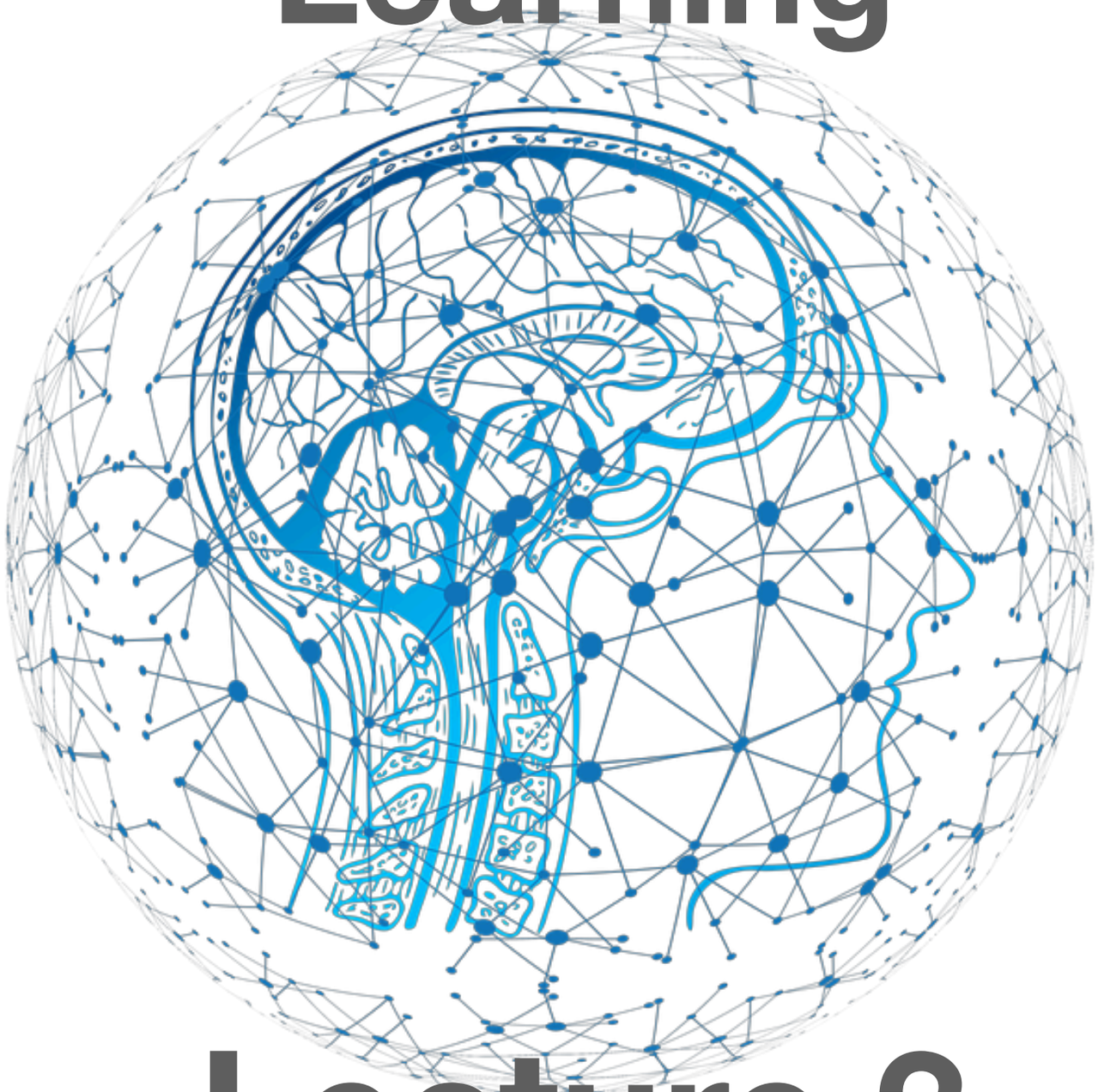
Course 395: Introduction to Machine Learning



Lecture 3

Dr Antoine Cully

Course 395:
Introduction to Machine Learning

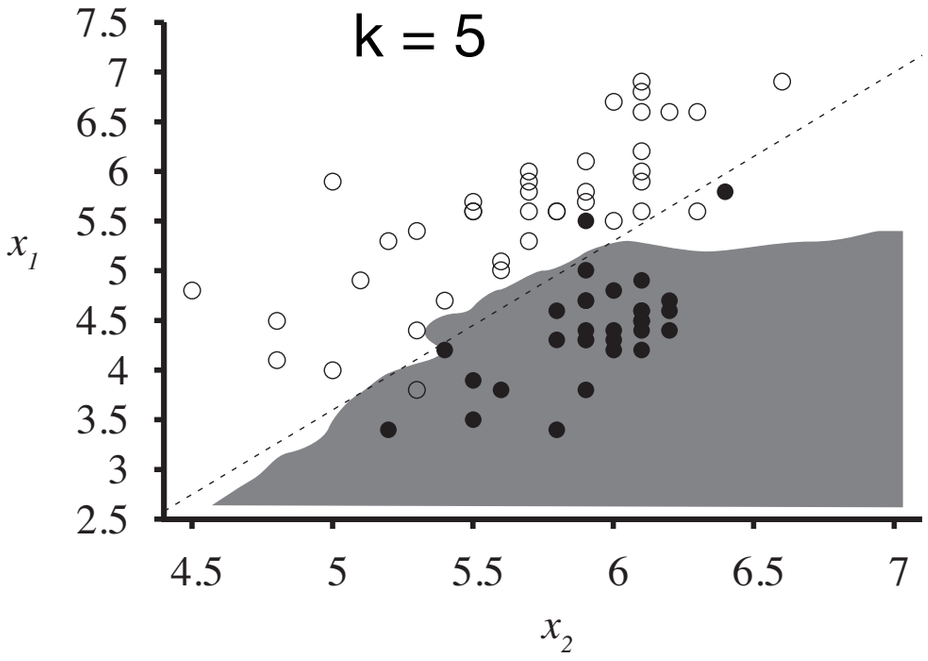
Lab sessions every week, starting next week.

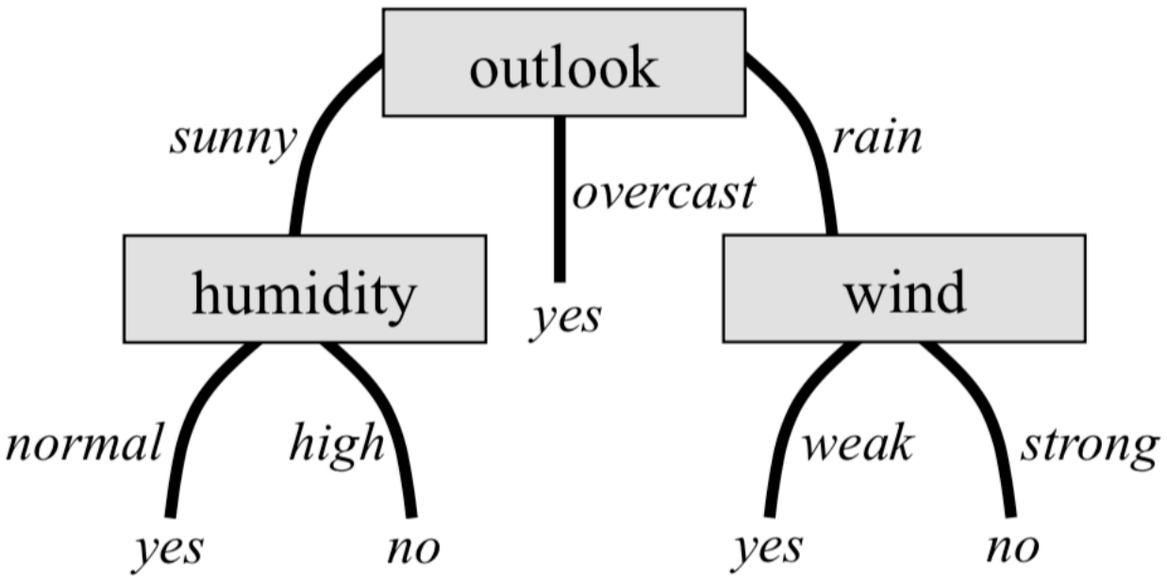
- Week 2: Intro and Instance Based Learning (*A. Cully*)
- Week 3: Decision Trees & CBC Intro (*A. Cully*)
- **Week 4 (today!): Evaluating Hypotheses (*A. Cully*)**
- Week 5: Artificial Neural Networks I (*N. Cingillioglu*)
- Week 6: Artificial Neural Networks II (*N. Cingillioglu*)
- Week 7: Unsupervised Learning and Density Estimation (*A. Cully*)
- Week 8: Genetic Algorithms (*A. Cully*)

Lecture Overview?

- Model evaluation:
  - Hold out
  - Cross validation
- Performance metrics
  - Classification rate
  - recall
  - precision
- Imbalanced datasets
- Overfitting
- Confidence interval
- Comparing algorithms

Evaluating a model/algorithm





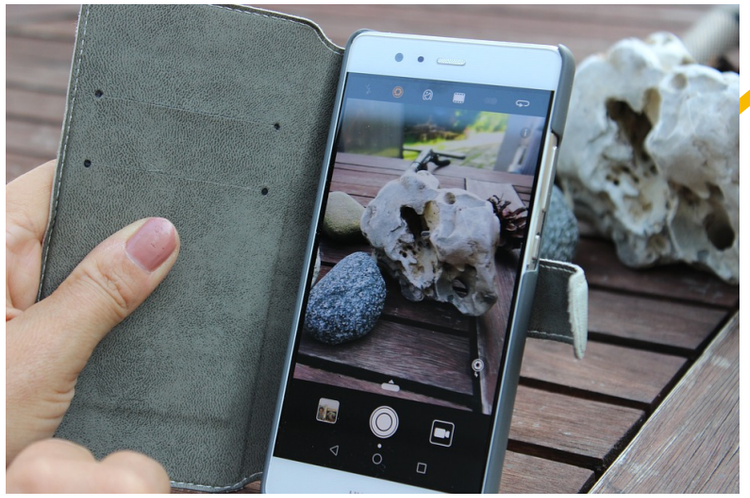
Which one is the best ?

