

03 – Modeling and Learning



Recap

- > What is the difference between Multi-label and Multi-Class?
- > What are steps involved in ML experiment process?
- > What is a feature function?
- > What is a Decision Tree?

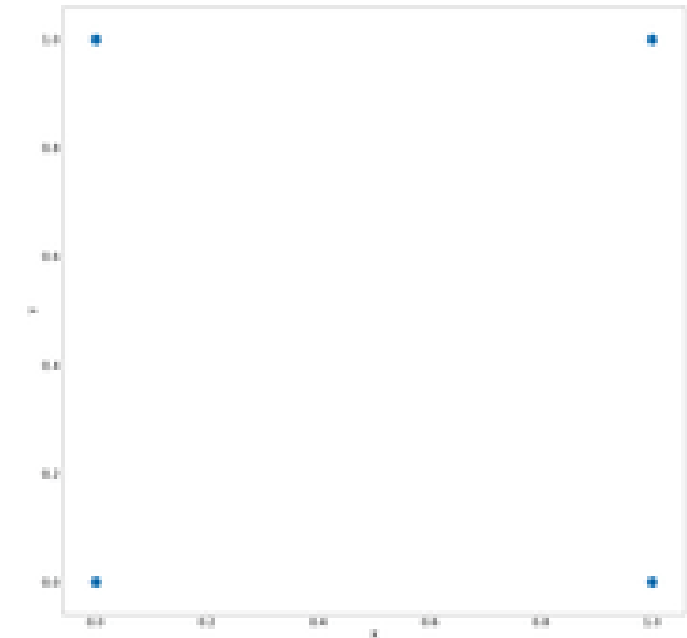
Agenda Today

- > How do Decision Trees learn?
- > Model Evaluation
- > Loss Function
- > Empirical Loss Minimization
- > Limits of Learning:
 - > Bias / Variance Tradeoff
- > Hyperparameters
- > From **Decision Trees** to **Random Forests**



Decision Tree Creation

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1



Decision Tree – Open Questions:

- > Does the model pick the *best* value?
- > Does the *order* of the features matter?
- > Is there *randomization* involved?
- > Can we do *better* than the Decision Tree?

Decision Tree Building Algorithm

> Based on the (training) data, find a good question (predictor) that discriminates (splits) the data optimally.

> How?

GINI Impurity:

$$Gini = 1 - \sum_{i=1}^n (P_i)^2 = \sum_{i=1}^n P_i(1 - P_i)$$

$$P_i = \frac{\text{\# of observations with class } i}{\text{total observations in node}}$$

