning models

How it works

- The figure uses the proportion of true positives (vertical axis) and the proportion of false positive (horizontal axis)
- These are equivalent to sensitivity and (1-specificity)
- Therefore, the diagram is also called sensitivity/specificity plot

The ROC curve



Discussion

- Classifier’s predictions are sorted by their estimated probability of positive classification, with largest values first
- Starting from the origin, each prediction’s impact on true positive and false positive rate will result on the curve tracing vertically (correct prediction) or horizontally (incorrect prediction)

Baselines

- A perfect classifier has 0% false positive rate with 100% true positive rate
- A classifier with no predictive value is a 45° line, i.e., it cannot discriminate
- Baseline for comparison (good classifiers should go above the 45° line)

Area under the ROC curve (AUC)

- Single number derived from the curve
- It’s the integral of the curve itself, i.e., the area under it
- For the 45° baseline, AUC=0.5
- For the perfect classifier, AUC=1
- AUC>0.5 go better than the baseline

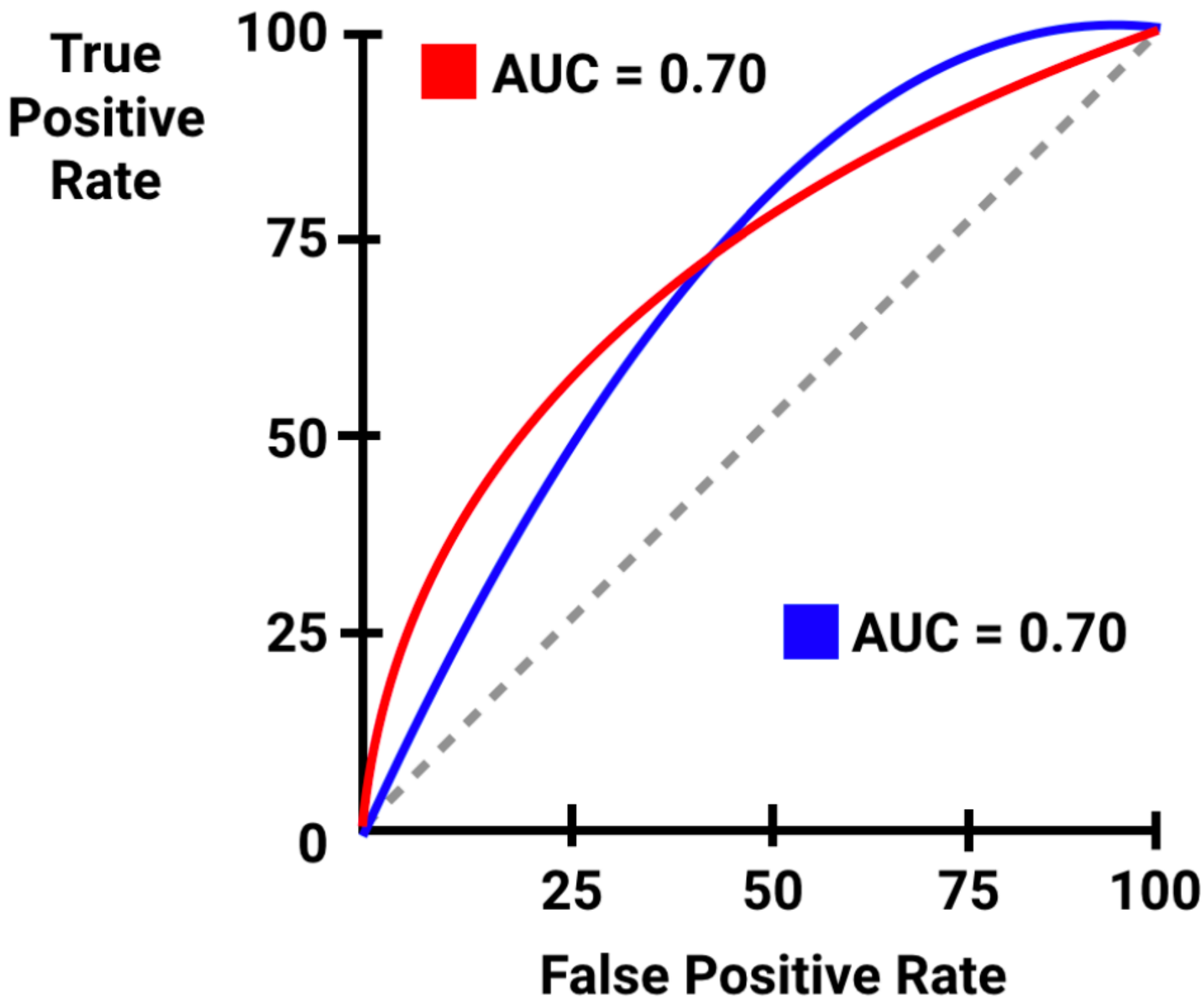
AUC scale

- A: Outstanding: 0.9 to 1.0
- B: Excellent/Good: 0.8 to 0.9
- C: Acceptable/Fair: 0.7 to 0.8
- D: Poor: 0.6 to 0.7
- E: No discrimination: 0.5 to 0.6

Note

The scale described before may be subjective, and the actual assessment depends from case to case

Different curves, same AUC



ROC by hand

- you need the classifier probabilities, rather than the classification
- then, you perform a classification for different cutoff thresholds (normally, it may be 0.5)
- once done it, compute, and store, sensitivity (as y value) and 1-specificity (as x value)
- finally, do a plot of y by x

For example

- If you have a sample where:
  P(ham)=0.4 and P(spam)=0.6

- With the thresholds
  0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6 0.7, 0.8, 0.9, 1

- The spam classification will be
  False, False, False, False, False, False, False, True, True, True, True

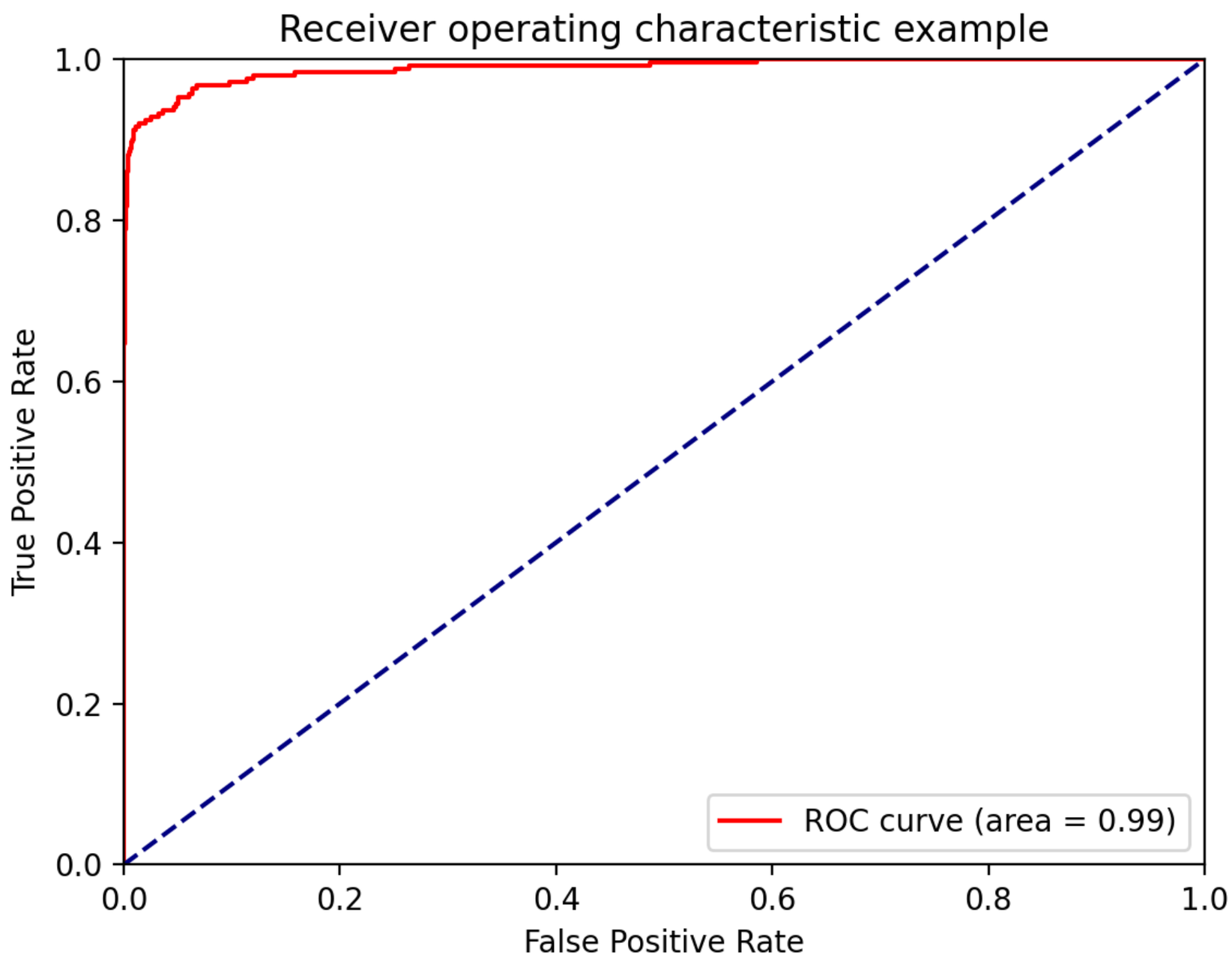
- Therefore the sensitivity and specificity can be computed accordingly

