Experimental protocols

Andrea Esuli

Designing experiments

Experiments

Experiments in a **controlled environment** are a good tool to predict the effectiveness of a method and to compare competing ideas.

In text analytics the controlled environment is usually defined by one or more **annotated corpora** that model the task of interest.

Data in a corpora can be exploited in many ways in order to perform an experiment, following different **experimental protocols**.

The most used protocol, and the basic element of many others, is the one based one **train and test**.

Every protocol has pros and cons with respect to **computational cost, reproducibility, and statistical relevance of results**.

Training & Test

The train and test protocol splits the dataset in two parts:

- A training set, which is the actual data on which the ML algorithm is **trained** to produce a **model**.
- A test set, which is the data on which the **model** is **evaluated**.

Information from test set must be NEVER used to train the model or in any decision regarding the definition of the training process

otherwise we break the possibility to consider the results obtained on test set to be a good representative of the results obtainable on any data.

The train and test protocol is the basic element used in many other protocols, to perform experiments and also **optimization** in the training phase.

Optimization of parameters

A ML pipeline may have many parameters that must be set and that can have an impact on the quality of results:

- Which features to extract?
- What lexicons to use, how?
- Use of tagging, parsing. How to use it?
- Feature selection: measures and amount
- Weighting functions
- Learner and its parameters

Optimization of parameters

Which is the optimal choice for a specific task?

We cannot test all the possibilities on test set, as this would mean fitting the method on test data, i.e.:

- cheating with respect of any method that has not been optimized on test set
- weakening the generalizability of the result to unseen data

Any decision regarding the method we are working on must not be based on test data.

Optimization of parameters

Optimization of parameters can be done using **only the training data,** by considering the training set as a smaller dataset on which we can use the results on the test part (typically called **validation set**) to choose the best parameters.

Once the best parameters are identified, the whole training set is used to fit a new model using such best parameters, and then the model is evaluated on test data.

Similarly to main experiments, also experiments for the optimization of parameters can use any protocol.

Experimental protocols

Train and test

Data is split once and for all in a single training set and a single test set.
E.g.: Reuters21578, RCV1 v2, IMDB dataset.

Pros:

- easy to reproduce
- reasonable to do on time-related data (training data comes before test data)
- experiments are quick to run

Cons:

- risk of overfitting validation data on the long run
- risk of low statistical relevance (test set must be large)

K-fold validation

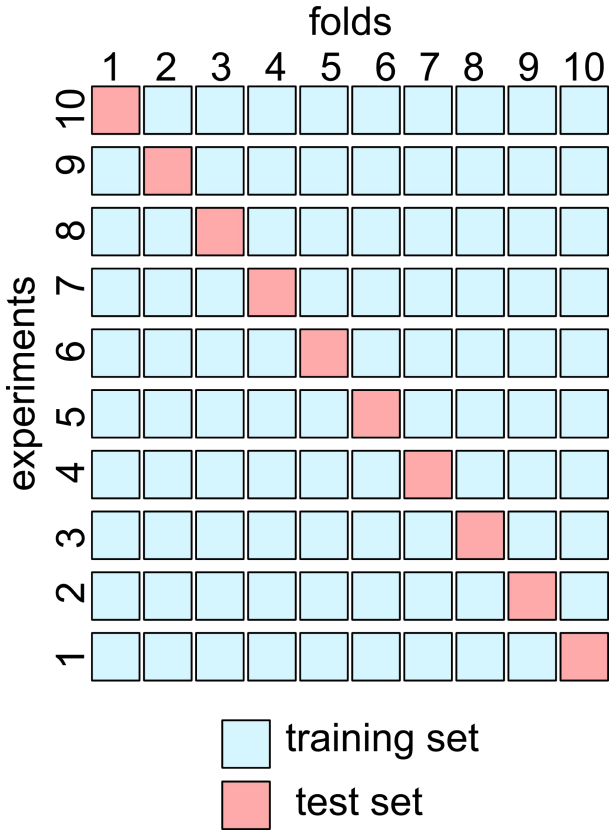
In k-fold validation, data is split in *k* equal sized sets. For k times, k-1 sets are used as the training set and the remaining one as the set set.
E.g., 20 newsgroups, ANY dataset.

Pros:

- improved statistical relevance (the whole dataset is a test set)

Cons

- reproducible by knowing how splits are made
- must check fold composition
- cost of experiment grows linearly with k

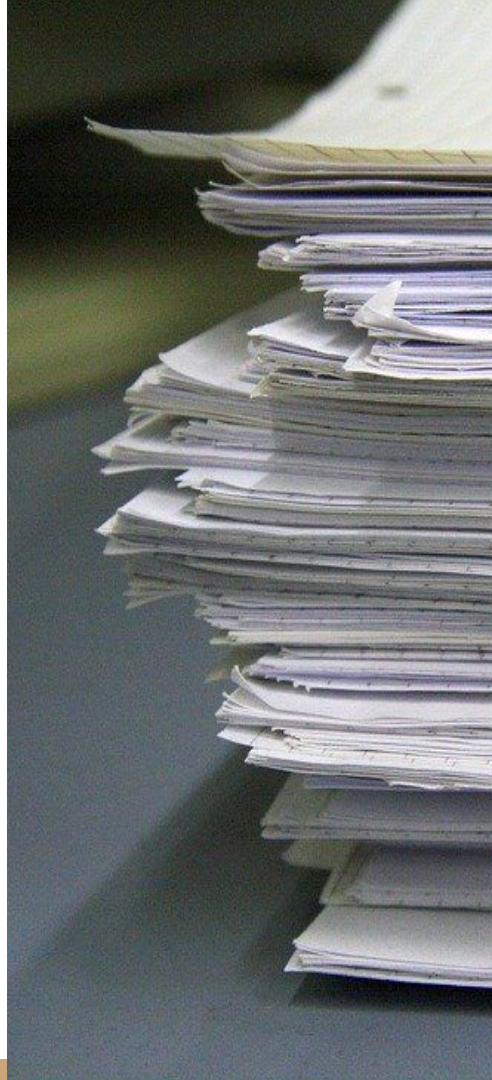


Stratification

Many text classification problems are **highly unbalanced**, i.e., very few positive examples exists for a label.