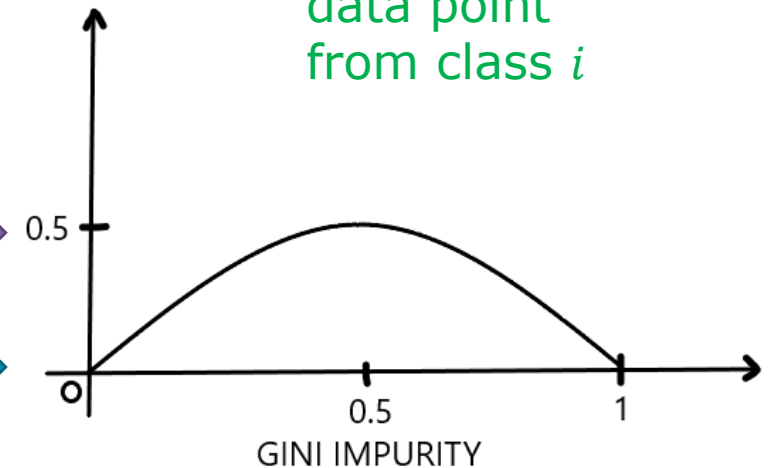
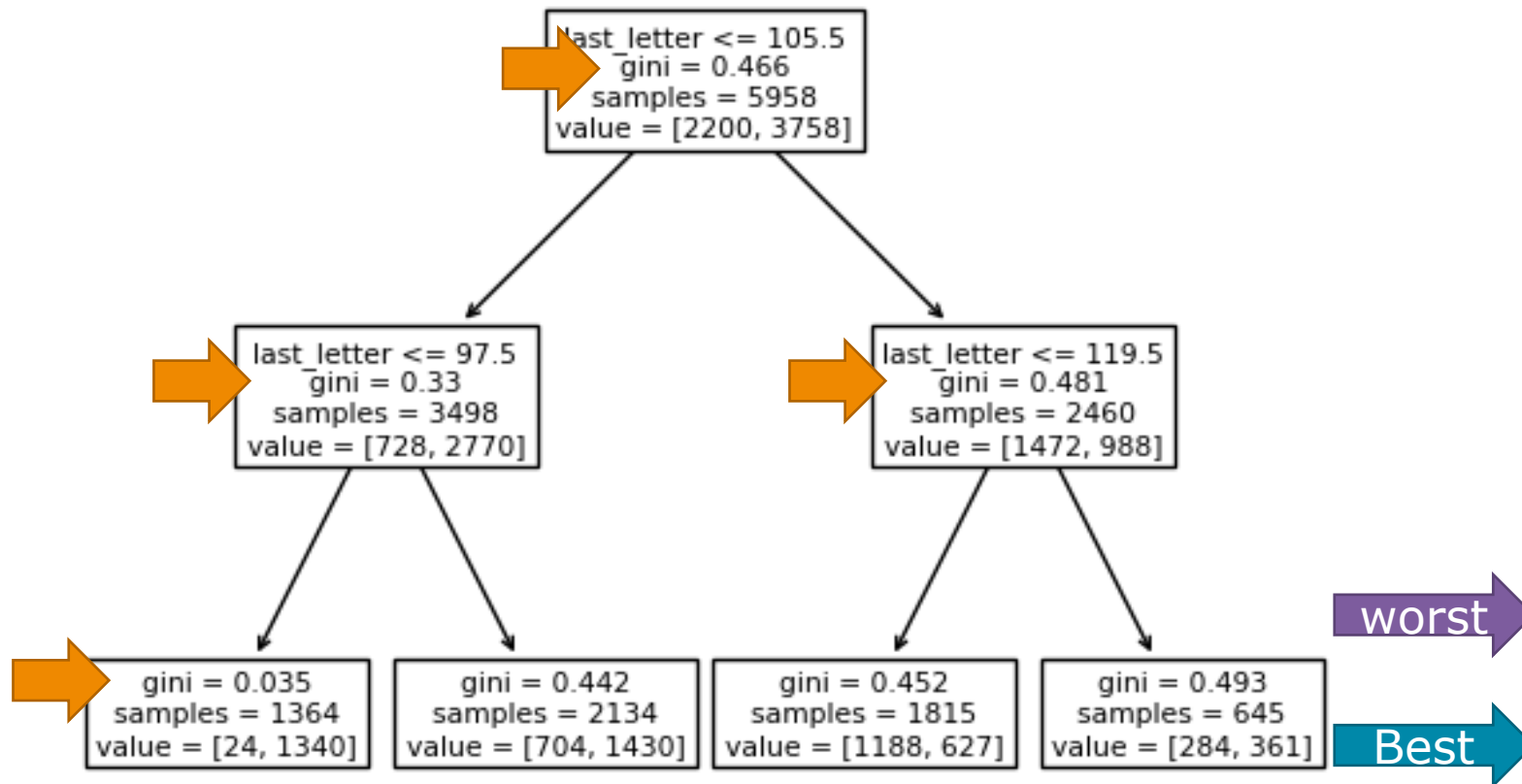
GINI Impurity

$$P_i = \frac{\text{\# of observations with class } i}{\text{total observations in node}}$$

