Evaluation

Connectionist and Statistical Language Processing

Frank Keller

keller@coli.uni-sb.de

Computerlinguistik
Universität des Saarlandes

Overview

- training set, validation set, test set
- holdout, stratification
- crossvalidation, leave-one-out
- comparing machine learning algorithms
- comparing against a baseline
- precision and recall
- evaluating numeric prediction

Literature: Witten and Frank (2000: ch. 6), Mitchell (1997: ch. 5).

Training and Test Set

For classification problems, we measure the performance of a model in terms of its *error rate:* percentage of incorrectly classified instances in the data set.

We build a model because we want to use it to classify new data. Hence we are chiefly interested in model performance on new (unseen) data.

The *resubstitution error* (error rate on the training set) is a bad predictor of performance on new data.

The model was build to account for the training data, so might *overfit* it, i.e., not generalize to unseen data.

Test and Validation Set

Use two data set: the *training set* (seen data) to build the model (determine its parameters) and the *test set* (unseen data) to measure its performance (holding the parameters constant).

Sometimes, we also need a *validation set* to tune the model (e.g., for pruning a decision tree). The validation set can't be used for testing (as it's not unseen).

All three data set have to be representative samples of the data that the model will be applied to.

Holdout

If a lot of data data are available, simply take two *independent samples* and use one for training and one for testing.

The more training, the better the model. The more test data, the more accurate the error estimate.

Problem: obtaining data is often expensive and time consuming. Example: corpus annotated with word senses, experimental data on subcat preferences.

Solution: obtain a limited data set and use a *holdout procedure.* Most straightforward: random split into test and training set. Typically between 1/3 and 1/10 held out for testing.

Stratification

Problem: the split into training and test set might be unrepresentative, e.g., a certain class is not represented in the training set, thus the model will not learn to classify it.

Solution: use *stratified holdout*, i.e., sample in such a way that each class is represented in both sets.

Example: data set with two classes A and B. Aim: construct a 10% test set. Take a 10% sample of all instances of class A plus a 10% sample of all instances of class B.

However, this procedure doesn't work well on small data sets.

Crossvalidation

Solution: *k-fold crossvalidation* maximizes the use of the data.

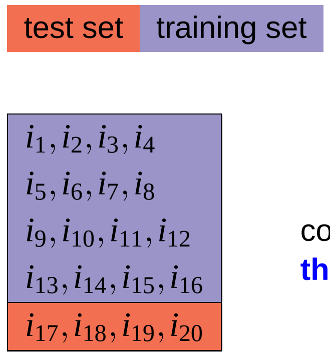
- Divide data randomly into k folds (subsets) of equal size.
- Train the model on $k-1$ folds, use one fold for testing.
- Repeat this process k times so that all folds are used for testing.
- Compute the average performance on the k test sets.

This effectively uses all the data for both training and testing. Typically $k = 10$ is used.

Sometimes *stratified k-fold crossvalidation* is used.

Crossvalidation

Example: data set with 20 instances, 5-fold crossvalidation


test set
training set
i_1, i_2, i_3, i_4
i_5, i_6, i_7, i_8
$i_9, i_{10}, i_{11}, i_{12}$
$i_{13}, i_{14}, i_{15}, i_{16}$
$i_{17}, i_{18}, i_{19}, i_{20}$

compute error rate for each fold
then compute average error rate

Leave-one-out

Leave-one-out crossvalidation is simply k-fold crossvalidation with k set to n, the number of instances in the data set.

This means that the test set only consists of a single instance, which will be classified either correctly or incorrectly.

Advantages: maximal use of training data, i.e., training on $n-1$ instances. The procedure is deterministic, no sampling involved.

Disadvantages: unfeasible for large data sets: large number of training runs required, high computational cost. Cannot be stratified (only one class in the test set).

Comparing Algorithms

Assume you want to compare the performance of two machine learning algorithms A and B on the same data set.

You could use crossvalidation to determine the error rates of A and B and then compute the difference.

Problem: sampling is involved (in getting the data and in crossvalidation), hence there is variance for the error rates.

Solution: determine if the difference between error rates is statistically significant.

If the crossvalidations of A and B use the same random division of the data (the same folds), then a *paired t-test* is appropriate.

Comparing Algorithms

Let k samples of the error rate of algorithms A be denoted by $x_1, \ldots, x_k$ and k samples of the error rate of B by $y_1, \ldots, y_k$.

Then the t for a paired t-test is:

$$t = \frac{\bar{d}}{\sqrt{\sigma_d^2 / k}} \tag{1}$$

Where $\bar{d}$ is the mean of the differences $x_n - y_n$, and σ_d is the standard deviation of this mean.

