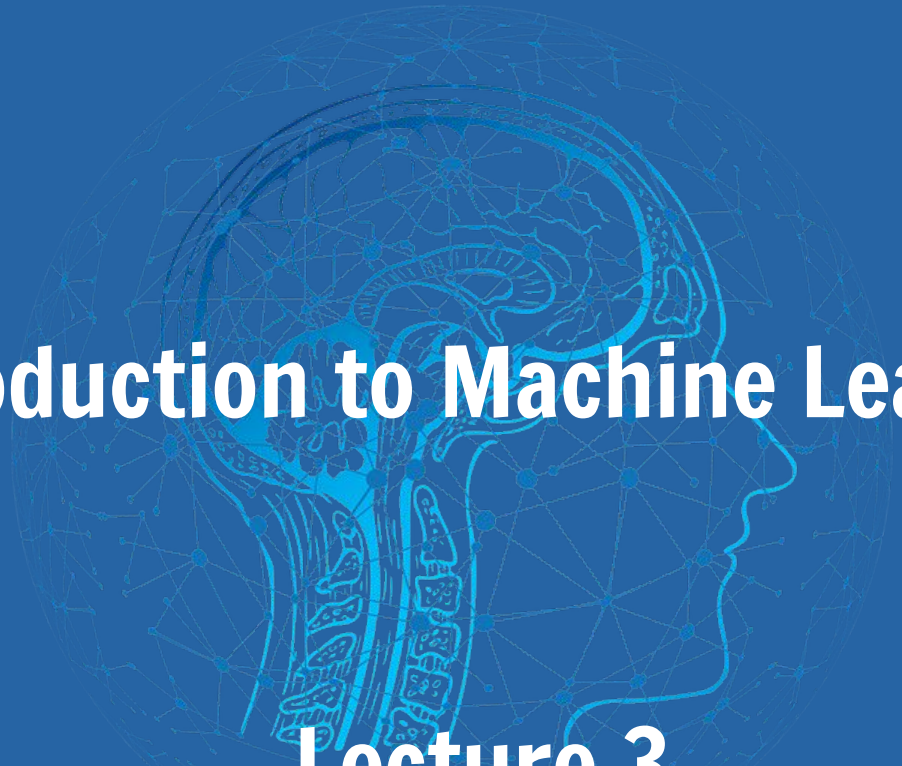
Imperial College London



Introduction to Machine Learning

Lecture 3

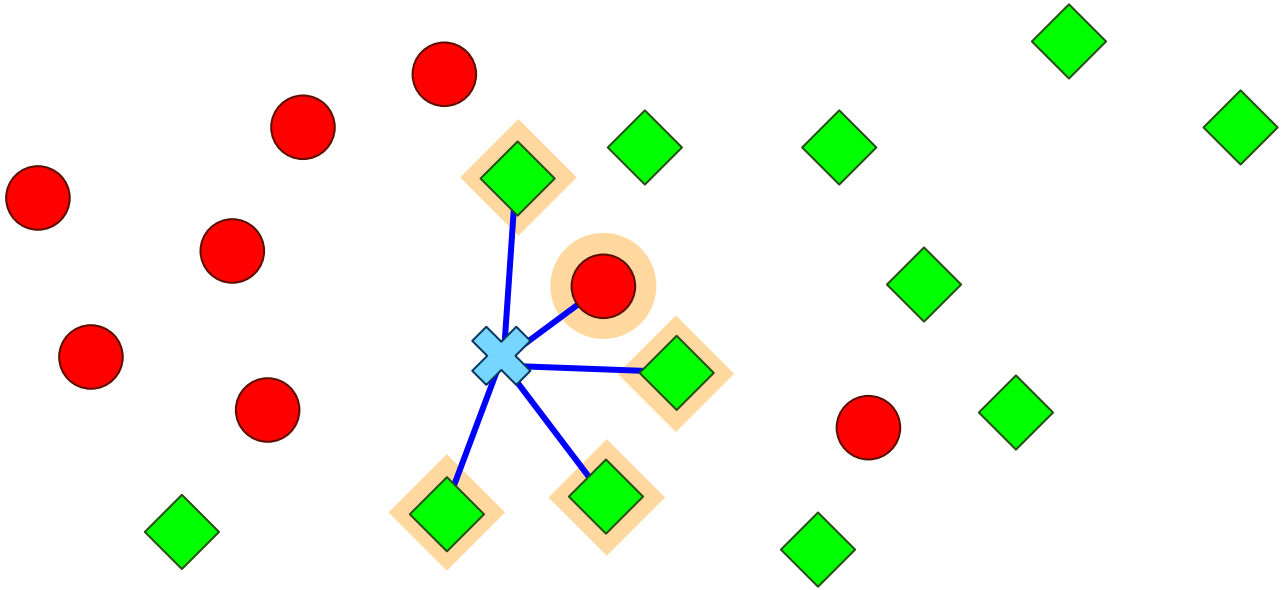
Antoine Cully & Marek Rei & Josiah Wang

In the previous lecture...

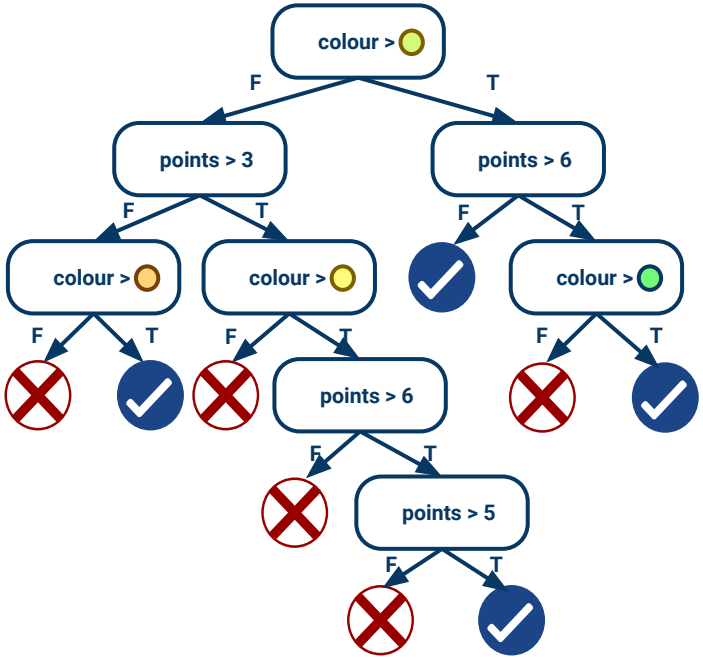
Recap: k-NN

k-Nearest Neighbours (k-NN) classifier:

Classify a test instance based on the class labels of the nearest (most similar) training instances.

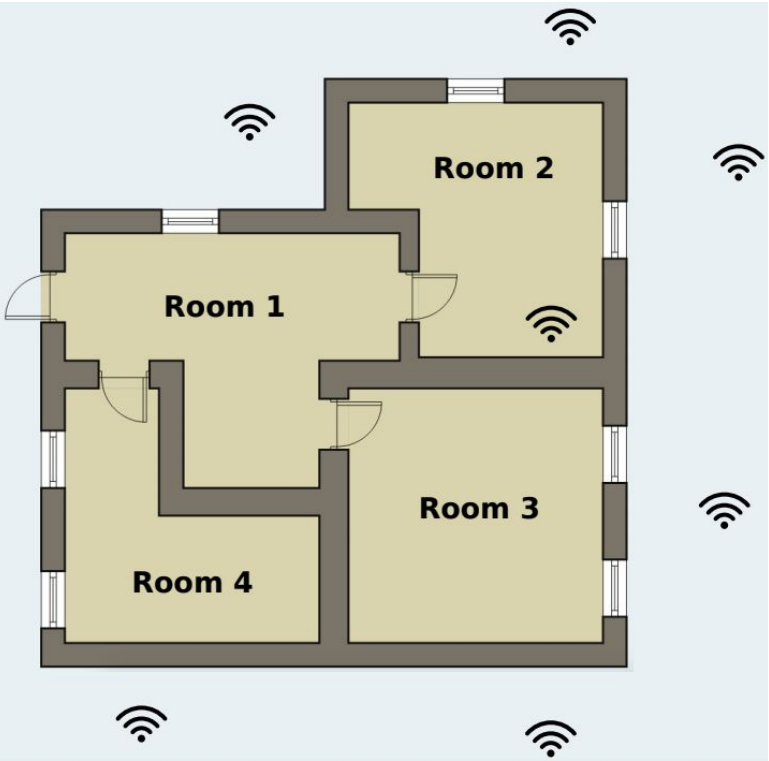


Recap: Decision trees



1. Search for an ‘optimal’ splitting rule on training data
2. Split your dataset according to your chosen splitting rule (e.g., information gain)
3. Repeat 1. and 2. on each new splitted subset

