Machine Learning for Text Analytics

Andrea Esuli

Machine Learning

Machine learning (ML) is a discipline of Computer Science that is focused on how to enable a computer to perform a task without explicitly programming it.

There are three main models of ML:

- Supervised ML
- Unsupervised ML
- Reinforcement learning

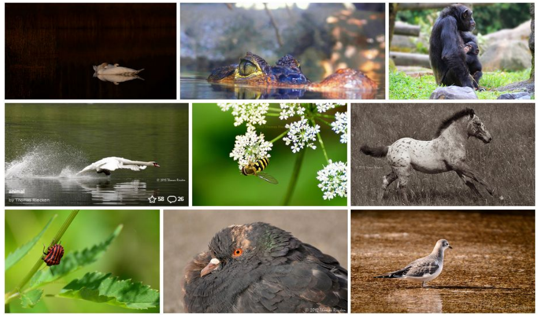
Supervised ML

The supervised ML model follows a learning by example metaphor.

To teach babies to recognize cats, you show them example pictures of what is a cat and what is not, so that each baby develops a internal model of the "cat" concept and uses it for recognition.

cat =

not cat =



The correct assignments of "cat" and "not cat" labels are the **supervised information** needed to learn the model.

Unsupervised ML

In the unsupervised ML model there is no explicit goal reach.

The machine is presented with an input which can be processed freely in order to:

- give structure to the input data
- extract useful information that is latent in the input data

The output can be in human-readable form or used as the input of another process.

Semi-supervised ML

Semi-supervised ML is a in-between scenario between the supervised and unsupervised ones.

The data the machine can work on is split into two parts:

- a set of labeled data, usually small
- a set of unlabeled data, usually large

Semi-supervised ML

Semi-supervised approaches may vary a lot, but they all share some intuitions:

- leveraging on the few labeled examples to model what the goal is
- heuristically applying the labels on unlabeled data exploiting the similarities with labeled data
- use the enlarged labeled data set to better *generalize* the model

Reinforcement Learning

The reinforcement learning model is a biologically-inspired model in which an *agent* acts in an *environment*, observing the outcome of its actions and adapting its behavior to *fit* in the environment.

There is no concept of example, like in supervised learning.

There is a concept of *reward* (either positive or negative) with respect to the actions taken during the exploration.

ML for Text Mining

Unsupervised TM

Unsupervised TM applications are typically oriented toward a relatively low-cost inspection of the gathered data, in order to:

- explore the data composition and quality
- visualize some of its properties
- support the definition of a (supervised) classification schema
- annotate latent properties of data for successive use by other TM methods

Unsupervised TM

Some unsupervised TM processes:

- topic modeling
- clustering
- lexical analysis
- summarization (also done in a supervised way)

Clustering

Clustering is the task of grouping a set of items by their similarity so that items in the same group (i.e., a cluster) are more similar to each other than to the other objects.

The similarity model is a key element of the process.

For example, similarity can be expressed as:

- a (metric) distance between items
- closeness in terms of graph walk distance
- position in a space modeled as density regions or probability distributions

Text Clustering

Document clustering can be performed in order to support various tasks, e.g.:

- ease document browsing
- visualize documents distribution
- support the definition of a classification schema
- improve the efficiency of (web) search
  - reduced index size, quicker search
  - enrich search results

Text Clustering

Clustering in scikit-learn:

```
...
pipeline = Pipeline([
    ('vect', CountVectorizer(analyzer=analyzer, min_df=5)),
    ('tfidf', TfidfTransformer()),
])

X = pipeline.fit_transform(texts)

clusterer = KMeans(n_clusters=args.clusters)

labels = clusterer.fit_predict(X)
```

Supervised TM

Supervised text mining problem have an explicit goal, modeled by a set of training examples provided in input:

- classification
- regression
- extraction
- summarization
- quantification

Text classification

The simplest text classification (TC) problem is the one of *binary classification*:

Given a document d and a class c, determine if $d \in c$ or $d \in \neg c$.

- d consists of a piece of text, usually with limited, if any, structure information (e.g., title, sections, hyperlinks, typographic styles).
- c is a symbolic label that denotes a shared characteristic among the documents that have that label assigned.
- $\neg c$ is the complement of c, i.e., the set of all documents that do not have the characteristic denoted by c.

