



ELIXIR-IIB Machine Learning for Biologists 2017

San Michele all'Adige (TN) — Sep 6, 2017



Machine learning for the working (biomedical) researcher

Marco Chierici

Fondazione Bruno Kessler



@MarcoChierici

mpba.fbk.eu



Summary

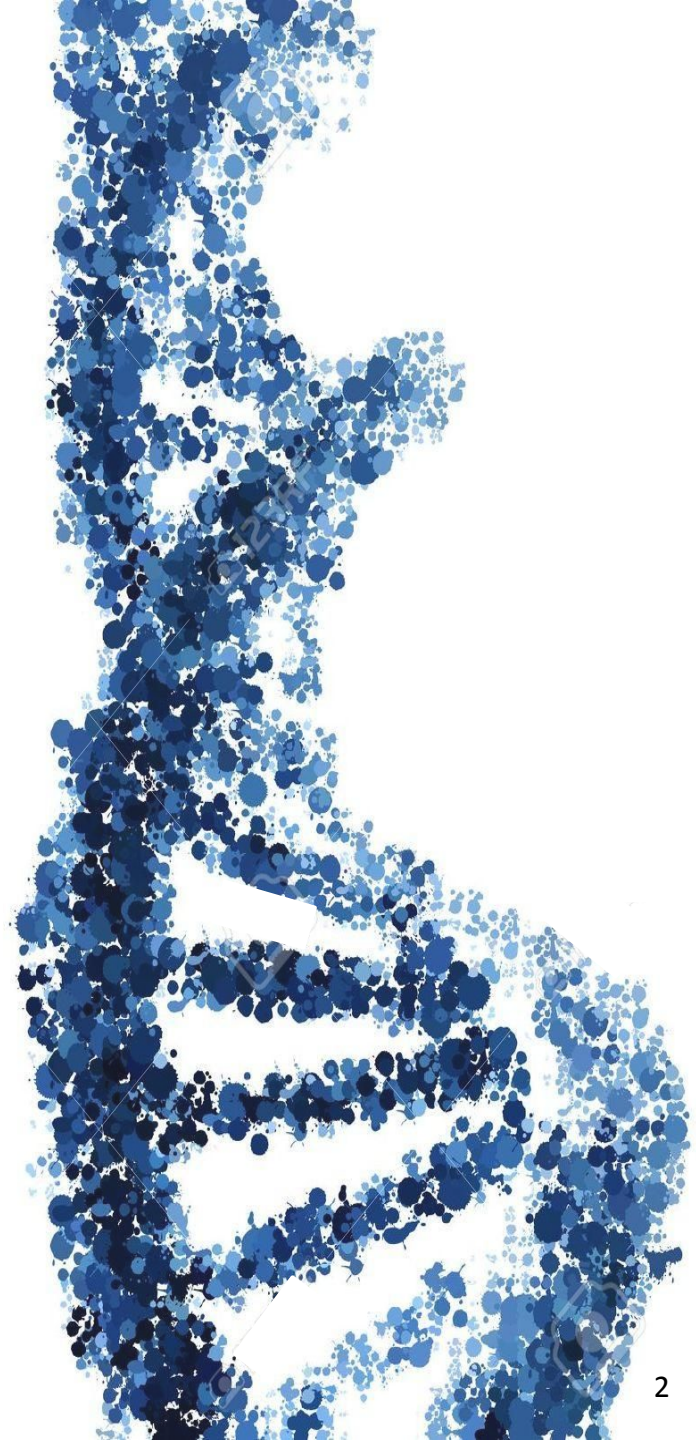
Learning problems: recap

A clinical dataset

Unsupervised learning: PCA

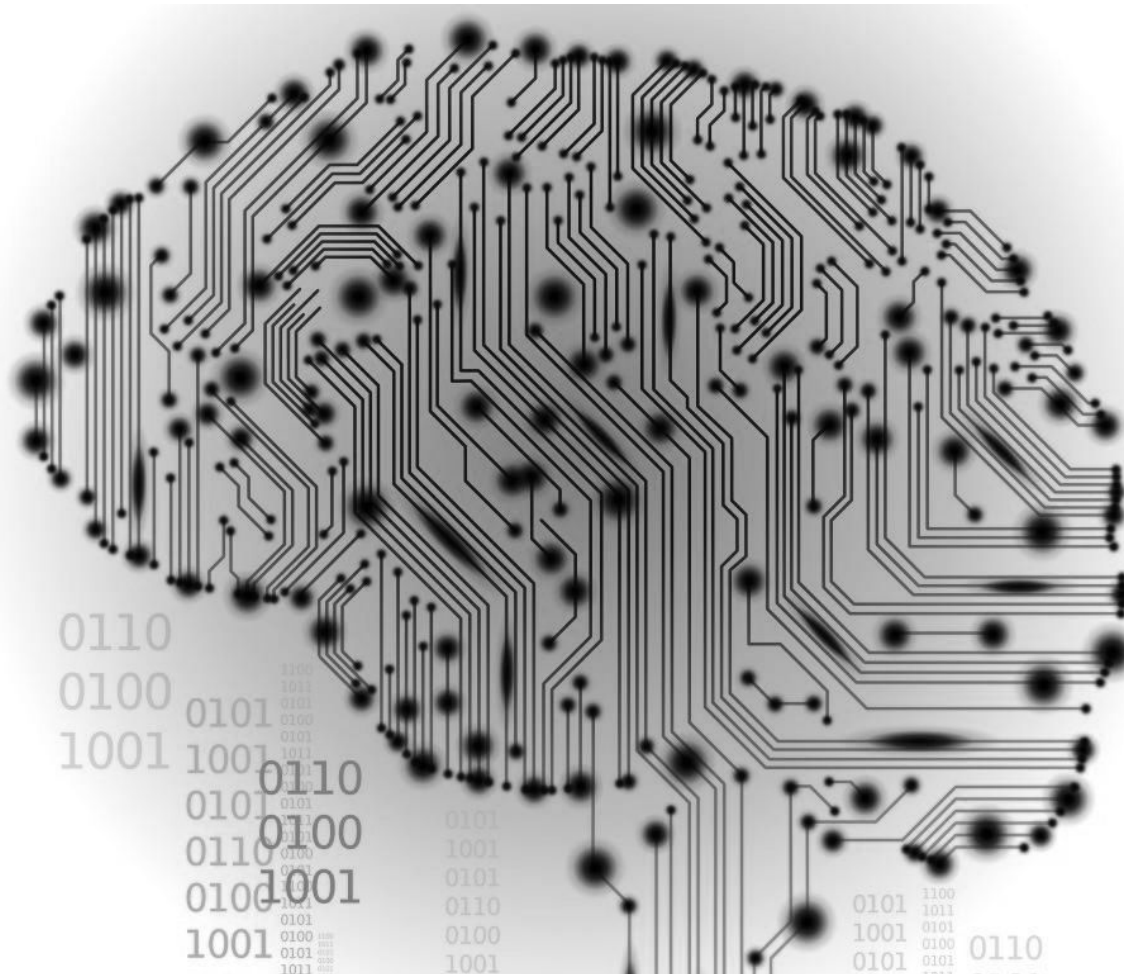
Supervised learning: classification

Guidelines for classifiers
development



Recap: Machine learning in a nutshell

Machine learning is about **making prediction from data**



Recap: The problem setting

A learning problem considers a set of n samples of data with several attributes (**features**)

Training data

and then tries to predict the properties of **unknown data**

Test data

Data can come with or without additional **targets (labels)** that we want to predict on test data

Training labels

Classic example: The Iris dataset

Sepal length	Sepal width	Petal length	Petal width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
6.3	3.3	6.0	2.5
5.8	2.7	5.1	1.9
7.1	3.0	5.9	2.1
...

Classic example: The Iris dataset

Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	<i>Iris setosa</i>
4.9	3.0	1.4	0.2	<i>Iris setosa</i>
4.7	3.2	1.3	0.2	<i>Iris setosa</i>
7.0	3.2	4.7	1.4	<i>I. versicolor</i>
6.4	3.2	4.5	1.5	<i>I. versicolor</i>
6.9	3.1	4.9	1.5	<i>I. versicolor</i>
6.3	3.3	6.0	2.5	<i>I. virginica</i>
5.8	2.7	5.1	1.9	<i>I. virginica</i>
7.1	3.0	5.9	2.1	<i>I. virginica</i>
...



FEATURES

CLASS
LABELS

Sepal length	Sepal width	Petal length	Petal width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
6.3	3.3	6.0	2.5
5.8	2.7	5.1	1.9
7.1	3.0	5.9	2.1
...

Species
<i>Iris setosa</i>
<i>Iris setosa</i>
<i>Iris setosa</i>
<i>I. versicolor</i>
<i>I. versicolor</i>
<i>I. versicolor</i>
<i>I. virginica</i>
<i>I. virginica</i>
<i>I. virginica</i>
...



FEATURES

CLASS LABELS

SAMPLES (train)

Sepal length	Sepal width	Petal length	Petal width
5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
6.3	3.3	6.0	2.5
5.8	2.7	5.1	1.9
7.1	3.0	5.9	2.1
...

Species
<i>Iris setosa</i>
<i>Iris setosa</i>
<i>Iris setosa</i>
<i>I. versicolor</i>
<i>I. versicolor</i>
<i>I. versicolor</i>
<i>I. virginica</i>
<i>I. virginica</i>
<i>I. virginica</i>
...



SAMPLES (test)

5.1	2.5	3.0	1.1
5.7	2.8	4.1	1.3
5.9	3.0	5.1	1.8

??
??
??

Recap: Learning problems

Unsupervised

No attributes/target values are available for training data

Goals

- ★ Discover groups of similar examples within the data
- ★ Project the data from a high- to low-dimensional space

Techniques

PCA, clustering, neural networks



Supervised

Data comes with additional attributes that we want to predict

Goal


Predict attributes of new unlabeled data

Techniques

Support Vector Machines
Random Forest [...]