Recap: Coursework 1

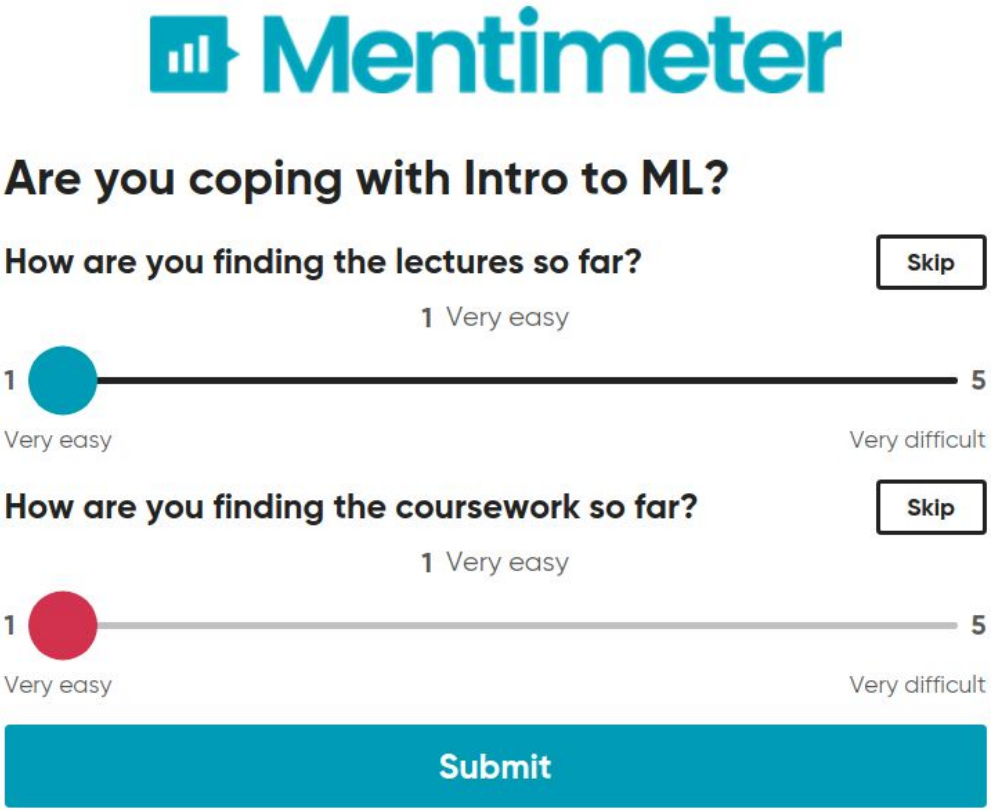


Implement a decision tree algorithm that is able to learn how to determine your position, based on the strength of wifi signals.

Deadline: 6th Nov 2020 (Friday) 7pm

Today...

Send your feedback



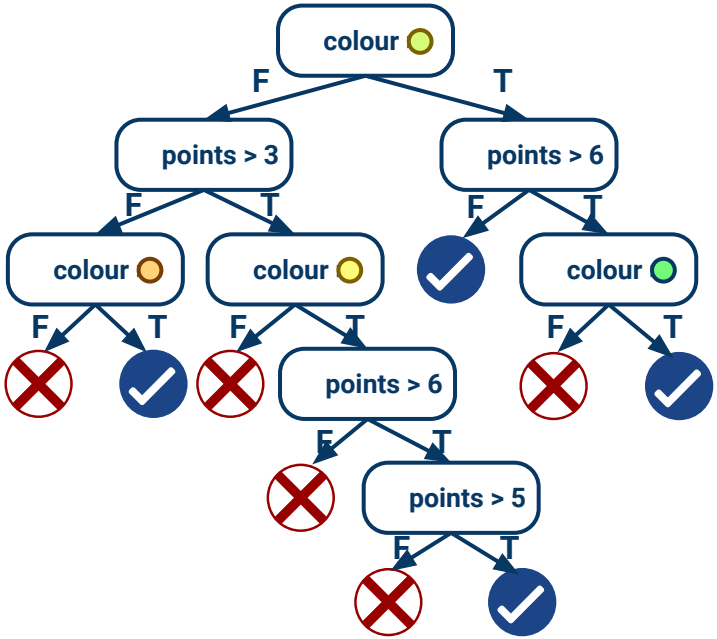
menti.com/skabhxxpg2

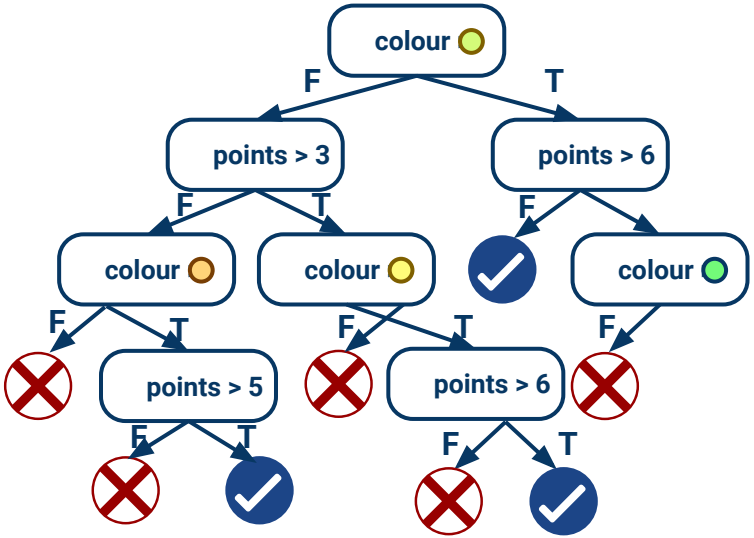
Course plan

	Lecture	Lecturer
Week 2	Introduction to ML + Classification	*Josiah*
Week 3	Instance-based Learning + Decision Trees	*Antoine*
Week 4	Machine Learning Evaluation	*Marek*
Week 5	Artificial Neural Networks I	*Marek*
Week 6	Artificial Neural Networks II	*Marek*
Week 7	Unsupervised Learning	*Antoine*
Week 8	Genetic Algorithms	*Antoine*

Today's lecture

- Evaluation set-up
  - Held-out dataset
  - Cross validation
- Performance metrics
  - Accuracy
  - Precision & Recall
  - F1 measure
- Imbalanced datasets
- Overfitting
- Confidence intervals
- Significance testing



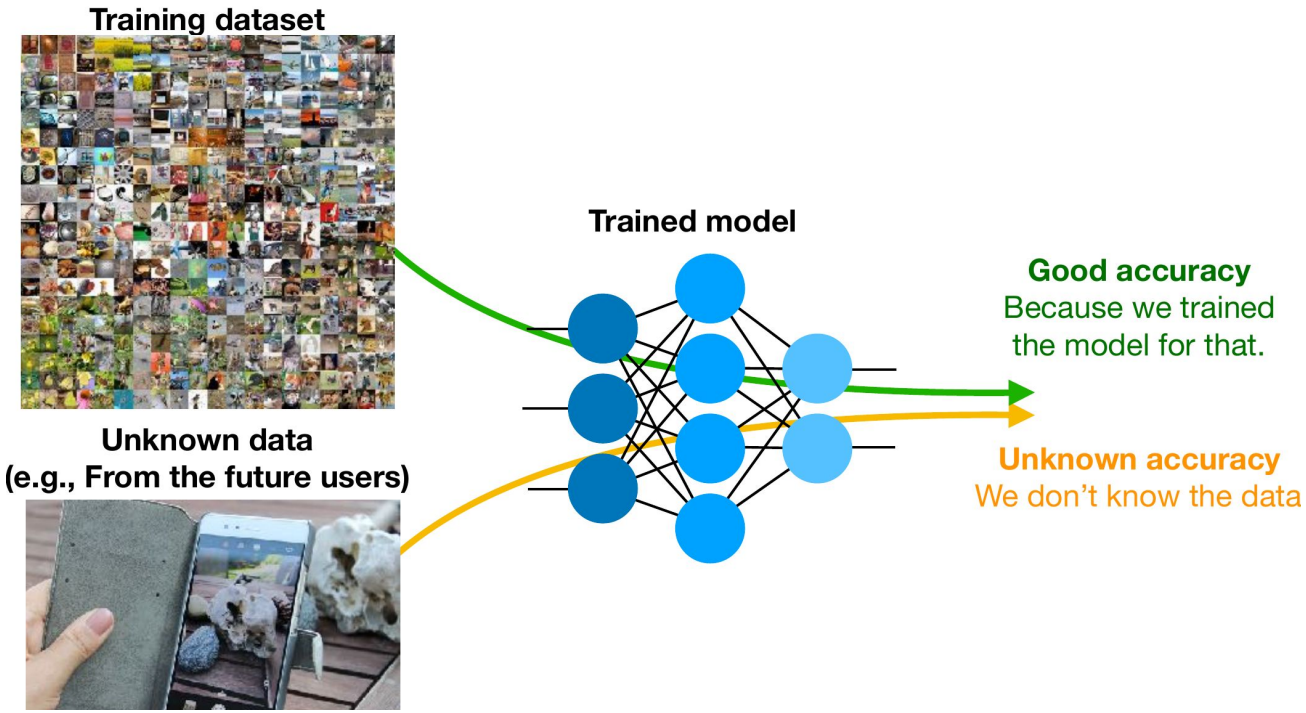


How good is a model?
Which model is the best?