Evaluating a model/algorithm

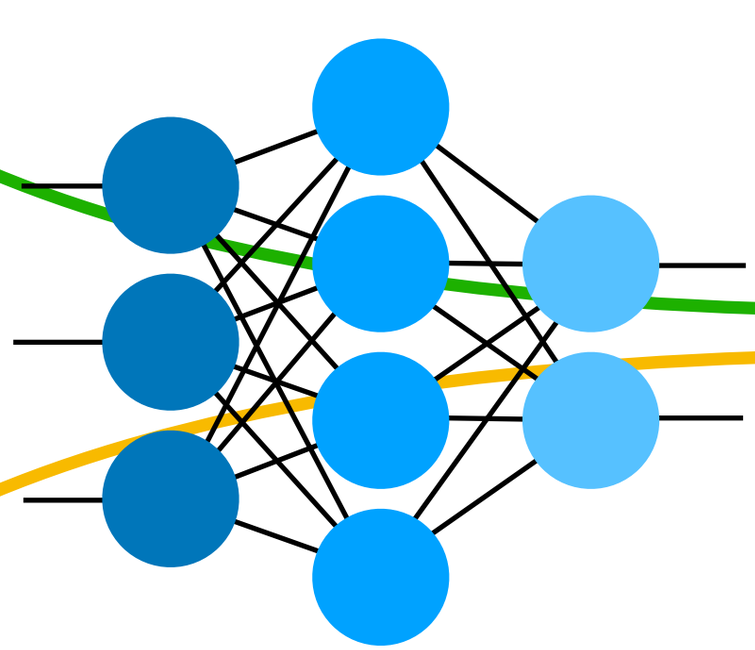
The ultimate goal in machine learning is create models/ algorithms that can generalise to unknown data.

Training dataset

Unknown data (e.g., From the future users)



Trained model

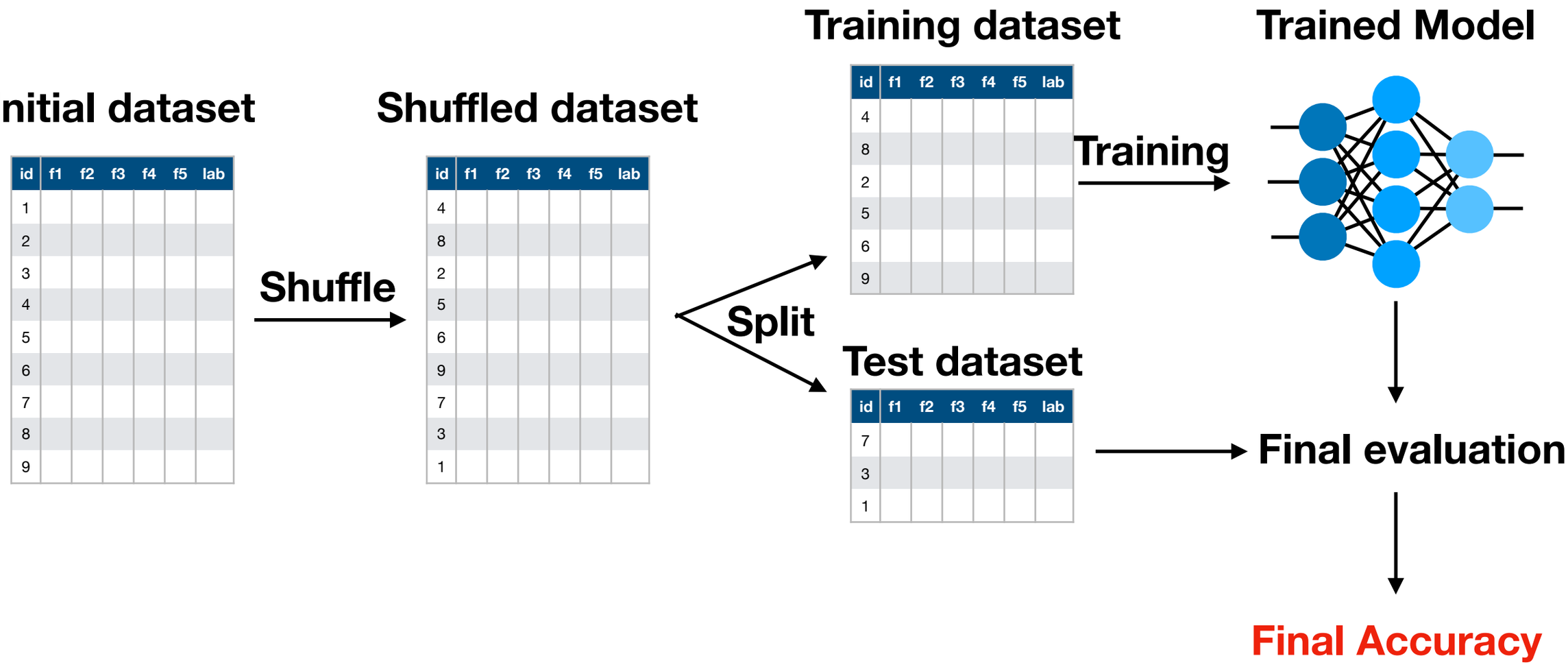


Good accuracy
Because we trained the model for that.

Unknown accuracy
We don't know the data

Evaluating a model/algorithm

To ensure meaningful evaluation, the essential requirement is to use a **test dataset**



The test dataset should **NEVER** be used to train the model. It is meant to simulate unknown data and thus estimate the model accuracy on unknown data.

Parameter tuning

- How to select the values of the algorithm's parameters? (For instance the k of the k-NN algorithm).
- The motivation is the same: Finding good parameter values that work with unknown data (i.e., that generalise well).

Naive approach:

- Try different values on the training dataset, and select the best according to the accuracy on the **training dataset** (and then evaluate on the test dataset).

- Issue: Not sure it will generalise well.

Parameter tuning

Wrong approach:

- Try different values on the training dataset, and select the best according to the accuracy on the **test dataset**.

- Issue: the test dataset is now part of your training process (because you made a decision according to it) therefore we can’t evaluate how your algorithm can generalise to unknown data.

- You should never do that!

Parameter tuning

Right approach:

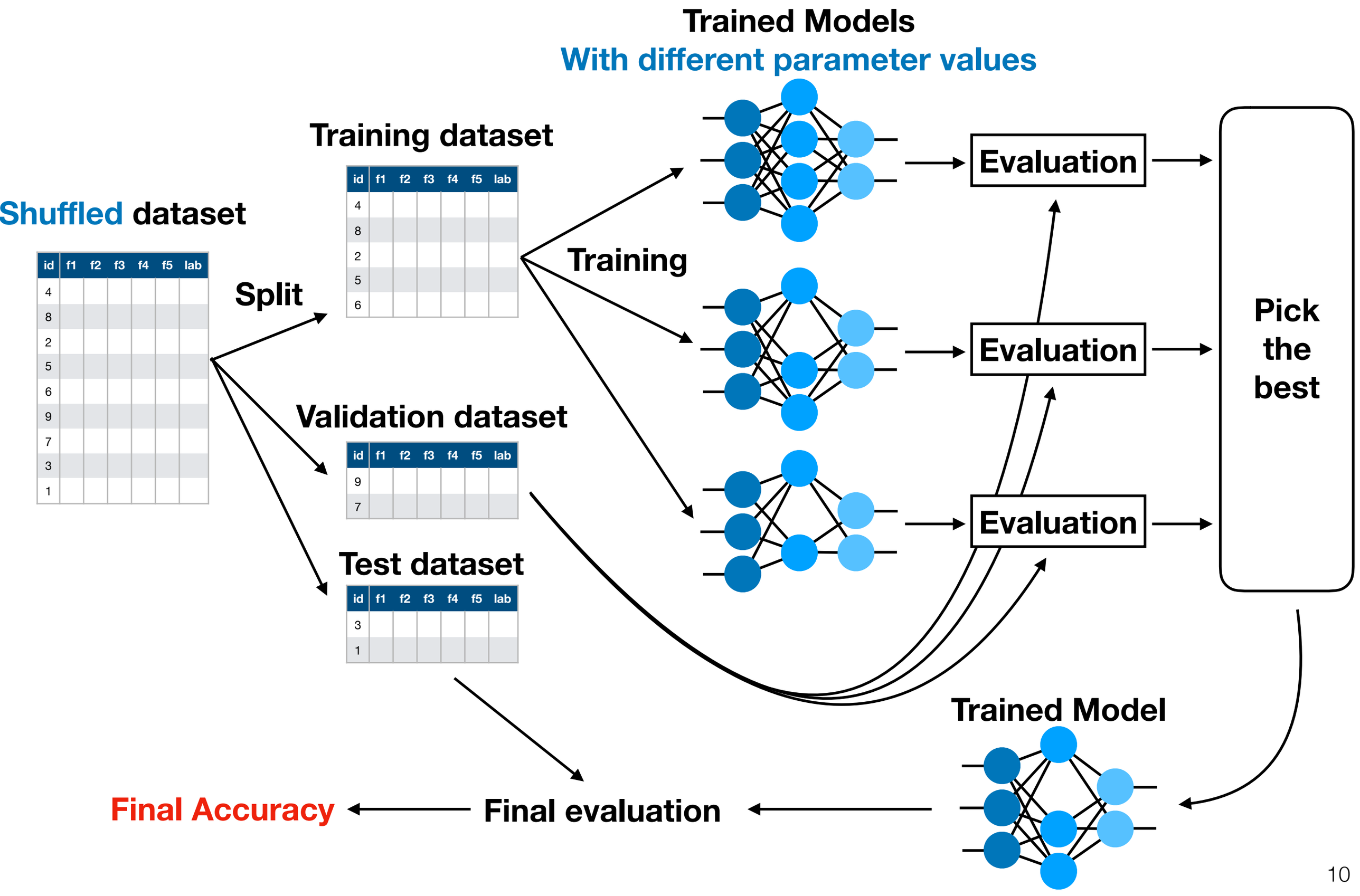
- Split your dataset in 3: training/validation/test

- If a lot of data is available then you can try 50%/25%/25% otherwise 60%/20%/20%.

