DMI

Machine Learning II

Data Mining and Data integration in Biomedicine
Master in Bioinformatics

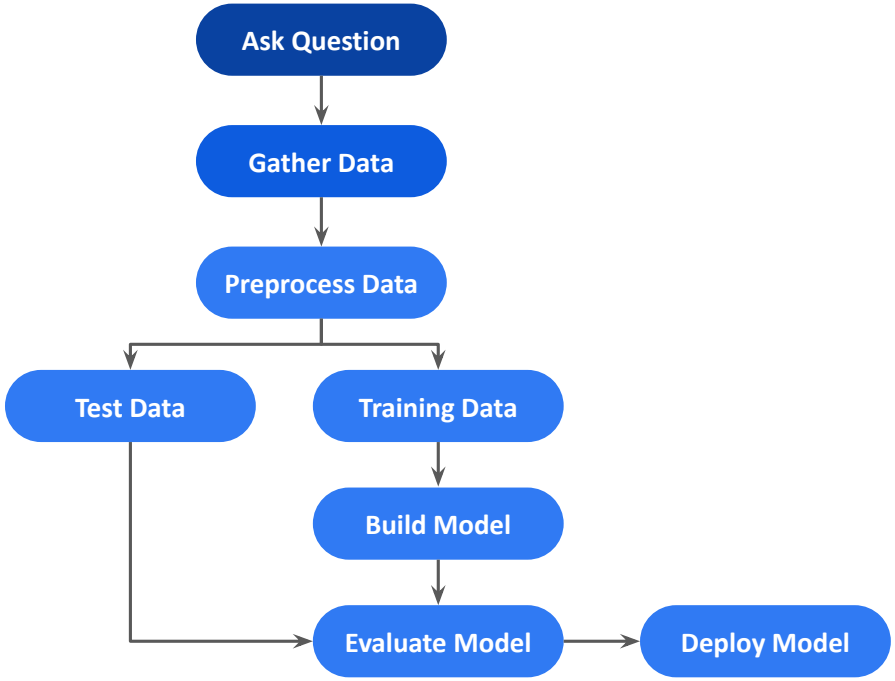
Janet Piñero
Medbioinformatics Solutions SL
2025-2026

Outline

- Brief recap of previous lesson
- Bias -Variance trade-off
- Resampling methods
- Random forest algorithms
- General recommendations
- Hands-on session
- Bibliography

Simplified view of the machine learning pipeline

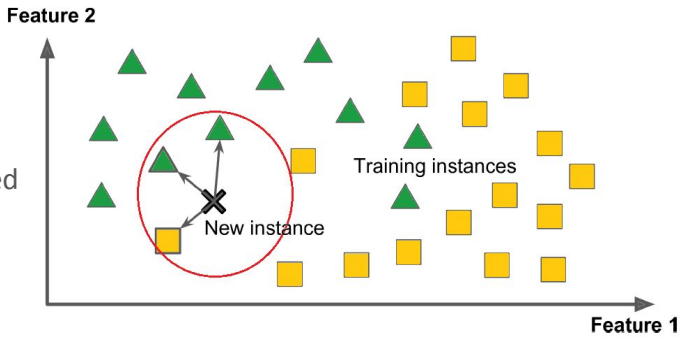
DMI



KNN revisited

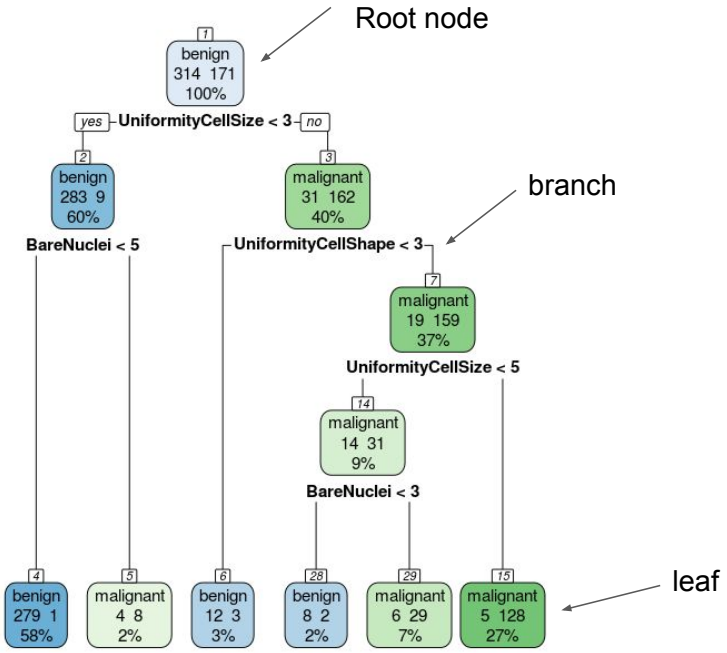
DMI

- ✓ Simple algorithm in which each observation is predicted based on its “similarity” to other observations.
- ✓ The k-nearest neighbors (kNN) algorithm can be used for regression as well as classification.
- ✓ In order to provide accurate classification, KNN requires a lot of observations relative to the number of predictors—that is, n much larger than p.
- ✓ The performance of KNNs is very sensitive to the choice of k: In general, low values of k typically overfit and large values often underfit.



When using KNN for classification, it is best to assess odd numbers for k to avoid ties in the event there is equal proportion of response levels

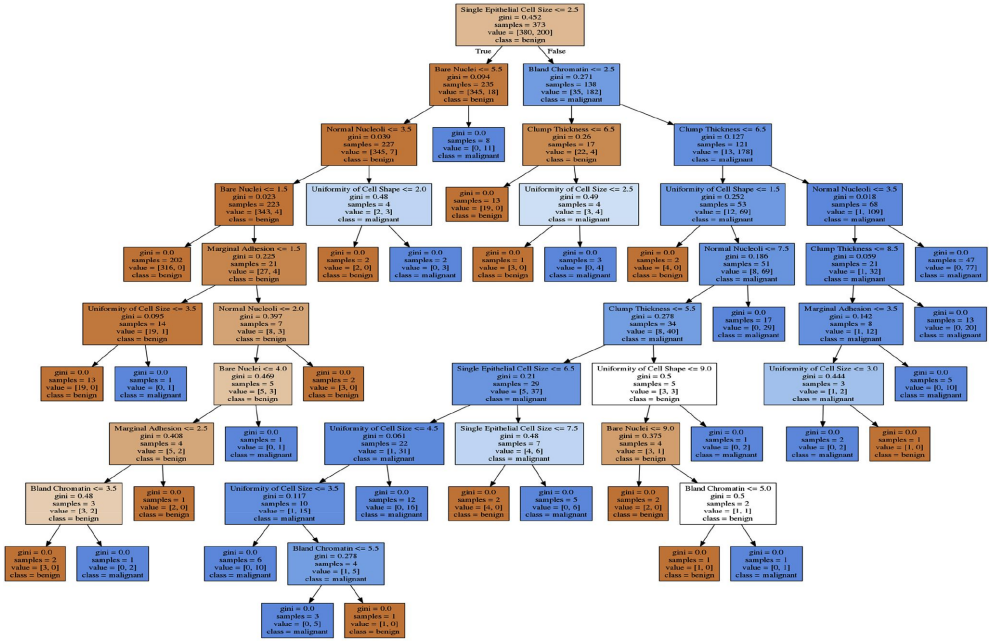
Decision Trees revisited



Basic algorithm

- Start with all variables in one group
- Find the variable/split that best separates the outcomes
- Divide the data into two groups ("leaves") on that split ("node")
- Within each split, find the best variable/split that separates the outcomes
- Continue until the groups are too small or sufficiently "pure"

Decision trees recap



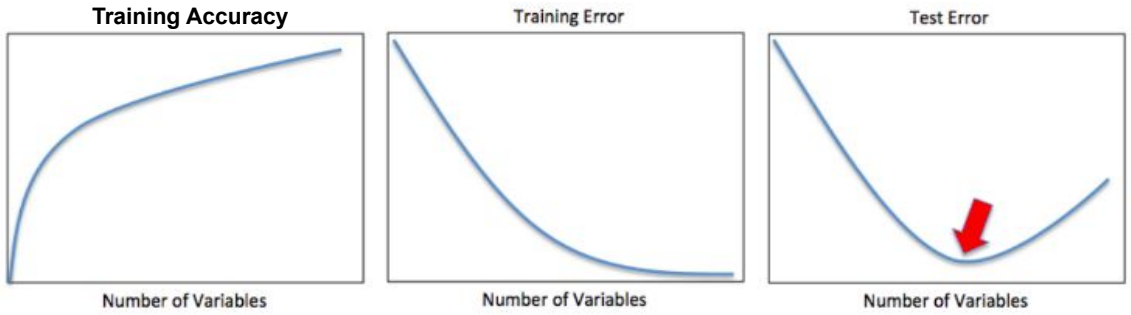
(a)

What happens with new data?