Implementation in Python

```
import matplotlib.pyplot as plt
predicted_prob = clf.predict_proba(X_new_tfidf)
print(y_test=='spam')
print(predicted_prob[:,1])
fpr,tpr,thresholds=metrics.roc_curve(y_test=='spam',predicted_prob[:,1])
roc_auc=metrics.auc(fpr, tpr)

plt.plot(
    fpr,
    tpr,
    color="red",
    label="ROC curve (area = %0.2f)" % roc_auc,
)
plt.plot([0, 1], [0, 1], color="navy", linestyle="--")
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("Receiver operating characteristic example")
plt.legend(loc="lower right")
plt.show()
```

Result


Receiver operating characteristic example
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (area = 0.99)

Estimating future performance

Resubstitution error

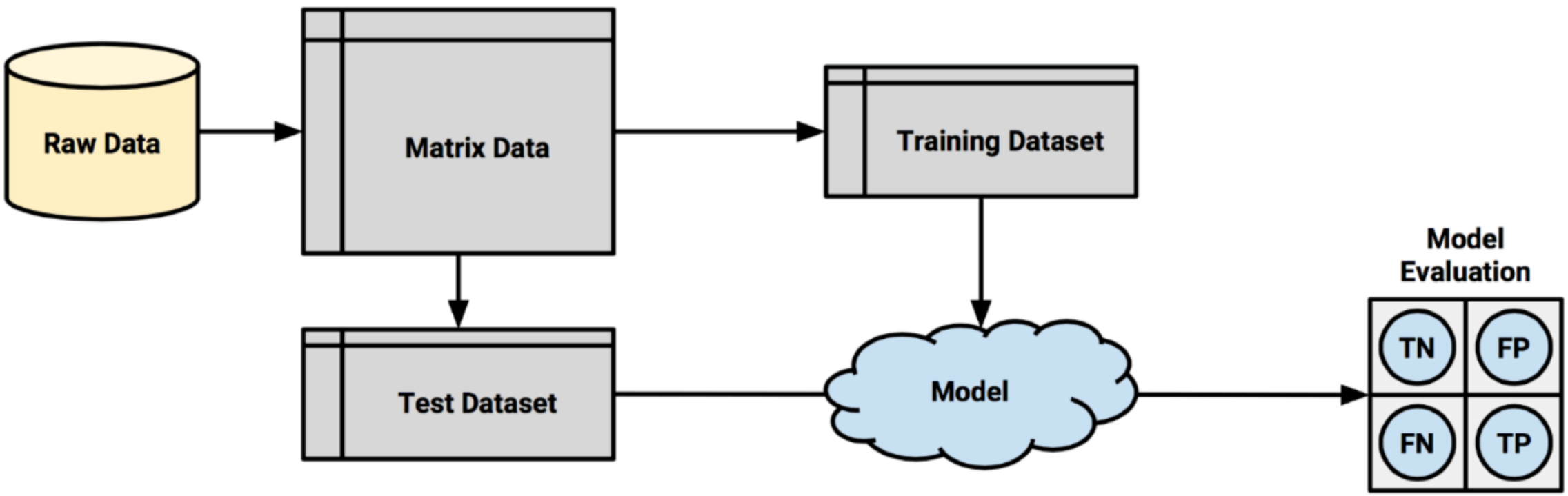
- When training and performance evaluation is just performed on the same data, we are simply evaluating the resubstitution error, i.e., the extent to which the model fails in predicting the data on which it has been trained