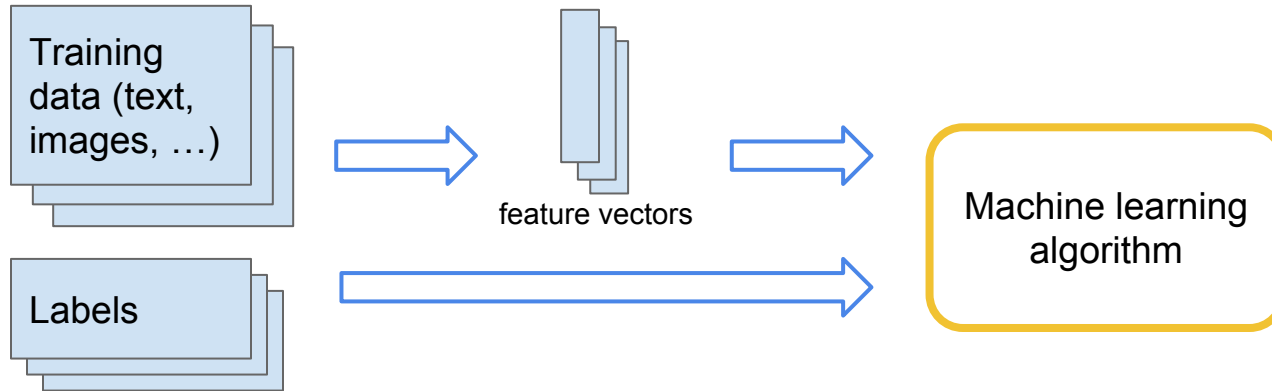
Supervised predictive modeling: a conceptual workflow



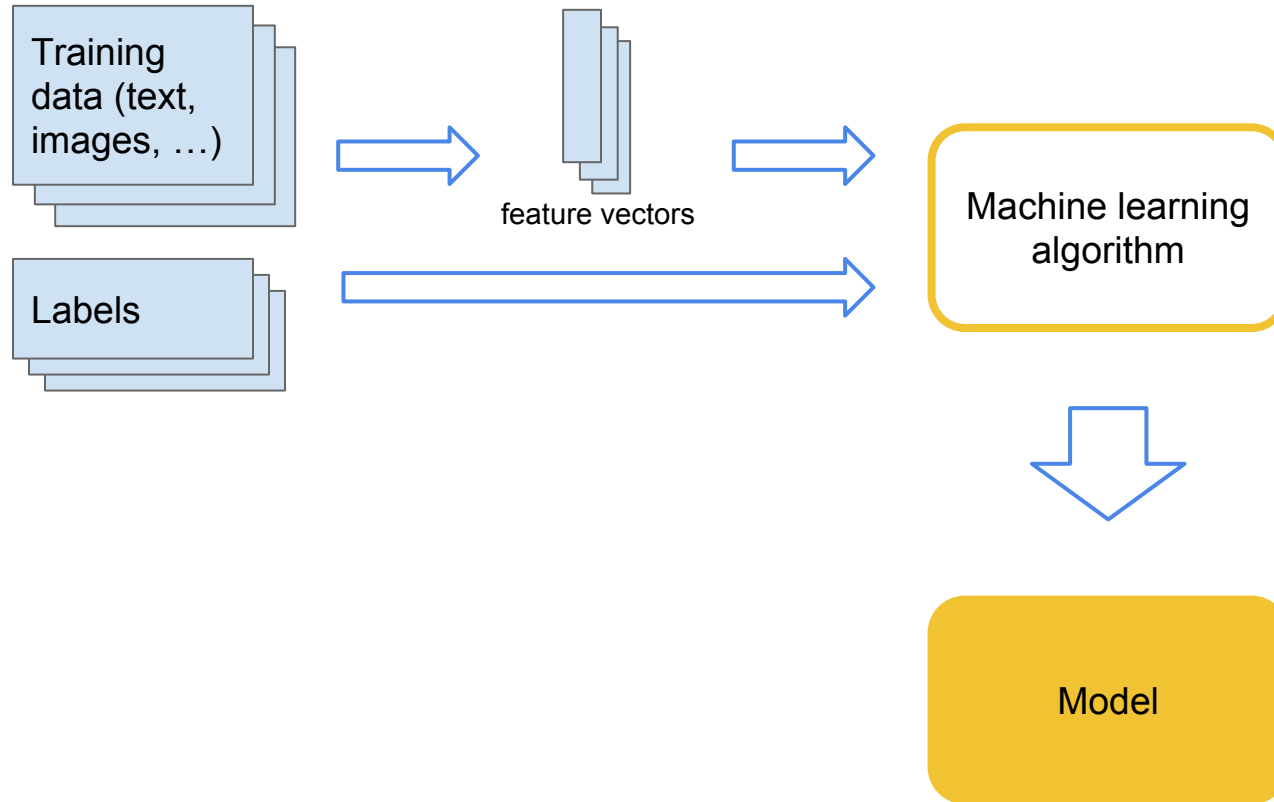
Training
data (text,
images, ...)

Labels

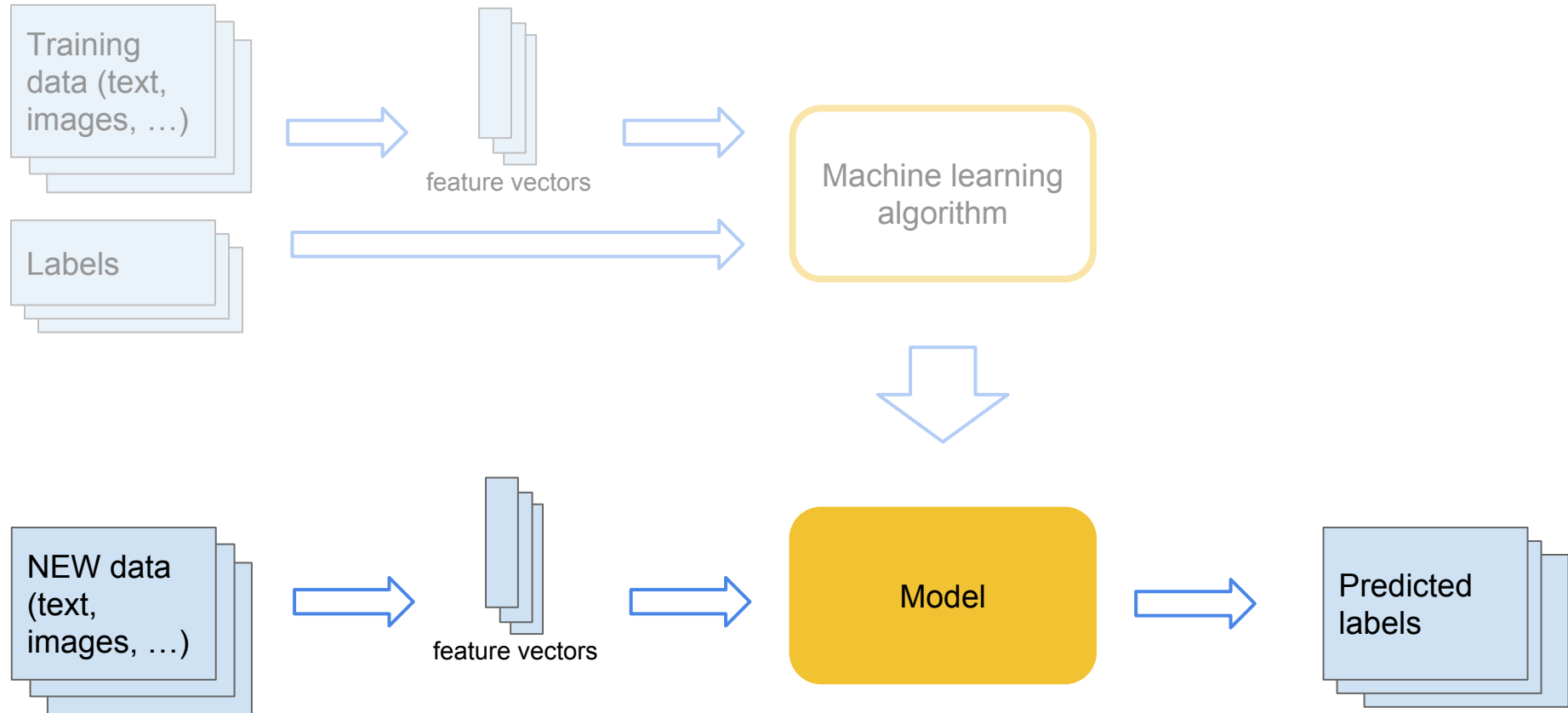
Supervised predictive modeling: a conceptual workflow



Supervised predictive modeling: a conceptual workflow



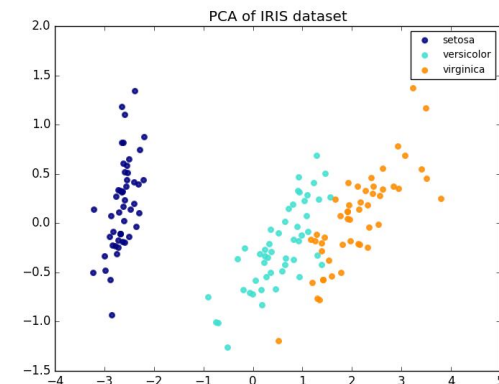
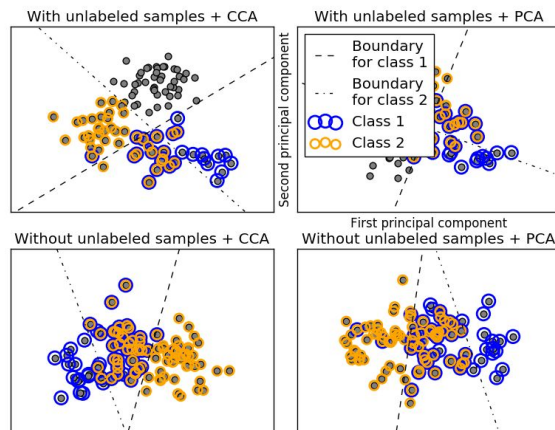
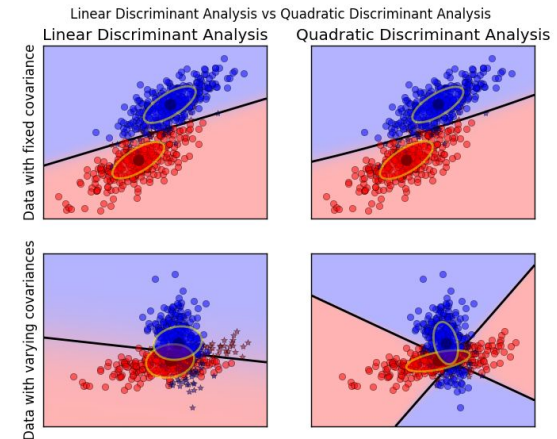
Supervised predictive modeling: a conceptual workflow





Tools: Scikit-learn

- Library of Machine Learning algorithms
- Open Source (BSD)
- Simple fit / predict / transform API
- Python / NumPy / SciPy / Cython



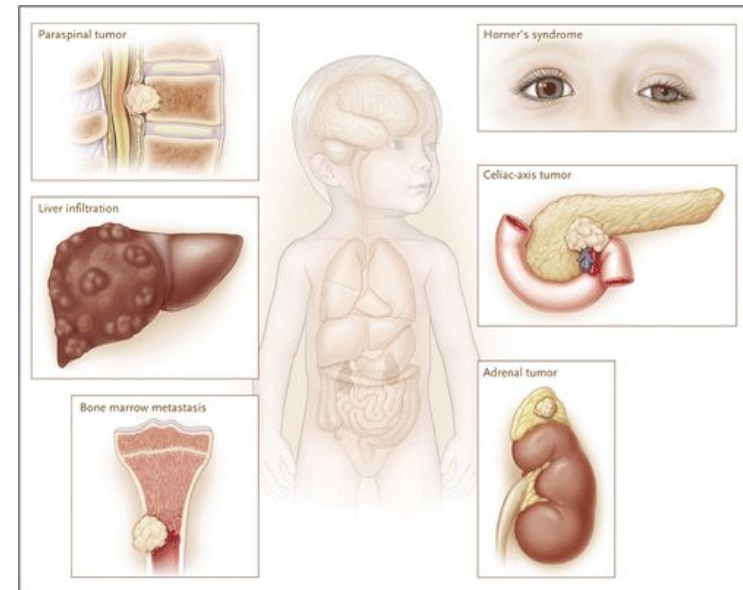
Application: Neuroblastoma

Neuroblastoma (NB) is a **pediatric tumor** of the sympathetic nervous system

- ★ NB develops from immature neuroblasts of the sympatho-adrenal lineage
- ★ NB accounts for **~700 new cases** in the USA per year

Aim: to separate patients with contrasting clinical courses by **machine learning** on **gene expression** data

[Maris, *NEJM*, 2010]



Possible locations of NB tumors.