Text classification

Example: determine if these documents belongs to the class c =*Computing*:

```
d1 = 'Django is a free and open source web application framework,
written in Python, which follows the model/view/controller (MVC)
architectural pattern.'

d2 = 'Python is a widely used general-purpose, high-level
programming language.'

d3 = 'Django is a 1966 Italian Spaghetti Western film directed by
Sergio Corbucci and starring Franco Nero in the eponymous role.'

d4 = 'Monty Python and the Holy Grail is a 1975 British comedy
film written and performed by the comedy group of Monty Python'
```

Historical notes

Before the affirmation of machine learning-based TC, TC was a *Knowledge Engineering* problem, in which a *domain expert* **manually** defined a set of rules to assign documents to classes.

```
if('open source'∈ d or 'programming language'∈ d)
then c
else ¬c
```

This process requires a domain expert thas is also expert in rules formulation.

Rules easily grow out of control for complex domains and large inputs.

```
'Big Buck Bunny is an open source movie from the Peach Open Movie
Project.'
```

Supervised learning for classification

Given a document domain D and a class c, the classification problem is represented by a - mostly unknown - classification function

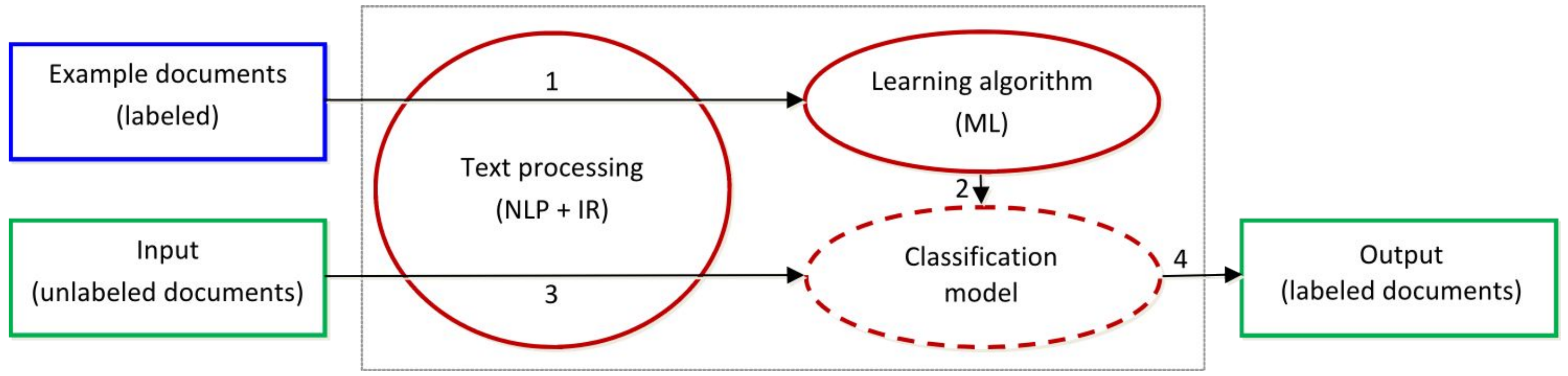
$$Y_c : D \rightarrow -1,+1$$

The learning algorithm learns an *approximated* classification function (also called model)