- However, this is not enough to see whether a model is good in predicting on unseen data

The holdout method

- It’s the procedure for partitioning data into training and test set
- The proportion of training and test can vary, sometimes 2/3 and 1/3, but not fixed
- In no way we should build multiple model and then choose the best one based on its performance on the test set

The holdout method



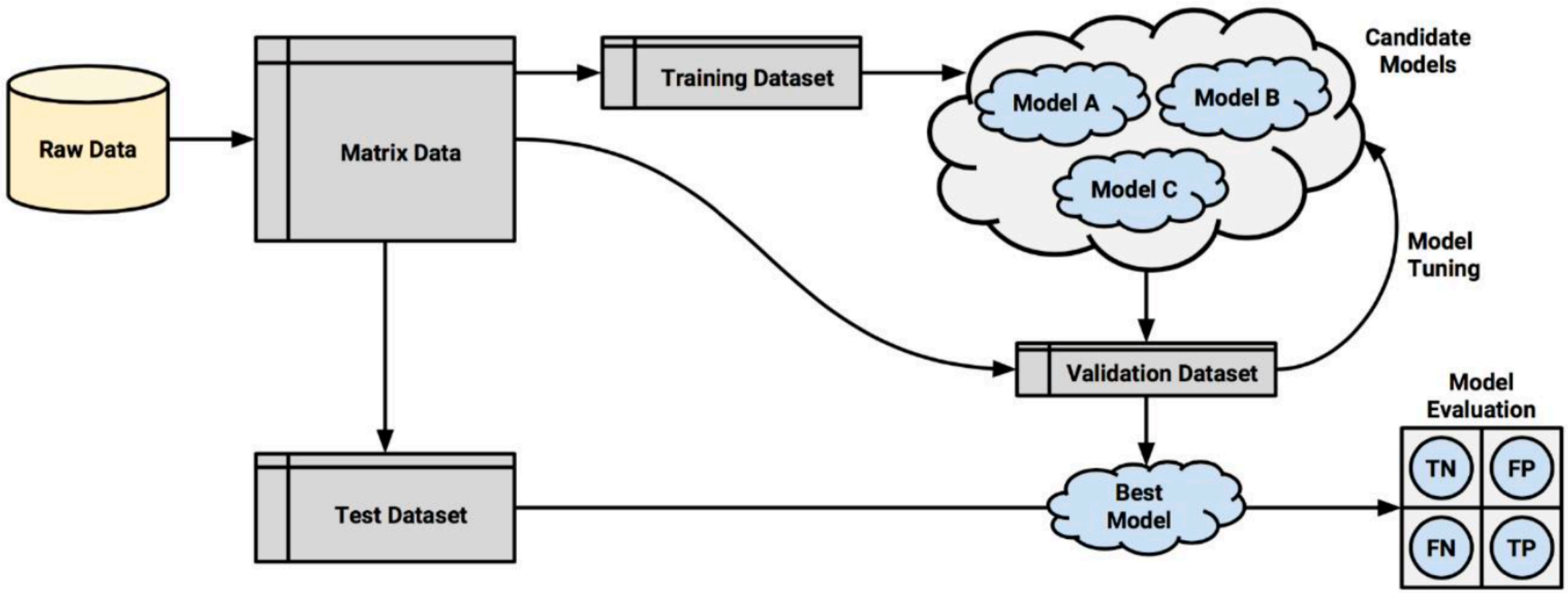
Problem: Where to tune?