Evaluation set-up

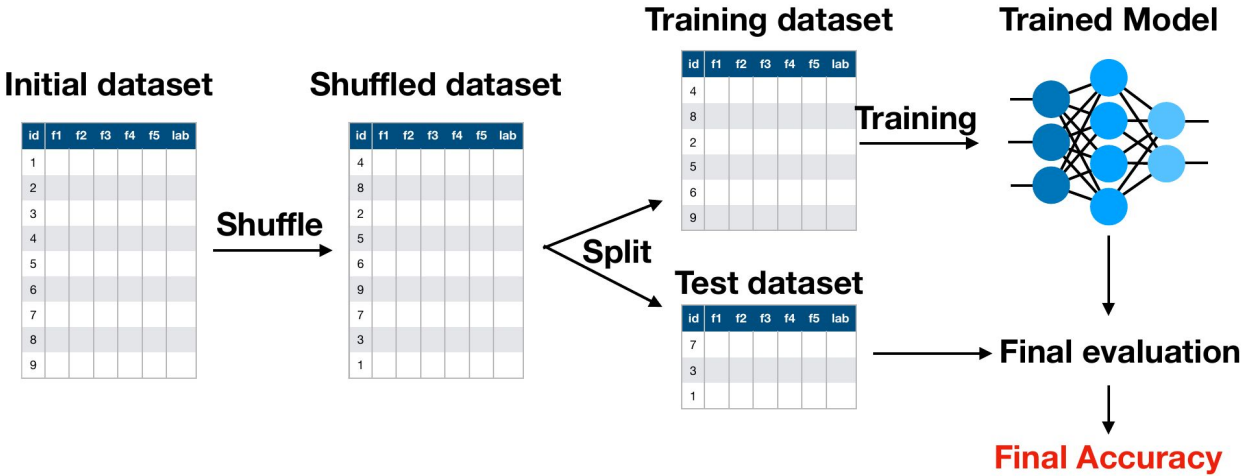
Evaluating a model / algorithm

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



Evaluating a model / algorithm

To ensure a meaningful evaluation, the essential requirement is to use a held-out **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.

Hyperparameter tuning

- Hyperparameters - model parameters that are chosen before the training.

  For example: the k of the k-NN algorithm.

- **Overall objective:** Finding the hyperparameter values that lead to the best performance

- The motivation is the same:
  Finding good hyperparameter values that work with unknown data (i.e., that generalise well).

Hyperparameter tuning

Naive approach:

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

- Issue: Usually doesn’t generalise well

Hyperparameter tuning

Wrong approach:

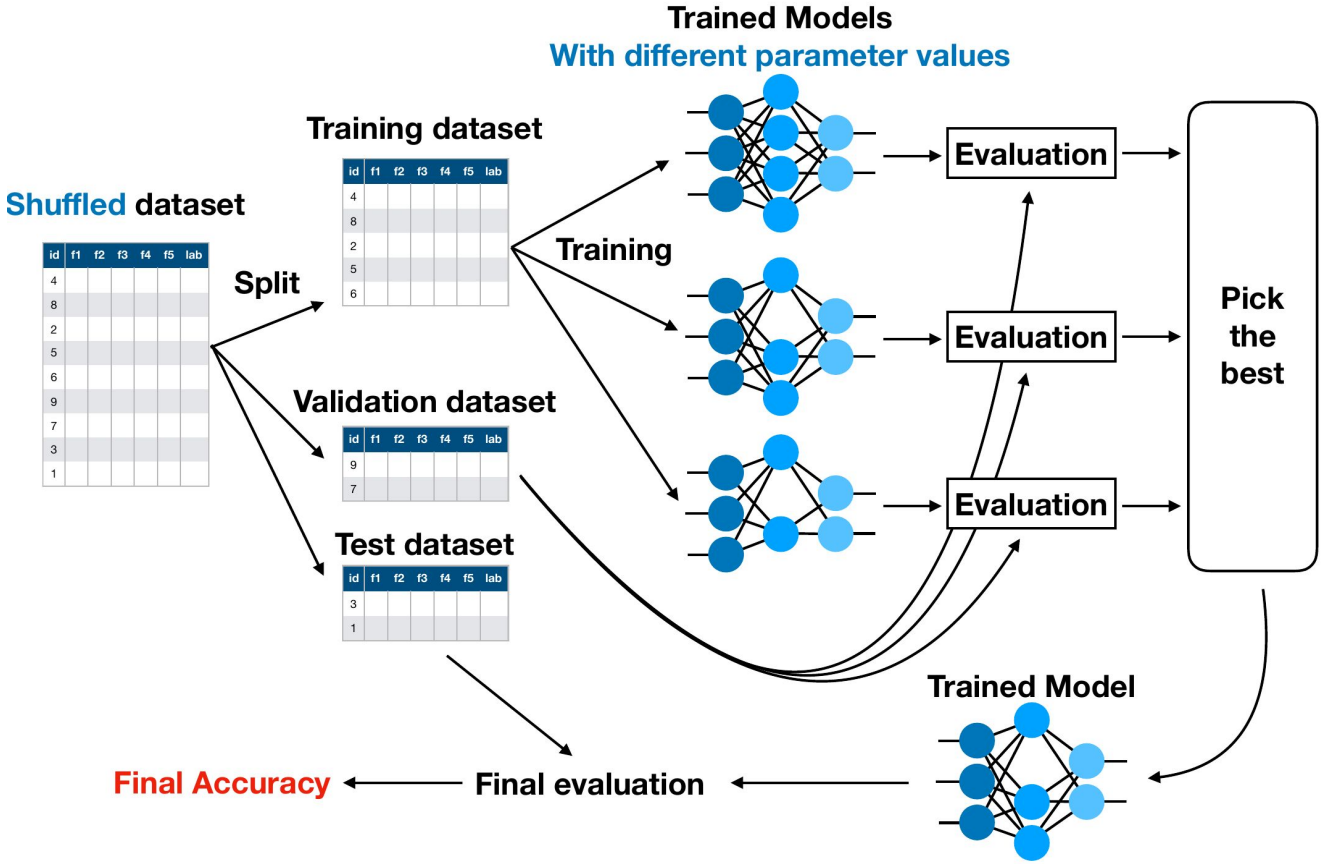
- Try different values on the training dataset, then select the best according to the accuracy on the test dataset.
- Issue: The test dataset is now part of your training process (because you made a decision according to it) therefore we can’t evaluate how your algorithm can generalise to unknown data.
- You should never do that!

Hyperparameter tuning

Right approach:

- Split your dataset in 3: training/validation/test
- Common splits between 60%/20%/20% and 80%/10%/10%
- Try different hyperparameter values on the training dataset, then select the best according to the accuracy on the **validation dataset**.
  Perform the final evaluation on the **test dataset**.
- Advantage: your hyperparameter choice takes into account how the model would generalise, and the final evaluation still only uses previously unseen data.