$$\tilde{Y}_c : D \rightarrow -1,+1$$

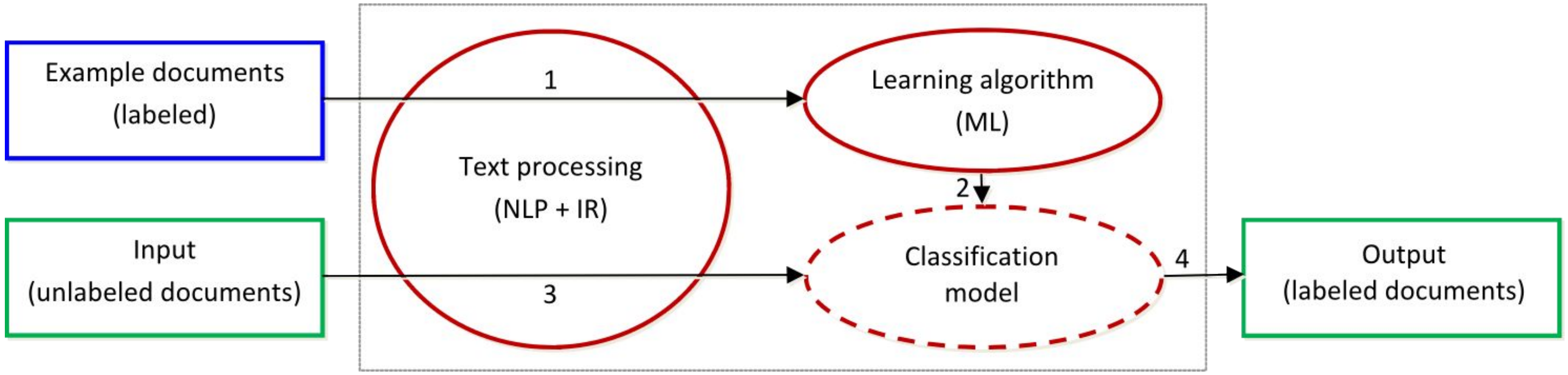
by *observing* a set of known classification cases (*training set*), i.e., documents for which the true classification label ($Y_c(d)$) is known.

The pipeline



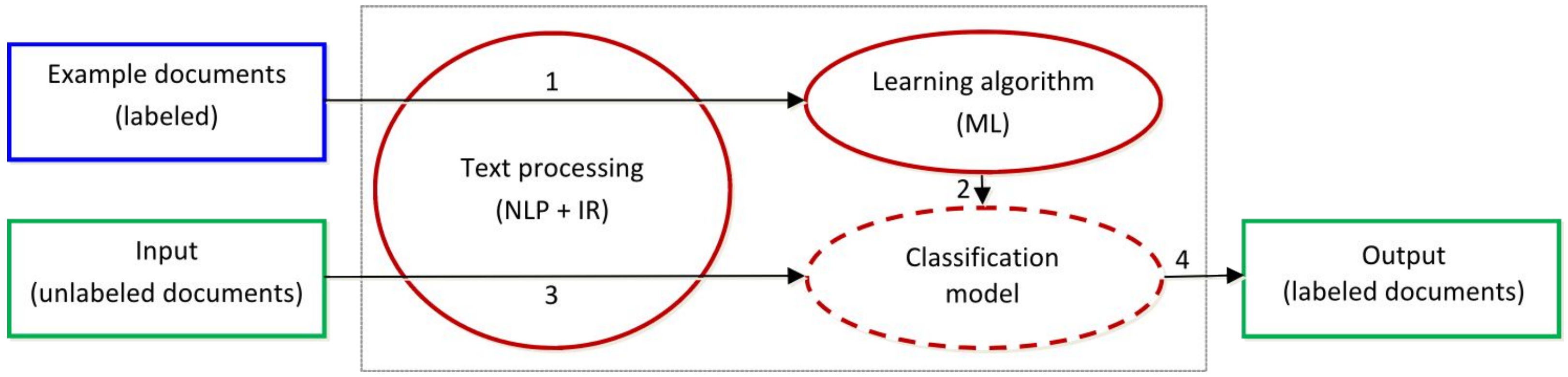
- (Indexing - 1) Labeled documents (training set) are processed by means of NLP and IR techniques to create a vectorial representation of its content.
- (Training - 2) Vectors and labels are fed to a machine learning algorithm that learns a model of the classification problem.

The pipeline



- (Indexing - 3) Unlabeled documents are processed in order to create vectorial representations that are consistent with those created from the training set.
- (Classification - 4) The classification model is applied to the test set, labeling its documents.

The pipeline



Note that the Indexing step 1, not only transforms text, but also defines ("fits") the transformation function (e.g., feature space, idf values).

That exact transformation function is then used in Indexing step 3.

ML algorithms

*Most** machine learning algorithm cannot take plain text as input.