- So far, we tried to improve the performances on the same data used to test the model
- This is against what we stated above
- What would be a better strategy?

Creating a validation set

- Instead of just splitting the data into training and test set, we create three splits:
  - training set: to build the model
  - validation set: to test different models/hyperparameters for performance improvement
  - test test: to evaluate the performance (best model only)
- We could do a 50/25/25 split, or something similar

Validation set



Problem with random sampling

- So far, we used random sampling to create training and test set
- However, we may not be sure to proportionally sample different classes for the dependent variable
- To reduce the chance of this occurring, we need to use a stratified random sampling
- This means sampling, in proportion, among the different classes

In Python

- The train_test_split function we have used already performs a stratified sampling by default

```
print(dataset['type'].value_counts())
```

```
ham     4812
spam     747
Name: type, dtype: int64
```

```
print(y_train.value_counts())
```

```
ham     3229
spam     495
Name: type, dtype: int64
```

- In both cases the proportions do not change

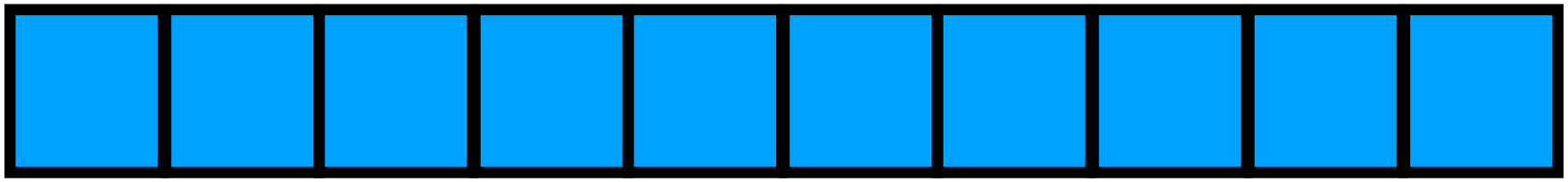
Repeated Holdouts

- Mitigates the randomness problem of training and test dataset (i.e., results may depend too much on the specific, random choice)
- Repeated holdouts performs multiple holdouts and averages their results
- In scikit-learn there is a dedicated module for that, named RepeatedKFold

Cross-Validation

- Known as k-fold cross-validation
- Standard for estimating model performance
- Rather than taking repeated random samples (which may use the same record once), it randomly divides the data into k separated partitions named folds
- Typically, but not necessarily, k=10