Comparing against a Baseline

An error rate in itself is not very meaningful. We have to take into account how hard the problem is.

This means comparing against a *baseline model* and showing that our model performs significantly better than the baseline.

The simplest model is the *chance baseline*, which assigns a classification randomly.

Example: if we have two classes A and B in our data, and we classify each instance randomly as either A or B, then we will get 50% right just by chance (in the limit).

In the general case, the error rate for the chance baseline is $1 - \frac{1}{n}$ for an n-way classification.

Comparing against a Baseline

Problem: a chance baseline is not useful if the distribution of the data is skewed.

We need to compare against a *frequency baseline* instead. A frequency baseline always assigns the most frequent class.

Its error rate is $1 - f_{\max}$, where $f_{\max}$ is the percentage of instances in the data that belong to the most frequent class.

Example: determining when a cow is in oestrus (classes: yes, no). Chance baseline: 50%, frequency baseline: 3% (1 out of 30 days).

More realistic example: assigning part-of-speech tags for English text. A tagger that assigns to each word its most frequent tag gets 20% error rate. Current state of the art is 4%.

Confusion Matrix

Assume a two way classification. Four classification outcomes are possible, which can be displayed in a *confusion matrix:*

		predicted	class
		yes	no
actual class	yes	true positive	false negative
	no	false positive	true negative

True positives (TP): class members classified as class members
True negatives (TN): class non-members classified as non-members
False positives (FP): class non-members classified as class members
False negatives (FN): class members classified as class non-members

Precision and Recall

The error rate is an inadequate measure of the performance of an algorithm, it doesn't take into account the *cost of making wrong decisions.*

Example:

Based on chemical analysis of the water try to detect an oil slick in the sea.

False positive: wrongly identifying an oil slick if there is none.
False negative: fail to identify an oil slick if there is one.

Here, false negatives (environmental disasters) are much more costly than false negatives (false alarms). We have to take that into account when we evaluate our model.

Precision and Recall

Measures commonly used in information retrieval, based on true positives, false positives, and false negatives:

Precision: number of class members classified correctly over total number of instances classied as class members.

$$\text{Precision} = \frac{|\text{TP}|}{|\text{TP}| + |\text{FP}|} \quad (2)$$

Recall: number of class members classified correctly over total number of class members.

$$\text{Recall} = \frac{|\text{TP}|}{|\text{TP}| + |\text{FN}|} \quad (3)$$

Precision and Recall

Precision and recall can be combined in the *F-measure:*

$$F = \frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}} = \frac{2|\text{TP}|}{2|\text{TP}| + |\text{FP}| + |\text{FN}|} \tag{4}$$

Example:

For the oil slick scenario, we want to maximize recall (avoiding environmental disasters). Maximizing precision (avoiding false alarms) is less important.

The F-measure can be used if precision and recall are equally important.

Evaluating Numeric Prediction

Error rate, precision, recall, and F-measure are mainly used for *classification tasks.* They are less suitable for tasks were a *numeric quantity* has to be predicted.

Examples: predict the frequency of subcategorization frames. We want to measure how close the model is to the actual frequencies.

$p_1, \dots, p_n$: predicted values of the for instances $1, \dots, n$
$a_1, \dots, a_n$: actual values of the for instances $1, \dots, n$

There are several measure that compare a_i and p_i.

Mean Squared Error

Mean squared error measures the *mean difference* between actual and predicted values. Used e.g. for connectionist nets:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(a_i - p_i)^2 \tag{5}$$

Often also the *root mean squared error* is used:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_i - p_i)^2} \tag{6}$$

Advantage: has the same dimension as the original quantity.

Summary

No matter how the performance of the model is measured (precision, recall, MSE, correlation), we always need to *measure on the test set,* not on the training set.

Performance on the training only tells us that the model learns what it's supposed to learn. It is not a good indicator of performance on unseen data.

The test set can be obtained using an *independent sample* or *holdout techniques* (crossvalidation, leave-one-out).

To meaningfully compare the performance of two algorithms for a given type of data, we need to compute if a difference in performance is *significant.* We also need to compare performance against a *baseline* (chance or frequency).

References

Mitchell, Tom. M. 1997. *Machine Learning.* New York: McGraw-Hill.

Witten, Ian H., and Eibe Frank. 2000. *Data Mining: Practical Machine Learing Tools and Techniques with Java Implementations.* San Diego, CA: Morgan Kaufmann.