They require the text to be converted into a machine processable format:

- a set of items,
- a probability distribution,
- a vector of real values, depending on the algorithm type,

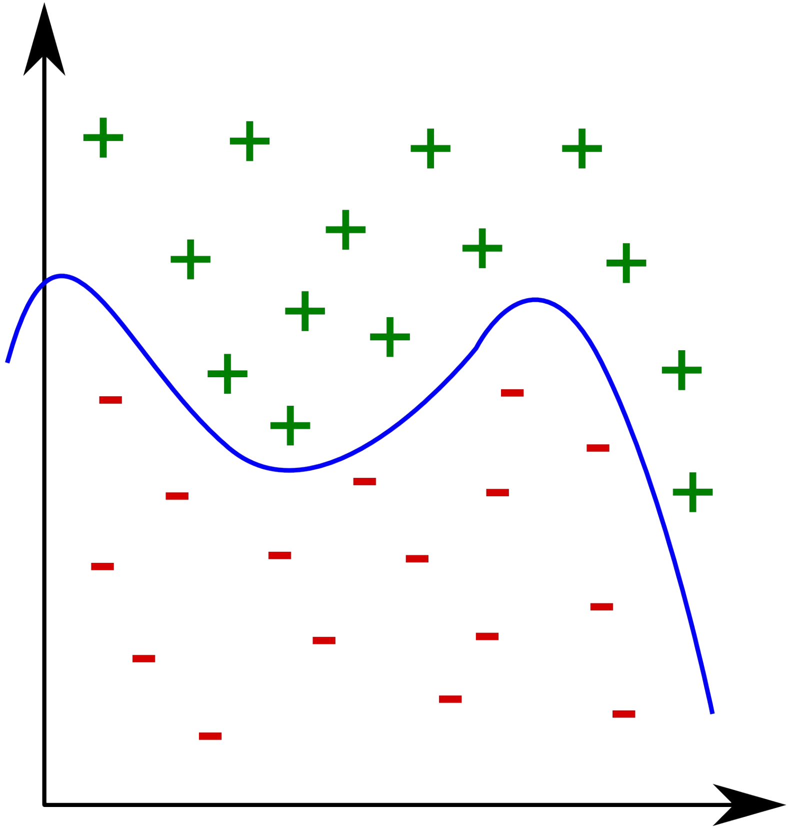
depending of the type of model the algorithm is designed to fit.

* for example, Neural Networks can actually perform better** when processing text as a sequence of characters
** A LOT of text is required to make it work.

Learning

The learning algorithm observes the labeled examples and determines from them the parameters of a classification model.

The classification model is configured in order to achieve the best separation between positive ($d \in c$) and negative ($d \in \neg c$) examples.



Probabilistic: Naïve Bayes

Naïve Bayes is a probabilistic learner that uses the Bayes Theorem:

$$p(c, d) = p(c|d)p(d) = p(d|c)p(c)$$

$$p(c|d) = \frac{p(d|c)p(c)}{p(d)}$$

making a strong independence assumption between the features.

$$p(c|d) \propto p(c) \prod_{f \in F} p(w_{df}|c)$$

It is fast, but it has a **strong bias** and is usually suboptimal on text, since words are often correlated (N-grams models partially counter this problem).

Naïve Bayes: log space

The multiplication of many (very) small probability values may lead to underflow errors.

Usually the models use the logs of the probabilities to work in a linear space:

$$log(\Pi_i P(w_i)) = \Sigma_i \, log(P(w_i))$$

The classifier is thus formulated as:

$$NB(d)=arg\,max_j \, log(P(c_j))+\Sigma_{w^i \in d} \, log(P(w_i|c_j))$$

Where $P(c_j)$ and $log(P(w_i|c_j)$ are precomputed on train data (plus smoothing for unknown words).

