ID2214/FID3214 2020

Programming for Data Science – Methodology

Henrik Boström

Prof. of Computer Science - Data Science Systems
Division of Software and Computer Systems
Department of Computer Science
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
bostromh@kth.se

Outline

Empirical investigations

Recipe for selecting a model and estimating its performance using a hold-out set

Comparing model performance to a baseline level

Comparing model performance to a baseline model

Recipe for selecting a model and estimating its performance using cross-validation

Comparing multiple algorithms

Empirical investigations

What is to be investigated?

- What is the predictive performance of *model* M on new (unseen) data?
- Is there a (significant) difference between *model* M_1 and M_2 (or between M_1, M_2, M_3, $\ldots$)?
- Is there a (significant) difference between *algorithm* A_1 and A_2 (or between A_1, A_2, A_3, $\ldots$)?

Recipe for selecting a model and estimating its performance using a hold-out set

1. Divide the dataset into a training and test set
2. Further divide the training set into a proper training set and validation set[1]
3. Use the various data preparation techniques, learning algorithms and hyper-parameters to generate models from the proper training set
4. Find the best performing configuration w.r.t. the validation set
5. Measure the performance of the model trained using this configuration and the full training set on the test set
6. Generate a new model from the full dataset using the same configuration[2]

[1] possibly repeatedly, e.g., by cross-validation

[2] often results in a stronger model, but also in a biased performance estimate

1. How to divide the dataset?

- We need a sufficiently large training set; otherwise we will not have enough data to train and validate the candidate models
- We need a sufficiently large test set; a too small sample will lead to an uncertain estimate of the performance
- In case both the above cannot be satisfied, cross-validation may be considered
- In case there is class imbalance, stratified sampling may be considered, i.e., keeping the class proportions the same in the training and test sets
- In case there are temporal or other dependencies between the instances in the dataset, then the sampling procedure should reflect this, so that the relation between the test set and the training set is as similar as possible to that between the training set and production data

2. (Why) do we need a validation set?

- Some learning algorithms allow for estimating the performance without setting aside a validation set, e.g., using out-of-bag predictions[3]
- For other algorithms, we could still do without a validation set if we are only interested in finding the best performing model, but not interested in an estimate of its performance; the test set would then serve the purpose of the validation set, i.e., to choose the best model
- Note however that if the best model is chosen based on test set performance, its estimated performance may be highly biased

[3]This however often underestimates the performance of the model

3. Why can't we just do data preparation, parameter tuning, etc. first?

- Any choice that could affect the predictive performance of the resulting model cannot be based on the test instances[4]; otherwise we risk overfitting the test set and the performance estimate will be biased
- Such choices include all the data preparation techniques, e.g., feature selection, imputation, discretization
- These choices also include specification of the models, e.g., hyper-parameters, architectures, etc.
- You can hence not use the full dataset for searching for the best feature set, parameter settings, etc., and then use part of that data for evaluation

[4]at least not their labels; in some (transductive) scenarios, we may have access to the feature vectors of the test instances

4. How do we estimate the predictive performance of a model?

- We need to carefully think about what we would like to optimize, e.g., accuracy or ranking performance (AUC) for classification tasks, and mean-squared error or correlation for regression tasks
- Note that a point estimate does not capture the uncertainty associated with it, due to the sampling error
- How can we tell that an observed improvement compared to some baseline is not just due to chance?

Comparing model performance to a baseline level using a confidence interval

- Given a test set, we may infer a *confidence interval* for the performance, i.e., by following the procedure to generate the intervals, we will with high probability obtain an interval that contains the true mean performance of the model
- Based on the confidence interval, we may perform statistical hypothesis testing, e.g., conclude that the mean model performance is significantly different from (or higher than) some baseline level, i.e., if the baseline falls outside (or above) the confidence interval
- Note however that the choice of method for calculating a confidence interval is not trivial; a proper choice depends on the performance metric, sample size, assumptions of normality, etc., and should be carefully documented, including the made assumptions

Generating confidence intervals for continuous measurements using the t-distribution