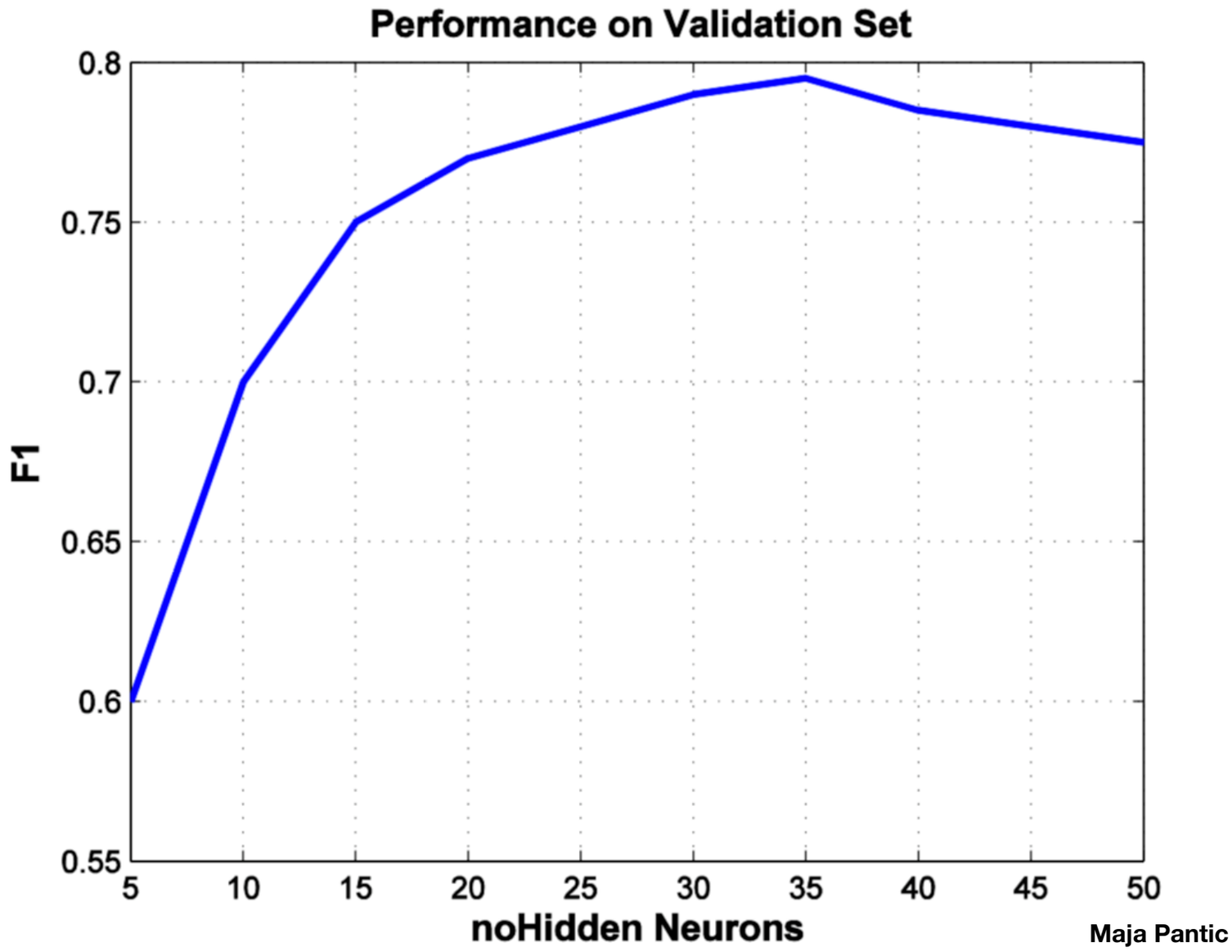
- Try different values on the training dataset, and select the best according to the accuracy on the **validation dataset** and perform the final evaluation on the **test dataset**

- Advantage: your parameter choice takes into account how the model would generalise, and the final evaluation only considers unknown data.

Evaluating a model/algorithm



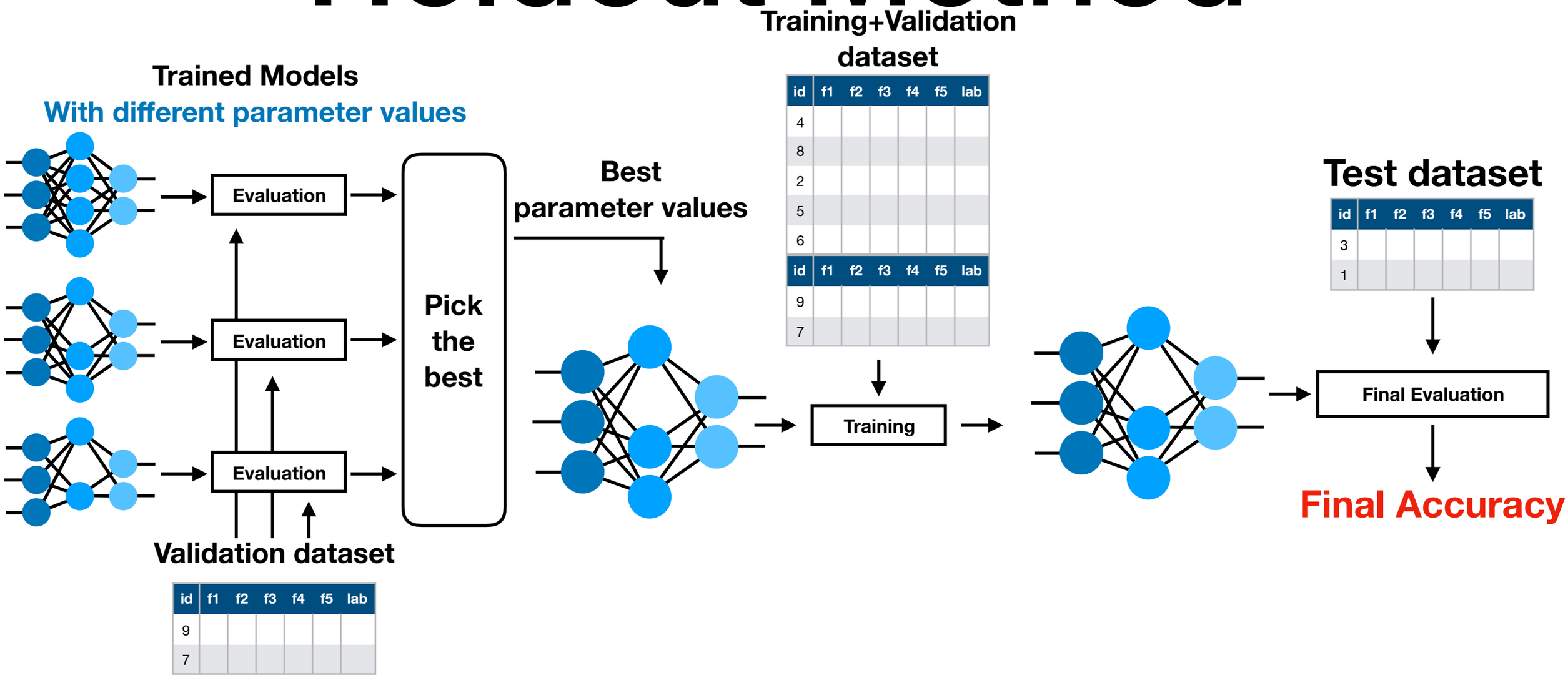
Holdout Method



- Keep the classifier that leads to the maximum performance on the validation set (in this example the one trained with 35 hidden neurons).

- This is called parameter optimization/tuning, since you select the set of parameters that have produced the best classifier.

Holdout Method



For the final evaluation

- Now that the parameters are fixed, you can either use the model trained on the training dataset, or combine the training and validation datasets and train a new model (with the same parameters).

- The final evaluation is always done on the test dataset.

Model evaluation

- The test set should **NOT** be used for training or validation. It is used **ONLY** in the end for estimating the performance on unknown examples, i.e. how well your trained classifier generalises.

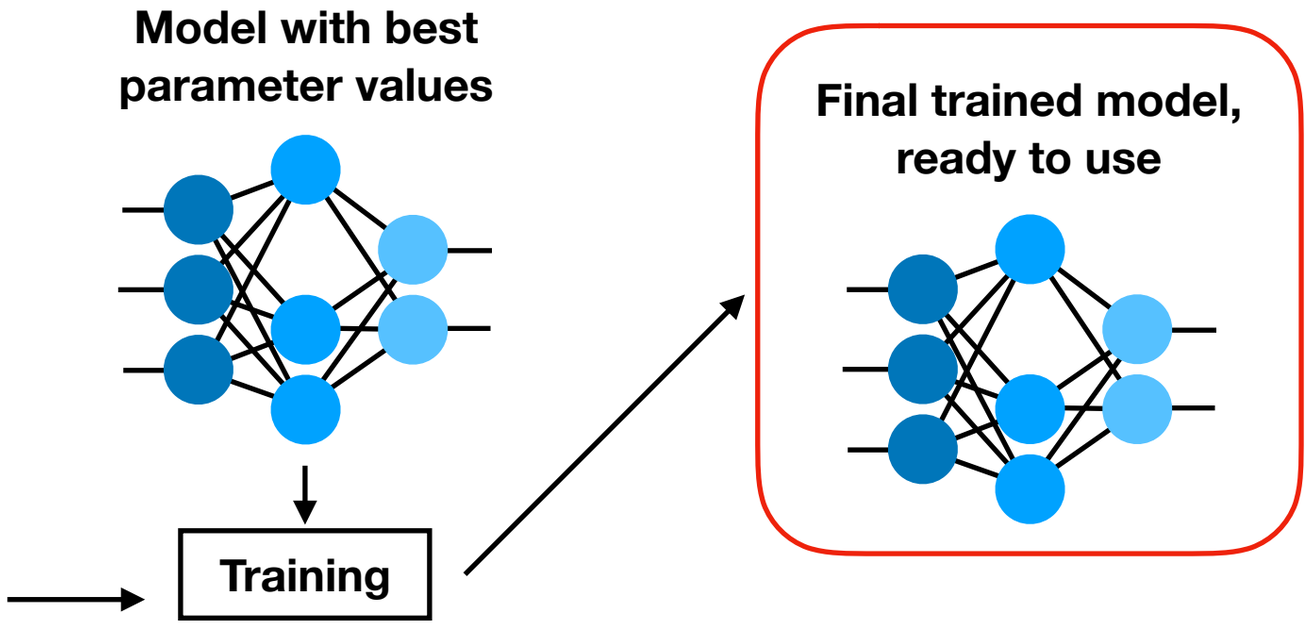
- You should assume that you do not know the labels of the test set and only after you have trained your classifier they are given to you.