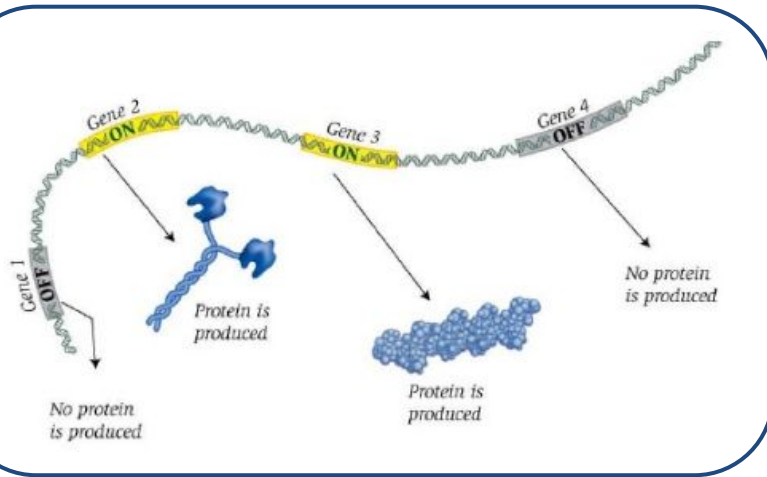
NB tumors begin in the adrenal gland, then may metastasize through liver, spine, orbits, intestine and bone.

Data from



UNIKLINIK
KÖLN

Measuring gene expression

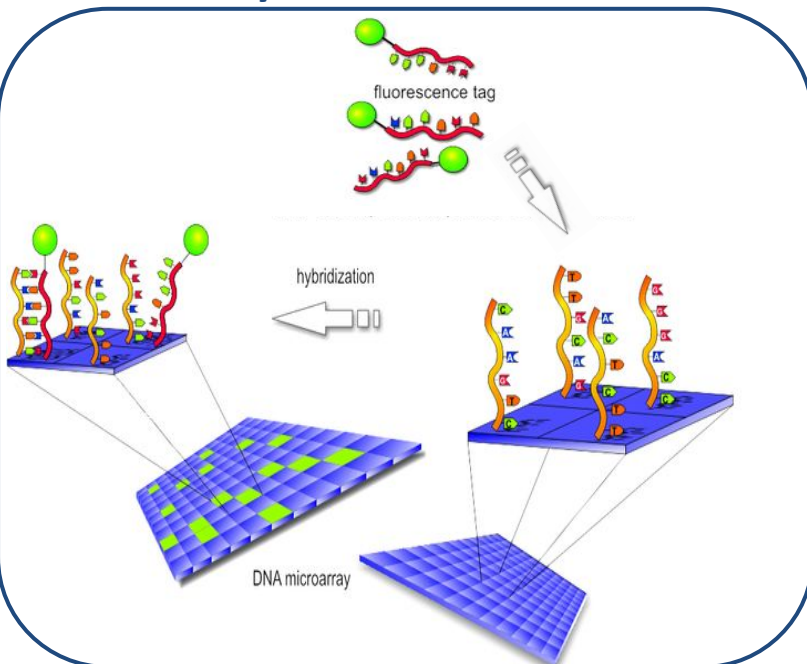


- **Gene:** segment of a DNA molecule that holds information for building a protein/RNA molecule
- When a gene is “on” and its protein/RNA product is made, the gene is expressed
- The on and off states of all cell’s genes is known as **gene expression profile**



How is it measured?

Microarray



- A collection of synthetic DNA segments (probes) attached to designated locations on a solid surface
- The probes bind to complementary "target" DNA sequences with fluorescent dyes
- The bound DNA will be detected by its fluorescent dye by a scanner
- The signal intensity is proportional to gene expression

Dataset

498 neuroblastoma
tumor tissue **samples**

~16,000 features (genes)

Endpoint	Training set			Validation set		
	# samples	1	-1	# samples	1	-1
Progression, relapse or death (Event yes/no)	249	89	160	249	94	155
Death from disease (Death yes/no)	249	51	198	249	54	195
Extreme disease outcome (Unfavorable/Favorable)	136	45	91	136	46	90

Loading the data

```
# Import required packages
import numpy as np
import pandas as pd
```

Load the training set data:

```
data_tr = pd.read_csv("data/nb_train.txt.gz", dtype=str, sep='\t', index_col=0)
```

```
data_tr.head()
```

	class	15E1.2	2'-PDE	7A5	A2BP1	A2LD1	A2M
sampleID							
NB001	1	10.4660823398966	8.12017940297329	7.65443720698381	10.0509080936873	8.20309820315593	14.418262495368
NB003	1	10.0433806860637	8.17757422776342	7.42951341977216	8.67841313806572	8.56271420871427	14.562891291508
NB005	1	10.2530057854764	8.56140102139205	7.60193363768839	10.1466351236573	9.18055741255828	14.911622584073
NB011	-1	9.84423602402605	8.10487703077007	7.63407268165348	11.3806517869924	8.1950865978724	16.279612025258
NB013	1	10.3727277348889	8.1403882657906	7.40405497125135	10.1629373821085	8.53913808430587	16.377943970020

5 rows x 16394 columns

```
x_tr = data_tr.values[:, 1:].astype(float)
y_tr = data_tr["class"].values.astype(int)
y_tr[:10]
```

```
array([ 1,  1,  1, -1,  1, -1, -1, -1, -1, -1])
```

```
data_ts = pd.read_csv("data/nb_test.txt.gz", dtype=str, sep='\t', index_col=0)
x_ts = data_ts.values[:, 1:].astype(float)
y_ts = data_ts["class"].values.astype(int)
```

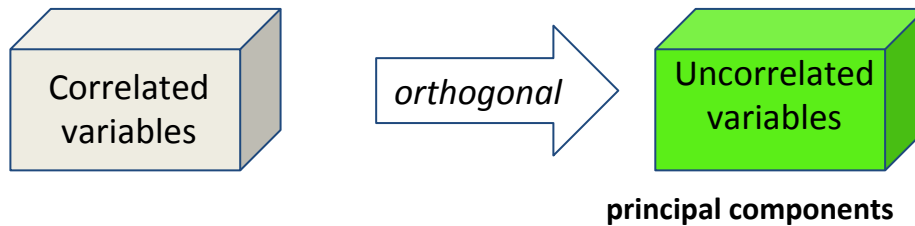
Training data

Training labels

Test data
and labels

Unsupervised transformations: Principal Component Analysis

(Pearson, 1901 & Hotelling, 1930)



AIM: reveal the internal structure of the data in a way that best explains the variance

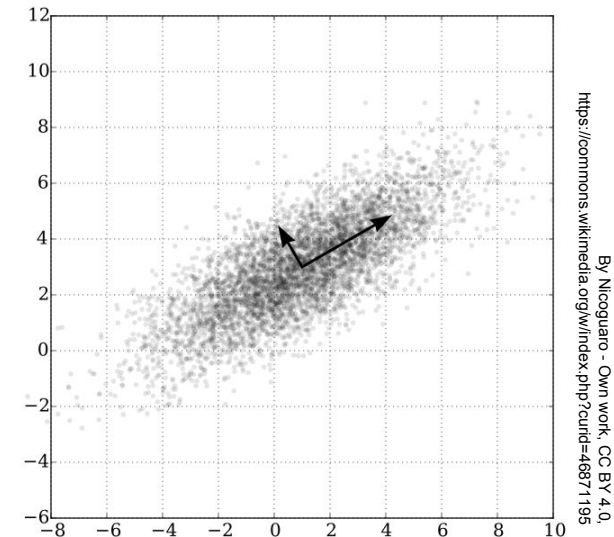
General algorithm:

1. first principal component has the largest possible **variance** (and thus explains the largest variability)
2. each succeeding component in turn has the highest variance possible under the constraint that it is **orthogonal** to the preceding components

Gory details:

Mathematically, this is the **eigenvalue decomposition** of a data **covariance** (or **correlation**) matrix or **singular value decomposition** of a **data matrix**, usually after mean centering (and normalizing or using **Z-scores**) the data matrix for each attribute.

The principal components are orthogonal because they are the **eigenvectors** of the **covariance/correlation matrix**, which is **symmetric**.



PCA of a

- multivariate Gaussian distribution centered at (1,3)
- with a standard deviation of 3 in roughly the (0.866, 0.5) direction
- and of 1 in the orthogonal direction.

The vectors shown are the eigenvectors of the covariance matrix scaled by the square root of the corresponding eigenvalue, and shifted so their tails are at the mean.

Unsupervised transformations:

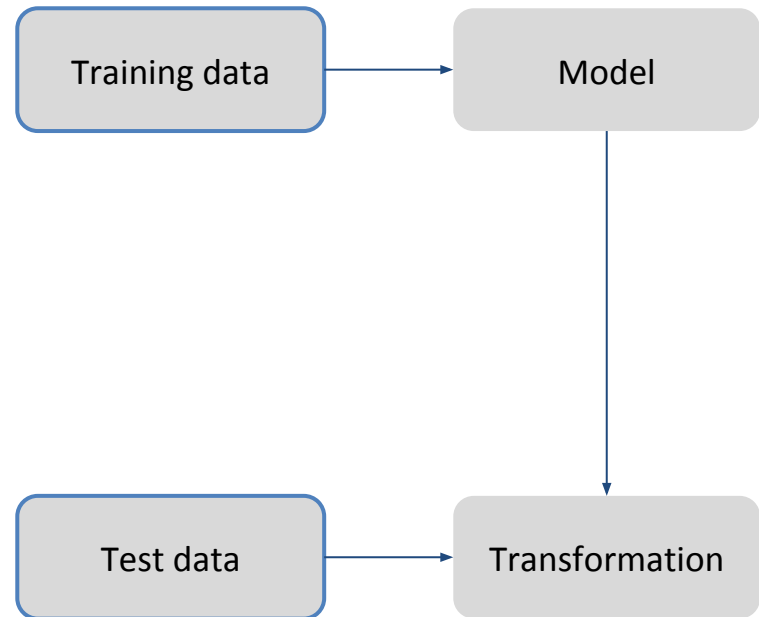
PCA in practice

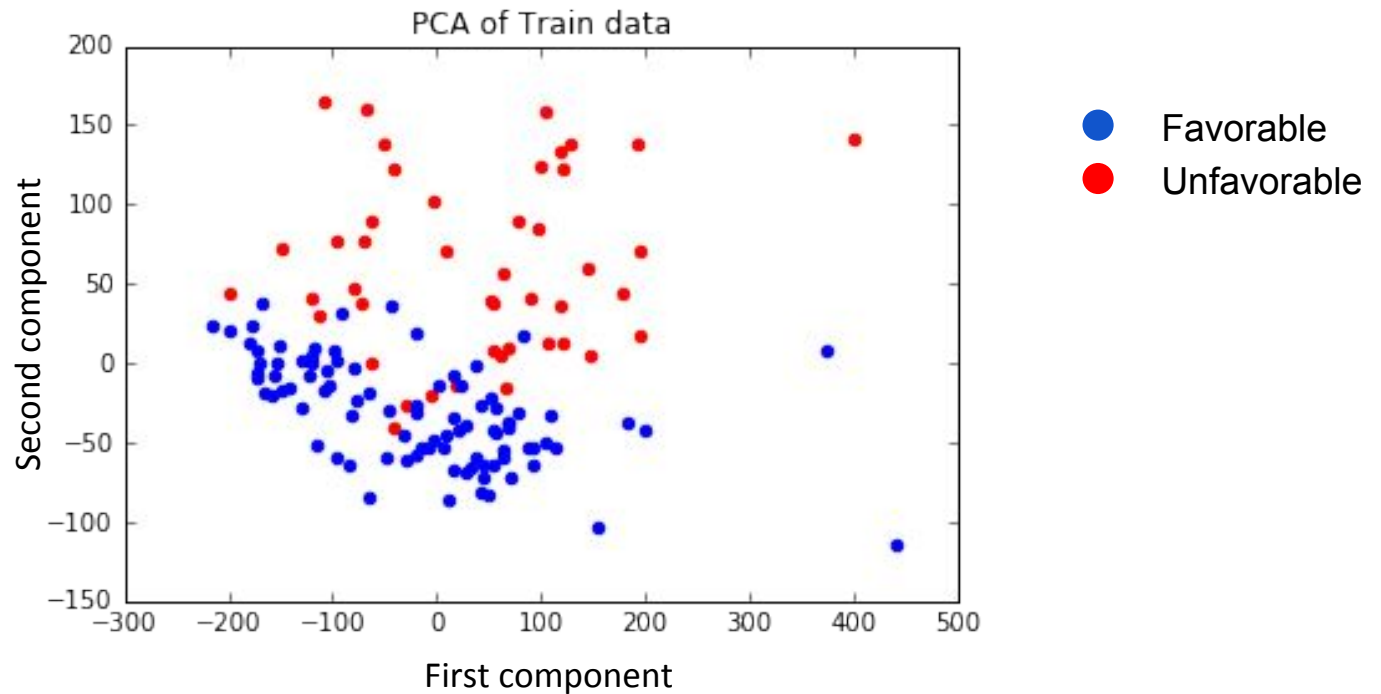
```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=2)
```

```
z_tr = pca.fit_transform(x_tr)
```

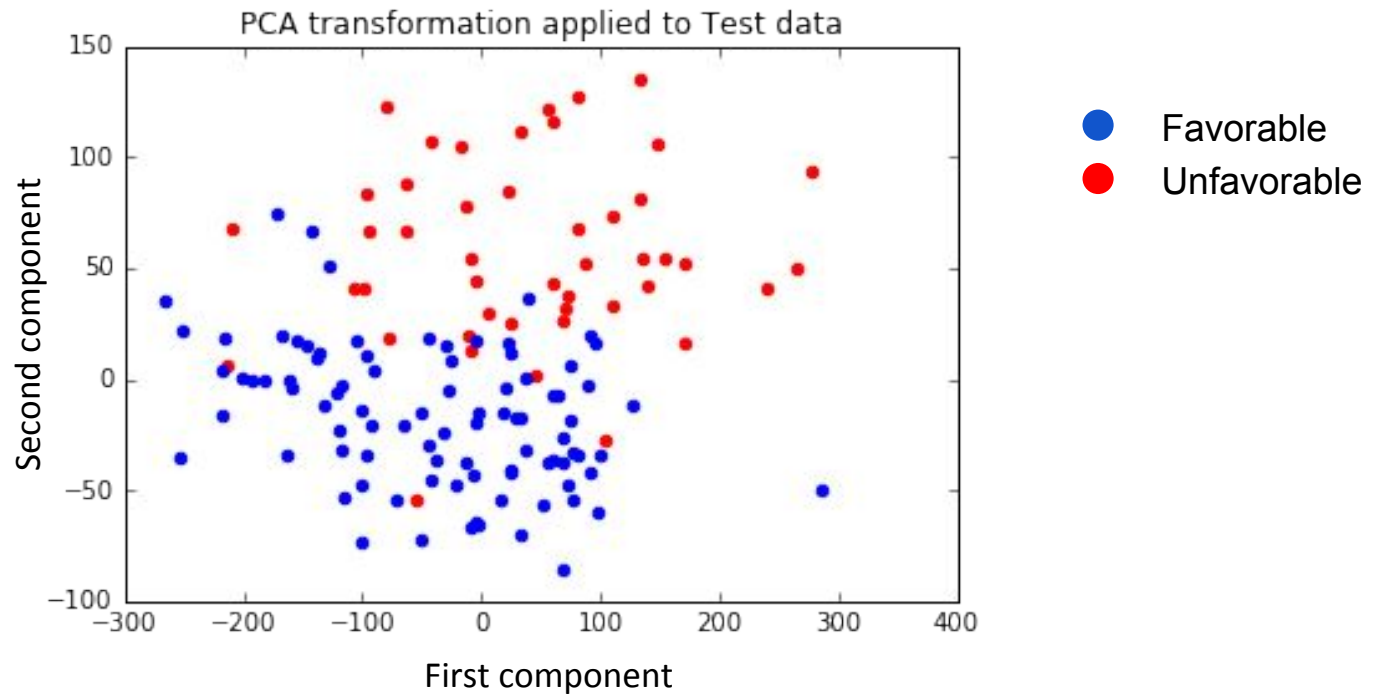
```
z_ts = pca.transform(x_ts)
```





```
import matplotlib.pyplot as plt
```

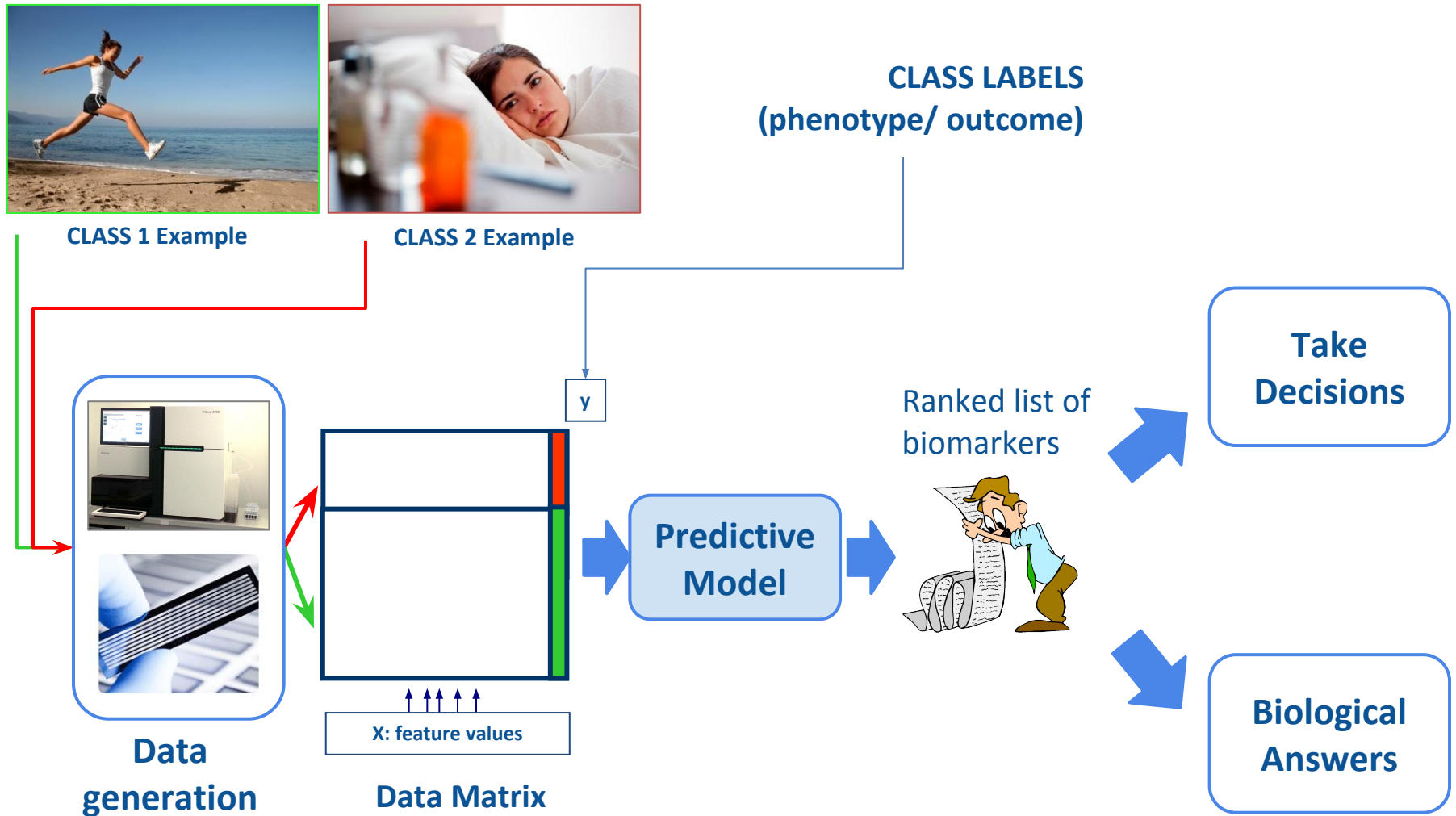
```
plt.figure()  
plt.scatter(z_tr[y_tr == 1, 0], z_tr[y_tr == 1, 1], color="r")  
plt.scatter(z_tr[y_tr == -1, 0], z_tr[y_tr == -1, 1], color="b")  
plt.title("PCA of Train data")  
plt.show()
```



```
import matplotlib.pyplot as plt
```

```
plt.figure()  
plt.scatter(z_ts[y_ts == 1, 0], z_ts[y_ts == 1, 1], color="r")  
plt.scatter(z_ts[y_ts == -1, 0], z_ts[y_ts == -1, 1], color="b")  
plt.title("PCA transformation applied to Test data")  
plt.show()
```

Supervised learning



Classification in practice (1a)