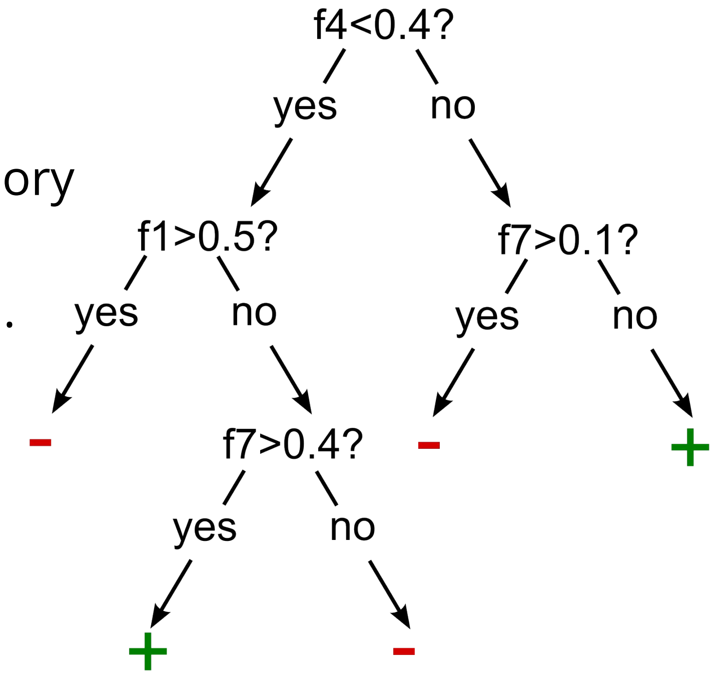
Decision Trees

A *Decision Tree* (DT) generates a tree whose nodes are *yes/no* question on features.

DTs pick as nodes the most informative features (considering parent nodes), using information theory measures, e.g., those used for feature selection.
In fact, DTs ***implicitly perform a feature selection***.

DT are prone to overfitting.

Random Forest learns a committee of decision trees, each one trained on a randomized sample of the training set.



SVM

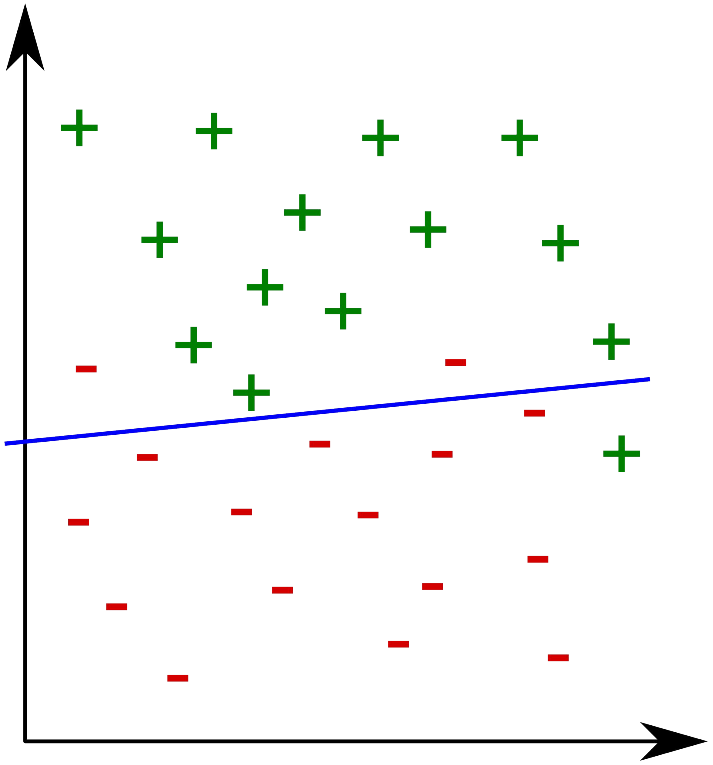
Linear Support Vector Machine (SVM) uses a model of the form:

$$w \cdot x - b = 0$$

to determine the hyperplane that better separates the examples.

Cannot work on non-linearly separable problems.

Usually works fine with text (high-dimensional, sparse).