- Think about exams for students! The grades would be meaningless if the students already know the exam questions.

What happen when we go in production?

Entire dataset

id	f1	f2	f3	f4	f5	lab
4						
8						
2						
5						
6						
id	**f1**	**f2**	**f3**	**f4**	**f5**	**lab**
9						
7						
id	**f1**	**f2**	**f3**	**f4**	**f5**	**lab**
3						
1						

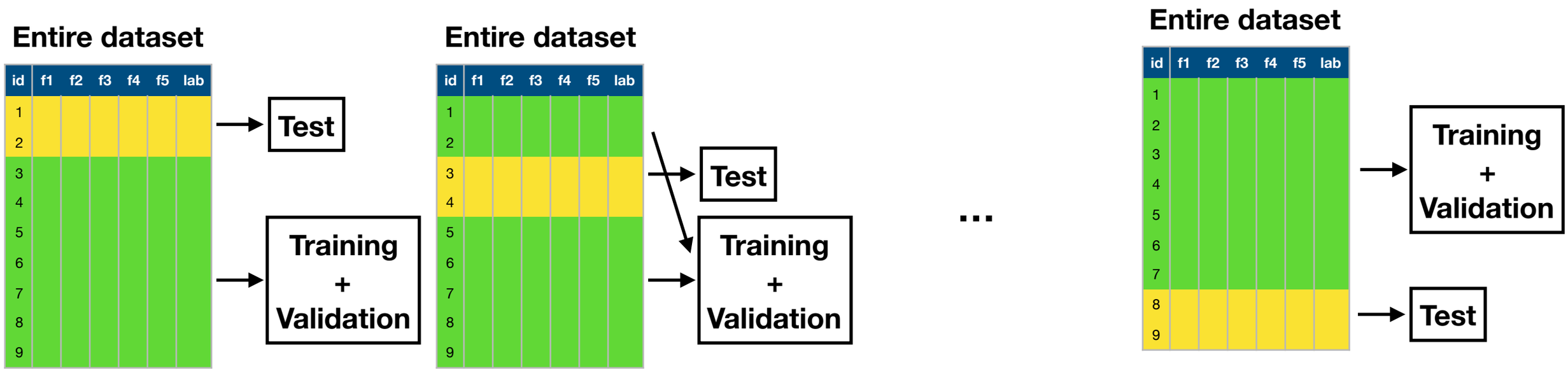


- Now that the parameters are fixed, and that we have an estimation of the performance of the model, we are ready to use it.

- For that, we want the best model, so we can **retrain** it on the entire dataset (training+validation+test) using the optimal set of parameters

Another approach

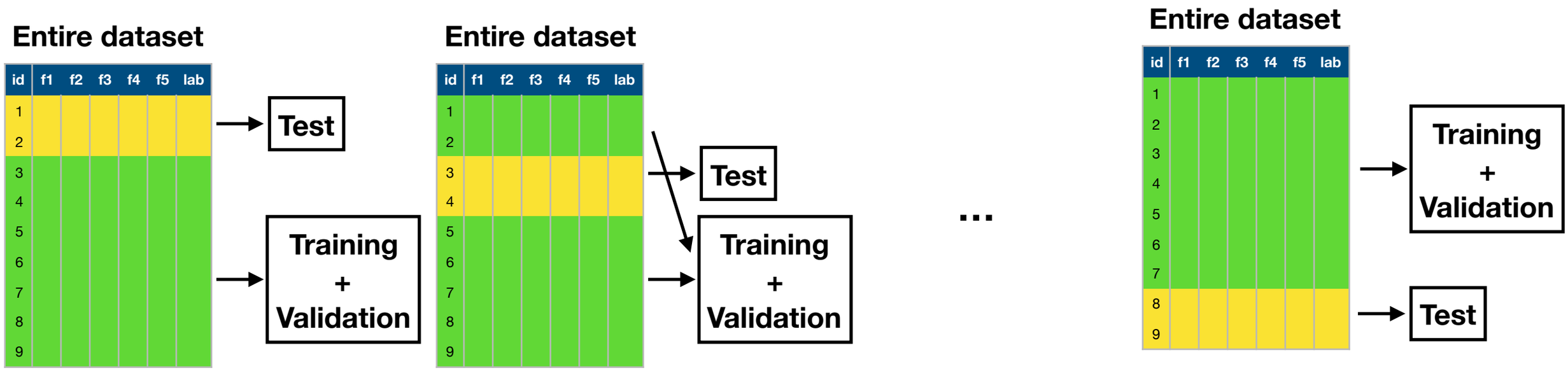
- When we have a lot of examples then the division into training/validation/test datasets is sufficient.
- When we have a small sample size then a good alternative is **cross validation**.



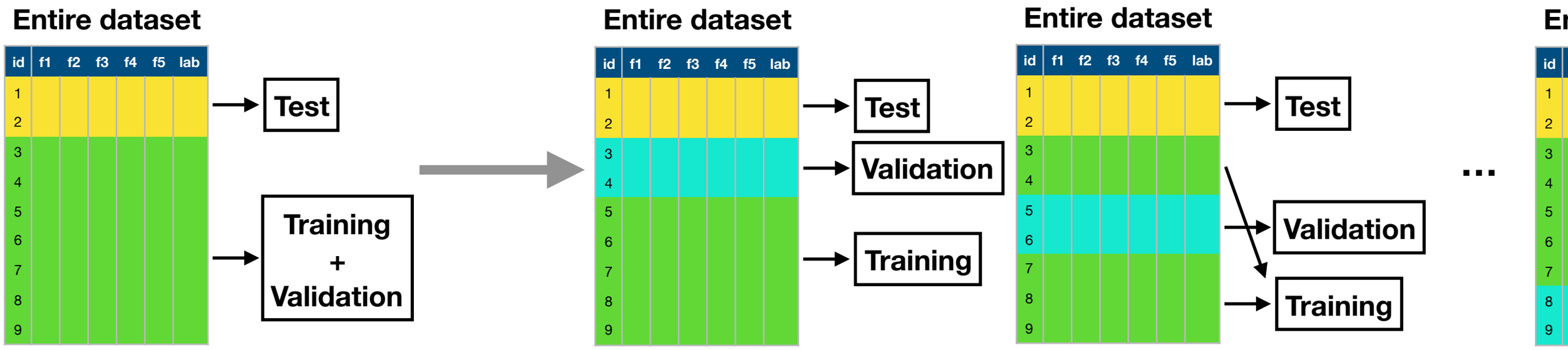
Cross Validation

- Divide dataset into *k* (usually 10) folds using *k*-1 for training+validation and one for testing
- Test data between different folds should **never** overlap!
- Training+Validation and test data in the same iteration should never overlap!
- In each iteration the error on the left-out test set is estimated
- Error estimate: average of the *k* errors

$$\textbf{Global error estimate} = \frac{1}{N}\sum_{i=1}^{N} e_i$$



Cross Validation: Parameter tuning

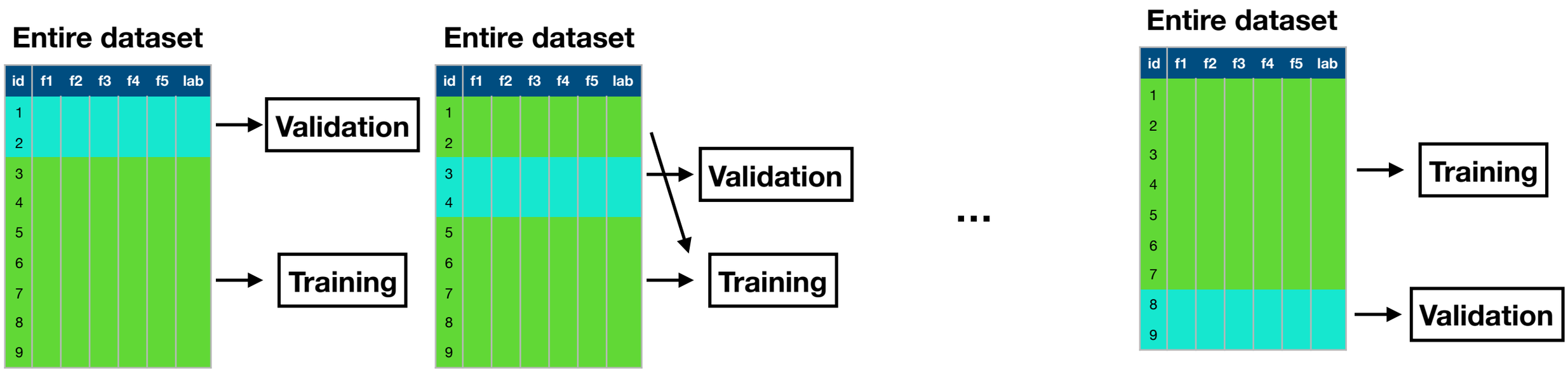


- The *k-1* folds should be divided into training and validation folds, e.g. *k-2* folds for training and 1 for validation.
- Train on the training set, optimise parameters on the validation set and test on the test set.
- We can only estimate the test set performance. In other word we evaluate how our implementation (and the way we optimise the parameters) generalises on unknown test sets.
- We know nothing about the optimal set of parameters.
  We find a different set of optimal parameters in each fold.

Cross Validation: Parameter tuning
(Without performance evaluation)

Before going into production (and now that we know the average accuracy of the model):

- We can use the **entire dataset** and cross validation to estimate the optimal set of parameters.
- *k-1* folds for training, 1 fold left out for validation
- For each parameter set run the *k* fold cross-validation
- Select the parameters that result in the best average performance over all *k* left out folds



Parameter Optimisation– Performance Estimation - Summary

CASE 1: A lot of data are available (Holdout Method)
1) Tune parameters on validation set

2) Estimate generalisation performance using the test set

3) Train on entire dataset using optimal set of parameters

CASE 2: Data are limited (Cross validation)
1) Run cross validation to estimate the test set performance

- Training, validation, test folds

- Optimise parameters in each iteration

2) Run cross validation to estimate optimal parameters

- Training, Validation folds only

3) Train on entire dataset using optimal set of parameters

The metrics

How to quantify the accuracy of the model?