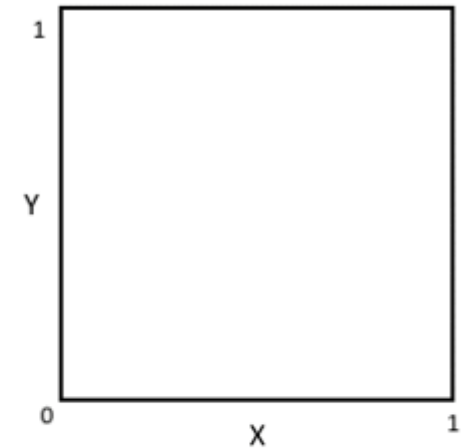
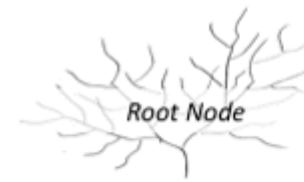
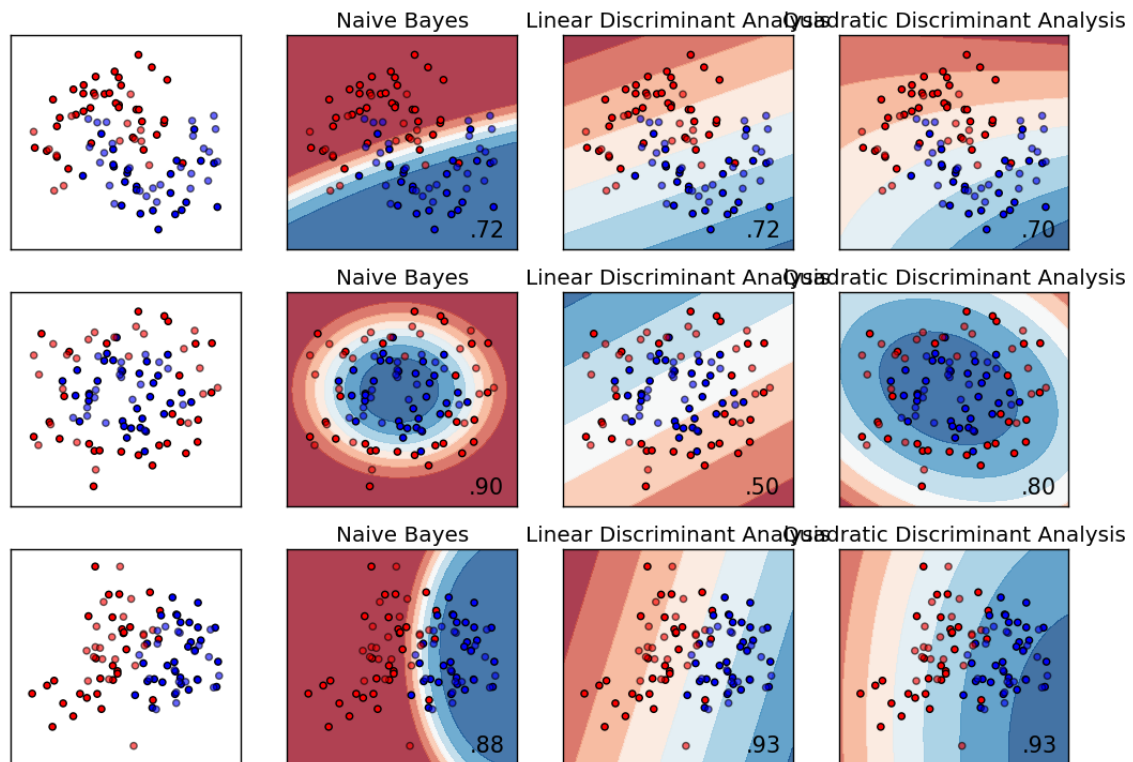
Probability of picking a data point from class i