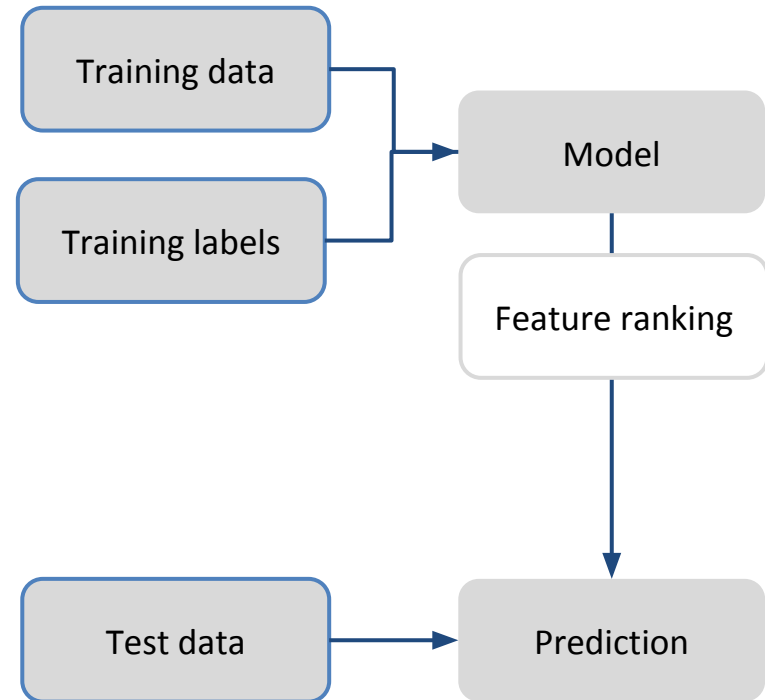
Support Vector Machine

```
from sklearn.svm import SVC
```

```
clf = SVC(kernel='linear', C=1.0)
```

```
clf.fit(x_tr, y_tr)
```

```
y_pred = clf.predict(x_ts)
```



Classification in practice (1b)

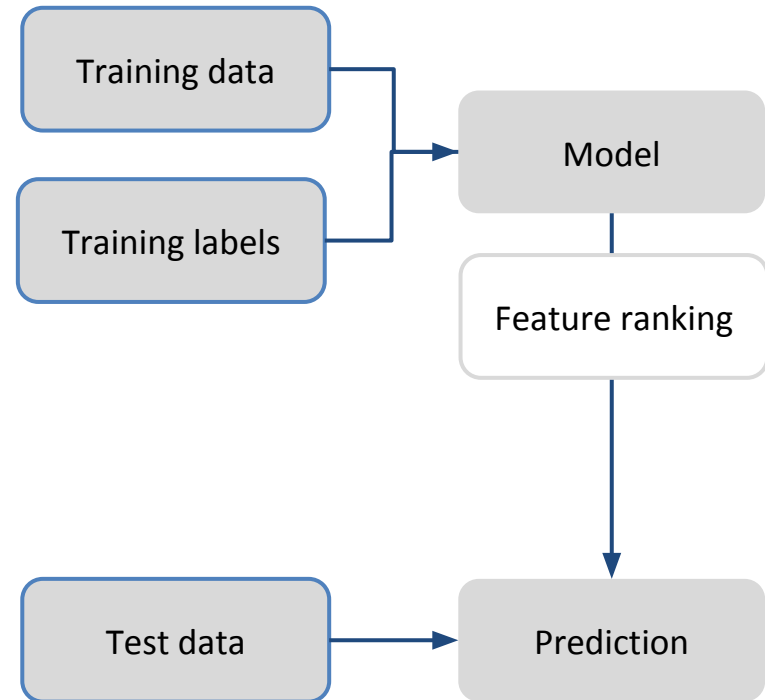
Random Forest

```
from sklearn.ensemble import RandomForestClassifier
```

```
clf = RandomForestClassifier()
```

```
clf.fit(x_tr, y_tr)
```

```
y_pred = clf.predict(x_ts)
```



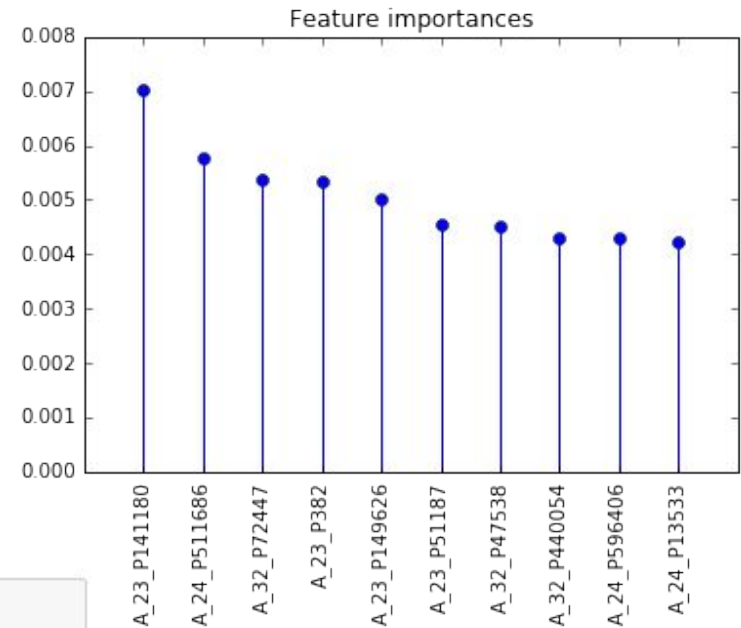
Feature ranking

```
importances = clf.feature_importances_  
indices = np.argsort(importances)[::-1]  
  
print("Feature ranking:")  
for f in range(10):  
    print("%d. feature %s (%f)" % (f + 1, feat_tr[indices[f]], importances[indices[f]]))
```

Feature ranking:

```
1. feature A_23_P141180 (0.007036)  
2. feature A_24_P511686 (0.005759)  
3. feature A_32_P72447 (0.005383)  
4. feature A_23_P382 (0.005325)  
5. feature A_23_P149626 (0.005002)  
6. feature A_23_P51187 (0.004552)  
7. feature A_32_P47538 (0.004529)  
8. feature A_32_P440054 (0.004315)  
9. feature A_24_P596406 (0.004294)  
10. feature A_24_P13533 (0.004211)
```

```
plt.figure()  
plt.title("Feature importances")  
plt.stem(range(10), importances[indices[:10]], align="center")  
plt.xticks(range(10), feat_tr[indices[:10]], rotation="vertical")  
plt.xlim([-1, 10])  
plt.show()
```



Classification in practice (2)

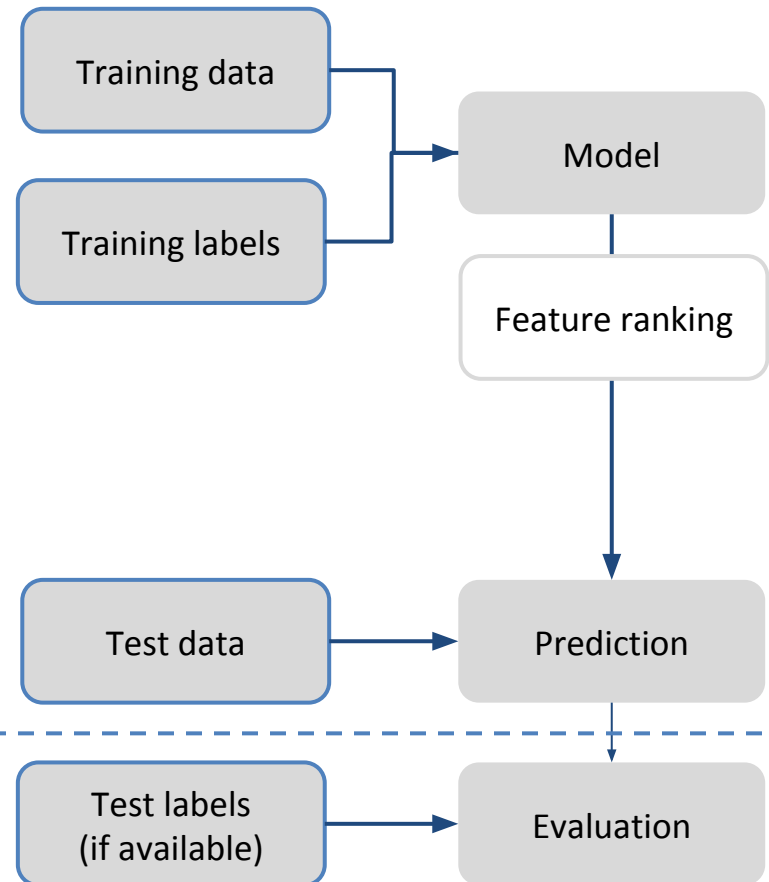
```
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
```

```
clf = RandomForestClassifier()
```

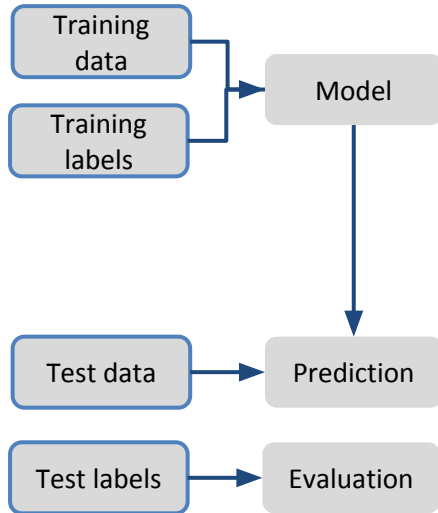
```
clf.fit(x_tr, y_tr)
```

```
y_pred = clf.predict(x_ts)
```

```
from sklearn import metrics
accuracy = metrics.accuracy_score(y_ts,
y_pred)
```



Performance evaluation: metrics



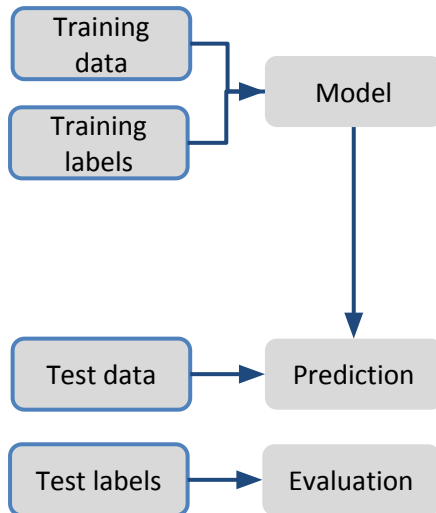
Sample name	Test labels (y_{ts})	Predicted labels (y_{pred})	
S1	1	1	True Positive (TP)
S2	1	-1	False Negative (FN)
S3	-1	1	False Positive (FP)
S4	-1	-1	True Negative (TN)
S5	-1	1	
S6	1	1	
...	



Confusion matrix

		Predicted	
		1	-1
True	1	TP	FN
	-1	FP	TN

Performance evaluation: metrics



Sample name	Test labels (y_{ts})	Predicted labels (y_{pred})	
S1	1	1	True Positive (TP)
S2	1	-1	False Negative (FN)
S3	-1	1	False Positive (FP)
S4	-1	-1	True Negative (TN)
S5	-1	1	
S6	1	1	
...	