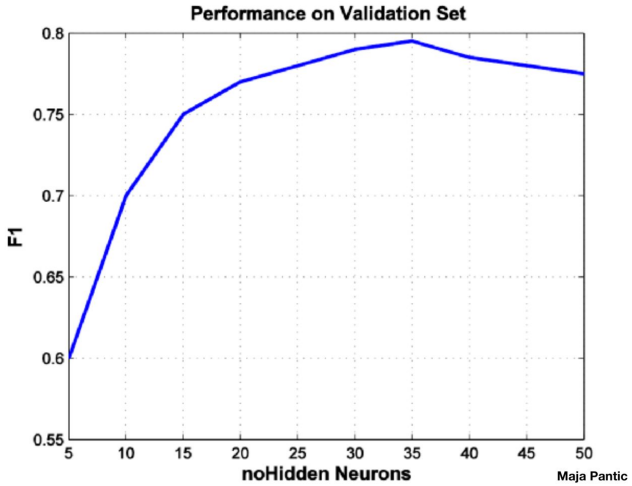
Hyperparameter tuning



Hyperparameter tuning

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

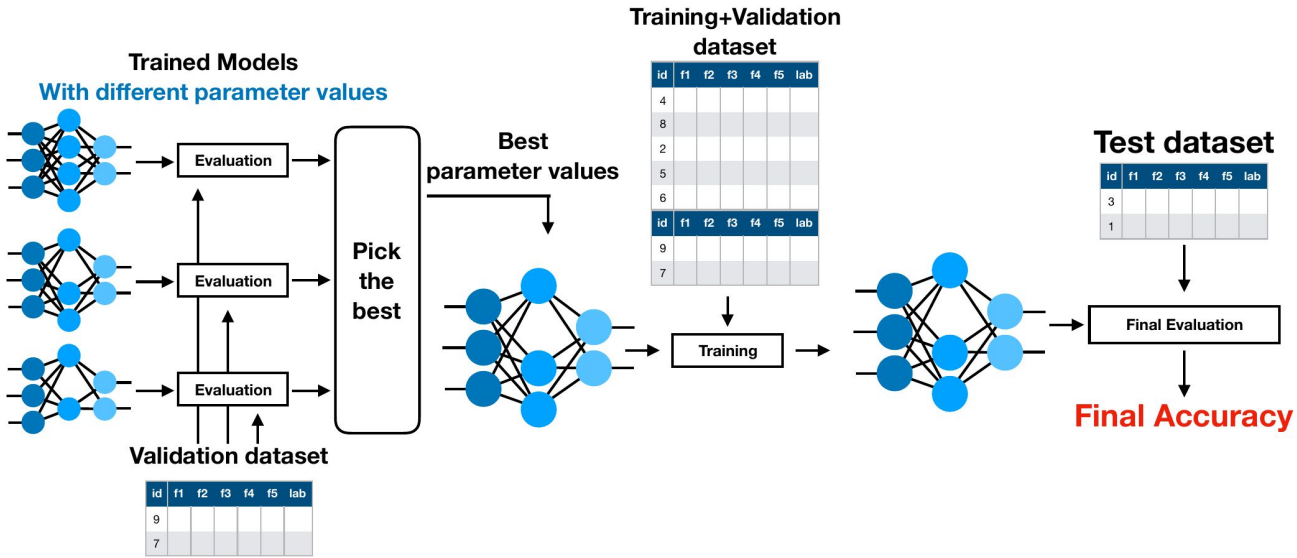
- This is called hyperparameter tuning/optimization, since you select the set of parameters that have produced the best classifier.



For the final evaluation

Now that the hyperparameters 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.



Case Study 1: Sentence Classifier

Company is given a dataset and asked to develop a classifier.

They write manual rules, based on how well each rule works on the dataset.

The final classifier is delivered with over 85%+ accuracy.

The client evaluates it on a new dataset, gets 25% accuracy.

Developing and evaluating the model based on the same data
-> Overfitting the model to the same dataset

Case Study 2: Preterm Birth Prediction

Many studies report near perfect results at predicting the risk of preterm birth for a patient.

The dataset is very imbalanced, so the researchers used oversampling to balance the classes.

They then split some of the data off for a dedicated test set.

Case Study 2: Preterm Birth Prediction

Many studies report near perfect results at predicting the risk of preterm birth for a patient.

The dataset is very imbalanced, so the researchers used oversampling to balance the classes.

They then split some of the data off for a dedicated test set.

Copies of examples can overlap between the training and test set
-> Training and testing on the same data

Case Study 2: Preterm Birth Prediction

Overly Optimistic Prediction Results on Imbalanced Data: Flaws and Benefits of Applying Over-sampling

Gilles Vandewiele[1], Isabelle Dehaene[2], György Kovács[4], Lucas Sterckx[1], Olivier Janssens[1], Femke Ongenae[1], Femke De Backere[1], Filip De Turck[1], Kristien Roelens[2], Johan Decruyenaere[3], Sofie Van Hoecke[1], and Thomas Demeester[1]

[1] IDLab, Ghent University – imec
Technologiepark-Zwijnaarde 126, Ghent, Belgium
[2] Department of Gynaecology and Obstetrics, Ghent University Hospital
Corneel Heymanslaan 10, Ghent, Belgium
[3] Department of Intensive Care Medicine, Ghent University Hospital
Corneel Heymanslaan 10, Ghent, Belgium
{firstname}.{lastname}@ugent.be
[4] Analytical Minds Ltd.
Arpad street 5, Beregsurany, Hungary
gyuriofkovacs@gmail.com

Copies of examples can overlap between the training and test set -> Training and testing on the same data

Model evaluation

To recap:

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

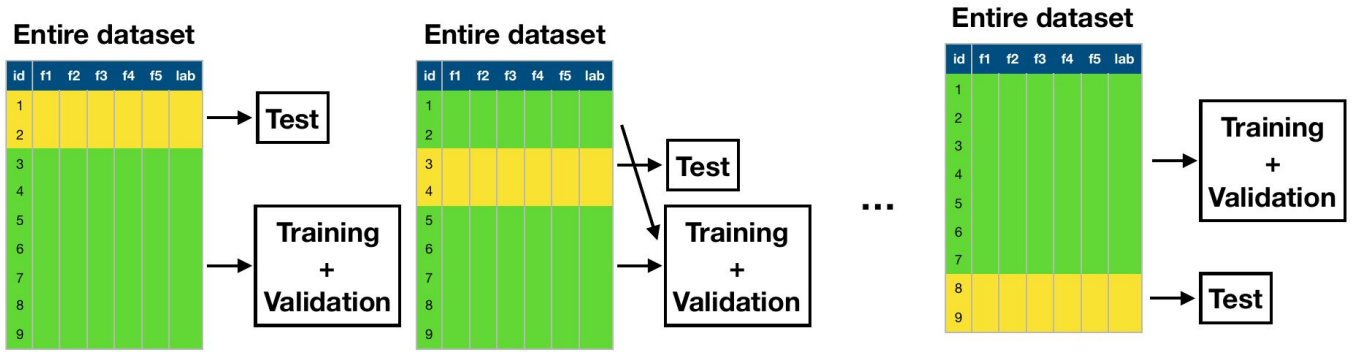
- You should assume that you do not know the labels of the test set and they are given to you only after you have trained your classifier.

Cross-validation

Another approach

When we have a lot of examples then the division into training/validation/test datasets is sufficient.

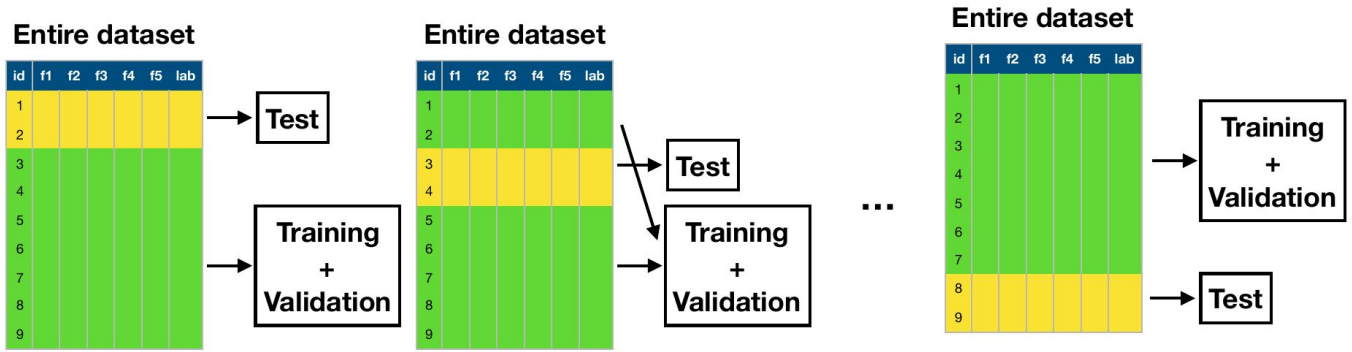
When we have only a small dataset available then a good alternative is **cross-validation**.



Another approach

- Divide dataset into k (usually 10) equal folds/splits. Use k-1 folds for training+validation and one for testing.
- Iterate k times, each time testing on a different portion of the data.
- Performance on all k held-out test sets can then be averaged

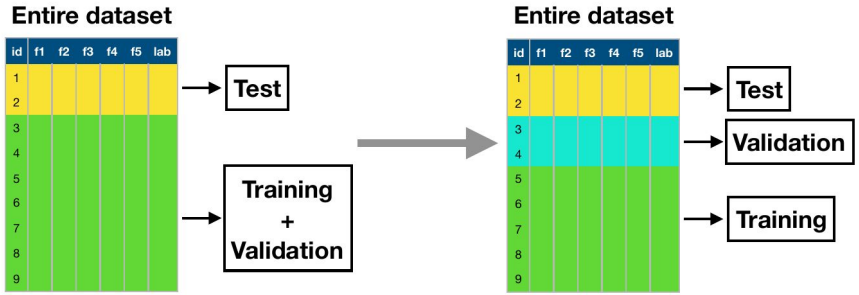
Global error estimate $= \frac{1}{N} \sum_{i=1}^{N} e_i$



Cross-validation: Parameter tuning

How do we estimate the hyperparameters when doing cross-validation?