- Assume a sample of measurements $V = \{v_1, \ldots, v_n\}$, with a sample mean $m = (\sum_{i=1}^{n} v_i)/n$ and sample standard deviation $s = \sqrt{(\sum_{i=1}^{n}(v_i - m)^2)/(n-1)}$
- The confidence interval is $(m - t_{\alpha/2,n-1}s/\sqrt{n}, m + t_{\alpha/2,n-1}s/\sqrt{n})$ where $t_{\alpha/2,n-1}$ is the value of the t-distribution, given α (1-confidence) and the degree of freedom (df), which in this case is $n-1$.
- For example, if we have $n = 81$ measurements, a sample mean $m = 25$, a sample standard deviation $s = 9$, then a 95% confidence interval, generated using $t_{0.025,80} = 1.99$, is $(23.01, 26.99)$

Generating confidence intervals for continuous measurements using the t-distribution (cont.)

- The proper interpretation of a say, 95%, confidence interval is not that it contains the mean of the population with a probability of 95%, but rather that if we apply the procedure of generating confidence intervals, the interval will contain the mean of the population in 95% of the cases
- Generation of confidence intervals using the t-distribution assumes that the sample mean follows a normal distribution, given by the population mean and variance equal to the sample variance divided by the sample size.
- The procedure *may* still provide valid intervals when the assumption is violated, in particular for large sample sizes, as the sample mean will then tend to being normally distributed[5]

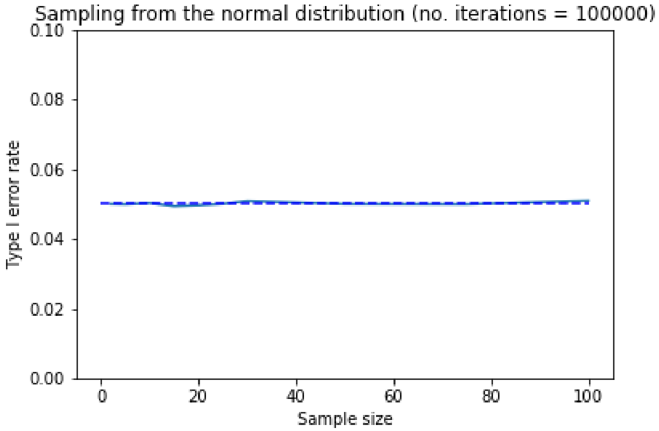
[5] which follows from the central limit theorem

Statistical hypothesis testing

Decision	Truth value for H_0	
	True	False
Reject H_0	Error of type I	Correct
Do not reject H_0	Correct	Error of type II

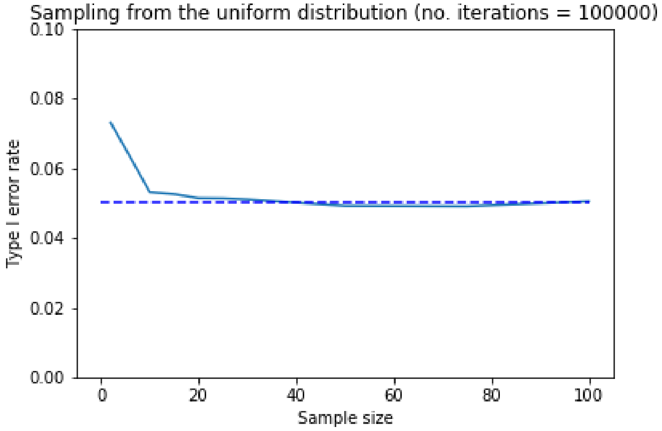
Observed type I errors: sampling from N(0,1)