Confusion matrix

		Predicted	
		1	-1
True	1	TP	FN
	-1	FP	TN

Metrics

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

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

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

Matthews Correlation Coefficient (MCC)

Range [-1, 1] : +1 perfect prediction
0 random prediction
-1 inverse prediction

Gory details:

$$MCC = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Baldi P. et al. Assessing the accuracy of prediction algorithms for classification: an overview.
Bioinformatics, 2000

Good for unbalanced classes

Example: 15 samples of class +1, 5 samples of class -1.
Suppose the smaller class gets almost entirely misclassified:

True	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	-1	-1	-1	-1	-1
Pred	1	1	1	1	1	1	1	-1	1	1	1	1	1	1	1	1	1	-1	1	1

Accuracy = 0.75 (seems good!)

MCC = 0.19 (ouch!)

Metrics in scikit-learn

```
from sklearn import metrics
```

Classification metrics

See the [Classification metrics](#) section of the user guide for further details.

<code>metrics.accuracy_score</code> (y_true, y_pred[, ...])	Accuracy classification score.
<code>metrics.auc</code> (x, y[, reorder])	Compute Area Under the Curve (AUC) using the trapezoidal rule
<code>metrics.average_precision_score</code> (y_true, y_score)	Compute average precision (AP) from prediction scores
<code>metrics.brier_score_loss</code> (y_true, y_prob[, ...])	Compute the Brier score.
<code>metrics.classification_report</code> (y_true, y_pred)	Build a text report showing the main classification metrics
<code>metrics.confusion_matrix</code> (y_true, y_pred[, ...])	Compute confusion matrix to evaluate the accuracy of a classification
<code>metrics.f1_score</code> (y_true, y_pred[, labels, ...])	Compute the F1 score, also known as balanced F-score or F-measure
<code>metrics.fbeta_score</code> (y_true, y_pred, beta[, ...])	Compute the F-beta score
<code>metrics.hamming_loss</code> (y_true, y_pred[, classes])	Compute the average Hamming loss.
<code>metrics.hinge_loss</code> (y_true, pred_decision[, ...])	Average hinge loss (non-regularized)
<code>metrics.jaccard_similarity_score</code> (y_true, y_pred)	Jaccard similarity coefficient score
<code>metrics.log_loss</code> (y_true, y_pred[, eps, ...])	Log loss, aka logistic loss or cross-entropy loss.
<code>metrics.matthews_corrcoef</code> (y_true, y_pred)	Compute the Matthews correlation coefficient (MCC) for binary classes
<code>metrics.precision_recall_curve</code> (y_true, ...)	Compute precision-recall pairs for different probability thresholds
<code>metrics.precision_recall_fscore_support</code> (...)	Compute precision, recall, F-measure and support for each class
<code>metrics.precision_score</code> (y_true, y_pred[, ...])	Compute the precision
<code>metrics.recall_score</code> (y_true, y_pred[, ...])	Compute the recall
<code>metrics.roc_auc_score</code> (y_true, y_score[, ...])	Compute Area Under the Curve (AUC) from prediction scores
<code>metrics.roc_curve</code> (y_true, y_score[, ...])	Compute Receiver operating characteristic (ROC)
<code>metrics.zero_one_loss</code> (y_true, y_pred[, ...])	Zero-one classification loss.
<code>metrics.brier_score_loss</code> (y_true, y_prob[, ...])	Compute the Brier score.

Metrics in scikit-learn

```
from sklearn import metrics

clf = RandomForestClassifier()

clf.fit(x_tr, y_tr)

y_pred = clf.predict(x_ts)

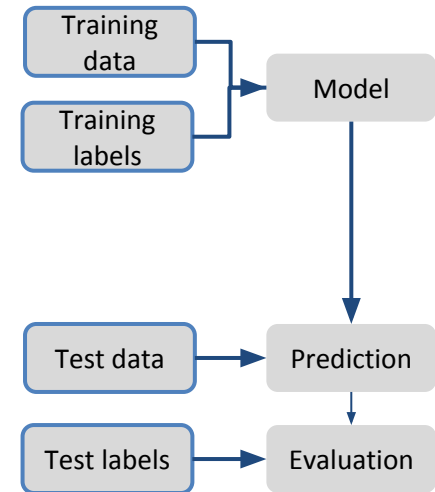
accuracy = metrics.accuracy_score(y_ts,
y_pred)
```

```
metrics.accuracy_score(y_ts, y_pred)
```

0.948529411765

```
metrics.matthews_corrcoef(y_ts, y_pred)
```

0.88787329751



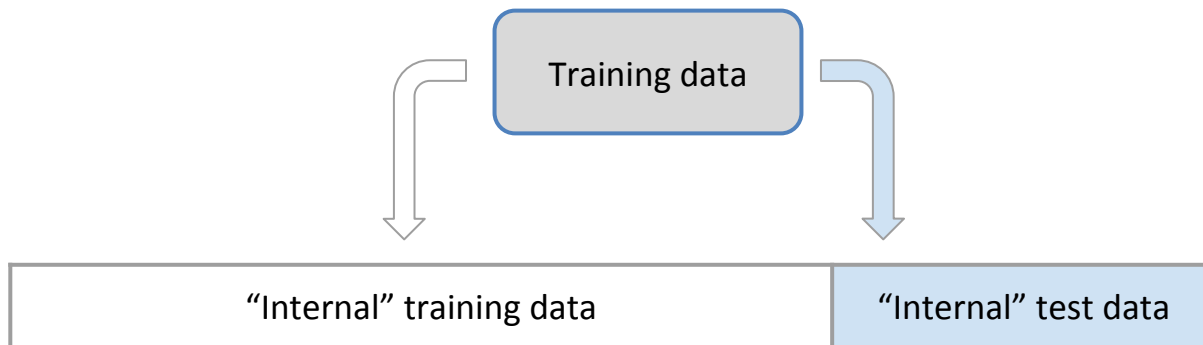
The model performs well!

BUT...

... how this model will generalize to an independent dataset?

Data partitioning

Randomly split dataset into two groups (hold-out strategy)



```
X_train, X_test, y_train, y_test = train_test_split(x_tr, y_tr, test_size=0.25, random_state=0)
```

Generally:

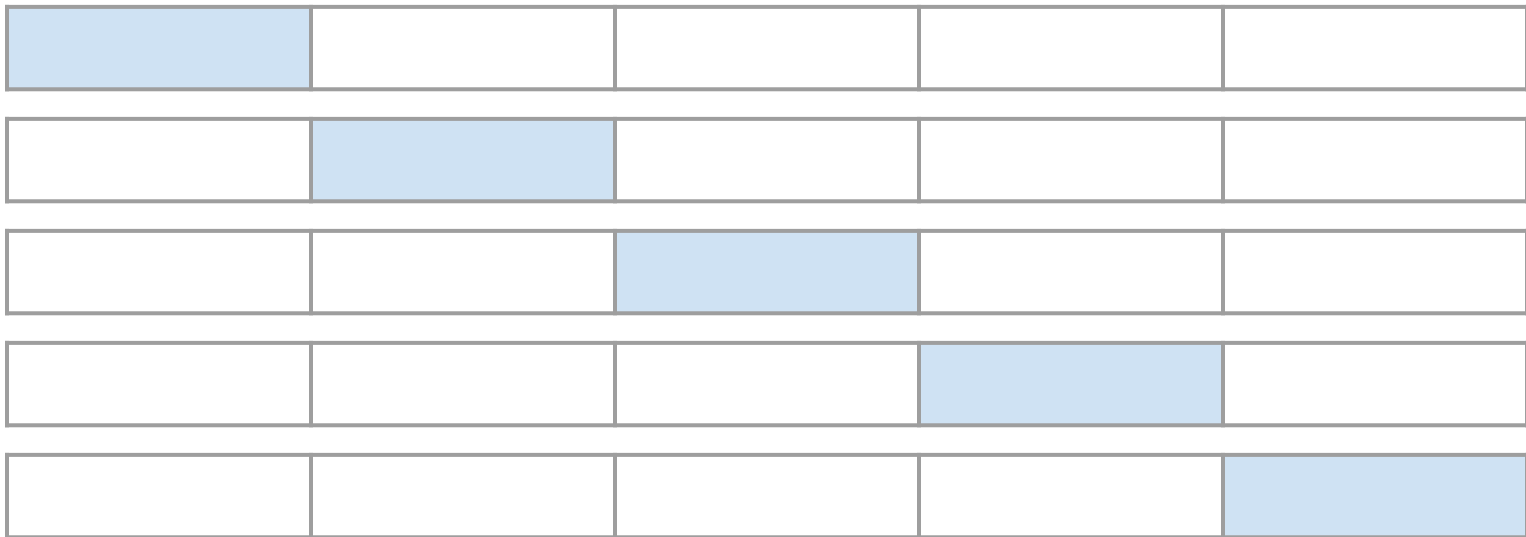
- The larger the training data the better the classifier
- The larger the test data, the more accurate the error estimate

Cross-validation

Split dataset into K equal size partitions (the “folds”)

- K-1 folds as “internal” training, to learn classification rules
- 1 fold as “internal” test, to evaluate classification performance

Example: K=5 (5-fold Cross-validation)

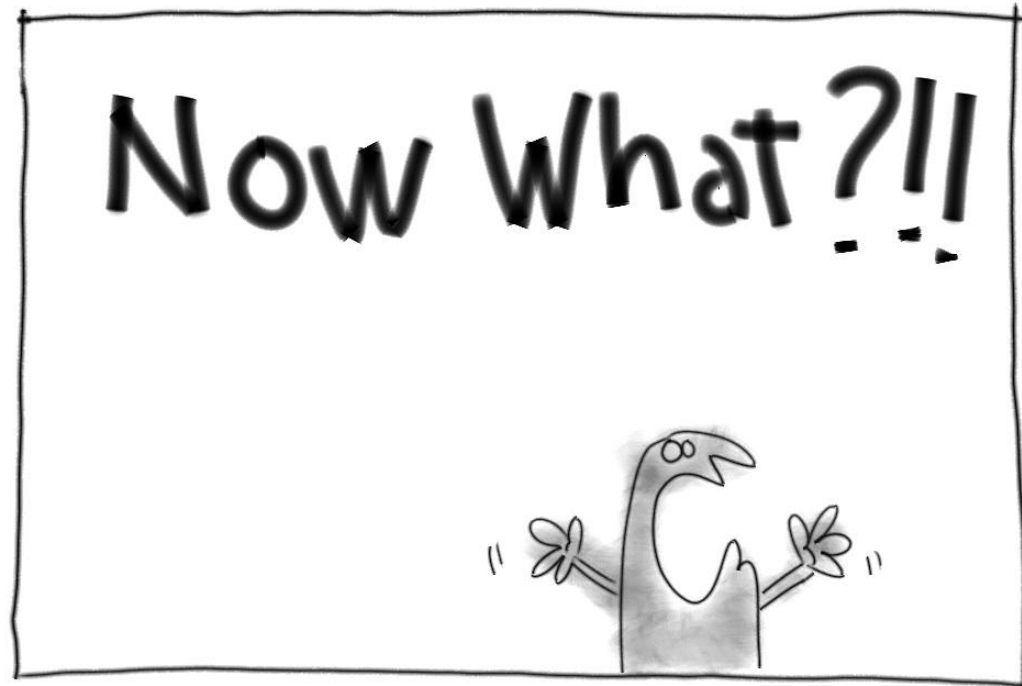


```
skf = model_selection.StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
for idx_tr, idx_ts in skf.split(x_tr, y_tr):
    X_train, Y_train = x_tr[idx_tr], y_tr[idx_tr]
    X_test, Y_test = x_tr[idx_ts], y_tr[idx_ts]
```