The problem with training error

DMI

As the model complexity increases….



... the training error decreases and the accuracy increases!

But the test error might not!

Model Complexity, Training Error, and Test Error

DMI

Model complexity is a key concept in supervised learning:

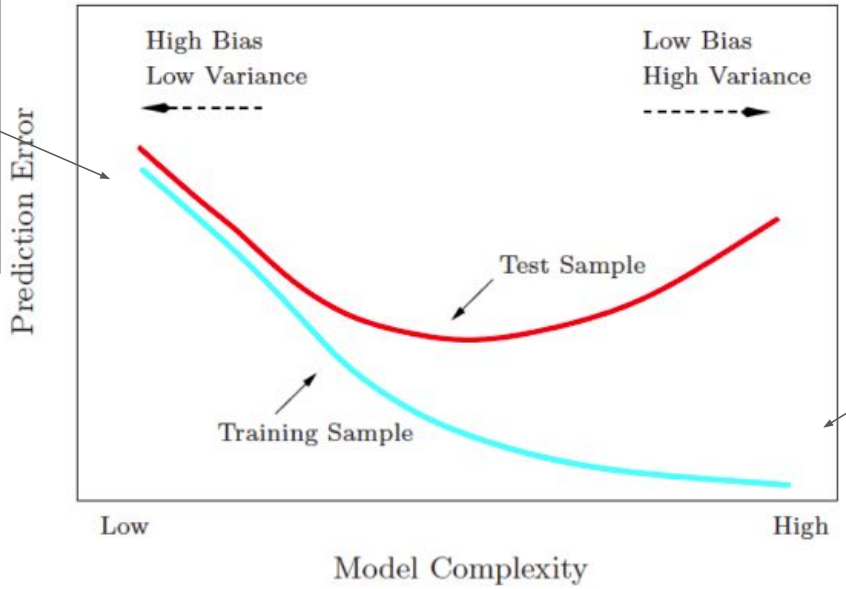
- As model complexity increases, training error always decreases.
- But the test error might not.

With four parameters I can fit an elephant and with five I can make him wiggle his trunk. John von Neumann

Bias-Variance Trade-Off

DMI

Model with high bias pays very little attention to the training data and oversimplifies the model. It leads to high error on training and test data (underfitting).



Overfitting:
A model is over complex and performs very well on training data but poorly on unseen data.

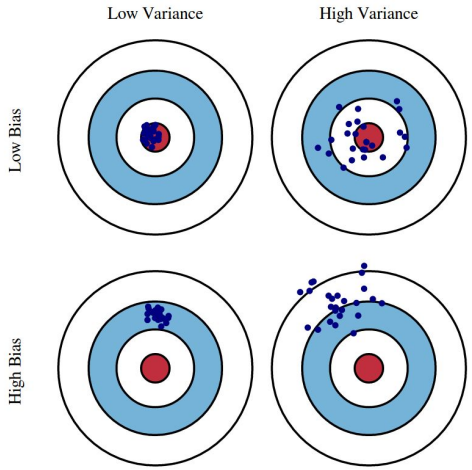
Bias-Variance Trade-Off

DMI

Bias: difference between the expected prediction value and the correct response value

Variance: the variability of the prediction values when we construct models multiple times with different training sets for the same problem.

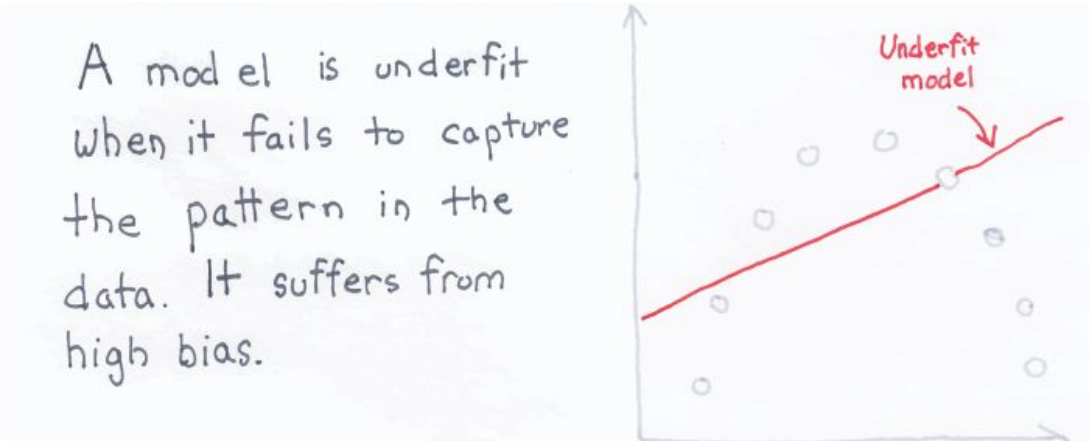
Note that the variance does not depend of the correct value of the test cases!



https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print12_toc.pdf

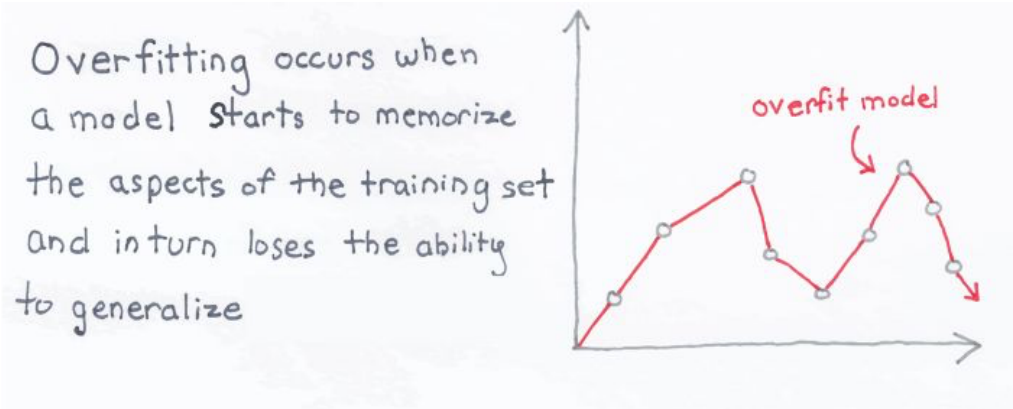
Underfitting

DMI



Underfitting = tree too shallow

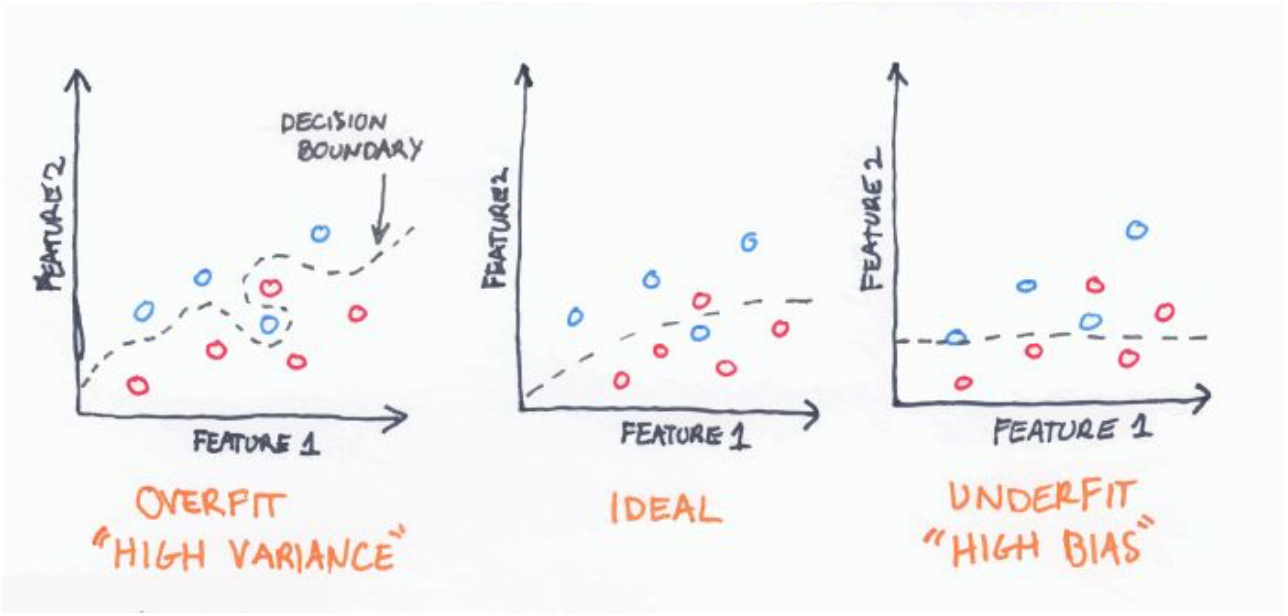
Overfitting



Overfitting = tree too deep

Overfit vs underfit

DMI



Bias-Variance Trade-Off

DMI

Because we typically like to explain away as much variation in our data as possible, and because we often have many more variables than are important for our problem, underfitting is less frequently a problem than overfitting.