t-test (`scipy.stats.ttest_1samp`); $H_0 = 0$; $\alpha = 0.05$

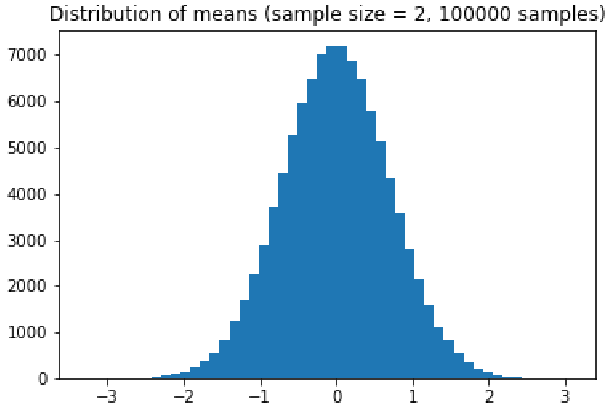


Observed type I errors: sampling from U(0,1)-0.5

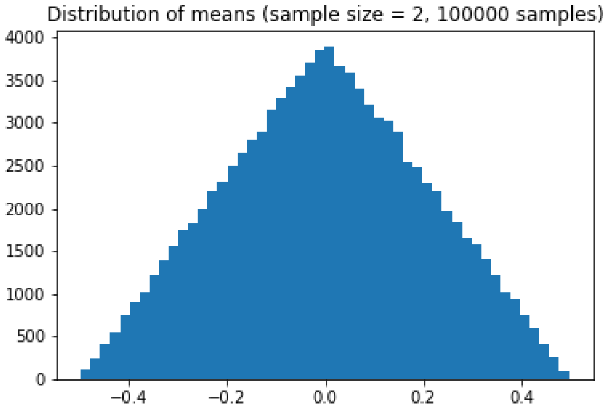
t-test (`scipy.stats.ttest_1samp`); $H_0 = 0$; $\alpha = 0.05$



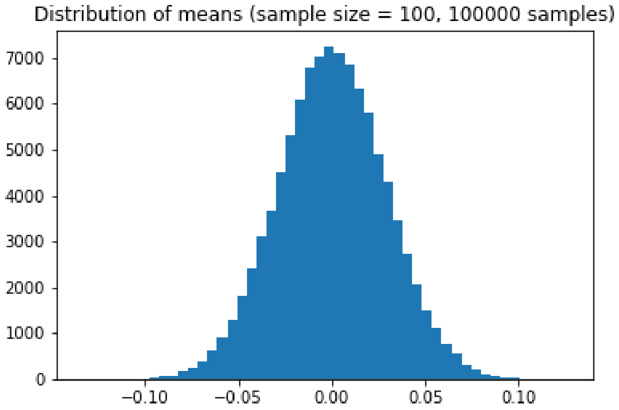
Small sample means from N(0,1)



Small sample means from U(0,1)-0.5

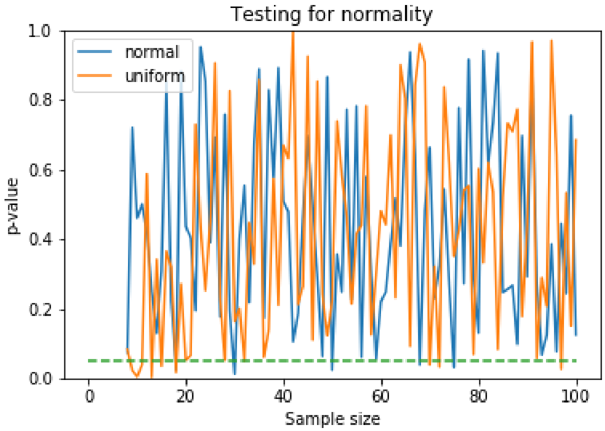


Larger sample means from U(0,1)-0.5



Testing for normality

normality test (`scipy.stats.normaltest`); 10 000 samples

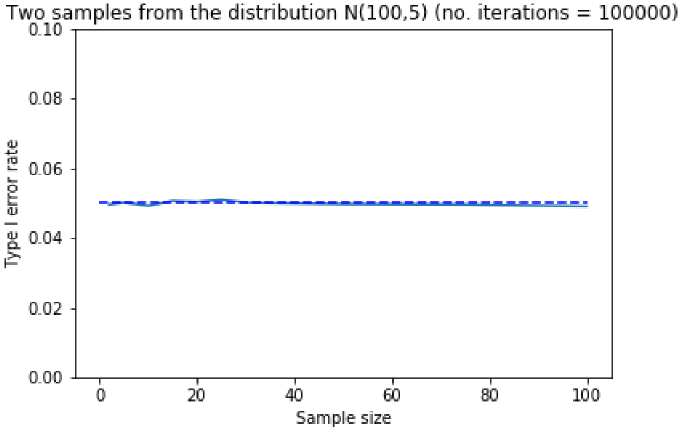


Comparing model performance to a baseline model using a confidence interval

- Given a test set, we observe the predictions for both the model and the baseline model, and then for each test instance calculate the difference between the performance of the model and baseline model; we may then infer a confidence interval for the mean difference
- Based on the confidence interval, we may reject the null hypothesis, i.e., we conclude that the mean model difference is significantly different from 0 (no difference), if 0 falls outside the confidence interval
- Note however that we again have to be careful with the choice of method for calculating the confidence interval