SVM

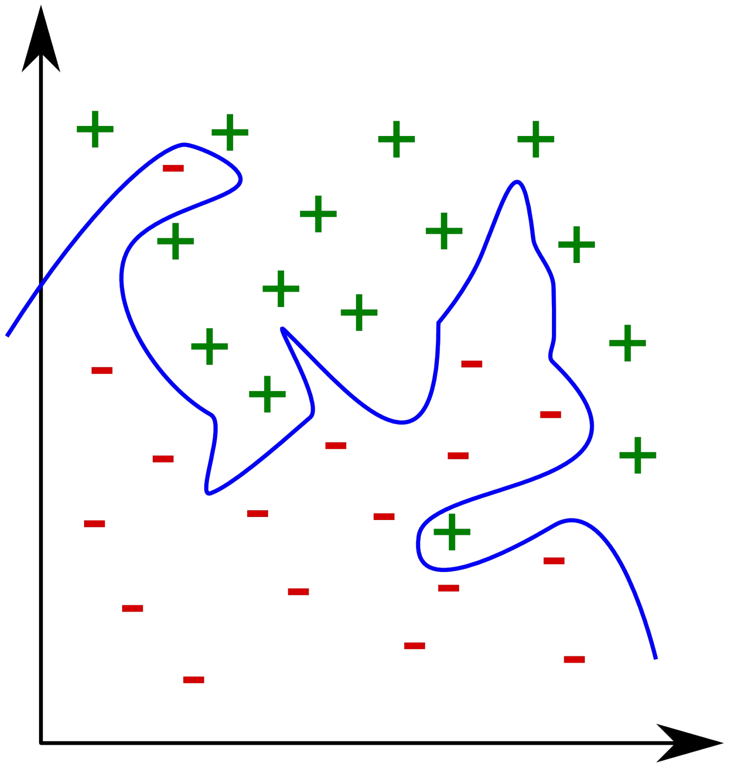
SVMs use kernels $K(\mathbf{x}, \mathbf{x}')$
to project data in **transformed spaces**
so as to learn more complex models.

- Polynomial: $(\mathbf{x}^\mathsf{T}\mathbf{x}' + c)^d$
- Radial basis function:

$$\exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$$

- Sigmoid: $\tanh(\mathbf{x}^\mathsf{T}\mathbf{x}') + r$

Must avoid **overfitting**.



Binary vs Multi-class

When the classification labels are **n=2** we face a **binary** classification problem.

When the classification labels are **n>2**, we face a **multi-class** classification problem, which can be further defined as:

- **multi-class multi-label**: a document can be assigned to any number of labels l, i.e., $0 \leqslant l \leqslant n$.
- **multi-class single-label:** a document must be assigned to one and only one label.

Multi-class

Multi-class multi-label classifiers are typically trained learning **n** independent binary classifiers.

There are two main methods to train a multi-class single-label classifier:

One-vs-one (OvO):

- for every possible pair of classes c_j,c_i a binary classifier is trained on the subset of positive example for those two classes
- the classification is done by applying all the pair-wise classifiers and assigning the labels that "wins" more classifications.

Multi-class

There are two main methods to train a multi-class single-label classifier:

One-vs-rest (alias One-vs-all, OvR, OvA):

- for every class c_i a binary classifier is trained using as the positive examples those belonging to c_i and as the negative examples those belonging to any other class
- the classification is done by applying all the classifiers and assigning the labels that obtains the highest scores (arg max).

One-vs-rest requires to train less classifiers, and it is usually preferred.

Explainability

"A decision tree model can be inspected and explained by a human, while an SVM model cannot be inspected and explained."

NOT TRUE

None of the two algorithms is designed for inspection and explainability.

On some problems (e.g., those with few features on "everyday" scales) DT models can expose in clear way how the features contribute to decisions.

Some SVM kernels (special those that perform best on low-dimensional, dense data, e.g., RBF) transform the vector space in a way that the model weights are difficult to correlate to output.

Explainability

Textual data produces many, many features, DT can become quite branched and deep.

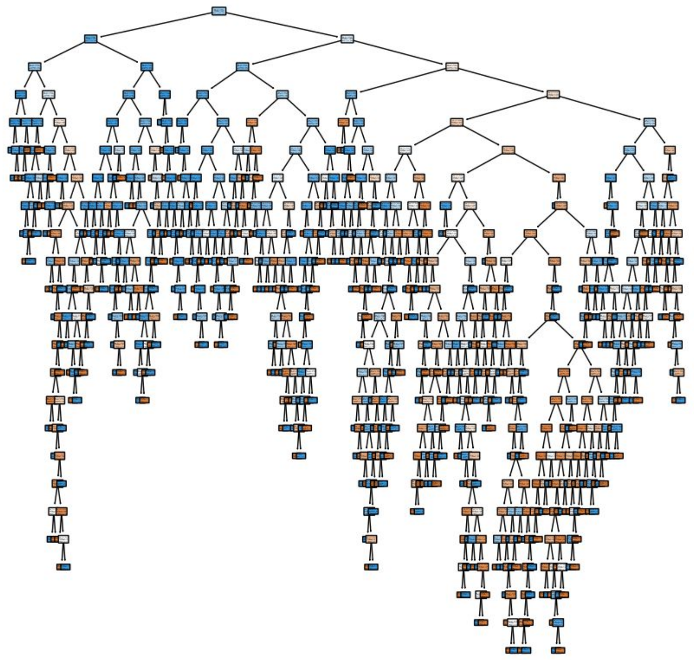
A path to a decision may be long, upper nodes may lead to many different decisions.