In such cases there is the risk that a k-fold protocol, or any protocol that samples the training set, may create train-test splits in which **all positive example are in the test set**, making that split almost useless for actual evaluation.

Stratified versions of protocols take care to keep the ratio of positive examples constant across splits, avoiding this issue.



Leave-one-out validation

Leave-one-out validation is an extreme setup of k-fold validation in which k is set to be equal to the dataset size. Test set for each fold is just one document.

Pros:

- really easy to reproduce
- good statistical relevance

Cons:

- very high cost

Random sampling

With random sampling a split proportion is determined, e.g., 80%/20%. For an arbitrary number of times a random train/test split is created and the accuracy measures are recorded.

Pros:

- high statistical relevance
- cost is flexible, can run it until you have resources

Cons:

- hard to reproduce exactly
- requires statistical analysis to put results together

Evaluation

Evaluation

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

Evaluation

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

Evaluation

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.

Evaluation

- 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).

Evaluation

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**

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

Metrics in SciKit-Learn

Parameter optimization

Tips on optimization

Optimization is made against a specific evaluation measure.

A grid search on all the candidate values of all the parameters can produce an explosion in combinations.

For example:

- 5 feature types, testing each feature independently, all together, and all possible pairs.
- 5 feature selection levels
- 10 values for the C parameter of SVM

produce a total of $(5_{single} + 1_{all} + 10_{pairs}) \cdot 5 \cdot 10 = 800$ configurations to be tested

Tips on optimization

Parameters with loose correlation can be optimized in sequence.

- First optimize feature selection amount then optimize C value for SVM

Parameters with lots of possible values can be optimized in two step: coarse search, and refinement.

$$k_{NN} \in \{1, 5, 10, 15, 20, 25, 30, 35, 40\} \rightarrow \{6, 7, 8, 9, 11, 12, 13, 14\}$$

For some numeric parameters a logarithmic search scale is fine.

$$C_{SVM} \in \{0.001, 0.01, 0.1, 1, 10, 10, 1000\}$$

Random-sampling for optimization

Once a grid of configuration for experiments is defined,

- all the experiments can be run exhaustively, or...
- configurations are randomly sampled from the grid, and the relative experiment is executed, until a given experiment budget is consumed.

Sklearn has implementations of both methods.

Optimization-overfitting-generalization

Optimizing too many parameters on the same validation set may lead to overfitting on that validation set.

Methods to avoid overfitting typically aim at building more general methods.

Yet, too general models can fail to capture key aspect of a problem.

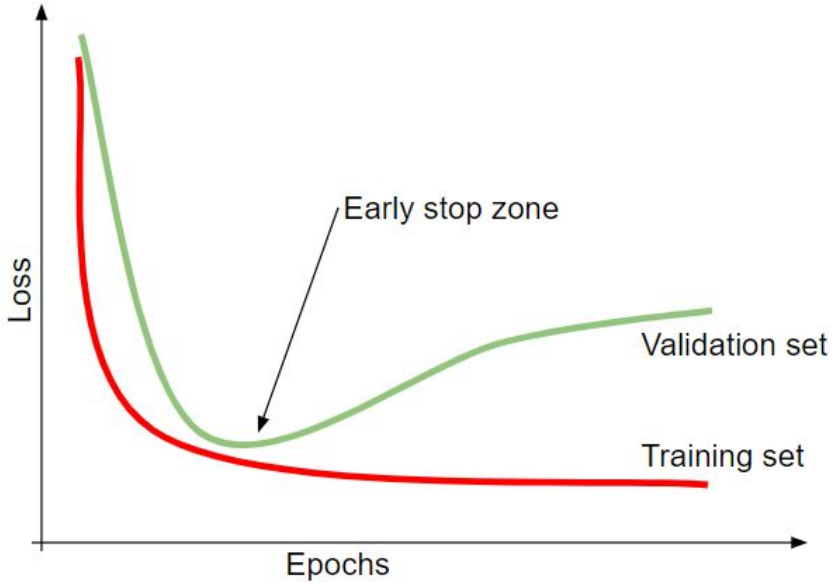
There is no objective/quantitative way to tell when the right overfitting/generalization trade-off is reached in machine learning.

The most solid, yet obvious, heuristic we can rely on is that more supervised information we have the better models we can expect to obtain.

Optimization-overfitting of Neural Networks

When training a neural network:

- **decrease** in **training loss** is the main indicator of the effectiveness of the learning process, yet...
- ...**overfitting** may occur: a **validation set** should be tested periodically in order to *guess* when it happens.



Most DL package implement tools and policies to control the learning process with respect to the performance on a validation set.

Beware of Machine Learning Gremlins!