$$\sum_{i=1}^n P_i(1 - P_i)$$

Probability of **not** picking a data point from class i

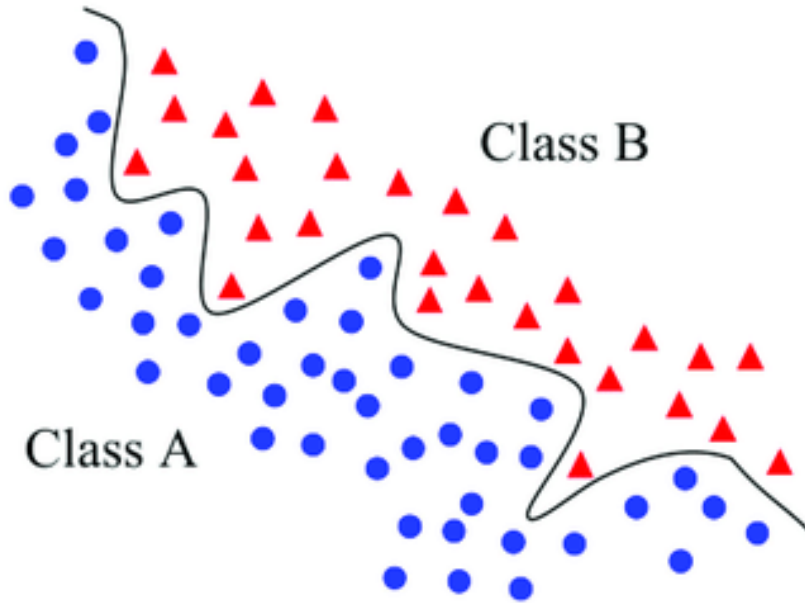


Decision Tree & Decision Boundary

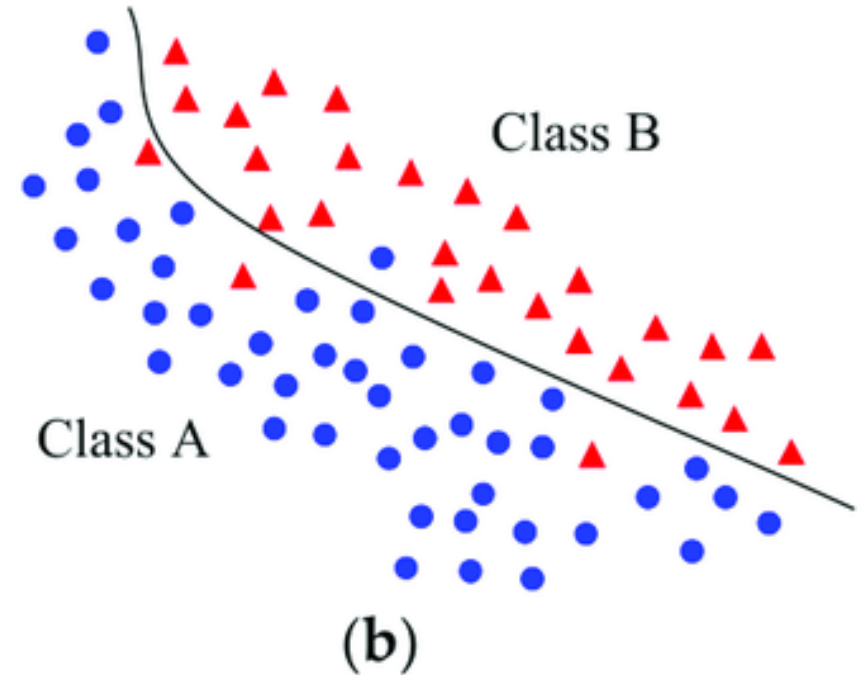


Decision Boundary - Classification

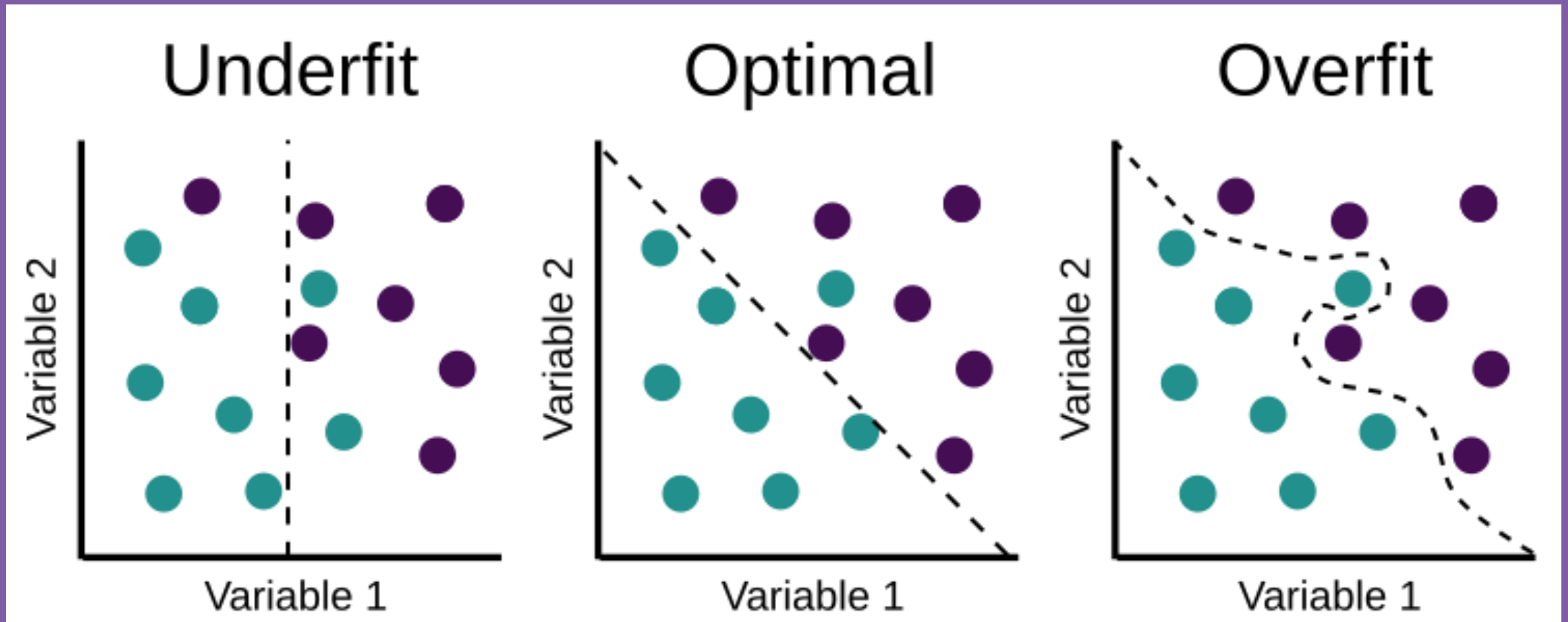
Decision boundary



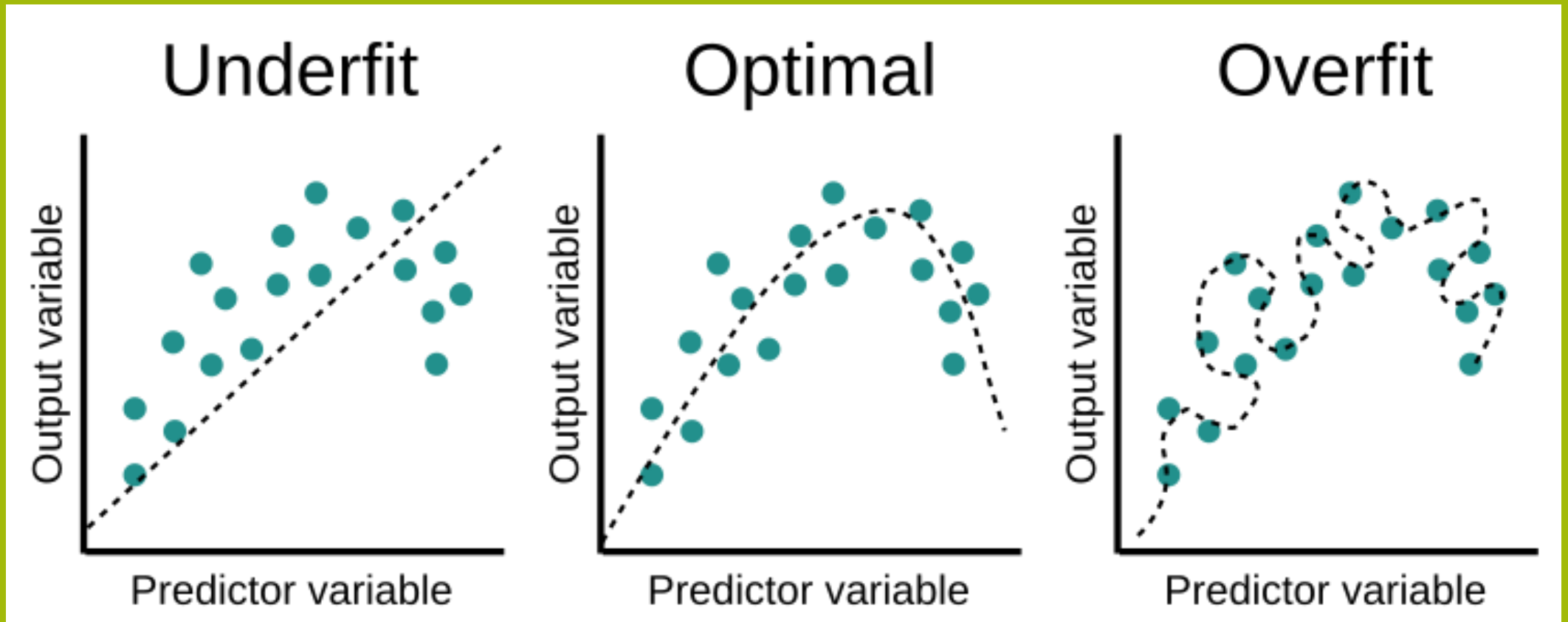
Decision boundary



Decision Boundary - Classification