Bias-Variance Trade-Off

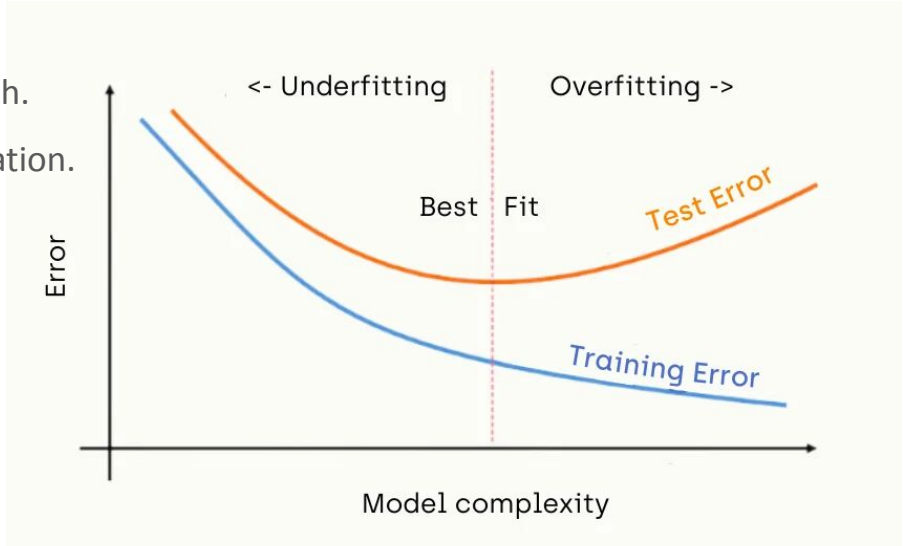
DMI

Since high variance models are more prone to overfitting, using **resampling procedures** are critical to reduce this risk. Moreover, many algorithms that are capable of achieving high generalization performance have lots of **hyperparameters** that control the level of model complexity (i.e., the tradeoff between bias and variance).

Three Ways to Estimate Test Error with Training Data

DMI

1. The validation set approach.
2. Leave-one-out cross-validation.
3. K-fold cross-validation.



Validation Set Approach

DMI

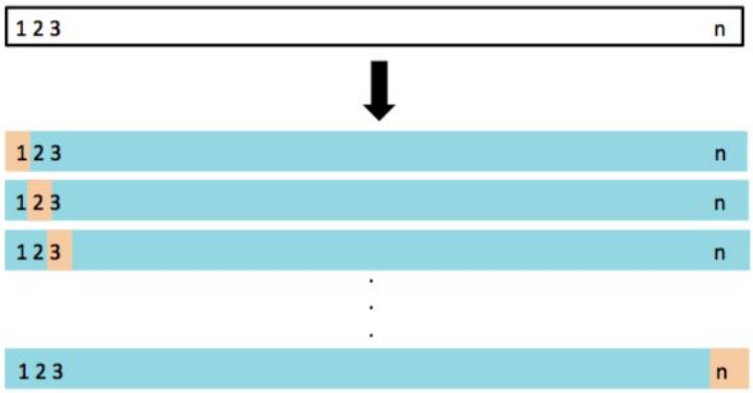
Split the n observations into two sets of approximately equal (80:20, 60:40) size.

Train on one set, and evaluate performance on the other.

- the validation estimate of the test error rate can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set.
- only a subset of the observations (those that are included in the training set rather than in the validation set) are used to fit the model.

Leave-One-Out Cross-Validation

Fit n models, each on n-1 of the observations. Evaluate each model on the left-out observation.

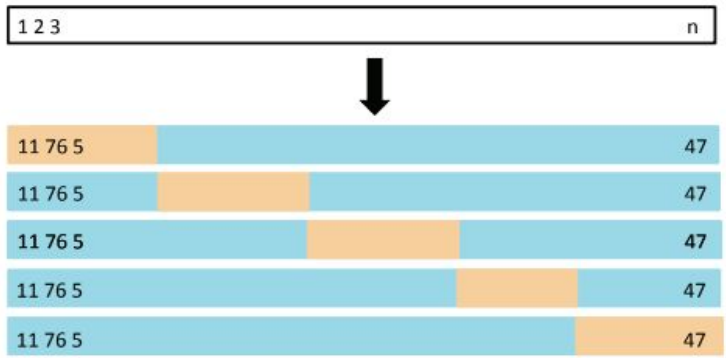


- has far less bias
- performing LOOCV multiple times will always yield the same results: there is no randomness in the training/validation set splits.

5-Fold Cross-Validation

DMI

Split the training data into 5 sets. Repeatedly train the model on 4 sets and evaluate its performance on the 5th.



Typically, one performs k-fold cross-validation using k = 5 or k = 10, as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance. k = 5 or k = 10 is also often computational feasible.

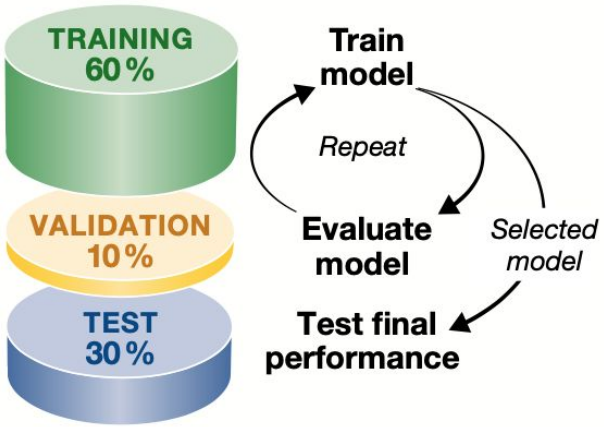
Tips For Evaluating Algorithms

DMI

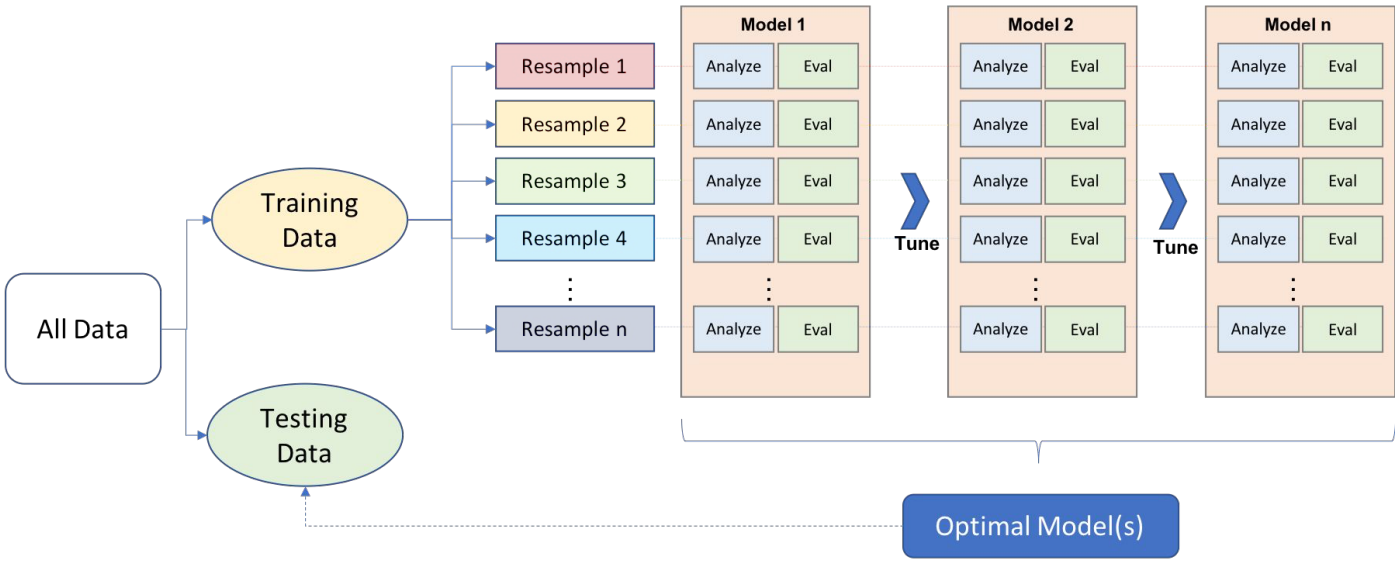
- Using a data split into a training and test set is a good idea when you have a lot of data and you are confident that your training sample is representative of the larger dataset.
- Using a data split is very efficient and is often used to get a quick estimate of model accuracy.
- Cross validation is a gold standard for evaluating model accuracy, often with k-folds set to 5 or 10 to balance overfitting the training data with a fair accuracy estimate.
- Repeated k-fold cross validation is preferred when you can afford the computational expense and require a less biased estimate.