Results from the test dataset

Id	Labels	Predictions
1	+	+
2	+	+
3	+	-
4	+	+
5	-	-
6	-	+
7	-	-
8	-	+

Confusion Matrix

Results from the test dataset

Id	Labels	Predictions
1	+	+
2	+	+
3	+	-
4	+	+
5	-	-
6	-	+
7	-	-
8	-	+

Confusion Matrix

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	**TP**: True Positive	**FN**: False Negative
Class 2 Actual	**FP**: False Positive	**TN**: True Negative

- We have two classes (Positive and Negative examples)
- Visualisation of the performance of an algorithm
- Allows easy identification of confusion between classes
  e.g. one class is commonly mislabelled as the other
- Most performance measures are computed from the confusion matrix

Confusion Matrix

Results from the test dataset

Id	Labels	Predictions
1	+	+
2	+	+
3	+	-
4	+	+
5	-	-
6	-	+
7	-	-
8	-	+

Confusion Matrix

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	TP: **3**	FN: **1**
Class 2 Actual	FP: **2**	TN: **2**

- This table also highlight the risk of each prediction.
- In some cases, it is more important to have low false negative than low false positive (e.g., diagnostic of diseases)

Classification Rate

Results from the test dataset

Id	Labels	Predictions
1	+	+
2	+	+
3	+	-
4	+	+
5	-	-
6	-	+
7	-	-
8	-	+

Confusion Matrix

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	TP: **3**	FN: **1**
Class 2 Actual	FP: **2**	TN: **2**

Classification Rate (or accuracy):

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

- Number of correctly classified examples divided by the total number of examples
- Classification Error = 1 – Classification Rate
- Classification Rate = Pr(correct classification)

Recall

Results from the test dataset

Id	Labels	Predictions
1	+	+
2	+	+
3	+	-
4	+	+
5	-	-
6	-	+
7	-	-
8	-	+

Confusion Matrix

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	TP: **3**	FN: **1**
Class 2 Actual	FP: **2**	TN: **2**

Recall: $\frac{TP}{TP + FN}$

- Number of correctly classified positive examples divided by the total number of positive examples
- High recall: The class is correctly recognised (small number of FN)
- Recall = Pr(correctly classified | positive example)

Precision

Results from the test dataset

Id	Labels	Predictions
1	+	+
2	+	+
3	+	-
4	+	+
5	-	-
6	-	+
7	-	-
8	-	+

Confusion Matrix

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	TP: **3**	FN: **1**
Class 2 Actual	FP: **2**	TN: **2**

Precision: $\frac{TP}{TP + FP}$

- Number of correctly classified positive examples divided by the total number of predicted positive examples
- High precision: An example labeled as positive is indeed positive (small number of FP)
- Precision = Pr(positive example | example is classified as positive)

Precision/Recall

Results from the test dataset

Id	Labels	Predictions
1	+	+
2	+	+
3	+	-
4	+	+
5	-	-
6	-	+
7	-	-
8	-	+

Confusion Matrix

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	TP: **3**	FN: **1**
Class 2 Actual	FP: **2**	TN: **2**

Precision: $\frac{TP}{TP + FP}$

Recall: $\frac{TP}{TP + FN}$

- High recall, low precision:
  Most of the positive examples are correctly recognised (low FN) but there are a lot of false positives.

- Low recall, high precision:
  We miss a lot of positive examples (high FN) but those we predict as positive are indeed positive (low FP).

- Think about what is important for your application!

Unweighted Average Recall

Results from the test dataset

Id	Labels	Predictions
1	+	+
2	+	+
3	+	-
4	+	+
5	-	-
6	-	+
7	-	-
8	-	+

Recall: $\frac{TP}{TP + FN}$

- We compute recall for class1 (R1) and for class2 (R2)
- Unweighted Average Recall (UAR) = mean(R1, R2)

Confusion Matrix

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	TP: **3**	FN: **1**
Class 2 Actual	FP: **2**	TN: **2**

F-measure/score

- It is sometimes useful to have one number to measure the performance of the classifier

$$F_\alpha = (1 + \alpha^2)\frac{Precision * Recall}{\alpha^2 * Precisions + Recall}$$

$$F_1 = 2\frac{Precision * Recall}{Precisions + Recall}$$

Going further

- This formulation is based on the concept of harmonic means (in this case, weighted by alpha).

Confusion Matrix: Multiple classes

Results from the test dataset

Id	Labels	Predictions
1	+	+
2	+	x
3	o	-
4	o	o
5	-	-
6	-	+
7	x	x
8	x	+

Confusion Matrix

	Class 1 Predict	Classe 2 Predicted	Classe 3 Predicted	Classe 4 Predicted
Class 1 actual	**TP**	**FN**	**FN**	**FN**
Class 2 Actual	**FP**	**TN**	?	?
Class 3 actual	**FP**	?	**TN**	?
Class 4 actual	**FP**	?	?	**TN**

- In the multiclass case it is still very useful to compute the confusion matrix.
- We can define one class as positive and the others as negative.
- We can compute the performance measures in exactly the same way.

Confusion Matrix: Multiple classes

Results from the test dataset

Id	Labels	Predictions
1	+	+
2	+	x
3	o	-
4	o	o
5	-	-
6	-	+
7	x	x
8	x	+

Confusion Matrix

	Class 1 Predict	Classe 2 Predicted	Classe 3 Predicted	Classe 4 Predicted
Class 1 actual	**TP**	**FP**	**FP**	**FP**
Class 2 Actual	**FP**	**TN**	?	?
Class 3 actual	**FP**	?	**TN**	?
Class 4 actual	**FP**	?	?	**TN**

- Classification Rate = number of correctly classified examples (trace) divided by the total number of examples.
- Recall and precision and F1 are still computed for each class.
- UAR = mean(R1, R2, R3,..., RN)

Balanced/imbalanced test set

What is the impact of an imbalanced class distribution in the test set?

Balanced test set

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	70	30
Class 2 Actual	10	90

CR: 80%
UAR: 80%

Recall (cl.1): 70%
Precision (cl.1): 87.5%
F1 (cl.1): 77.8%

Recall (cl.2): 90%
Precision (cl.2): 75%
F1 (cl.2): 81.8%

- Balanced Dataset: The number of examples in each class (of the test set) are similar
- All measures result in similar performance

Imbalanced test set

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	70	30
Class 2 Actual	10	90

CR: 80%
UAR: 80%

Recall (cl.1): 70%
Precision (cl.1): 87.5%
F1 (cl.1): 77.8%

Recall (cl.2): 90%
Precision (cl.2): 75%
F1 (cl.2): 81.8%

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	700	300
Class 2 Actual	10	90

CR: 71.8%
UAR: 80%

Recall (cl.1): 70%
Precision (cl.1): 98.6%
F1 (cl.1): 81.9%

Recall (cl.2): 90%
Precision (cl.2): 23.1%
F1 (cl.2): 36.8%

- Imbalanced Dataset: Classes are not equally represented
- CR goes down, is affected a lot by the majority class
- Precision (and F1) for Class 2 are significantly affected :
  30% of class1 examples are misclassified -> leads to a higher number of FN than TN due to imbalance

Imbalance: One class is completely misclassified

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	70	30
Class 2 Actual	10	90

Recall (cl.1): 70%
Precision (cl.1): 87.5%
F1 (cl.1): 77.8%

CR: 80%
UAR: 80%

Recall (cl.2): 90%
Precision (cl.2): 75%
F1 (cl.2): 81.8%

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	700	300
Class 2 Actual	100	0

Recall (cl.1): 70%
Precision (cl.1): 87.5%
F1 (cl.1): 77.8%

CR: 63.6%
UAR: 35%

Recall (cl.2): 0%
Precision (cl.2): 0%
F1 (cl.2): Not defined

- **CR is misleading**, class 2 is completely misclassified.
- F1 for class 2 shows that something is wrong.
- UAR also detects that there is a problem.

Imbalanced test set: Conclusion

- CR can be misleading, simply follows the performance of the majority class

- UAR is useful and can help to detect that one class is completely misclassified but it does not give us any information about FP

- F1 is useful as well but is also affected by the class imbalance problem
  - We are not sure if the low score is due to one class being misclassified or class imbalance

- That’s why we should always have a look at the confusion matrix

Imbalanced test set: Solutions

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	700	300
Class 2 Actual	10	90

CR: 71.8%
UAR: 80%

Recall (cl.1): 70%
Precision (cl.1): 98.6%
F1 (cl.1): 81.9%

Recall (cl.2): 90%
Precision (cl.2): 23.1%
F1 (cl.2): 36.8%

Divide by the total number of examples per class

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	0.7	0.3
Class 2 Actual	0.1	0.9

CR: 80.0%
UAR: 80%

Recall (cl.1): 70%
Precision (cl.1): 87.5%
F1 (cl.1): 77.8%

Recall (cl.2): 90%
Precision (cl.2): 75%
F1 (cl.2): 81.8%

Report performance ALSO on the "normalised matrix"

Imbalanced test set: Solutions

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	700	300
Class 2 Actual	10	90

Divide by the total number of examples per class

	Class 1 Predicted	Classe 2 Predicted
Class 1 Actual	0.7	0.3
Class 2 Actual	0.1	0.9

- These would be the results if we had the same number of examples and the performance of the classifier remained the same

- We don’t have the same number of examples and there is no guarantee that the performance will remain the same (but still it’s one solution to the problem)

Imbalanced test set: Solutions

- Upsample the minority class

- Downsample the majority class
  - e.g. select randomly the same number of examples as the minority class.