Need for guidelines to ensure unbiased model estimates

```
from sklearn.ensemble import RandomForestClassifier
```

```
y_pred = clf.predict(X_test)
```



```
X_train, Y_train = x_tr[idx_tr], y_tr[idx_tr]  
X_test, Y_test = x_tr[idx_ts], y_tr[idx_ts]
```

```
metrics.matthews_corrcoef(y_ts, y_pred)
```

```
ie, random_state=0)
```

The MAQC-II/SEQC initiatives

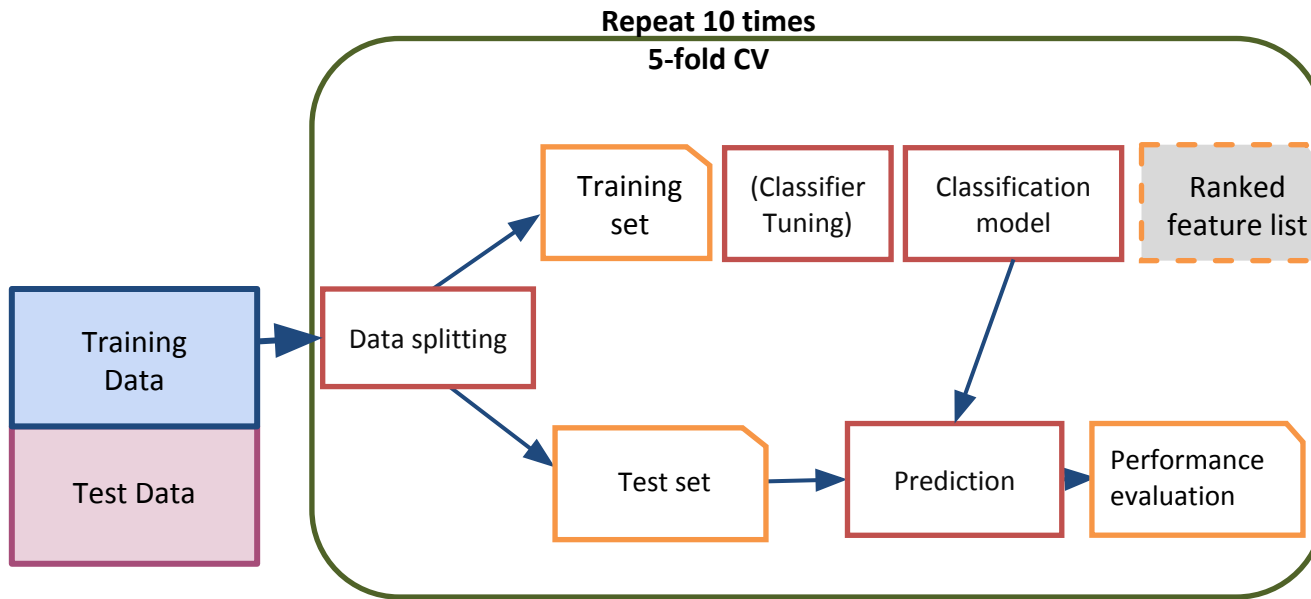
36 independent teams from 14 countries

*Identifying a set of guidelines
for predictive profiling
(microarray and NGS)
through a suite of testbed
datasets*

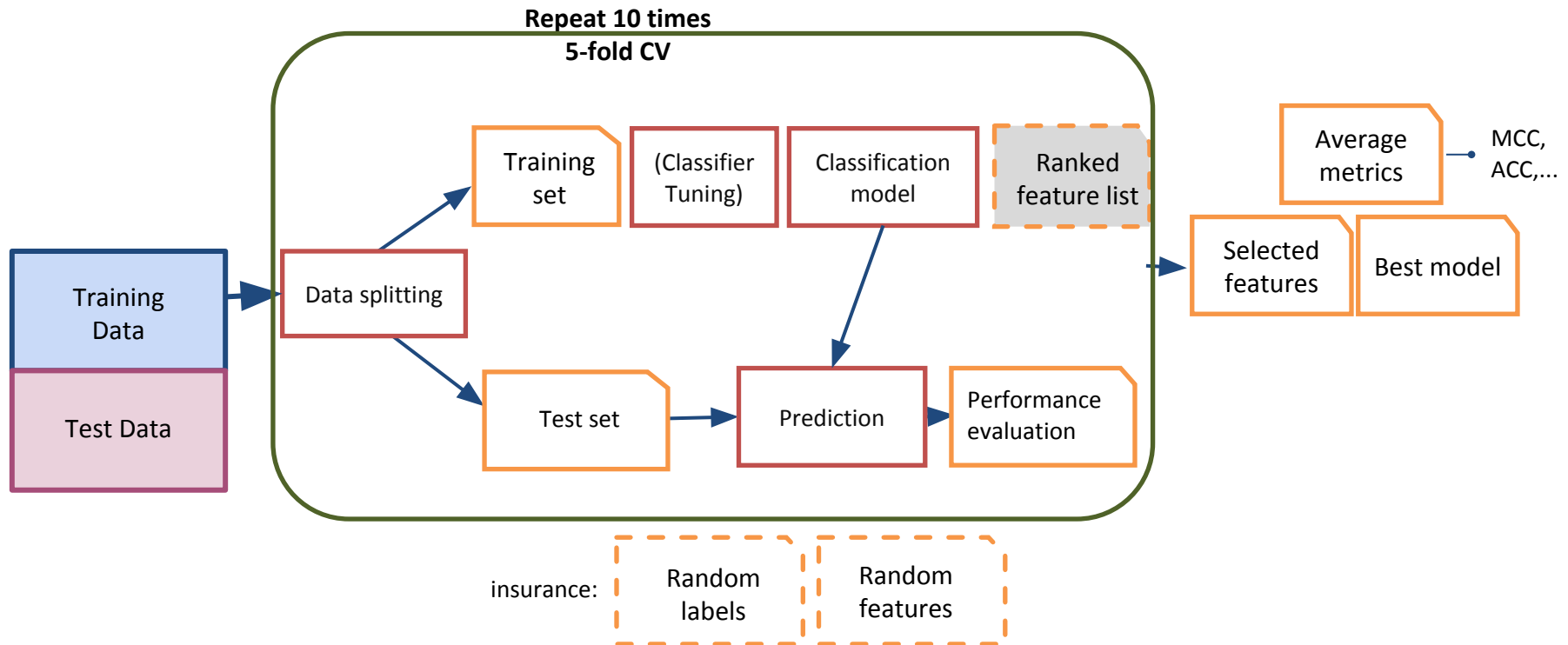
1. Predictive models can be derived from high-throughput data,
2. But they need to be carefully developed and independently tested
3. Reproducibility requires substantial effort.