Option 1:

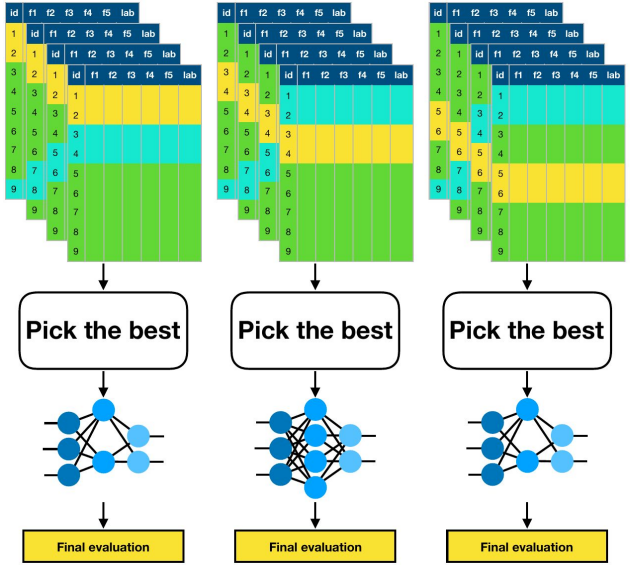


- At each iteration use 1 fold for testing, 1 fold for validation and k-2 folds for training.
- Whenever we are using multiple training sets, we are evaluating the algorithm, not a particular model.
- **We find a different set of optimal parameters in each fold.**

Cross-validation: Parameter tuning

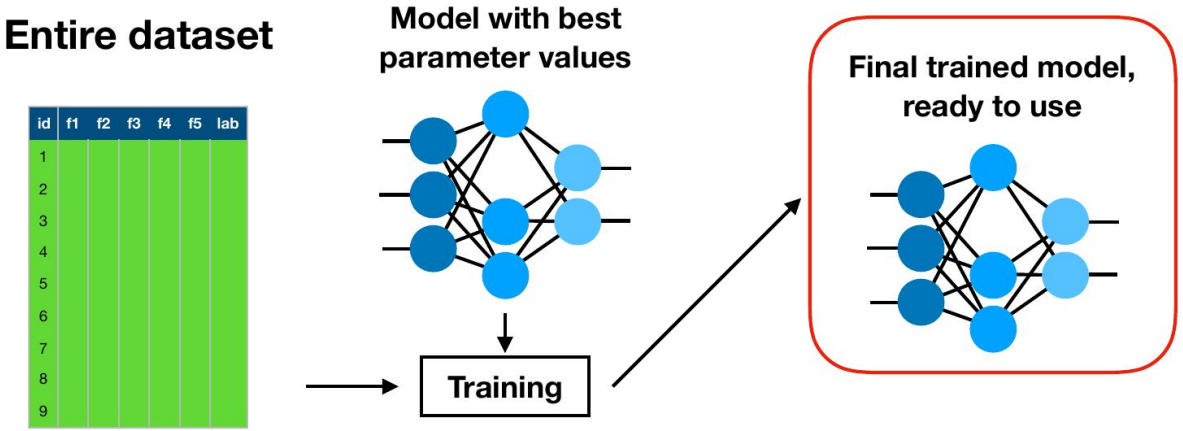
Option 2:

- At each cross-validation step separate 1 fold for testing.
- Run an internal cross-validation over the remaining k-1 folds to find the optimal hyperparameters.
- Every fold will still have different hyperparameters, just chosen based on more data.



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

What happens when we go into production?



When all our experiments are done and we know the best hyperparameter values, we can retrain on the whole dataset.

Good: Can use all the available data for training the model.

Bad: We don't have a way of estimating the performance of the final trained model any more.

Parameter optimisation and performance estimation: in summary

CASE 1: Plenty of data available -> Held-out test set

1) Train algorithm on **training set**
2) Tune hyperparameters on **validation set**
3) Estimate generalisation performance using the **test set**

CASE 2: Limited data available -> Cross-validation

1) Separate dataset into **k folds**
2) Use 1 fold for testing and k-1 folds for **training+validation**
3) Repeat k times, using each fold as the **test set**
4) Estimate generalisation performance by **averaging results** across all the test folds

Evaluation metrics

How to quantify the quality 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	Class 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	Class 2 Predicted
Class 1 Actual	TP: **3**	FN: **1**
Class 2 Actual	FP: **2**	TN: **2**

- This table also highlights the risk of each prediction.
- In some cases, it is more important to have fewer false negatives than fewer false positives (e.g., diagnosing a disease)

Accuracy

Results from the test dataset

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

Confusion Matrix

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

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- Number of correctly classified examples divided by the total number of examples
- Classification error = 1 - accuracy

```
Accuracy = 5/8 = 0.625
         = 62.5%
```

Precision

Results from the test dataset

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

Confusion Matrix

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

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

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

```
Precision for class 1
 = 3/5 = 0.6 = 60%
```

Recall

Results from the test dataset

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

Confusion Matrix

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

$$\textbf{Recall} = \frac{TP}{TP + FN}$$

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

```
Recall for class 1
 = 3/4 = 0.75 = 75%
```

Precision vs Recall

Results from the test dataset

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

Confusion Matrix

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

$$\textbf{Precision} = \frac{TP}{TP + FP} \qquad \textbf{Recall} = \frac{TP}{TP + FN}$$

- High recall, low precision:
  Most of the positive examples are correctly recognised (low FN) but there are many false positives
- Low recall, high precision:
  We miss a lot of positive examples (high FN) but those we predict as positive are really positive (low FP)
- Think about what is important for your application!

Macro-averaged Recall

Results from the test dataset

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

Confusion Matrix

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

$$\textbf{Recall} = \frac{TP}{TP + FN}$$

- We can compute recall for class1 (R1) and for class2 (R2)
- Macro-averaged recall = mean (R1, R2)
- Can do the same for precision and F-measure

```
Recall for class 1 = 3/4
Recall for class 2 = 2/4
Macro-averaged recall = 62.5%
```

F-measure / F-score

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

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

Equal importance for precision and recall

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

Beta = 2 ==> Recall is twice importance than precision

Going further: This formulation is based on the concept of harmonic means (in this case, weighted by beta)

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	Class 2 Predicted	Class 3 Predicted	Class 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 we can also compute the confusion matrix.
- We can define one class as positive and the others as negative.
- We can then 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	Class 2 Predicted	Class 3 Predicted	Class 4 Predicted
Class 1 actual	TP	FN	FN	FN
Class 2 Actual	FP	TN	?	?
Class 3 actual	FP	?	TN	?
Class 4 actual	FP	?	?	TN

- Accuracy = number of correctly classified examples divided by the total number of examples.
- Precision, recall and F1 are computed for each class separately.
- We can then calculate macro-averaged precision, recall and F1.

Micro- vs macro-averaging

Macro-averaging: we are taking the average on the class level. We calculate metrics for each class and average them together.

$$P_{macro} = \frac{1}{4} \cdot (\frac{1}{3} + \frac{1}{1} + \frac{1}{2} + \frac{1}{2}) = 0.583$$

Micro-averaging: we are taking the average on the item level. For example, adding the TP, FP, TN, FN values for each class before calculating the metrics.

$$P_{micro} = \frac{1+1+1+1}{(1+1+1+1)+(2+0+1+1)} = 0.5$$

Note: For binary and multi-class classification, micro-averaged P, R and F1 will be just equal to accuracy. This can vary for more complex measures and multi-label classification.

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

Regression tasks

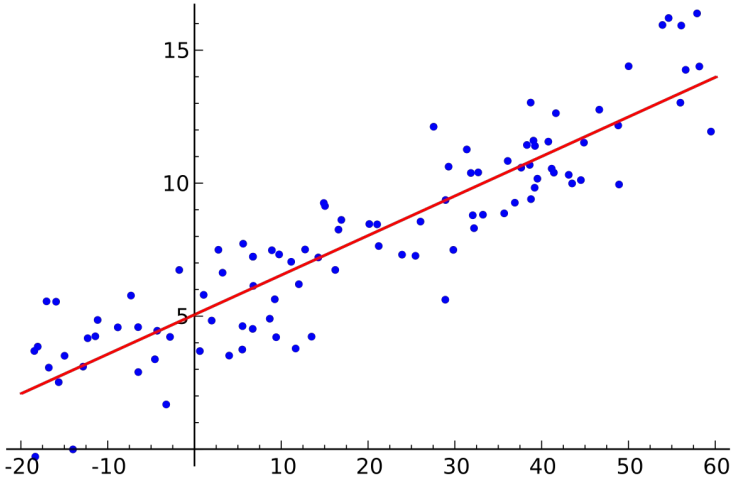
- Mean squared error (MSE):

$$\frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2$$

Y_i Sample from the dataset

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

Lower is better!



- Root mean squared error (RMSE):

$$\sqrt{(\mathbf{MSE})}$$

It’s not all about accuracy

We want the models to be

1. **Accurate:** making correct predictions
2. **Fast:** is fast to train and to query
3. **Scalable:** works with very large datasets
4. **Simple:** is understandable and robust
5. **Interpretable:** can explain its predictions

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/

Imbalanced data distribution

Balanced test set

	Class 1 Predicted	Class 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%

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

Acc: 80%

- 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	Class 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%

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

Acc: 80%

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

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%

Acc: 71.8%

- Imbalanced dataset: Classes are not equally represented
- Accuracy 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 as class 2

Imbalance: One class is completely misclassified

	Class 1 Predicted	Class 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%

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

Acc: 80%

	Class 1 Predicted	Class 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%

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

Acc: 63.6%

- Accuracy is misleading, class 2 is completely misclassified.
- F1 for class 2 shows that something is wrong.
- Macro-averaged recall also takes all classes into account.