Training-test-validation sets

- Training Dataset: The sample of data used to fit the model.
- Validation Dataset: The sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration.
- Test Dataset: The sample of data used to provide an unbiased evaluation of a final model fit on the training dataset.
- 60%/20%/20% is typical to split the data into three sets



The machine learning process



Hyperparameter tuning

DMI

What algorithms can learn, and what they must be told

Hyperparameters (aka tuning parameters) are the “knobs to twiddle” to control the complexity of machine learning algorithms and, therefore, the bias-variance trade-off.

Not all algorithms have hyperparameters; however, most have at least one or more.

Hyperparameters are all the training variables set manually with a predetermined value before starting the training.

A grid search is an automated approach to searching across many combinations of hyperparameter values

Hyperparameter tuning

DMI

- Babysitting (aka Trial & Error)
- Grid Search
- Random Search

Hyperparameter tuning

DMI

- Babysitting (aka Trial & Error)

Babysitting is also known as Trial & Error or Grad Student Descent in the academic field.
This approach is 100% manual and the most widely adopted by researchers, students, and hobbyists.

The end-to-end workflow is really quite simple: a student devises a new experiment that she follows through all the steps of the learning process (from data collection to feature map visualization), then will she/he iterates sequentially on the hyperparameters until she runs out time (usually due to a deadline) or motivation.

Hyperparameter tuning

DMI

- Grid Search

AKA "Just try everything!

It is a naive approach of simply trying every possible configuration.

The workflow:

Define a grid on n dimensions, where each of these maps for an hyperparameter.

e.g. n = (learning_rate, dropout_rate, batch_size)

For each dimension, define the range of possible values:

e.g. batch_size = [4, 8, 16, 32, 64, 128, 256]

Search for all the possible configurations and wait for the results to establish the best one:

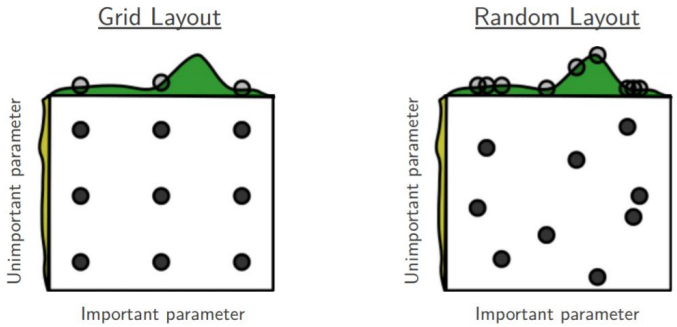
e.g. C1 = (0.1, 0.3, 4) -> acc = 92%, C2 = (0.1, 0.35, 4) -> acc = 92.3%, etc...

Hyperparameter tuning

DMI

- Random Search

The only real difference between Grid Search and Random Search is on the step 1 of the strategy cycle – Random Search picks the point randomly from the configuration space.



With grid search, nine trials only test three distinct places. With random search, all nine trails explore distinct values.

Example: the breast cancer dataset

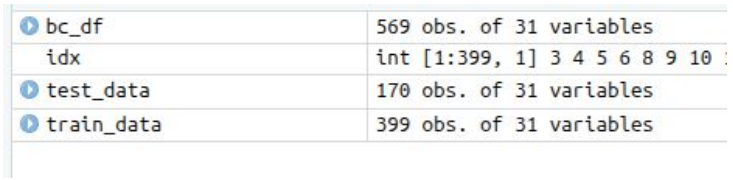
DMI

Id	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	diagnosis
1057013	8	4	5	1	2	4	7	3	1	Malignant
1096800	6	6	6	9	6	2	7	8	1	Benign
1183246	1	1	1	1	1	2	2	1	1	Benign
1184840	1	1	3	1	2	2	2	1	1	Benign
1193683	1	1	2	1	3	3	1	1	1	Benign
1193698	1	1	2	1	3	3	1	1	1	?

https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset

kNN in R

```
rm(list = ls())
library(caret)
library(recipes)
library(dplyr)
library(rpart)
library(rpart.plot)

bc_df<- read.csv("breast-cancer.csv")
bc_df <- bc_df %>%
  select(-id)
bc_df$diagnosis <- factor(bc_df$diagnosis)
set.seed(123)
idx <- createDataPartition(bc_df$diagnosis, p = 0.7, list = FALSE)
train_data <- bc_df[idx, ]
test_data  <- bc_df[-idx, ]
```

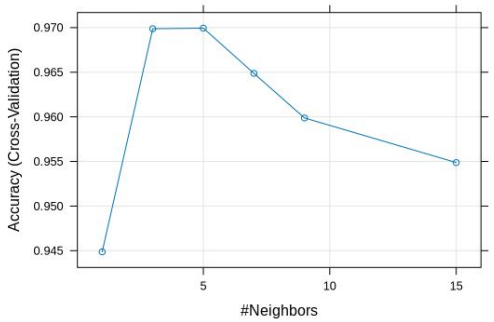


https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset

kNN in R

```
bc_recipe <- recipe(diagnosis ~ ., data = train_data) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())

ctrl <- trainControl(
  method = "cv",
  number = 10
)

knn_model <- train(
  bc_recipe,
  data = train_data,
  method = "knn",
  tuneGrid = data.frame(k = c(1, 3, 5, 7, 9, 15)),
  trControl = ctrl
)

knn_model
plot(knn_model)
```

```
k   Accuracy   Kappa
 1  0.9498077  0.8915986
 3  0.9724359  0.9402680
 5  0.9698718  0.9350759
 7  0.9623718  0.9185654
 9  0.9574359  0.9070828
15  0.9574359  0.9064716

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 3.
>
```



https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset

kNN in R

DMI

```
bc_recipe <- recipe(diagnosis ~ ., data = train_data) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors())

ctrl <- trainControl(
  method = "cv",
  number = 10
)

knn_model <- train(
  bc_recipe,
  data = train_data,
  method = "knn",
  tuneGrid = data.frame(k = c(1, 3, 5, 7, 9, 15)),
  trControl = ctrl
)

plot(knn_model)

pred <- predict(knn_model, newdata = test_data)
confusionMatrix(pred, test_data$diagnosis)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   B   M
         B 105   6
         M   2  57

               Accuracy : 0.9529
                 95% CI : (0.9094, 0.9795)
    No Information Rate : 0.6294
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.8978

 Mcnemar's Test P-Value : 0.2888

            Sensitivity : 0.9813
            Specificity : 0.9048
         Pos Pred Value : 0.9459
         Neg Pred Value : 0.9661
             Prevalence : 0.6294
         Detection Rate : 0.6176
   Detection Prevalence : 0.6529
      Balanced Accuracy : 0.9430

       'Positive' Class : B
```

https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset

Decision trees in R

DMI

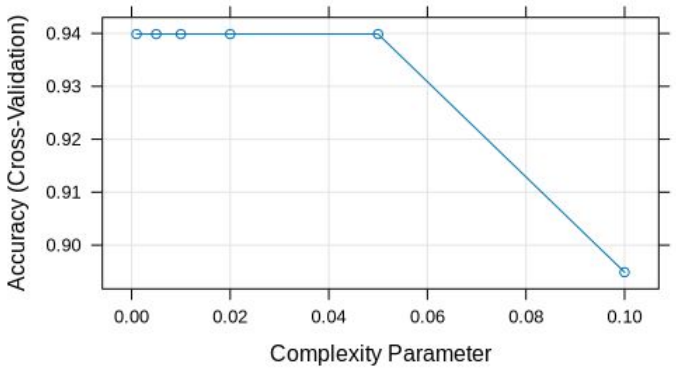
```
tree_recipe <- recipe(diagnosis ~ ., data = train_data)
ctrl <- trainControl(
  method = "cv",
  number = 10
)
cp_grid <- data.frame(  cp = c(0.1, 0.05, 0.02, 0.01, 0.005,
0.001) )

tree_model <- train(
  tree_recipe,
  data = train_data,
  method = "rpart",
  trControl = ctrl,
  tuneGrid = cp_grid
)
tree_model
```

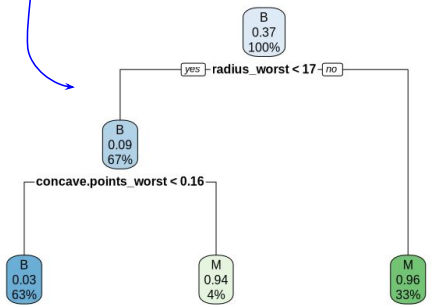
```
399 samples
 30 predictor
  2 classes: 'B', 'M'

Recipe steps:
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 360, 359, 359, 359, 359, 359, ...
Resampling results across tuning parameters:

  cp     Accuracy   Kappa
  0.001  0.9398718  0.8718510
  0.005  0.9398718  0.8715721
  0.010  0.9398718  0.8715721
  0.020  0.9398718  0.8721262
  0.050  0.9398718  0.8721262
  0.100  0.8948718  0.7685415

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.05.
```