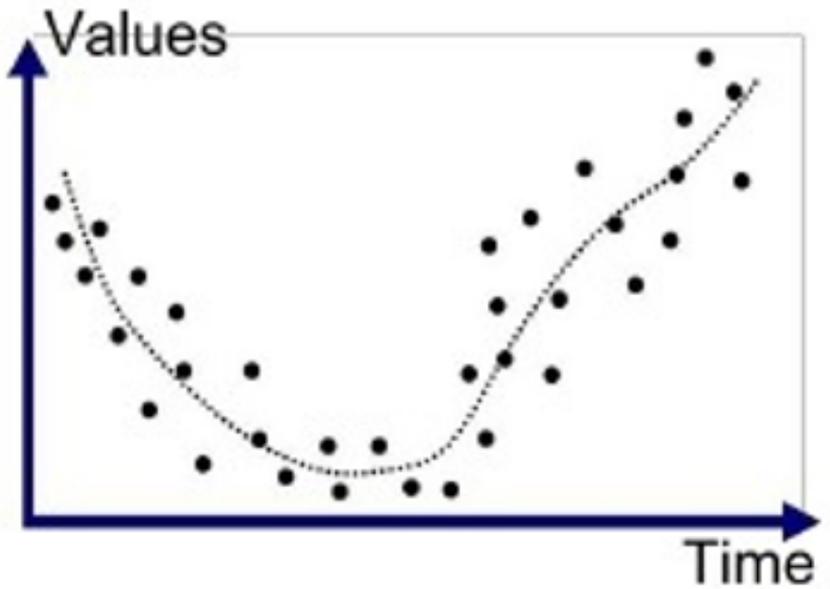
- Repeat this procedure several times and train a classifier each time with a different training set.

- Report the mean and st. dev. of the selected performance measure

- Japkowicz, Nathalie, and Shaju Stephen. "The class imbalance problem: A systematic study." Intelligent data analysis 6.5 (2002): 429-449.

Our experiments allowed us to conclude that the class imbalance problem is a relative problem that depends on: 1) the degree of class imbalance; 2) the complexity of the concept represented by the data; 3) the overall size of the training set; and 4) the classifier involved.

Regression tasks



- Mean squared error (MSE): $\frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2$
- Root mean squared error (RMSE): $\sqrt{(\textbf{MSE})}$

Y_i Sample from the dataset

$\hat{Y}_i$ Prediction from the model

It’s not all about accuracy

Interpretable

Simple

Accurate

Fast
(to train or query)

Scalable

It’s not all about accuracy

Innovation
by Mike Masnick
Fri, Apr 13th 2012
12:07am

Filed Under:
contest, data, recommendation algorithm, streaming
Companies:
netflix

Permalink.
Short link.

Why Netflix Never Implemented The Algorithm That Won The Netflix $1 Million Challenge

from the *times-change* dept

You probably recall all the excitement that went around when a group **finally won** the big Netflix $1 million prize in 2009, improving Netflix's recommendation algorithm by 10%. But what you might *not* know, is that **Netflix never implemented that solution itself**. Netflix recently put up a blog post **discussing some of the details of its recommendation system**, which (as an aside) explains why the winning entry never was used. First, they note that they *did* make use of an earlier bit of code that came out of the contest:

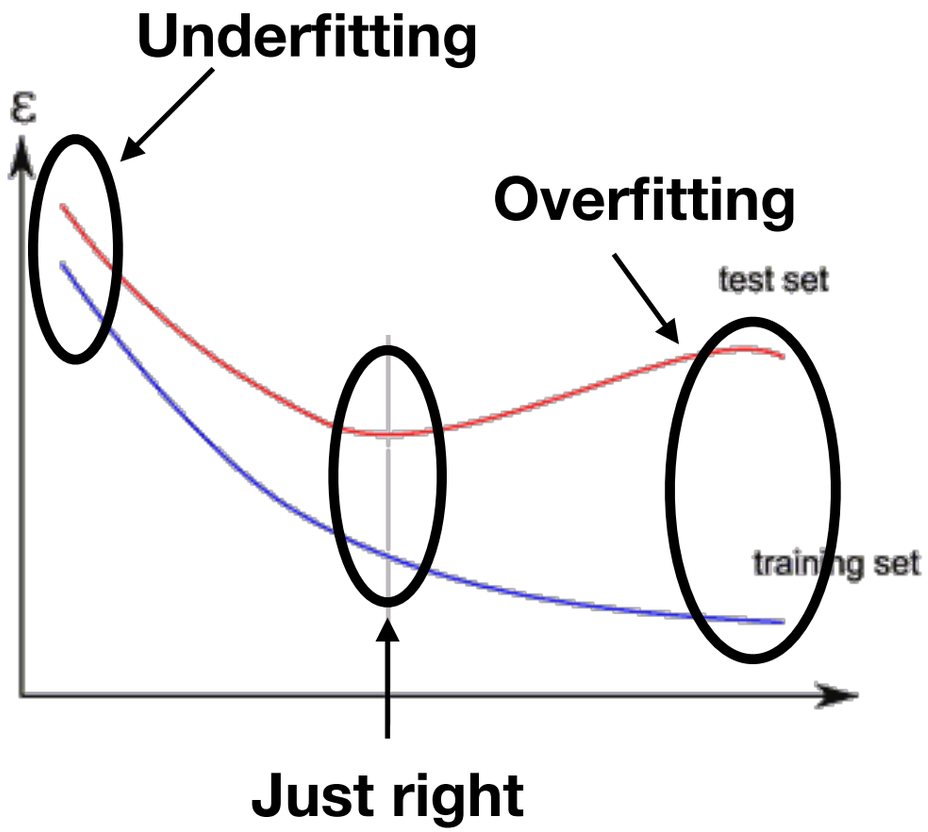
> *A year into the competition, the Korbell team won the first Progress Prize with an 8.43% improvement. They reported more than 2000 hours of work in order to come up with the final combination of 107 algorithms that gave them this prize. And, they gave us the source code. We looked at the two underlying algorithms with the best performance in the ensemble: Matrix Factorization (which the community generally called SVD, Singular Value Decomposition) and Restricted Boltzmann Machines (RBM). SVD by itself provided a 0.8914 RMSE (root mean squared error), while RBM alone provided a competitive but slightly worse 0.8990 RMSE. A linear blend of these two reduced the error to 0.88. To put these algorithms to use, we had to work to overcome some limitations, for instance that they were built to handle 100 million ratings, instead of the more than 5 billion that we have, and that they were not built to adapt as members added more ratings. But once we overcame those challenges, we put the two algorithms into production, where they are still used as part of our recommendation engine.*

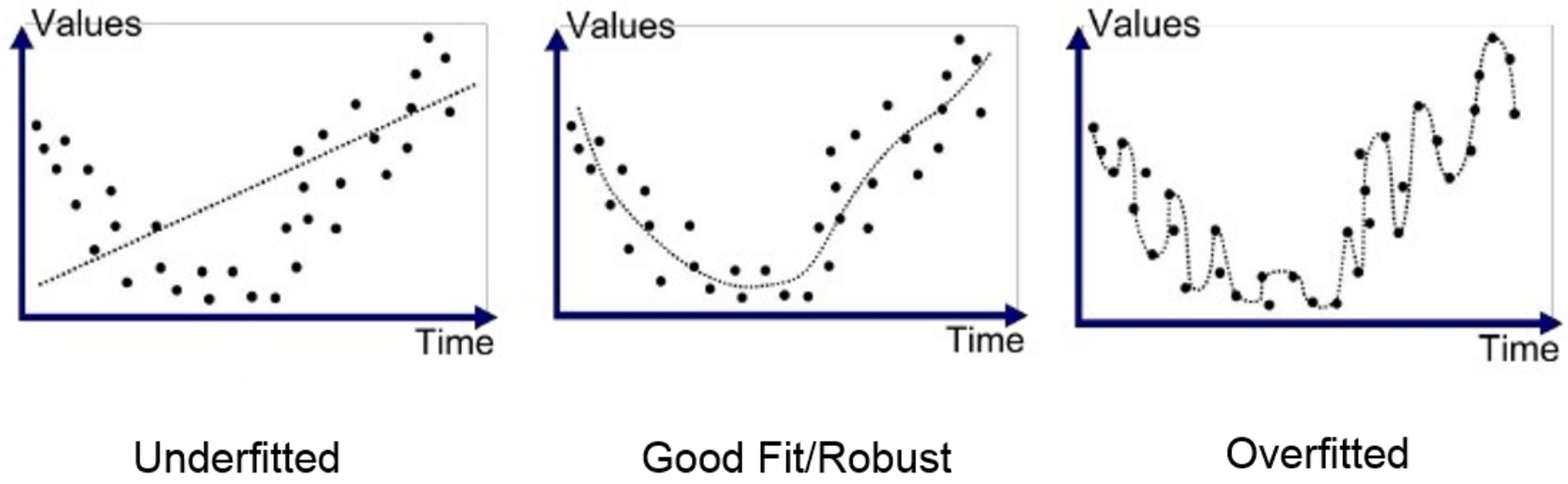
Neat. But the winning prize? Eh... just not worth it:

> *We evaluated some of the new methods offline but the additional accuracy gains that we measured did not seem to justify the engineering effort needed to bring them into a production environment.*

https://www.techdirt.com/articles/20120409/03412518422/

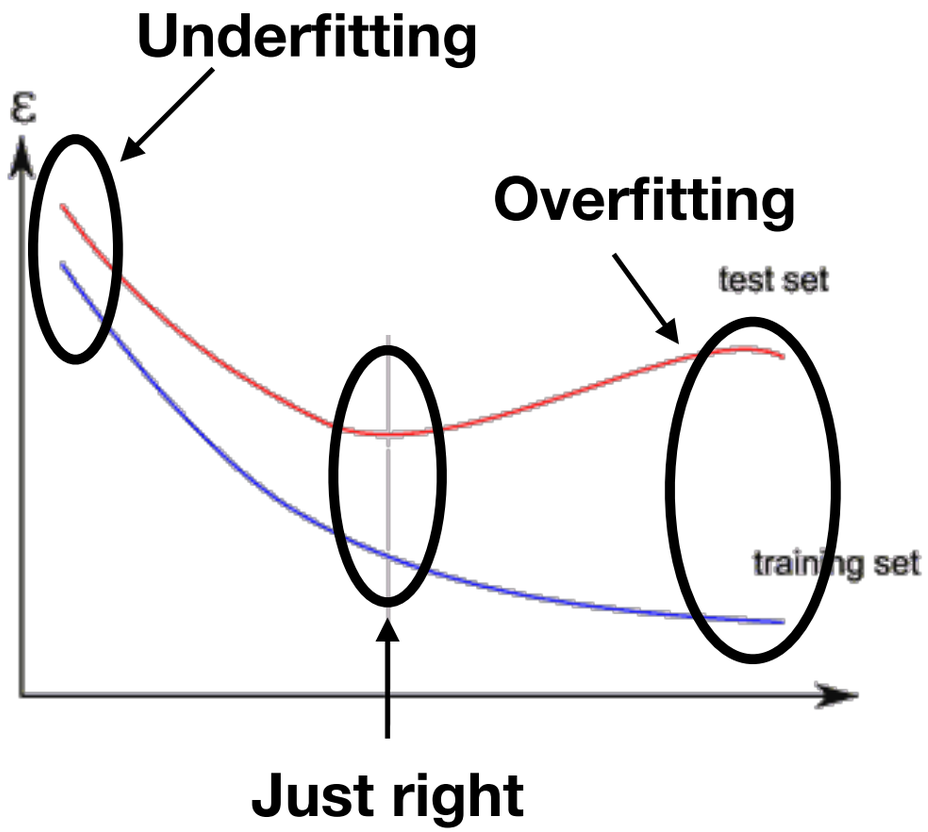
Overfitting

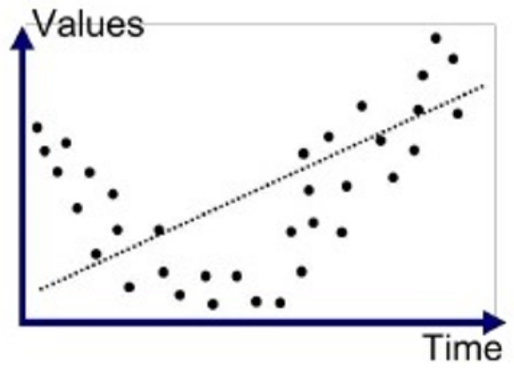


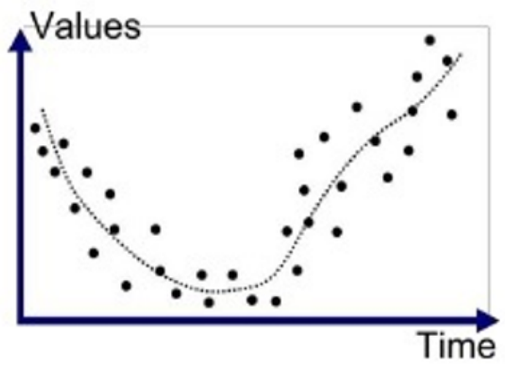


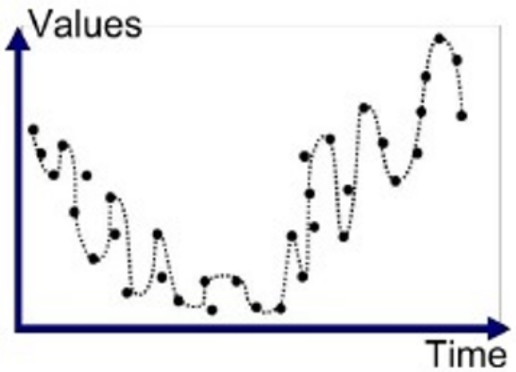
- **Overfitting**: Good performance on the training data, poor generalisation to other data.
- **Underfitting**: Poor performance on the training data and poor generalisation to other data

Overfitting







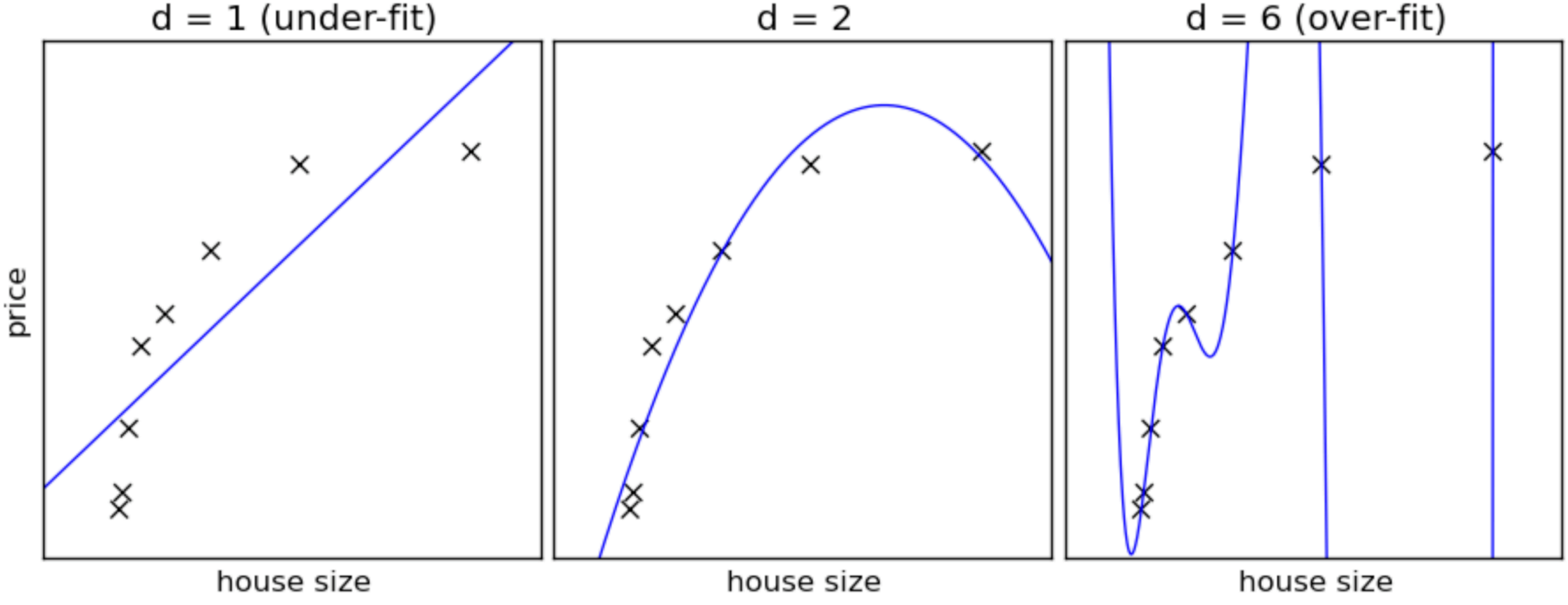


- Overfitting is like a student who learned all the answers of the previous exams, but did not understand the main concepts of the course.
- He/she would be excellent on questions that were already asked in the past (training dataset).
- But he/she would be unable to answer new questions or even variations of old questions (test dataset).

Overfitting

Overfitting can occur when:

- Learning is performed for too long (e.g., in Neural Networks).
- The examples in the training set are not representative of all possible situations.
- The model we use is too complex.

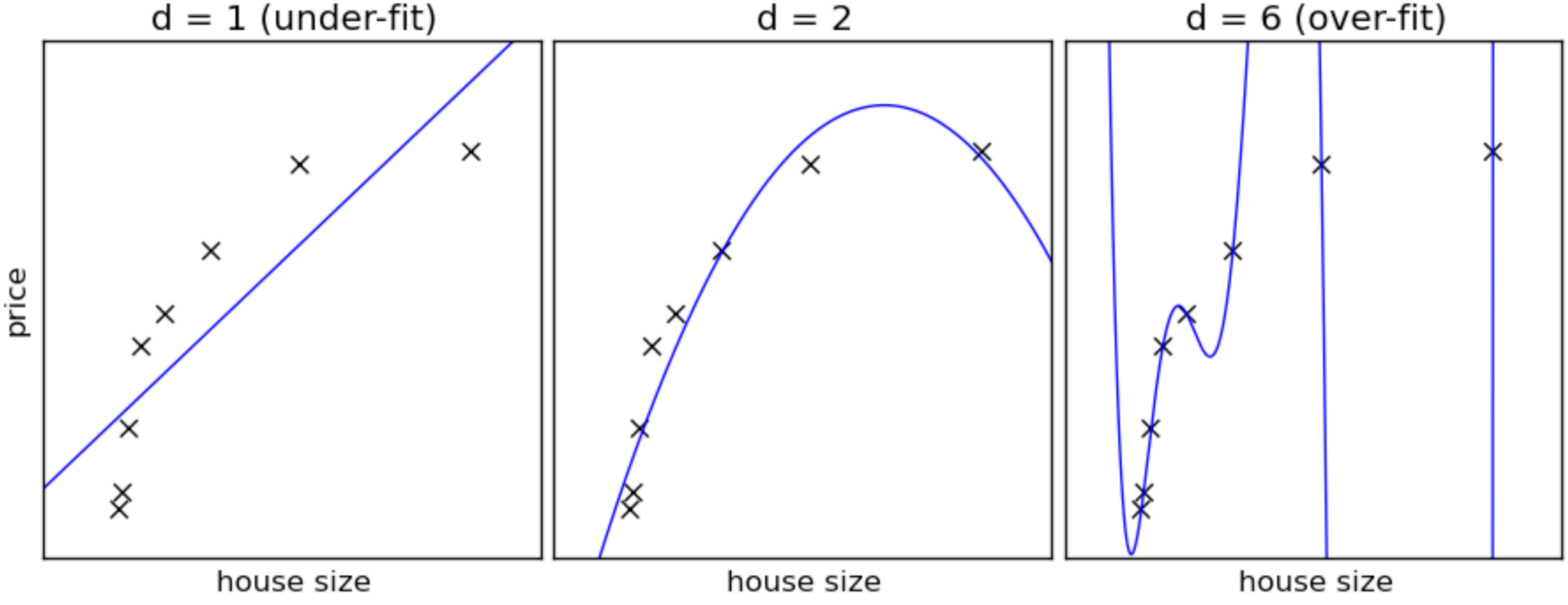


http://www.astroml.org/sklearn_tutorial/practical.html

How to fight overfitting

We can fight overfitting by:

- Stopping the training earlier (use the validation set to know when).
- Getting more data.
- Using the right level of complexity (again use the validation set).



http://www.astroml.org/sklearn_tutorial/practical.html

Summary so far

- We have seen how to evaluate our models: hold out or cross validation with training/validation/test datasets.