k-folds

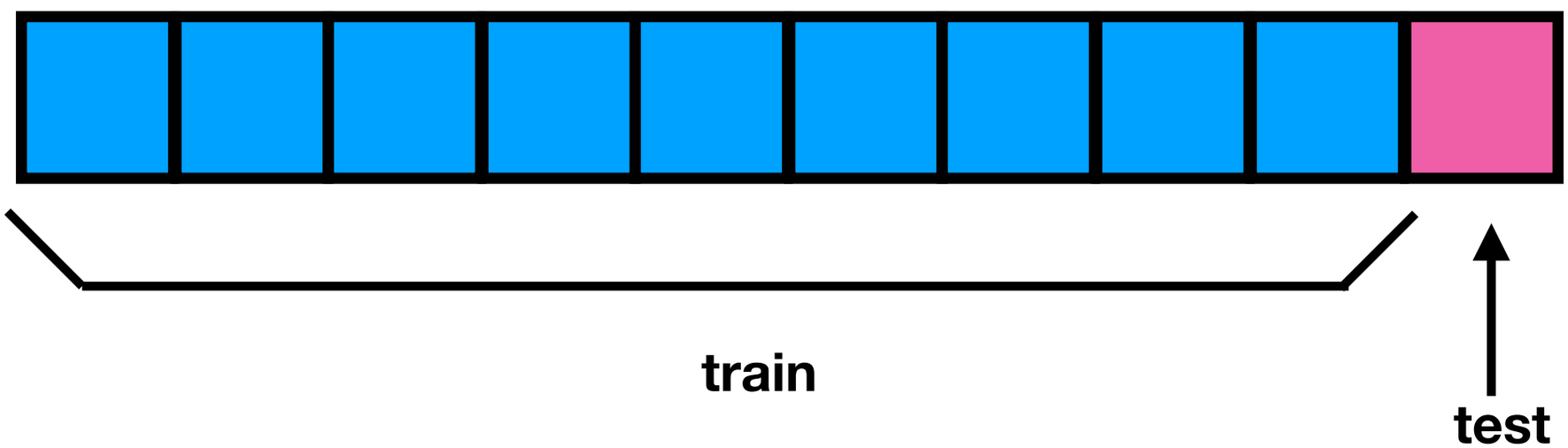


Then...

- We take one fold out, train on the remaining ones, and test on the extracted fold