Decision trees in R

DMI

```
plot(tree_model)

pred <- predict(tree_model, newdata = test_data)

confusionMatrix(pred, test_data$diagnosis)

rpart.plot(tree_model$finalModel)
```






```
Confusion Matrix and Statistics

          Reference
Prediction   B   M
         B 105   7
         M   2  56

               Accuracy : 0.9471
                 95% CI : (0.9019, 0.9755)
```

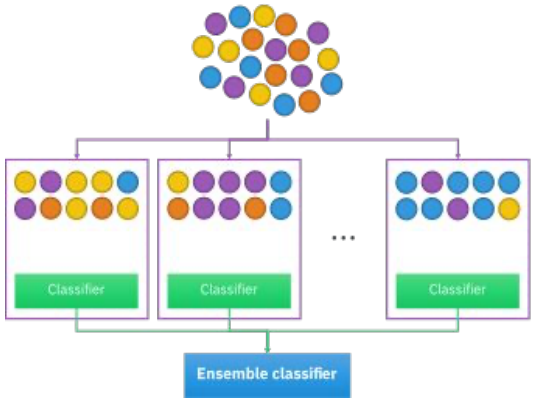
Decision trees

DMI

Weaknesses

- ✓ Overfitting: This problem gets solved by setting constraints on model parameters and pruning
- ✓ trees can be very non-robust: a small change in the data can cause a large change in the final estimated tree.
- ✓ individual decision trees generally do not often achieve state-of-the-art predictive accuracy.

Bootstrap aggregating (bagging)



Basic idea:

- Resample cases and recalculate predictions
- Average or majority vote

1. Decide how many sub-models you're going to train.
2. For each sub-model, randomly sample cases from the training set, with replacement, until you have a sample the same size as the original training set.
3. Train a sub-model on each sample of cases.
4. Pass new data through each sub-model, and let them vote on the prediction (OOB).
5. The modal prediction (the most frequent prediction) from all the sub-models is used as the predicted output.

Bootstrapping

DMI

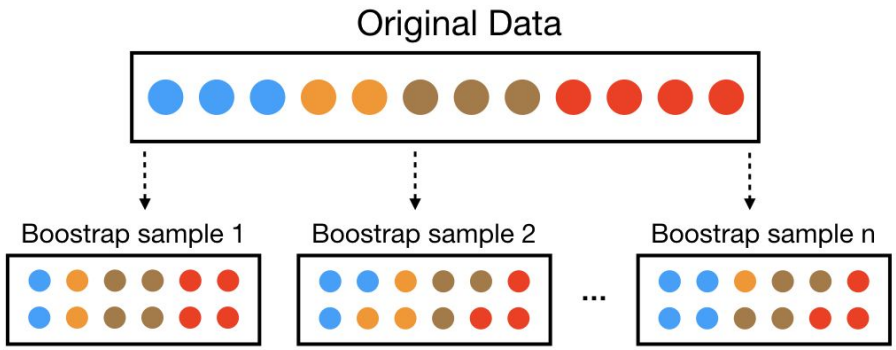
A bootstrap sample is a random sample of the data taken with replacement

Since samples are drawn with replacement, each bootstrap sample is likely to contain duplicate values.

In fact, on average, ≈ 63.21 % of the original sample ends up in any particular bootstrap sample.

The original observations not contained in a particular bootstrap sample are considered out-of-bag (OOB).

When bootstrapping, a model can be built on the selected samples and validated on the OOB samples



Bootstrap aggregating (bagging)

DMI

The general idea behind bagging is referred to as the “wisdom of the crowd” effect. It essentially means that the aggregation of information in large diverse groups results in decisions that are often better than could have been made by any single member of the group. The more diverse the group members are then the more diverse their perspectives and predictions will be, which often leads to better aggregated information. Think of estimating the number of jelly beans in a jar at a carnival. While any individual guess is likely to be way off, you’ll often find that the averaged guesses tends to be a lot closer to the true number.

Bootstrap aggregating (bagging)

DMI

Ensemble method is a technique that combines several base models in order to produce one optimal predictive model

It is a simple and powerful ensemble method.

Bagging is the application of the Bootstrap procedure to a high-variance machine learning algorithm, typically decision trees.

The idea is to create several subsets of data from the training sample chosen randomly with replacement (bootstrapping)

Each collection of subset data is used to train their decision trees.

This is repeated to create many models and every model is trained in parallel

Prediction is given based on the aggregation of predictions from all the models, which is more robust than a single decision tree.

Bagging in caret

DMI

Some models perform bagging for you, in train function consider method options

bagEarth

treebag

bagFDA

Alternatively you can bag any model you choose using the bag function

Notes:

- Often used with trees - an extension is random forests
- Several models use bagging in caret's *train* function
- Optimal performance is often found by bagging 50–500 trees.
- Data sets that have a few strong predictors typically require less trees; whereas data sets with lots of noise or multiple strong predictors may need more

Random forest

- Bootstrap samples
- At each split, bootstrap variables
- Grow multiple trees and vote or average those trees in order to get the prediction for a new outcome

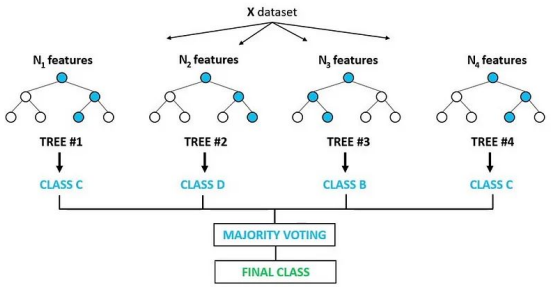
Pros:

- Accuracy

Cons:

- Speed
- Interpretability
- Overfitting

Random Forest Classifier



Random forest hyperparameters

DMI

An out-of-the-box feature or functionality (also called OOTB or off the shelf), particularly in software, is a feature or functionality of a product that works immediately after or even without any special installation without any configuration or modification

Although random forests perform well out-of-the-box, there are several tunable hyperparameters that should be considered when training a model.

1. The number of trees in the forest
2. The number of features to consider at any given split: **mtry**
3. The complexity of each tree
4. The sampling scheme

the largest impact on predictive accuracy and should always be tuned
With regression problems the default value is often mtry=p/3 and for classification mtry=$\sqrt{p}$

Random forest

DMI

- Random forests are usually one of the two top performing algorithms along with boosting in prediction contests.
- Random forests are difficult to interpret but often very accurate.
- Care should be taken to avoid overfitting

Shapley Additive exPlanations values (SHAP values), in the context of machine learning, are a way to allocate the contribution of each feature to the prediction for a specific instance in a fair and interpretable manner.

Random forest in R

DMI

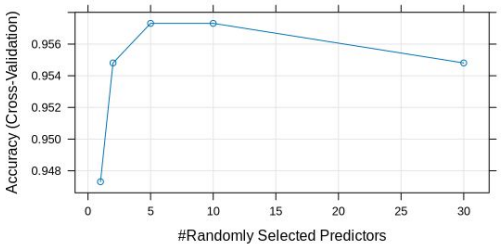
```
rf_recipe <- recipe(diagnosis ~ ., data = train_data)
ctrl <- trainControl(
 method = "cv",
 number = 10
)
p <- ncol(train_data) - 1  # exclude diagnosis
mtry_grid <- data.frame(    mtry = c(1, 2, floor(sqrt(p)), floor(p/3), p)  )
rf_model <- train(
 rf_recipe,
 data = train_data,
 method = "rf",
 trControl = ctrl,
 tuneGrid = mtry_grid,
 ntree = 500,
 importance = TRUE)
rf_model
```

```
Random Forest

399 samples
 30 predictor
  2 classes: 'B', 'M'

Recipe steps:
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 359, 359, 360, 359, 359, 359, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   1    0.9473077  0.8864213
   2    0.9548077  0.9025085
   5    0.9573077  0.9080581
  10    0.9573077  0.9089098
  30    0.9548077  0.9026489

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 5.
```

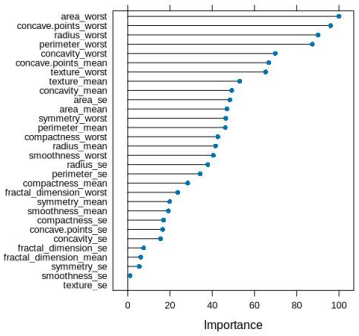
Random forest in R

DMI

```
plot(rf_model)

pred_rf <- predict(rf_model, newdata = test_data)

confusionMatrix(pred_rf, test_data$diagnosis)


varImp(rf_model)
plot(varImp(rf_model))
```




```
Confusion Matrix and Statistics

          Reference
Prediction   B   M
         B 105   3
         M   2  60

               Accuracy : 0.9706
                 95% CI : (0.9327, 0.9904)
    No Information Rate : 0.6294
    P-Value [Acc > NIR] : <2e-16
```