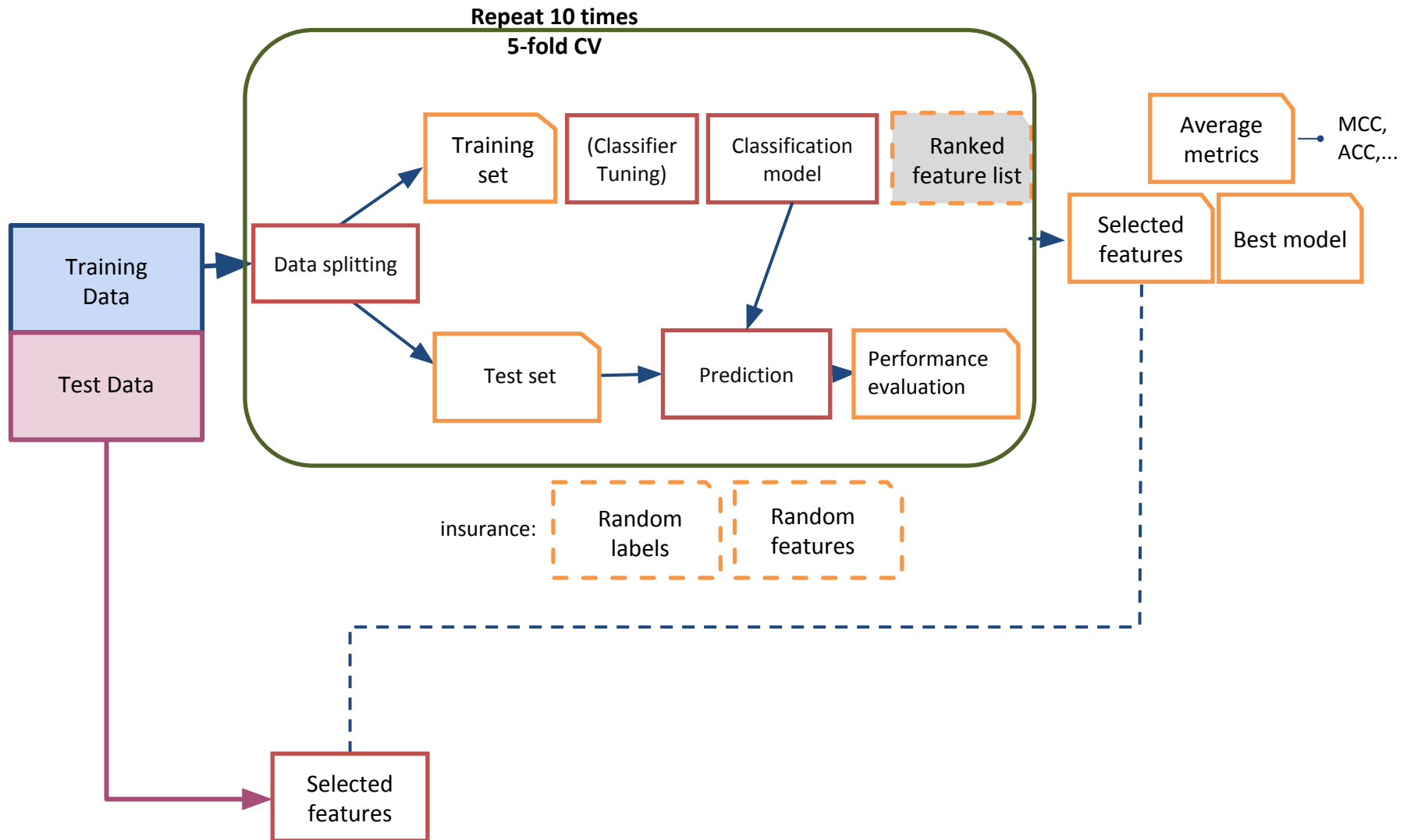
MAQC-II/SEQC Data Analysis Plan



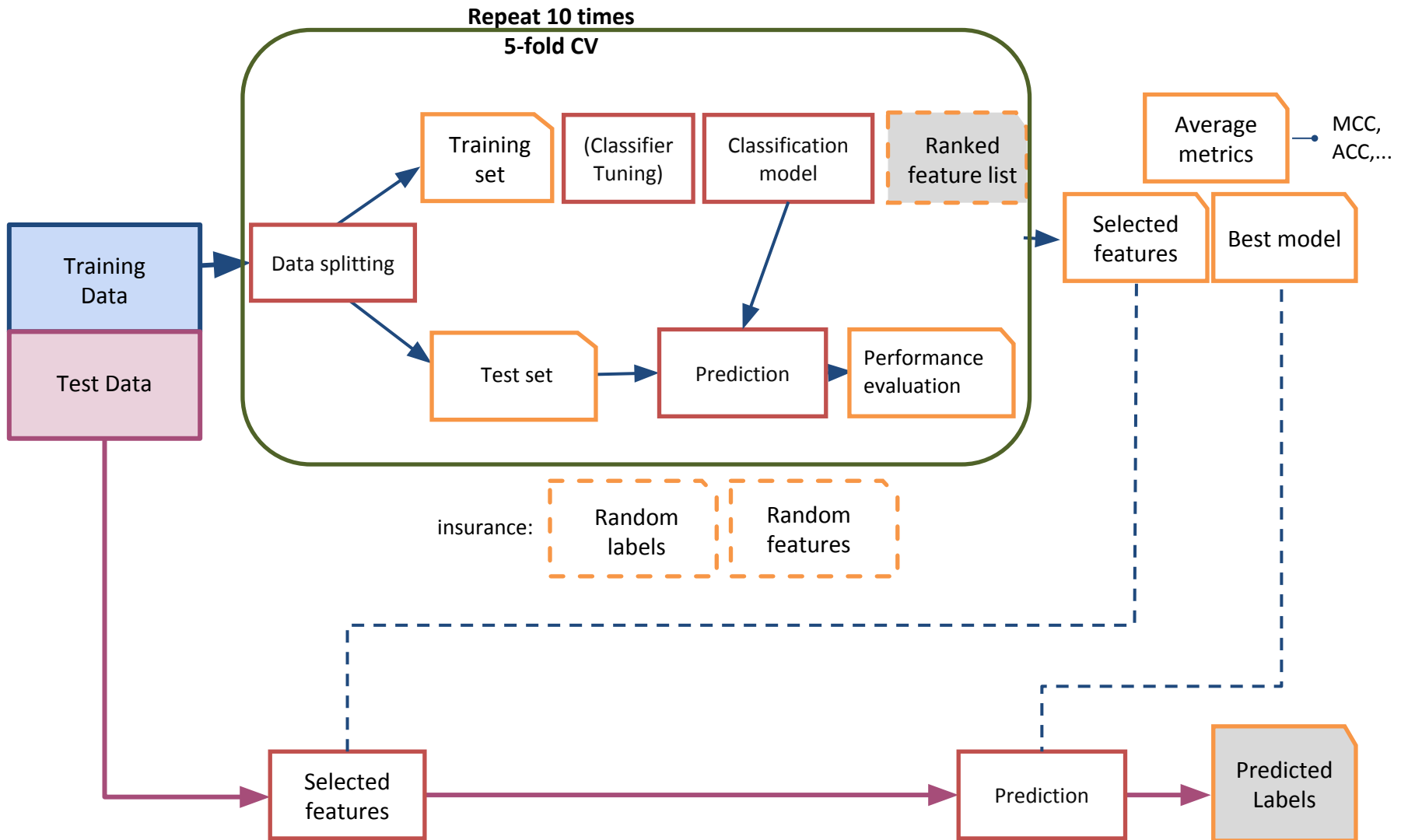
MAQC-II/SEQC Data Analysis Plan



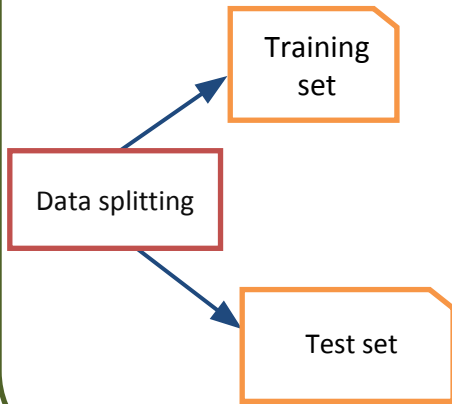
MAQC-II/SEQC Data Analysis Plan



MAQC-II/SEQC Data Analysis Plan



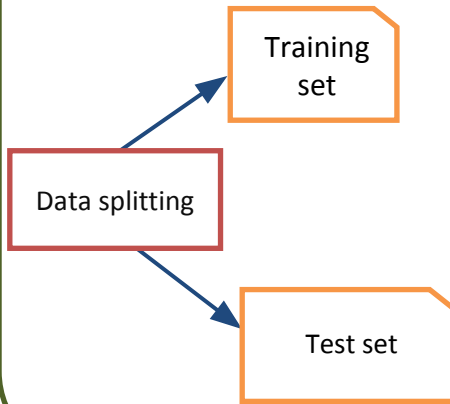
5-fold CV



```
skf = cross_validation.StratifiedKFold(n_splits=5, shuffle=True, random_state=n)
for idx_tr, idx_ts in skf.split(x_tr, y_tr):
    X_train, y_train = x_tr[idx_tr], y_tr[idx_tr]
    X_test, y_test = x_tr[idx_ts], y_tr[idx_ts]
```

Repeat 10 times

5-fold CV



```
for n in range(10):
```

```
    skf = cross_validation.StratifiedKFold(n_splits=5, shuffle=True, random_state=n)
```

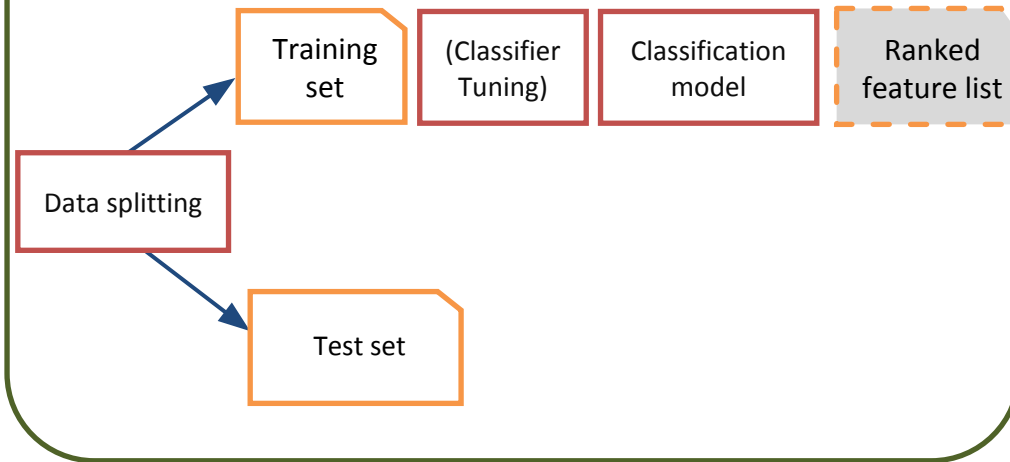
```
    for idx_tr, idx_ts in skf.split(x_tr, y_tr):
```

```
        X_train, y_train = x_tr[idx_tr], y_tr[idx_tr]
```

```
        X_test, y_test = x_tr[idx_ts], y_tr[idx_ts]
```

Repeat 10 times

5-fold CV



```
for n in range(10):
```

```
    skf = cross_validation.StratifiedKFold(n_splits=5, shuffle=True, random_state=n)
```

```
    for idx_tr, idx_ts in skf.split(x_tr, y_tr):
```

```
        X_train, y_train = x_tr[idx_tr], y_tr[idx_tr]
```

```
        X_test, y_test = x_tr[idx_ts], y_tr[idx_ts]
```

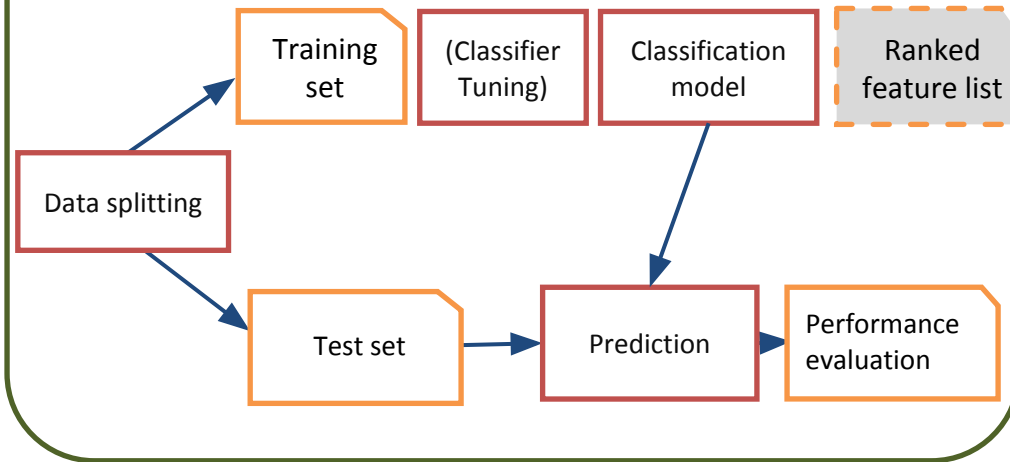
```
        clf = RandomForestClassifier(n_estimators=500, random_state=n)
```

```
        clf.fit(X_train, y_train)
```

```
        ranking = np.argsort( clf.feature_importances_ )[::-1]
```

Repeat 10 times

5-fold CV



```
for n in range(10):
```

```
    skf = cross_validation.StratifiedKFold(n_splits=5, shuffle=True, random_state=n)
```

```
    for idx_tr, idx_ts in skf.split(x_tr, y_tr):
```

```
        X_train, Y_train = x_tr[idx_tr], y_tr[idx_tr]
```

```
        X_test, Y_test = x_tr[idx_ts], y_tr[idx_ts]
```

```
        clf = RandomForestClassifier(n_estimators=500, random_state=n)
```

```
        clf.fit(X_train, Y_train)
```

```
        ranking = np.argsort( clf.feature_importances_ )[::-1]
```

```
        for s in [1, 5, 10, 25, 50, 100]:
```

```
            v = ranking[:s] # consider the top s ranked features
```

```
            X_tr_fs, X_ts_fs = X_train[:, v], X_test[:, v]
```

```
            clf.fit(X_tr_fs, Y_train)
```

```
            yp = clf.predict(X_ts_fs)
```

```
            mcc = matthews_corrcoef(Y_test, yp)
```

```
            [...]
```

Thank you!!



Questions?