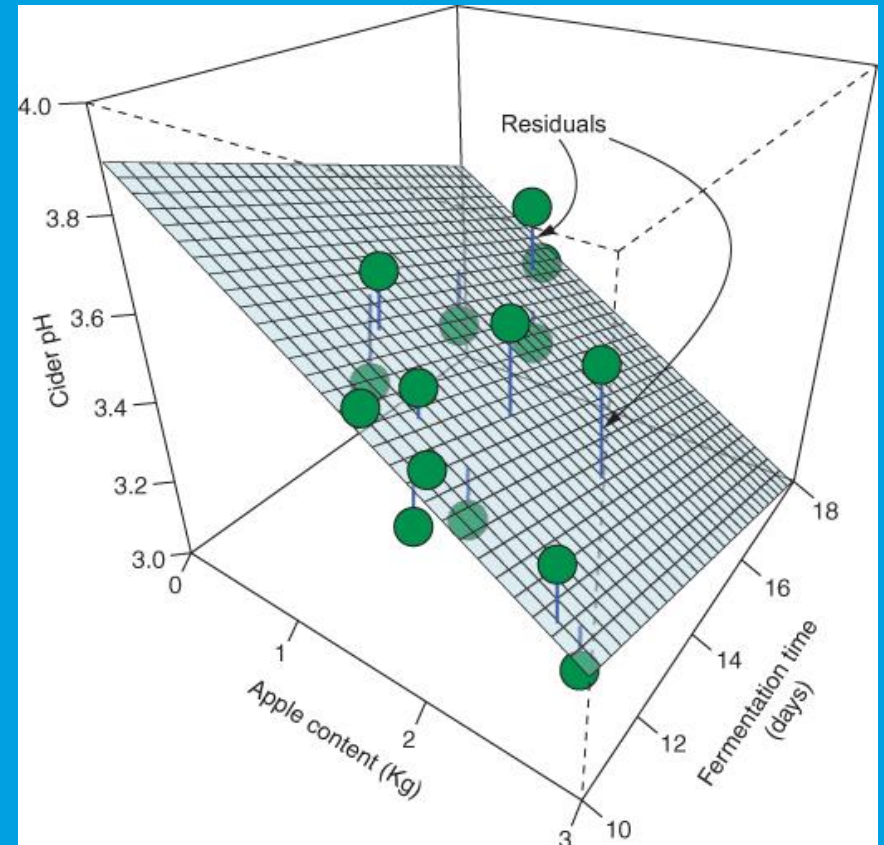
Decision Boundary - Regression



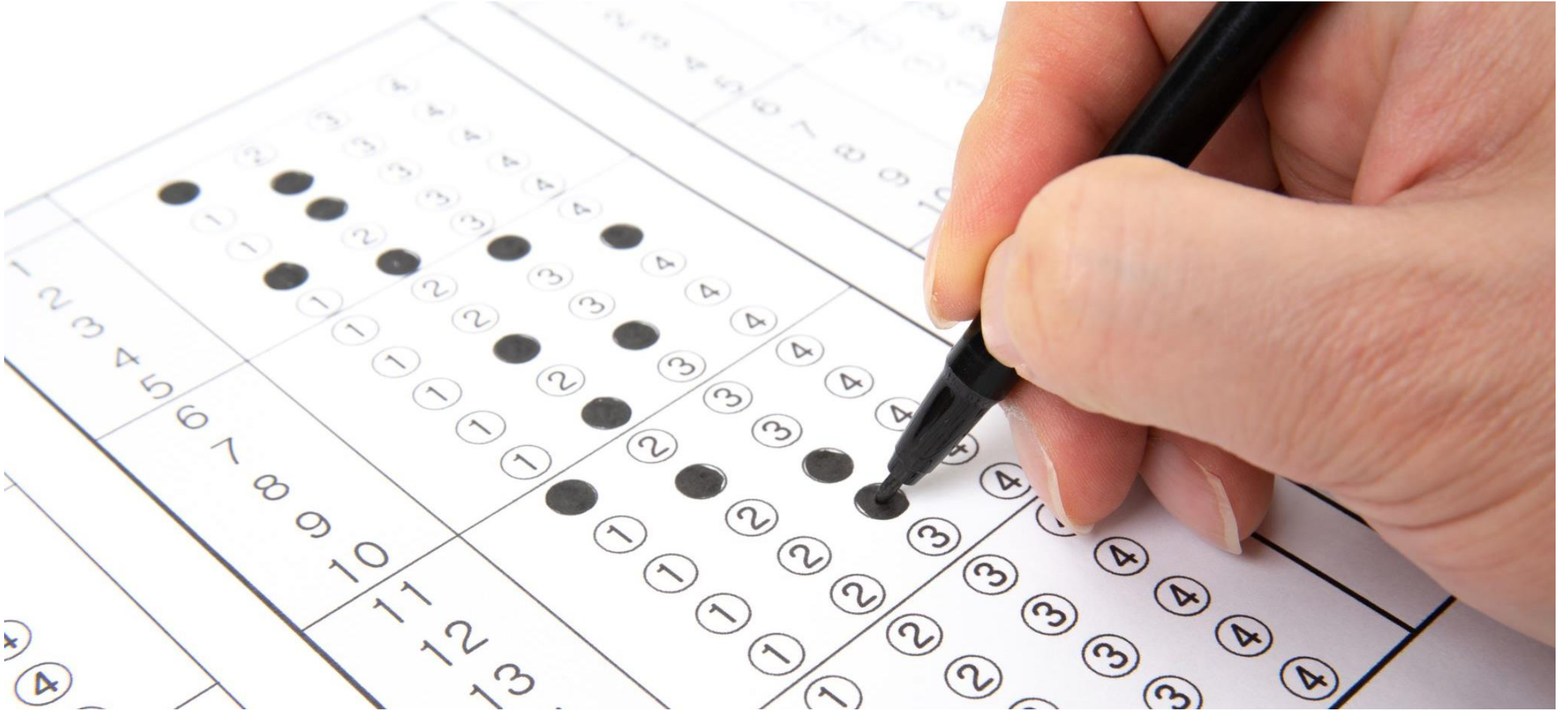
Decision Boundary in Higher Dimensions

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \dots \beta_k x_k + \epsilon$$

- Decision Boundary in higher dimensions is hard to visualize.
- We need a better way to evaluate our model.



Model Evaluation



Model Evaluation

- > The performance of the learning algorithm should be measured on unseen **"test" data**.

- > The data that our algorithm "sees" at **training** time and the one it "sees" at **test** time should be related:
 - Drawn from the same distribution.
 - (Hopefully represent the real-world data)

Model Evaluation

		Predicted Class	
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

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

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

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

$$\text{F1 Score} = \frac{2 * \text{Recall} * \text{Precision}}{\text{Recall} + \text{Precision}}$$

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

Evaluating Unbalanced Classes

Chicco and Jurman *BMC Genomics* (2020) 21:6
<https://doi.org/10.1186/s12864-019-6413-7>