Values of features are on unusual scales, not linked to human perception (e.g., tf-idf).

tf-idf('my name')>0.032

DT Forests have a lot of trees, how each of them contributes to the final answer?

Decision tree for detection of Italian native speakers writing in English:



Explainability

Data is usually sparse, so a linear kernel can be successfully used.

Weights of linear kernels are easy to understand in terms of positive/negative correlation (and its strength) of the feature with respect to the output.

E.g., SVM weights of trigram features for detection of Italian native speakers writing in English:

I_think_that:	2.89	a_lot_of:	2.27
agree_with_the:	1.81	I_'_m:	1.62
In_my_opinion:	1.53	._In_fact:	1.02
._However_,:	-0.56	,_however_,:	-0.52
._Also_,:	-0.52	should_not_be:	-0.45
._He_is:	-0.37	as_well_as:	-0.35

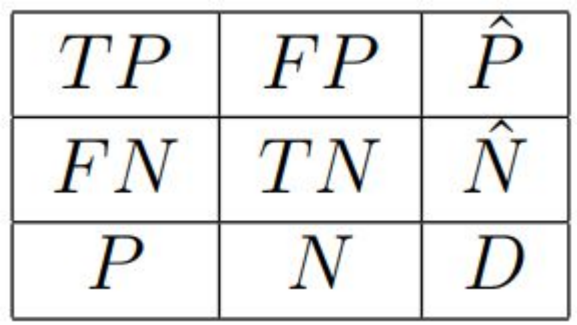
Evaluation

Contingency table

The learned classifier $\hat{Y}$ is applied to unlabeled documents.

Labels are in fact known, but kept hidden to the classifier (test set).

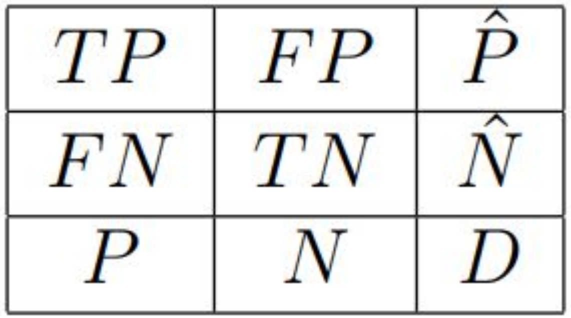
Predicted labels are compared to true labels from test set, building a contingency table:



TP	FP	$\hat{P}$
FN	TN	$\hat{N}$
P	N	D

Contingency table

Predicted labels are compared to true labels from test set, building a contingency table:



TP	FP	$\hat{P}$
FN	TN	$\hat{N}$
P	N	D

- TP = true positive, document correctly labeled with the category label
- FP = false positive, document wrongly labeled with the category label
- FN = false negative, document wrongly not labeled with the category label
- TN = true negative, document correctly not labeled with the category label

Accuracy

TP	FP	$\hat{P}$
FN	TN	$\hat{N}$
P	N	D

Various measures can be used to evaluate the classifier:

- Accuracy

$$\frac{TP + TN}{D}$$

Accuracy **is not fair on unbalanced sets**: if only 1% of test documents belong to the category, then saying always “no” yields a 99% accuracy.

Precision & Recall

- Recall, ability to find positive items

$$\rho = \frac{TP}{TP + FN} = \frac{TP}{P}$$

- Precision, accuracy on positive labels

$$\pi = \frac{TP}{TP + FP} = \frac{TP}{\hat{P}}$$

100% Recall can be achieved by saying always "yes".

- In that case precision will be P/(P+N).