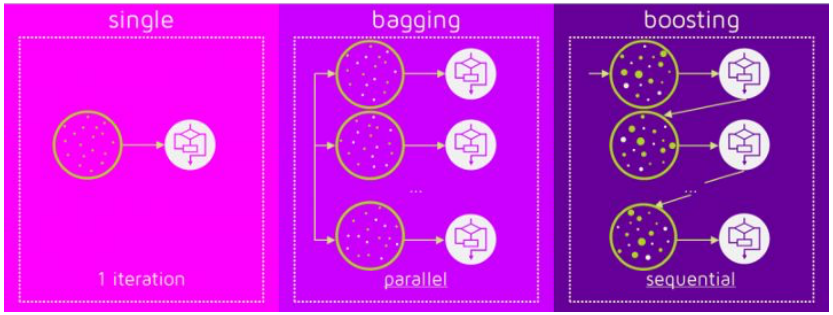


Boosting

'weak' learner (classifier, predictor) is just one which performs relatively poorly--its accuracy is above chance, but just barely.

Basic idea

1. Take lots of (possibly) weak predictors, sequentially, each trying to correct its predecessor
2. Weight them and add them up
3. Get a stronger predictor



Boosting: Adaboost

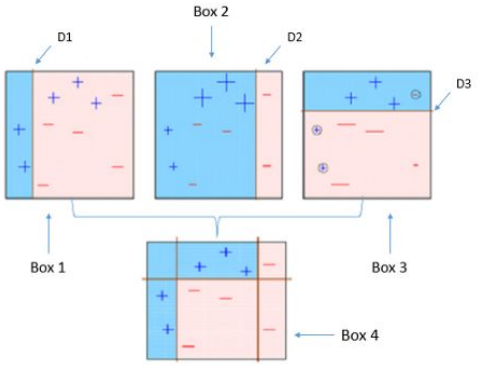
Adaboost combines multiple weak learners into a single strong learner.

The weak learners in AdaBoost are decision trees with a single split, call decision stumps.

When AdaBoost creates its first decision stump, all observations are weighted equally.

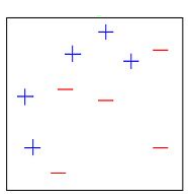
To correct the previous error, the observations that were incorrectly classified now carry more weight than the observations that were correctly classified.

AdaBoost algorithms can be used for both classification and regression problems.

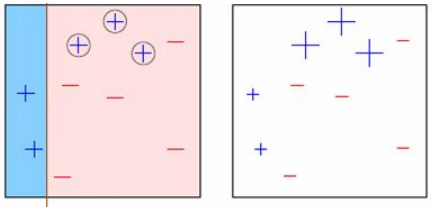


Boosting: Adaboost

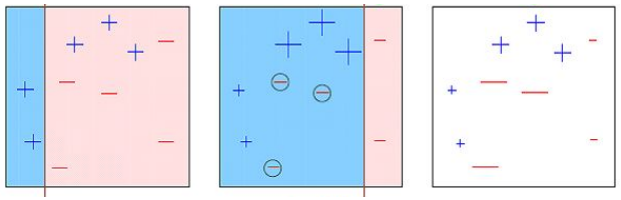
DMI



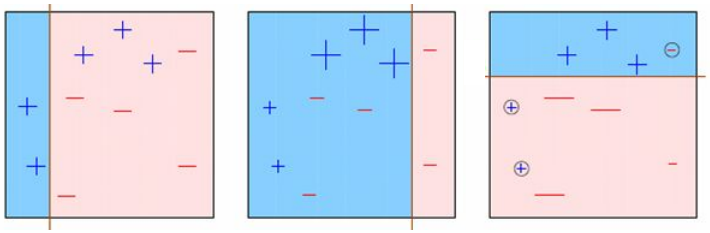
The first round:



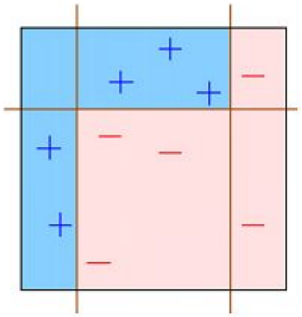
The second round:



The third round:



The final model:



$$\text{model weight} = 0.5 \times \ln\left(\frac{1 - p(\text{incorrect})}{p(\text{incorrect})}\right)$$

Boosting in R

DMI

- Boosting can be used with any subset of classifiers
- R has multiple boosting libraries. Differences include the choice of basic classification functions and combination rules.
    - gbm - boosting with trees.
    - mboost - model based boosting
    - ada - statistical boosting based on additive logistic regression
    - gamBoost for boosting generalized additive models
- Most of these are available in the caret package

Data leakage

DMI

typically where test data or future data gets mixed into the training data, and leakage in features, where something highly informative about the true label somehow gets included as a feature.

One very important cause of data leakage is performing some kind of pre-processing on the entire dataset whose results influence what is seen during training. This can include such scenarios as computing parameters for normalizing and rescaling or finding minimum and maximum feature values to detect and remove outliers and using the distribution of a variable across the entire dataset to estimate missing values in the training set, or perform feature selection.

Data leakage

DMI

practices that lead to data leakage

- Using the entire data set for Imputations. Always do imputation based on your training set only
- Using the entire data set for discretizations or normalizations/scaling or many other data-based transformations
- Using the entire data set for Feature Selection