Imbalanced test set: Conclusion

- Accuracy can be misleading, simply follows the performance of the majority class.
- Macro-averaged recall can also help detect if one class is completely misclassified, but it does not give us information about FP
- F1 is useful, but can also be affected by the class imbalance problem. We are not sure if the low score is due to one class being misclassified or class imbalance.
- We should look at several metrics, along with the confusion matrix

Imbalanced test set: solutions

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

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%

Acc: 71.8%

Divide by the total number of examples per class

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

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%

Acc: 80%

Can measure performance ALSO on the “normalised matrix”

Imbalanced test set: solutions

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

Divide by the total number of examples per class

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

- These would be the results if we had a balanced 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)

Can measure performance ALSO on the "normalised matrix"

Imbalanced test set: solutions

- We can downsample the majority class
  - e.g. select randomly the same number of examples as the minority class

- Or upsample the minority class (create add duplicates until both classes are the same size)

- This will help balance the classes, but the results will not reflect how well this model will generalise - the **real data** will still come from an imbalanced distribution

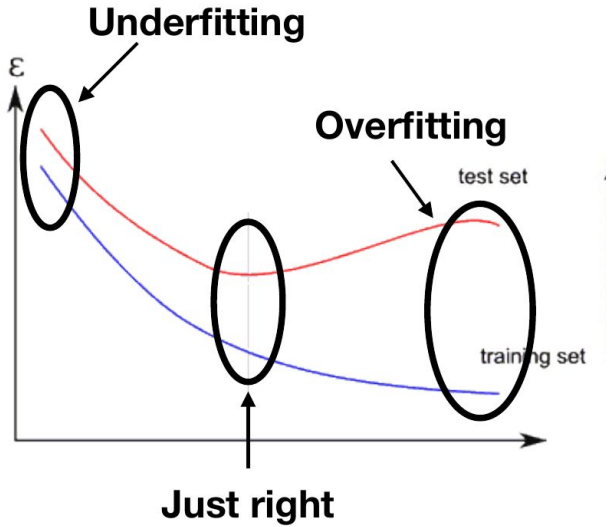
- Best to look at several different metrics and choose ones that reflect the intended behaviour of the model.

Overfitting

Overfitting

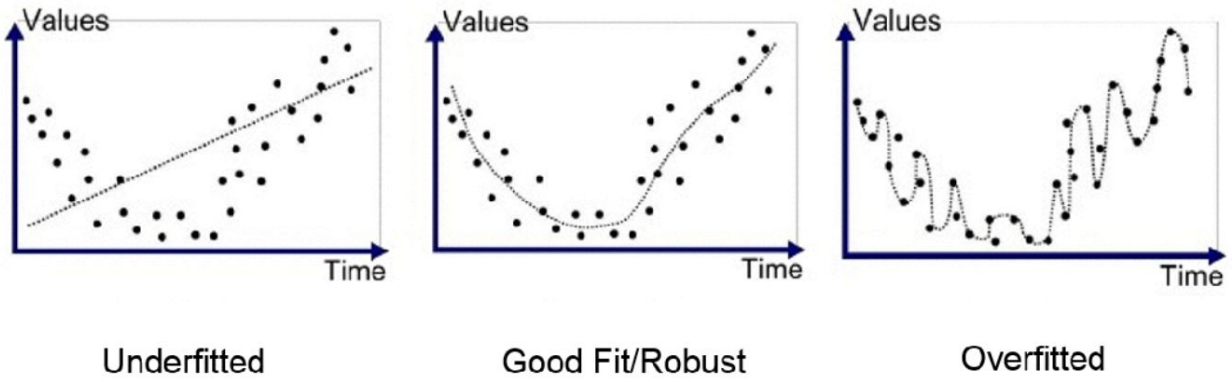
- **Overfitting:** Good performance on the training data, poor generalisation to other data.

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



Overfitting

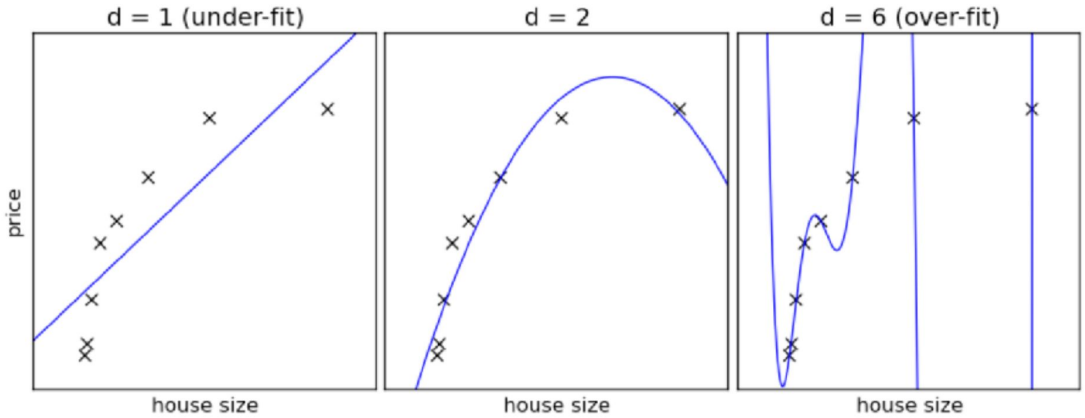
- **Overfitting:** Good performance on the training data, poor generalisation to other data.
- **Underfitting:** Poor performance on the training data and poor generalisation to other data.



How to fight overfitting

Overfitting can occur when:

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

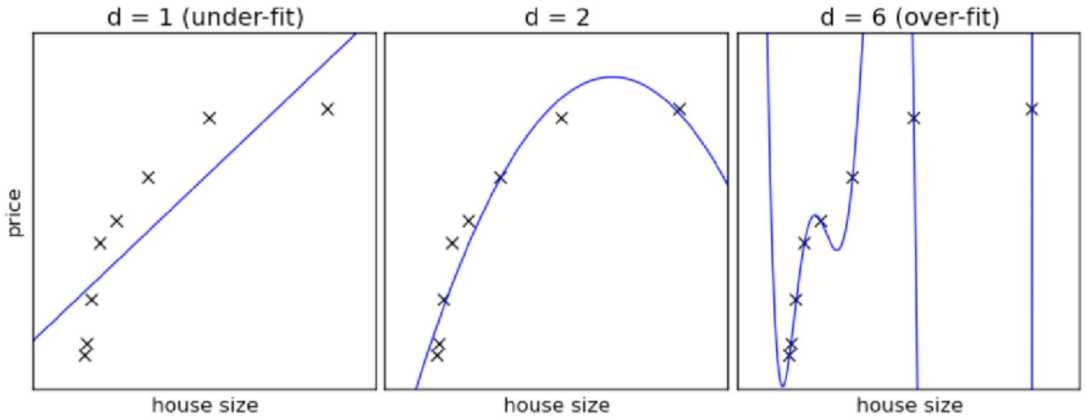


https://github.com/astroML/sklearn_tutorial/

How to fight overfitting

We can fight overfitting by:

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



https://github.com/astroML/sklearn_tutorial/

Summary so far

- We have seen how to evaluate our models: using a held-out test set or cross-validation
- How to measure the performance of the models: Accuracy, precision, recall, F1
- What is overfitting and how to mitigate it

Confidence intervals

Can we trust the performance metrics?

This is a critical question. The decisions of some ML algorithms can directly change people's lives.

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

Can we trust the performance metrics?

- The amount of data used in the test set also impacts our confidence of the performance evaluation:
    - 90% accuracy on a test set with 10 samples is different from 84% accuracy on a test set with 10000 samples.
- This can be captured by the confidence interval.

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

$$\delta(f(x), h(x)) = 1 \textbf{ if } f(x) \neq h(x)$$

$$\delta(f(x), h(x)) = 0 \textbf{ if } f(x) = h(x)$$

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

Confidence interval

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

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

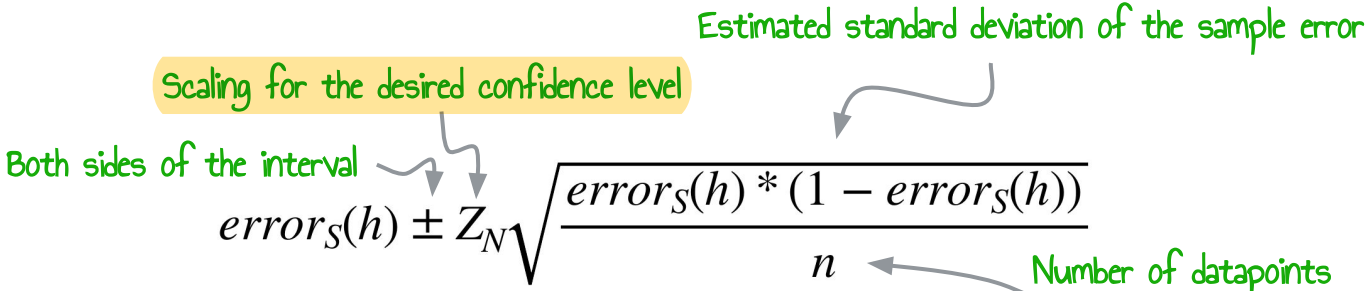
$$error_S(h) \pm Z_N \sqrt{\frac{error_S(h) * (1 - error_S(h))}{n}}$$

$N\%$	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

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

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



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 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 %

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

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

- For the Face modality, what is n? What is $error_s(h)$? $\text{error}(h) = 22\%, N=50$
- Exercise: compute the 95% confidence interval for this error.