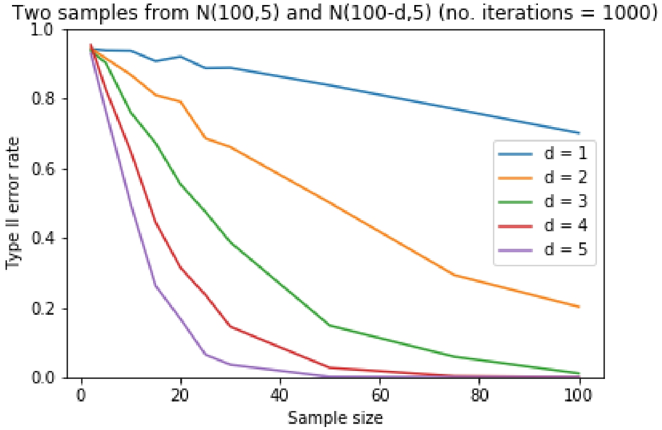
Observed type I errors: two samples from N(100,5)

t-test (`scipy.stats.ttest_rel`); H_0 = difference is 0; $\alpha = 0.05$



Observed type II errors: sampling from different distributions

t-test (`scipy.stats.ttest_rel`); $H_0 =$ no difference; $\alpha = 0.05$



What to do if the distribution of the sample mean is not normal?

- Consider increasing the sample size
- Consider an alternative to the t-test, e.g., Wilcoxon signed rank test (sample size should be at least 20)

The binomial test

The probability of succeeding exactly s times out of n trials, when the probability of success is p, is:

$$P(s, n, p) = \binom{n}{s} p^s (1-p)^{n-s}$$

The probability of succeeding s times or more is:

$$P(X \geq s, n, p) = \sum_{i=s}^{n} P(i, n, p)$$

One-sided test: if $P(X \geq s) < \alpha$, then reject the null hypothesis

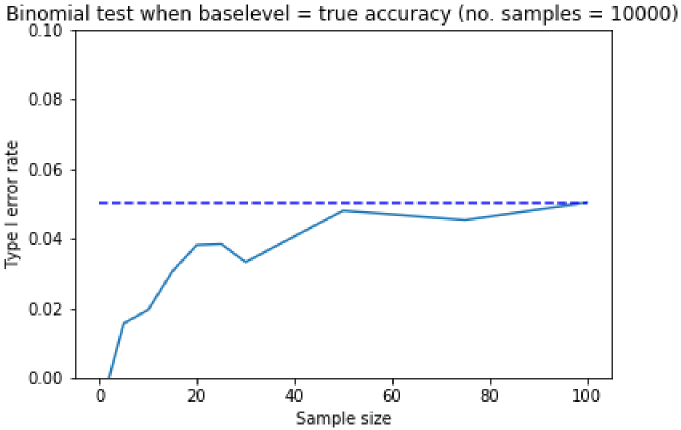
Two-sided test: There are competing approaches for calculating this, including *the Method of small p-values*

Comparing model performance using the binomial test

- Given a baseline level (p), if the probability for observing a certain number of correct (s) and incorrect (f) predictions, or more extreme values, is less than α (1 - a confidence level), the null hypothesis can be rejected
- Given a baseline method, if the probability of observing the number of times the method is correct and the baseline incorrect (s) and the number of times the method is incorrect and the baseline correct (f), or more extreme values, assuming that they are equally accurate ($p = 0.5$), is less than α, then the null hypothesis can be rejected - *this is known as McNemar's test*