- How to measure the performance of the models: Classification rate, Recall, Precision,…

- What is overfitting and how to mitigate it.

Can we trust the performance metrics?

- **This is a critical question.** The decision of some ML algorithms can directly change the life of people.

- All our conclusions are based on the assumption that future samples will be drawn from the same distribution as our training/testing dataset.

Sample error & true error

- The **True error** of the model h is the probability that it will misclassify a randomly drawn example x from distribution D:

$$error_D(h) \equiv Pr[f(x) \neq h(x)]$$

- The **Sample error** of the model h based on a data sample S:

$$error_S(h) \equiv \frac{1}{N} \sum_{x \in S} \delta(f(x), h(x))$$

n=number of samples

$\delta(f(x), h(x)) = 1$ **if** $f(x) \neq h(x)$

$\delta(f(x), h(x)) = 0$ **if** $f(x) = h(x)$

- We want to know the true error but we can only measure the sample error.

Confidence interval

- The amount of data used in the test set also impacts our confidence on the performance evaluation:

- 90% CR on a 10 samples test set is different from a 84% CR on a 10000 sample test set.

- This can be captured by the confidence interval

Confidence interval

- An N% confidence interval for some parameter p
  is an interval that is expected with probability N% to contain p.
  e.g. a 95% confidence interval [0.2,0.4] means that with probability 95% p lies between 0.2 and 0.4.

- Given a sample S with *n* >= 30 on, we can say with *N* % confidence, the true error lies in the interval:

$$error_S(h) \pm Z_N \sqrt{\frac{error_S(h) * (1 - error_S(h))}{n}}$$

$N\%$:	50%	68%	80%	90%	95%	98%	99%
z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

Confidence interval – example

- Given the following extract from a scientific paper on multimodal emotion recognition:

We trained the classifiers with 156 samples and tested with 50 samples from three subjects.

⋮

Table 3. Emotion recognition results for 3 subjects using 156 training and 50 testing samples.

	Attributes	Number of Classes	Classifier	Correctly classified
Face*	67	8	C4.5	78 %
Body*	140	6	BayesNet	90 %

From M. Pantic

- For the Face modality, what is n? What is errors(h)?

- **Exercise**: compute the 95% confidence interval for this error.

$$error_S(h) \pm Z_N \sqrt{\frac{error_S(h) * (1 - error_S(h))}{n}}$$

$N\%$:	50%	68%	80%	90%	95%	98%	99%
z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

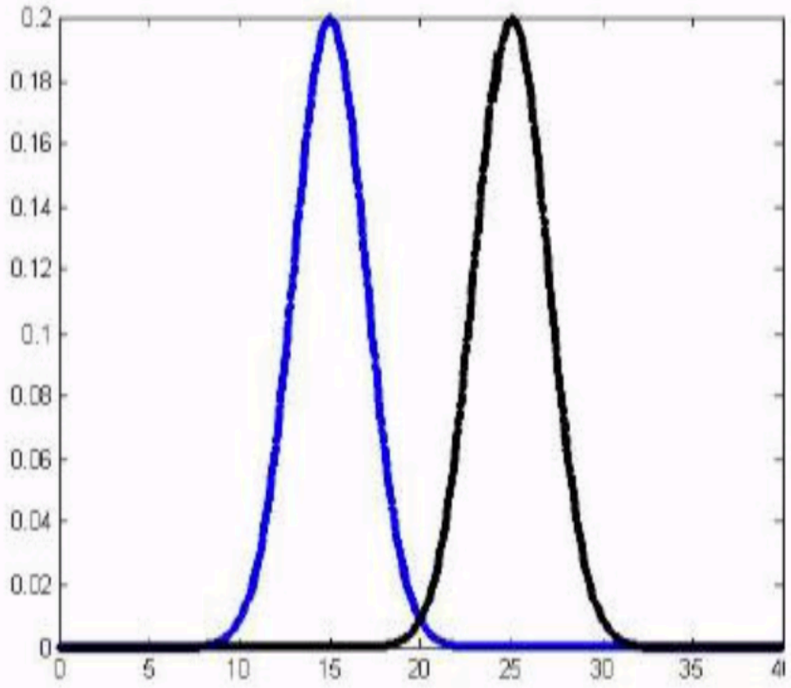
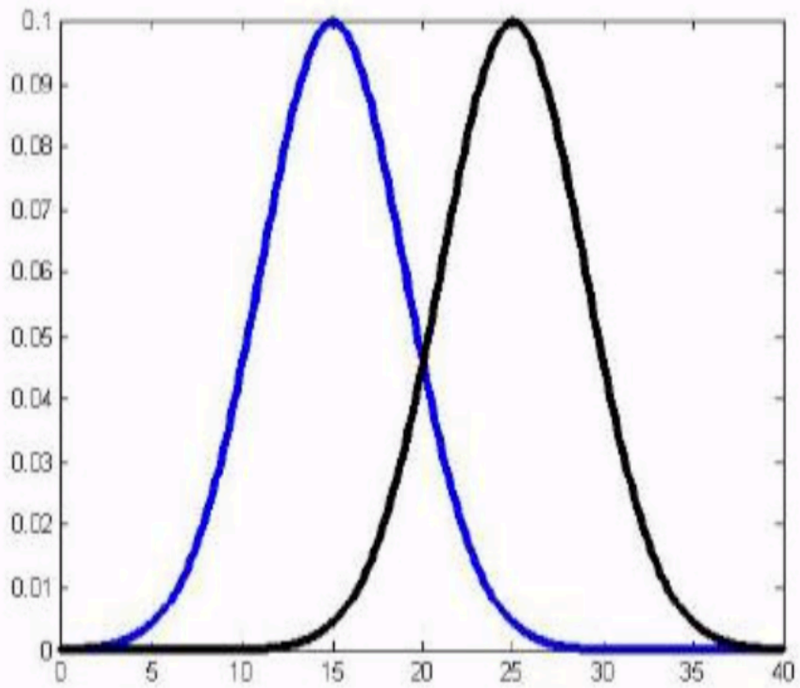
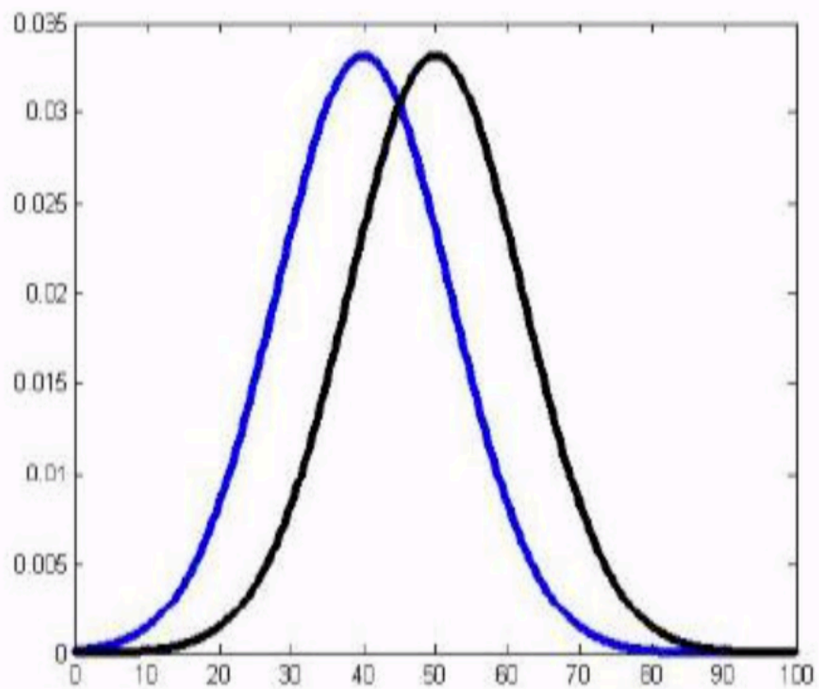
Confidence interval – example

- Given that errors(h)=0.22 and n= 50, and zN=1.96 for N = 95%, we can now say that with 95% confidence errorD(h) will lie in the interval:

$$\left[0.22 - 1.96\sqrt{\frac{0.22 * (1 - 0.22)}{50}}, 0.22 + 1.96\sqrt{\frac{0.22 * (1 - 0.22)}{50}}\right] = [0.11, 0.33]$$

- What will happen when n -> infinity?

Comparing two algorithms

- Consider the distributions of the classification errors of two different classifiers derived by cross-validation.

- The means of the distributions are not enough to say that one of the classifiers is better!! In all cases the mean difference is the same.

- That's why we need to run a statistical test to tell us if there is indeed a difference between the two distributions.

Comparing two algorithms

- There are several statistical tests: T-test, Wilcoxon rank-sum, Randomisation etc.

- A set of observations x and y (e.g. classification error) for each algorithm are needed.

- Two-sample T-test: x, y could be the classification errors on two different datasets

- Paired T-test: x, y could be the classification errors on the same folds of cross-validation from two different algorithms. The test folds are the same, i.e. they are matched.

- The statistical tests tell us if the means of the two sets are significantly different.

Comparing two algorithms

- The statistical tests return a “p-value”, which represents the probability that the “null hypothesis” can be rejected.

- The null hypothesis in our case means that there is no performance difference between the two algorithms tested.

- In short: we usually say that the performance difference between two algorithms is **statistically significant** if $p<0.05$

Comparing two algorithms

- A p-value > 0.05 **does not mean** that the two algorithms are similar. Simply that we cannot observe a statistical difference.

- For instance, collecting more data can change the p-value in one direction or the other.

- To evaluate if two algorithms are similar (in term of performance), a power analysis can be used.

Comparing two algorithms

- Keep in mind that the evaluation of a model is meant to represent what performance you can expect when applying this model (on similar data distribution).

- If the training of the model is stochastic (neural networks, genetic algorithms, …) then, you should **replicate the training** several times (e.g., 20) to know the distribution of the performance.

- Then, you can use statistical tests to compare the distributions of different algorithms.

- **Seed hacking**: finding the seed of the random generator that maximises the performance of your algorithm. **This is cheating**!