Confidence interval example

- Given that $error_S(h)=0.22$, $n=50$ and $Z_N=1.96$ for $N=95\%$, we can now say that with 95% confidence $error_D(h)$ will lie in the interval:

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

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

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

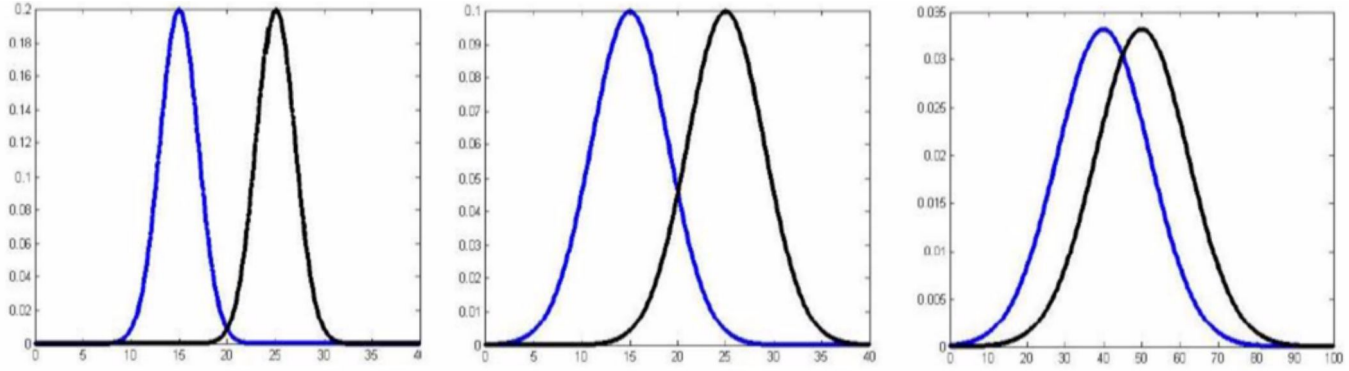
- What will happen when n -> infinity?

It will converge to 0.22 (Sample Error)

Testing for statistical significance

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

- The **statistical tests** tell us if the means of the two sets are significantly different.
- There are several statistical tests: Randomisation, T-test, Wilcoxon rank-sum, etc.
- Randomisation test: randomly switch some predictions from both models and measure how often the new performance difference is greater or equal than the original difference.
- Two-sample T-test: estimate the likelihood that two metrics (e.g. classification error) from different populations are actually different.
- Paired T-test: estimating significance over multiple matched results, for example classification error over the same folds in cross-validation

Comparing two algorithms

- The **null hypothesis**: the hypothesis that the two algorithms/models perform the same and the performance differences are only due to sampling error.

- The statistical tests return a **p-value**: the probability of obtaining observed performance differences (or more), assuming that the null hypothesis is correct.

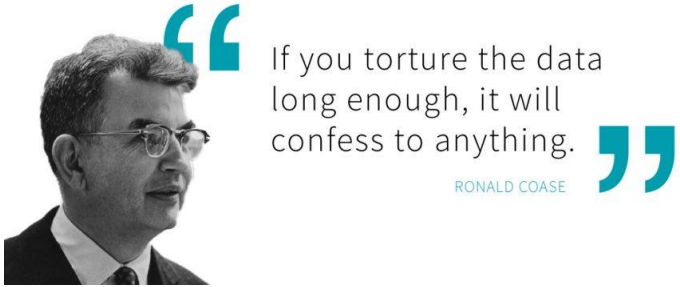
- Small p-value -> we can be more confident that one system is actually different from another.

Comparing two algorithms

- Performance difference is considered to be **statistically significant** if $p<0.05$.
- A p-value of > 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.

P-hacking

P-hacking is the misuse of data analysis to find patterns in data that can be presented as statistically significant when in fact there is no underlying effect.

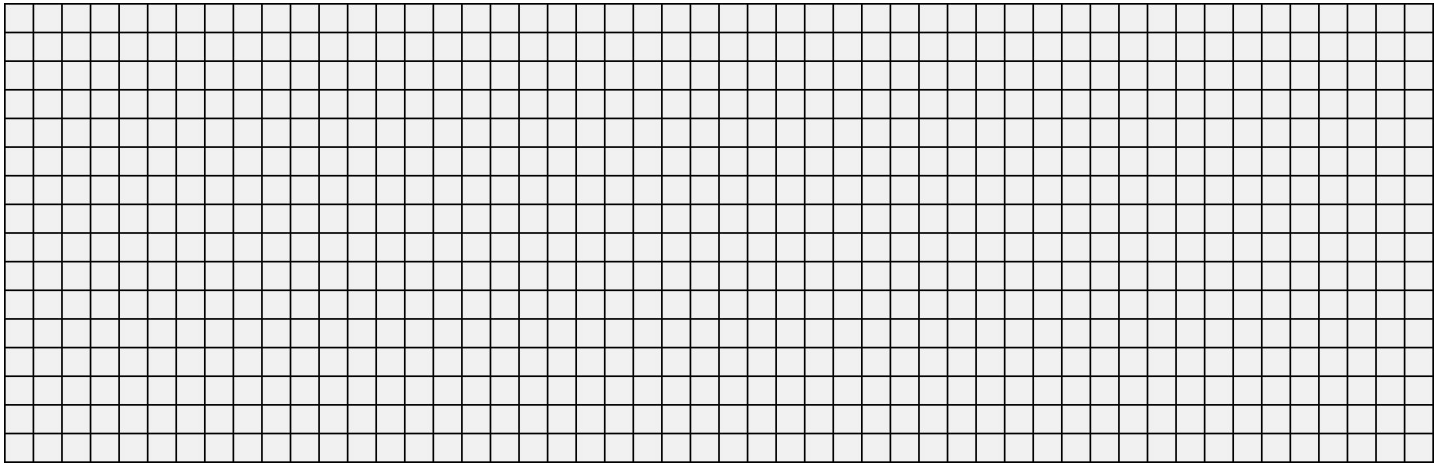


Done by running large numbers of experiments and only paying attention to the ones that come back with significant results.

Statistical significance is defined as being less than 5% likely that the result is due to randomness ($p < 0.05$).

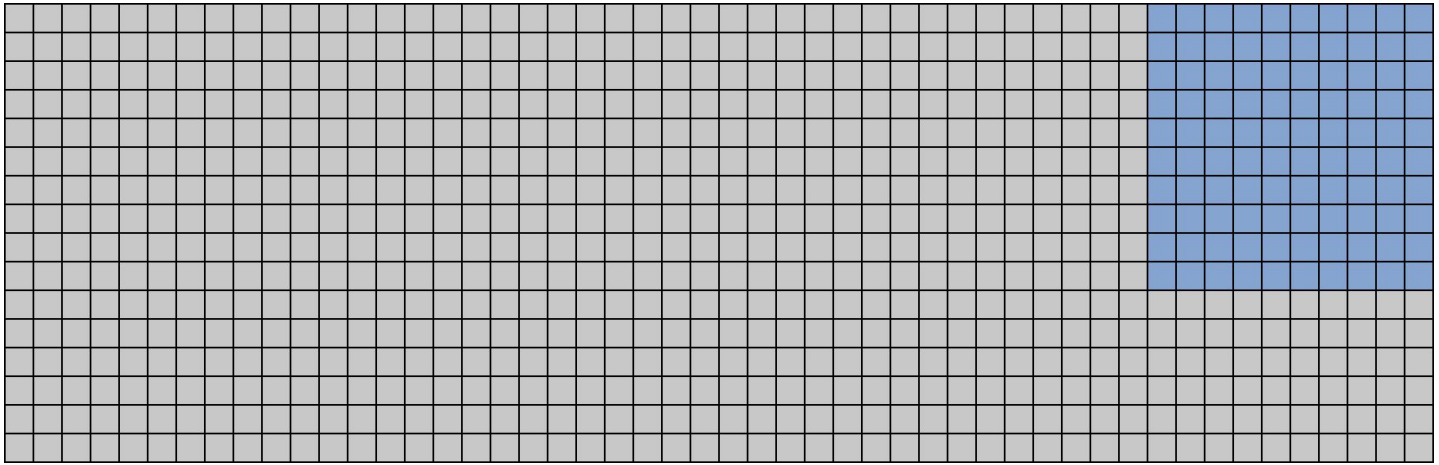
That means we accept that some “significant” results are going to be false positives!