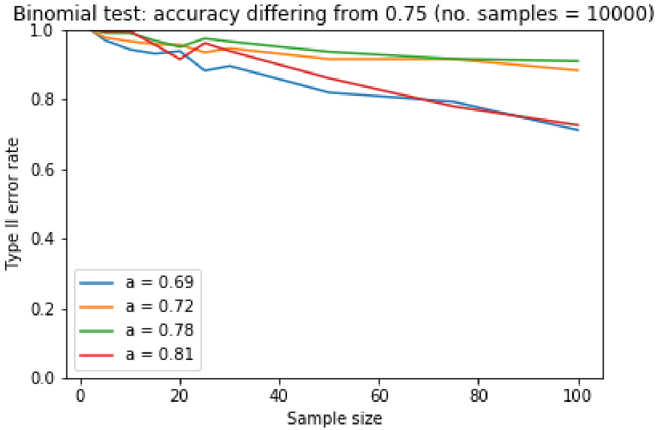
Observed type I errors: comparing to a base level

Binomial test (`scipy.stats.binom_test`);
H_0 = accuracy is 0.75; $\alpha = 0.05$



Observed type II errors: comparing to a base level

Binomial test (`scipy.stats.binom_test`);
H_0 = accuracy is 0.75; $\alpha = 0.05$



Comparing multiple models to a baseline

- The more models we compare to the baseline, i.e., the more statistical tests we make, the higher the risk of making a type I error
- For example, if we make 20 tests with a confidence level of 0.95, we can expect to make one type I error
- In order to avoid increasing the risk of a type I error, we may apply the Bonferroni correction, i.e., instead of using a confidence level of $1 - \alpha$ for n tests, we employ a confidence level of $1 - \alpha/n$

Recipe for selecting a model and estimating its performance using cross-validation

1. Split the dataset into k folds; each fold will serve once as a test set, where the $k-1$ other folds will be the training set
2. Further divide the training set into a proper training set and validation set (possibly repeatedly, e.g., by cross-validation)
3. Use the various data preparation techniques, learning algorithms and hyper-parameters to generate models from the proper training set and find the best performing configuration w.r.t. the validation set
5. Measure the performance of the model trained using this configuration and the full training set on the test set, and average the performances over all folds
6. Find the best configuration by cross-validation, and generate a new model from the full dataset using this configuration

Statistical hypothesis testing for cross-validation

One may think that the estimated performances from the folds is a sample from which a confidence interval may be created using the t-distribution; this is however problematic:

- The variance cannot be properly estimated[6]
- What is the underlying population from which the sample is drawn?

[6] Bengio, Y. and Grandvalet, Y., 2004. No unbiased estimator of the variance of k-fold cross-validation. *Journal of Machine Learning Research*, 5, pp. 1089–1105

Comparing multiple algorithms

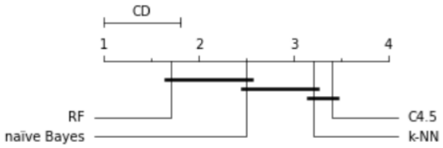
When comparing multiple algorithms over multiple datasets, the standard procedure is to employ a *Friedman test*, followed by some suitable *post hoc* test, e.g.,

- the *Nemenyi* test, if we compare all algorithms to each other
- the *Bonferroni-Dunn* test, if we compare all algorithms against a control algorithm, e.g., a newly proposed one

- Demšar, J., 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7, pp.1–30
- Garcia, S. and Herrera, F., 2008. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9, pp. 2677–2694.

Comparing multiple algorithms over multiple datasets

```
import Orange
import matplotlib.pyplot as plt
algorithms = ["RF","k-NN","naïve Bayes","C4.5"]
no_datasets = 34
average_ranks =  [1.7,3.2,2.5,3.4]
cd = Orange.evaluation.compute_CD(average_ranks,no_datasets)
Orange.evaluation.graph_ranks(average_ranks,algorithms,cd=cd,width=6,textspace=1.5)
plt.show()
```



Concluding remarks

- The choice of performance metric is (at least) as important as the choice of learning algorithm; careful consideration of what needs to be optimized is required
- Common traps should be avoided, e.g., dependencies between training and test data, and over-fitting the test set by repeated experimentation
- Statistical hypothesis testing requires careful formulation of one or more null hypotheses stating what exactly are to be compared, as well as careful application, making sure that samples are properly drawn from the targeted population