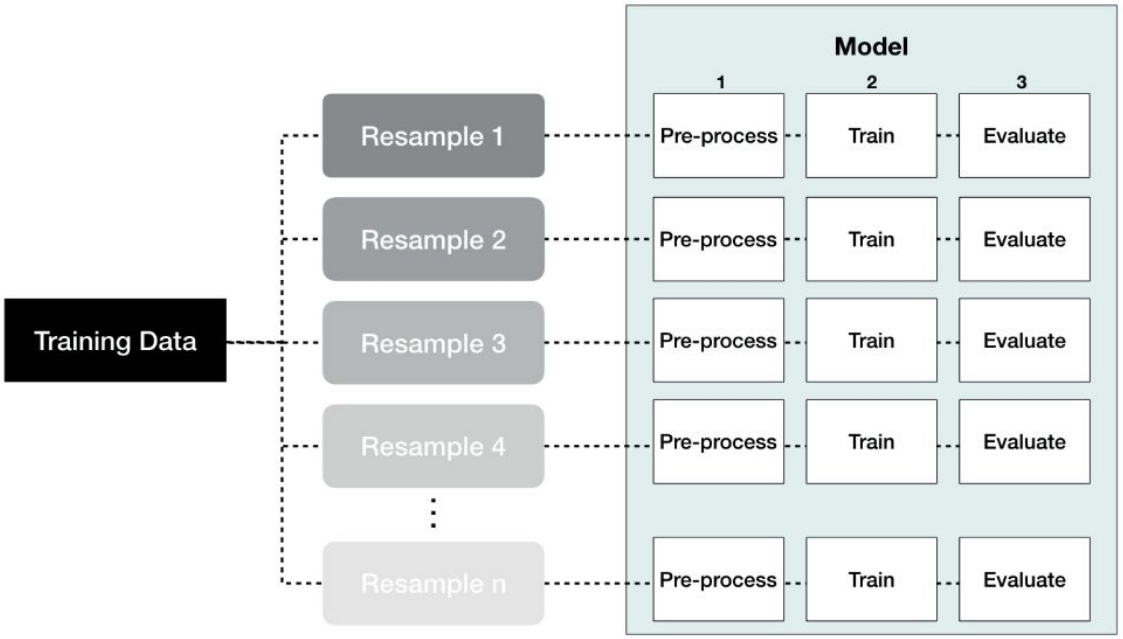
Data leakage

DMI

practices to help reduce the chance of data leakage

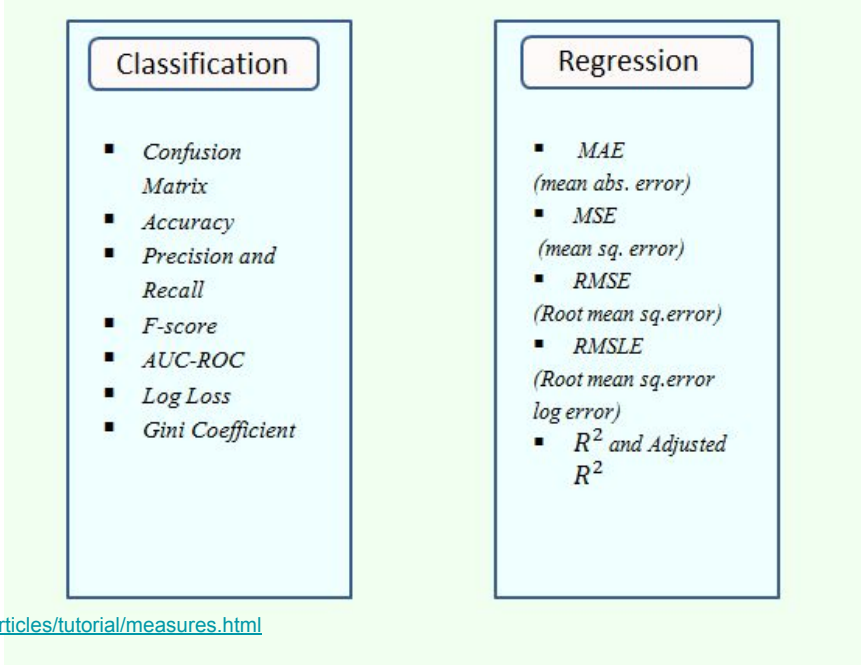
- perform any data preparation within each cross-validation fold, separately.
- scaling or normalizing features, any statistics or parameters that you estimate for this should only be based on the data available in the cross-validation split and not the entire data set.
- You should also make sure that you use these same parameters on the corresponding held out test fold.
- consider splitting off a completely separate test set before you even start working with a new dataset, and then evaluating your final model and this test data only as a very last step.

Preventing data leakage



Performance Metrics

DMI



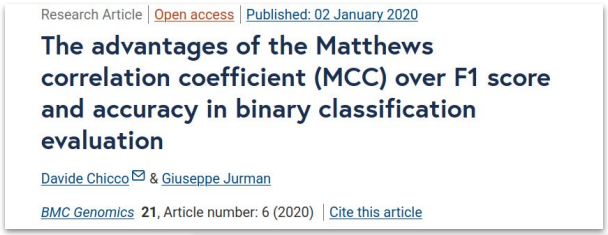
https://mlr.mlr-org.com/articles/tutorial/measures.html

Performance Metrics: The F1 score

DMI

$$F1 = \frac{2}{recall^{-1} + precision^{-1}} = 2\frac{precision*recall}{precision+recall} = \frac{2TP}{2TP+FP+FN}$$

Performance Metrics

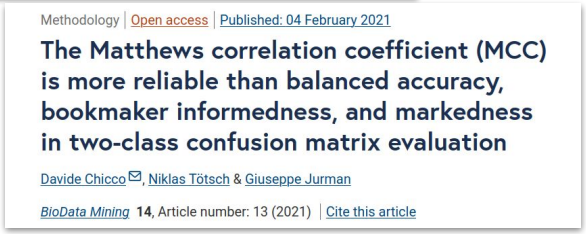


Research Article | Open access | Published: 02 January 2020

The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation

Davide Chicco & Giuseppe Jurman

BMC Genomics **21**, Article number: 6 (2020) | Cite this article

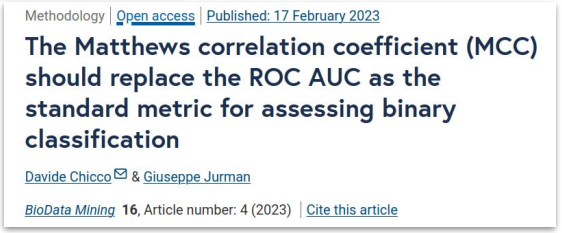


Methodology | Open access | Published: 04 February 2021

The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation

Davide Chicco, Niklas Tötsch & Giuseppe Jurman

BioData Mining **14**, Article number: 13 (2021) | Cite this article



Methodology | Open access | Published: 17 February 2023

The Matthews correlation coefficient (MCC) should replace the ROC AUC as the standard metric for assessing binary classification

Davide Chicco & Giuseppe Jurman

BioData Mining **16**, Article number: 4 (2023) | Cite this article

> [MCC] takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes [Wikipedia]

Performance Metrics

DMI

Matthews Correlation Coefficient (MCC)

$$\mathrm{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

$$F1 = \frac{2TP}{2TP + FP + FN}$$

Some final remarks

DMI

What algorithm should you use on your dataset? This is the most common question in applied machine learning. It's a question that can only be answered by trial and error. Evaluate a diverse set of algorithms on your dataset and see what works and drop what doesn't. (**No free lunch!**)

We cannot know beforehand the best algorithm representation or learning algorithm for that representation to use. We don't even know the best parameters to use for algorithms that we could try.

One way that you could choose an algorithm for a problem is to rely on experience. This could be your experience with working on similar problems in the past. It could also be the collective experience of the field where you refer to papers, books and other resources for similar problems to get an idea of what algorithms have worked well in the past. This is a good start, but this should not be where you stop.

Supervised Learning Checklist

DMI

Did you identify in advance your error measure?

Did you immediately split your data into training and validation?

Did you use cross validation, resampling, or bootstrapping only on the training data?

Did you create features using only the training data?

Did you estimate parameters only on the training data?

Did you fix all features, parameters, and models before applying to the validation data?

Did you apply only one final model to the validation data and report the error rate?

Ten quick tips for machine learning in computational biology

DMI

Tip 1: Check and arrange your input dataset properly

Each dataset is unique. Indeed, each dataset has domain-specific features, contains data strictly related to its scientific area, and might contain mistaken values hardly noticeable by inexperienced researchers.

Tip 2: Split your input dataset into three independent subsets (training set, validation set, test set), and use the test set only once you complete training and optimization phases

A common suggested ratio would be 50% for the training set, 30% for the validation set, and 20% for the test set.Tip 3: Frame your biological problem into the right algorithm category

Tip 4: Which algorithm should you choose to start? The simplest one!

Tip 5: Take care of the imbalanced data problem

Tip 6: Optimize each hyper-parameter

Tip 7: Minimize overfitting

Tip 8: Evaluate properly your algorithm performance*

Tip 9: Program your software with open source code and platforms (such as R, Python, and Weka)

Tip 10: Ask for feedback and help to computer science experts, or to collaborative Q&A online communities

One neuron versus deep learning in aftershock prediction

DMI

ARISING FROM P. M. R. DeVries et al. *Nature* https://doi.org/10.1038/s41586-018-0438-y (2018)

Forecasting the spatial distribution of aftershocks in the aftermath of large seismic events is of great importance for improving both our understanding of earthquake triggering and post-disaster management. Recently, DeVries et al.[1] attempted to solve this scientific problem by deep learning. Using the area under the curve (AUC) of receiver operating characteristic curves, the authors showed that a deep neural network (DNN) considerably outperformed classical Coulomb stress. Here we first clarify that similar performances had already been obtained (by the same authors, in 2017)[2] for various scalar stress metrics, suggesting that deep learning does not actually improve prediction. Second, we reformulate the 2017 results[2] using two-parameter logistic regression (that is, one neuron) and obtain the same performance as that of the 13,451-parameter DNN. We further show that a logistic regression based on the measured distance and mainshock average slip (instead of derived stresses) performs better than the DNN. This demonstrates that so far the proposed deep learning strategy does not provide any new insight (predictive or inferential) in this domain.

Operational aftershock forecasting has been possible for decades thanks to well established empirical laws[3–5]. Spatial patterns of aftershocks are often described as a power-law decay[5–8]. Physical models based on the Coulomb stress paradigm only outperform statistical methods when considered in physical/statistical hybrids with high-quality mainshock rupture data[9]. Coulomb stress models can on their own be as performant as statistical methods when additionally including receiver fault heterogeneities[10,11]. Meade et al.[2] performed a thorough analysis of various scalar stress metrics and showed that several of them outperform classic Coulomb failure stress. We are here concerned with their follow-up article[1], which presented similar results, but via deep learning. In this study, we aim to demonstrate that while defining larger and deeper DNNs usually does not hurt model

https://doi.org/10.1038/s41586-019-1582-8

Guidelines for supervised ML in Bio

DMI

Be on the lookout for	Consequences	Recommendation(s)
Data size & quality	Data not representative of domain application.	Data size & distribution is representative of the domain. (requirement)
Appropriate partitioning, dependence between train and test data.	Unreliable or biased performance evaluation.	Independence of optimization (training) and evaluation (testing) sets. (requirement)
Class imbalance		This is especially important for meta algorithms, where independence of multiple training sets must be shown to be independent from the evaluation (testing) sets.
No access to data	Cannot check data credibility.	Release data preferably using appropriate long-term repositories, including exact splits (requirement)