F1

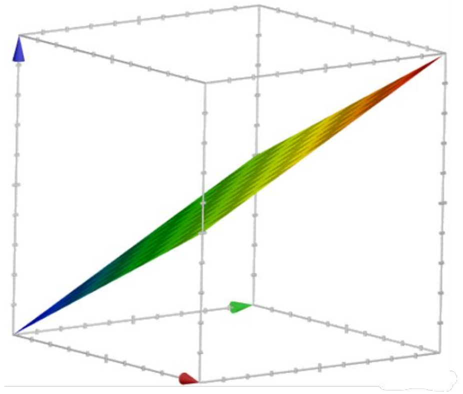
By combining precision and recall we can obtain a measure that cannot be cheated using trivial classifiers:

- F1 is the **harmonic mean of precision and recall**

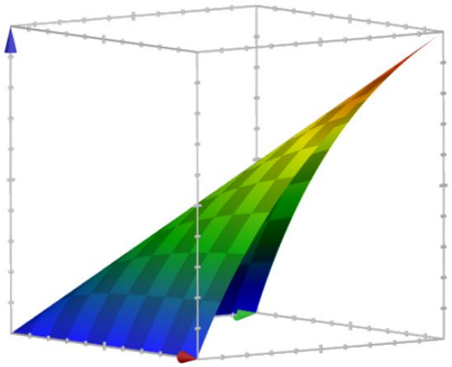
$$F_1 = 2\frac{\pi\rho}{\pi + \rho} = \frac{2TP}{2TP + FP + FN}$$

Evaluation in SciKit-Learn

Arithmetic mean



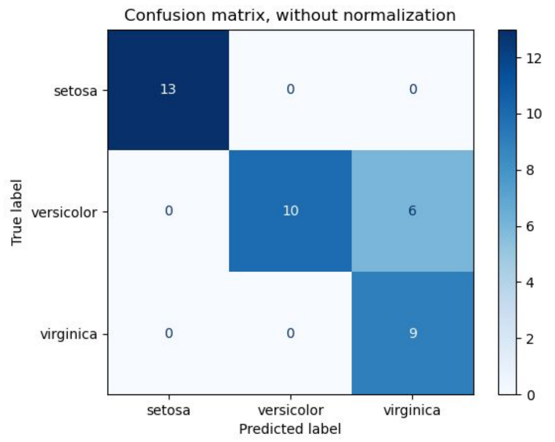
Harmonic mean

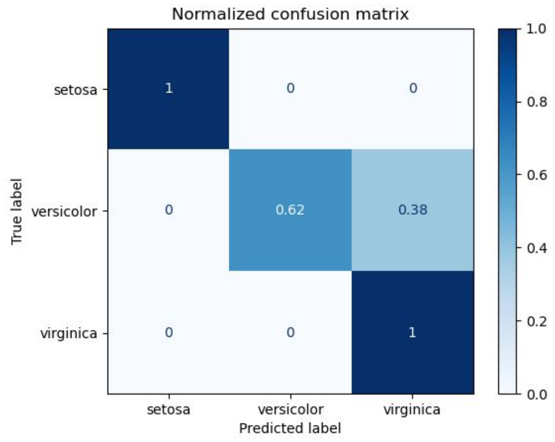


Confusion table

In multi-class single-label classification problem we can compute a **confusion table**, which show the counts of how many documents belonging to a class c_i (true label) have been assigned to a class c_j (predicted label).

Cells in a confusion table may reports raw counts (highlighting more populated classes) or row-normalized values (giving to every class equal importance).





Micro- and macro-averaging

Evaluation of multi-class multi-label problems is carried out by evaluating the binary classifier for each class.

For each class a contingency table T_i is computed.

Two evaluation models:

- **micro-averaged**: cells of all T_i tables are summed up in a single T table, then evaluation measures are applied to it. **Classes with more positive cases weight more** in the final result.
- **macro-averaged**: evaluation measures e_i are computed distinctly on each T_i, then the final e values are computed as the average of all e_i values. This gives **equal weight to all classes**.

Scikit-learn

Scikit-learn

Scikit-learn is a Python package that implements a rich amount of machine learning methods and tools (except complex neural networks).

It implements the concept of the processing pipeline through a shared interface for all of its components: tokenization, weighting, selection, learning, prediction.

```
pipeline = Pipeline([
    ('vect', CountVectorizer()),  # feature extraction
    ('sel', SelectKBest(chi2, k=5000)),  # feature selection
    ('weight', TfidfTransformer()),  # weighting
    ('learner', LinearSVC())  # learning algorithm
])
```