P-hacking



Each square represents one possible experiment.
Let’s say we have 800 different experiments we can try.

P-hacking

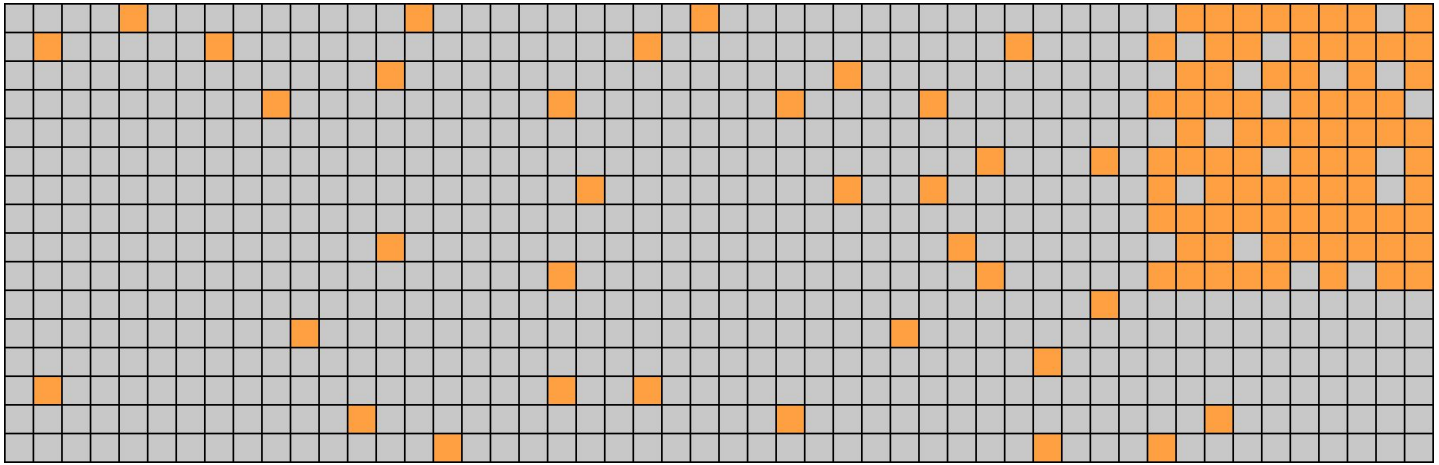


The **true** underlying distribution:

100 experiments are positive

700 experiments are negative

P-hacking

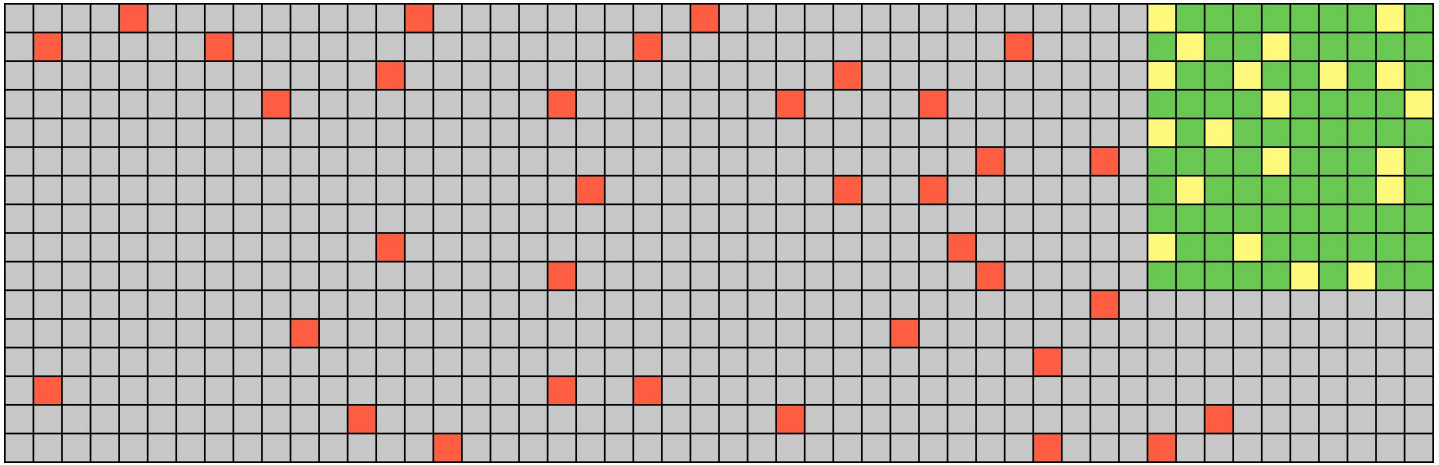


On our data sample, we get either:

Positive results OR Negative results

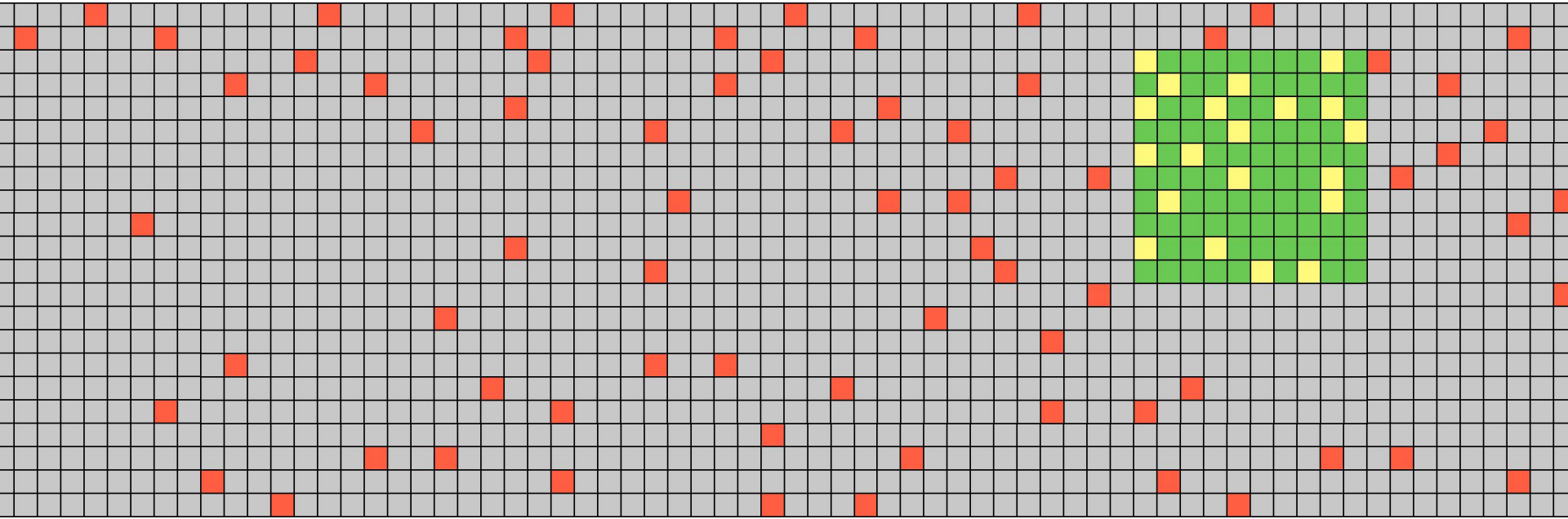
P(false positive) = 0.05 P(false negative) = 0.2

P-hacking



We made 80 true discoveries (TP)
We missed 20 potential discoveries (FN)
We made 35 false discoveries (FP)
False Discovery Proportion = 35 / 115 = 30%

P-hacking



If we do this over 2400 experiments:
We made 80 true discoveries
We made 115 false discoveries
False Discovery Proportion = 80 / 195 = 59%

Spurious correlations

A sample “study” with 54 people, searching over 27,716 possible relations.

Our shocking new study finds that ...

EATING OR DRINKING	IS LINKED TO	P-VALUE
Raw tomatoes	Judaism	<0.0001
Egg rolls	Dog ownership	<0.0001
Energy drinks	Smoking	<0.0001
Potato chips	Higher score on SAT math vs. verbal	0.0001
Soda	Weird rash in the past year	0.0002
Shellfish	Right-handedness	0.0002
Lemonade	Belief that “Crash” deserved to win best picture	0.0004
Fried/breaded fish	Democratic Party affiliation	0.0007
Beer	Frequent smoking	0.0013
Coffee	Cat ownership	0.0016
Table salt	Positive relationship with Internet service provider	0.0014

https://fivethirtyeight.com/features/you-cant-trust-what-you-read-about-nutrition/

Strategies against P-hacking

Distinguish between verifying a hypothesis and exploring the data.

Benjamini & Hochberg (1995) offer an adaptive p-value:

1. Rank p-values from M experiments.

$$p_1 \leq p_2 \leq p_3 \leq \ldots \leq p_M$$

2. Calculate the Benjamini-Hochberg critical value for each experiment.

$$z_i = 0.05\frac{i}{M}$$

3. Significant results are the ones where the p-value is smaller than the critical value.