https://www.nature.com/articles/s41592-021-01205-4

Guidelines for supervised ML in Bio

DMI

Be on the lookout for	Consequences	Recommendation(s)
Overfitting, underfitting and illegal parameter tuning	Over/under optimistic performance reported. Models noise or miss relevant relationships.	Clear statement that evaluation sets were not used for feature selection, pre-processing steps or parameter tuning. (requirement) Appropriate metrics to prove no over/under fitting, i.e. comparison of training and testing error. (requirement)
Imprecise parameters and protocols given.	Results are not reproducible.	Release definitions of all algorithmic hyper-parameters, parameters and optimization protocol. (requirement) Include explicit model validation techniques, such as N-fold Cross validation. (recommendation)

https://www.nature.com/articles/s41592-021-01205-4

Guidelines for supervised ML in Bio

DMI

Be on the lookout for	Consequences	Recommendation(s)
Unclear if black box or transparent model	A transparent model shows no explainable behaviour	Describe the choice of black box / interpretable model. If interpretable show examples of it doing so. (requirement).
No access to: resulting source code, trained models & data	Cannot cross compare methods, reproducibility, & check data credibility.	Release of: documented source code + models + executable + UI/webserver + software containers. (recommendation)
Execution time is impractical	Model takes too much time to produce results	Report execution time averaged across many repeats. If computationally tough compare to similar methods (recommendation)

https://www.nature.com/articles/s41592-021-01205-4

Guidelines for supervised ML in Bio

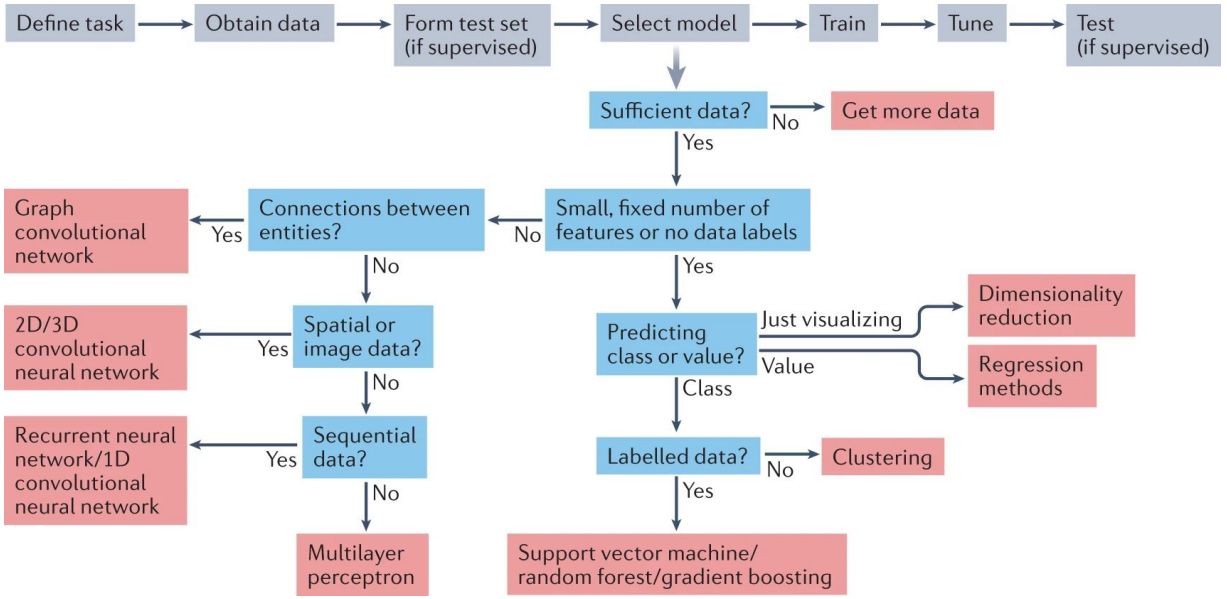
DMI

Be on the lookout for	Consequences	Recommendation(s)
Performance measures inadequate	Biased performance measures reported.	Compare with public methods & simple models (baselines). (requirement)
No comparisons to baselines or other methods	The method is falsely claimed as state-of-the-art.	Adoption of community validated measures and benchmark datasets for evaluation. (requirement)
		Comparison of related methods and alternatives on the same dataset. (recommendation)
Highly variable performance.	Unpredictable performance in production.	Evaluate performance on a final independent hold-out set. (recommendation)
		Confidence intervals/error intervals to gauge prediction robustness. (requirement)

https://www.nature.com/articles/s41592-021-01205-4

A guide to machine learning for biologists

DMI



https://www.nature.com/articles/s41580-021-00407-0

Hands on session

Go to this page and accept the assignment

Bibliography

Elements of Statistical Learning, Data Mining, Inference, and Prediction

Trevor Hastie, Robert Tibshirani, Jerome Friedman

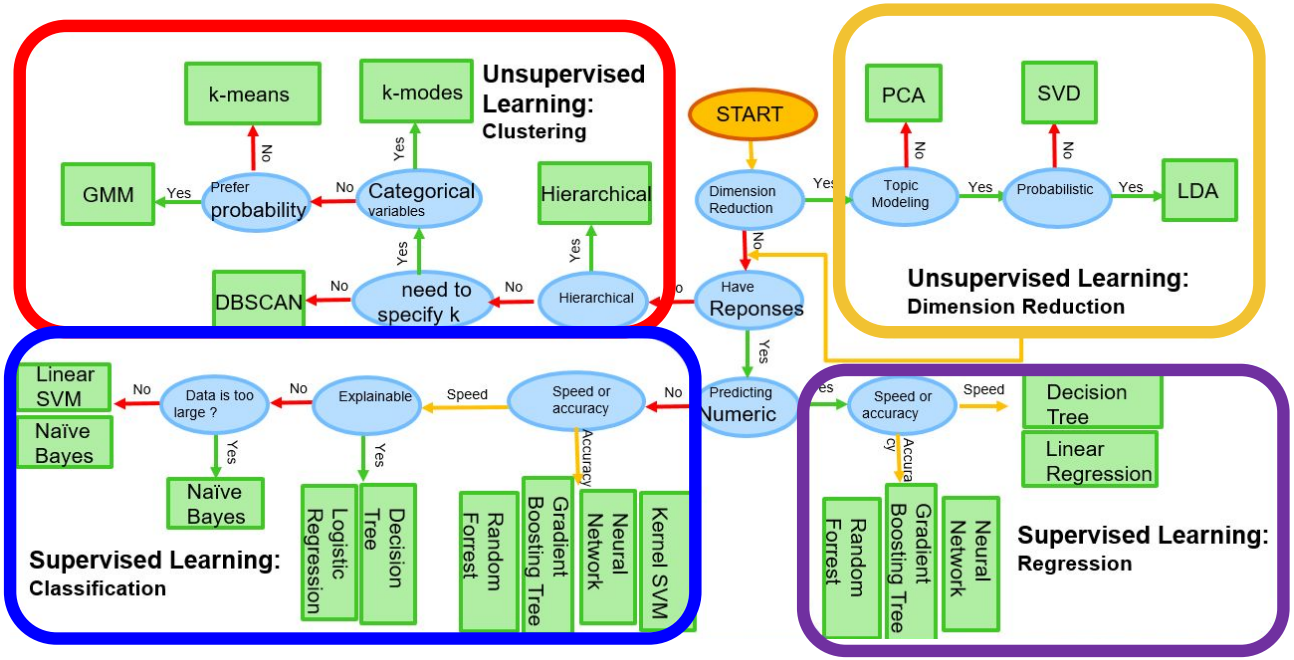
An Introduction to Statistical Learning, with Applications in R

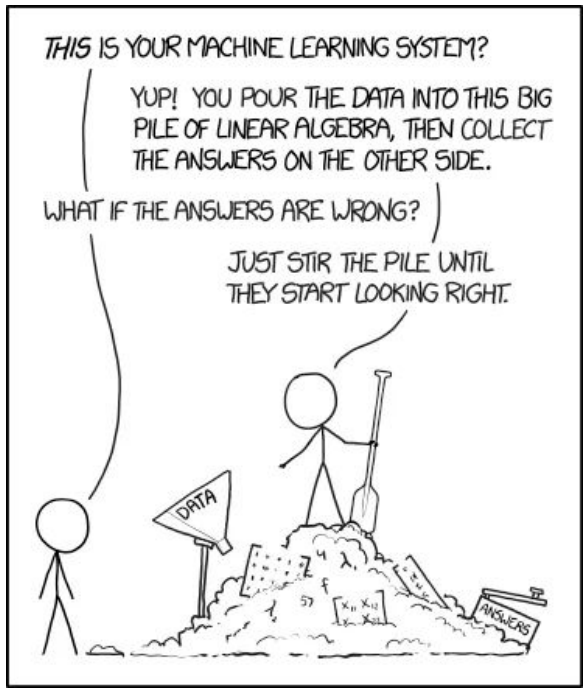
Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani

Steps to avoid overuse and misuse of machine learning in clinical research | Nature Medicine

Machine learning algorithm Cheat-sheet

DMI





https://xkcd.com/1838/