- Save the Model performances

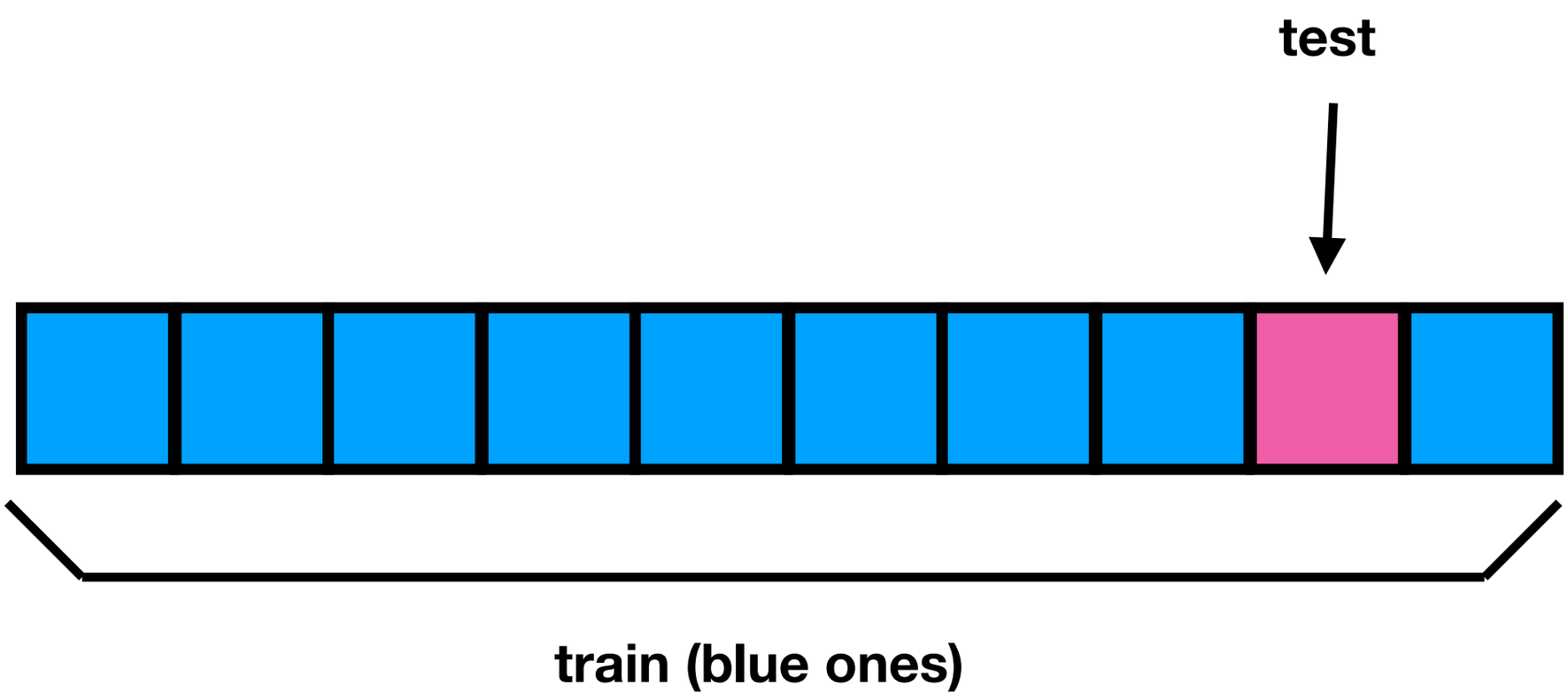
First iteration



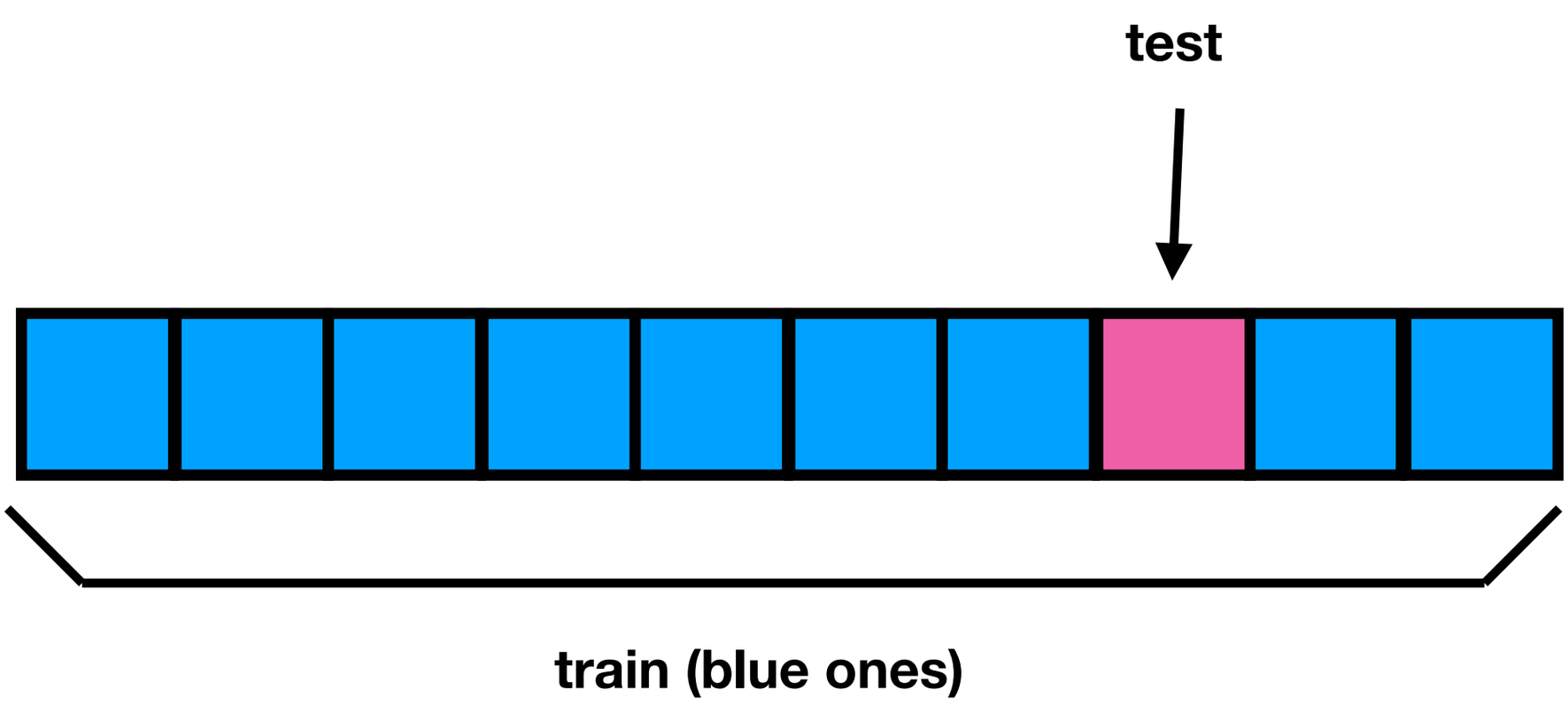
After…

We repeat the process taking out each fold…

Second iteration



Third iteration

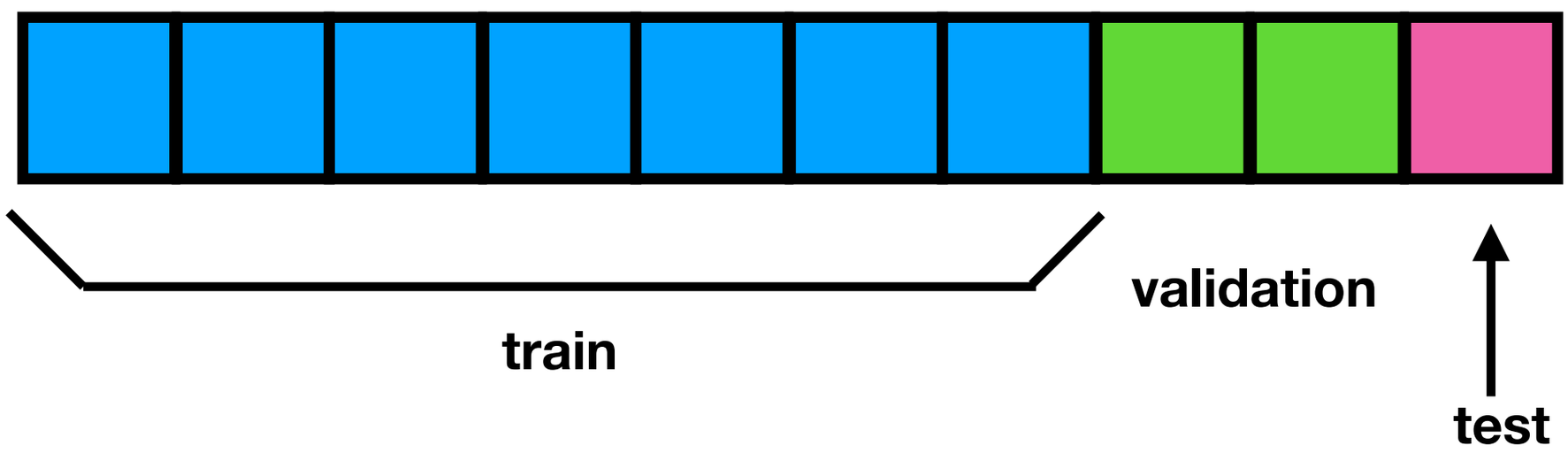


We repeat 10 times and then average the performances

With validation set

Typical rule:

- 7 folds for training
- 2 for validation
- 1 for test



Python Implementation

- You could implement this manually
- Or, you could use built-in functions
- Details available here: https://scikit-learn.org/stable/modules/cross_validation.html

Python implementation (only training and testing)

```
#applies transformText to all rows of text
dataset['text'] = dataset['text'].map(transformText)

#Build the counting corpus
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_counts = count_vect.fit_transform(dataset['text'])
y=dataset['type']

from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X = tfidf_transformer.fit_transform(X_counts)

from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
```

Python implementation (only training and testing)

```
from sklearn.model_selection import KFold
from sklearn import metrics
kf = KFold(n_splits=10)

totalPredicted=[]
totalActual=[]
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    clf.fit(X_train, y_train)
    predicted=clf.predict(X_test)
    totalPredicted+=predicted.tolist()
    totalActual+=list(y_test)

print(metrics.classification_report(totalActual,totalPredicted))
```

Just reporting an evaluation score…

```
from sklearn.model_selection import cross_val_score
cv = KFold(n_splits=10, random_state=1, shuffle=True)
scores = cross_val_score(clf, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
print(scores)
```

Note: accuracy is just a possible score, you can use precision, recall, etc.

Multiple scores

```
from sklearn.model_selection import cross_validate
scoring = {'acc': 'accuracy',
           'prec_macro': 'precision_macro',
           'rec_micro': 'recall_macro'}
scores = cross_validate(clf, X, y, scoring=scoring, cv=10, n_jobs=-1)
```