BMC Genomics

RESEARCH ARTICLE

Open Access

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



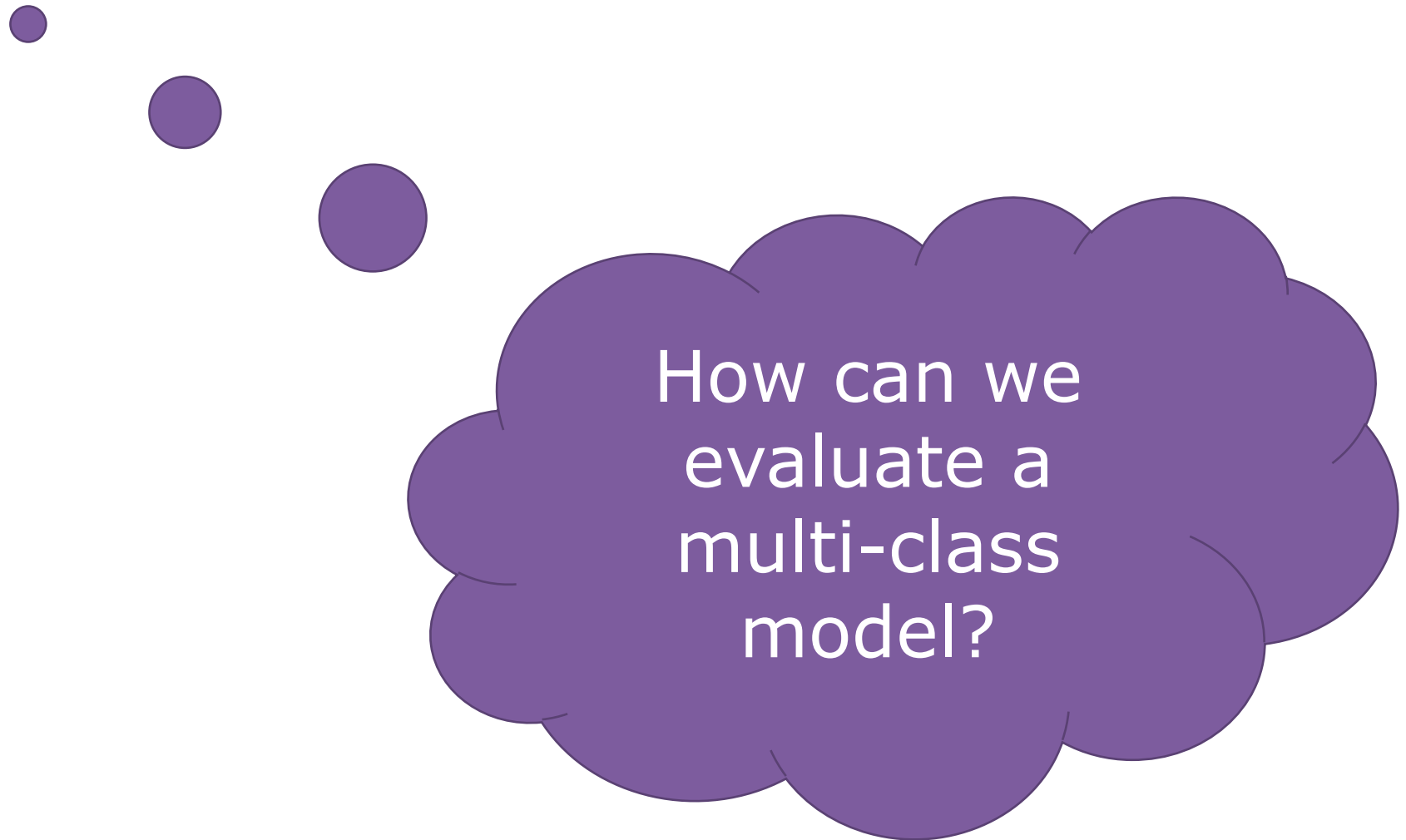
Davide Chicco^{1,2*}  and Giuseppe Jurman³ 

Evaluating Unbalanced Classes

Results: The Matthews correlation coefficient (MCC), instead, is a **more reliable statistical rate** which produces a high score only if the prediction obtained good results in all of the four confusion matrix categories (true positives, false negatives, true negatives, and false positives), proportionally both to the size of positive elements and the size of negative elements in the dataset.

Conclusions: In this article, we show how MCC produces a more informative and truthful score in evaluating binary classifications than accuracy and F_1 score, by first explaining the mathematical properties, and then the asset of MCC in six synthetic use cases and in a real genomics scenario. We believe that the **Matthews correlation coefficient should be preferred to accuracy and F_1 score in evaluating binary classification tasks by all scientific communities.**

From Binary Classification to Multi-Class



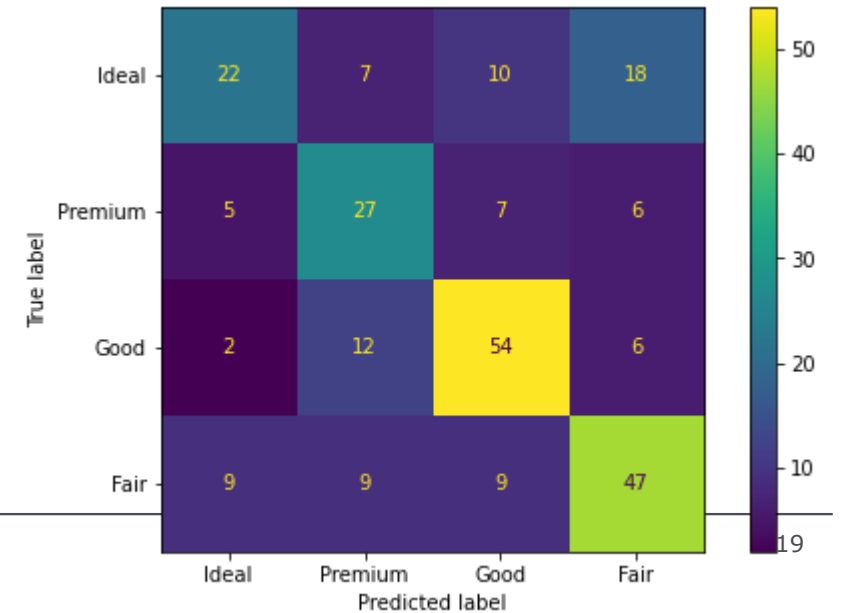
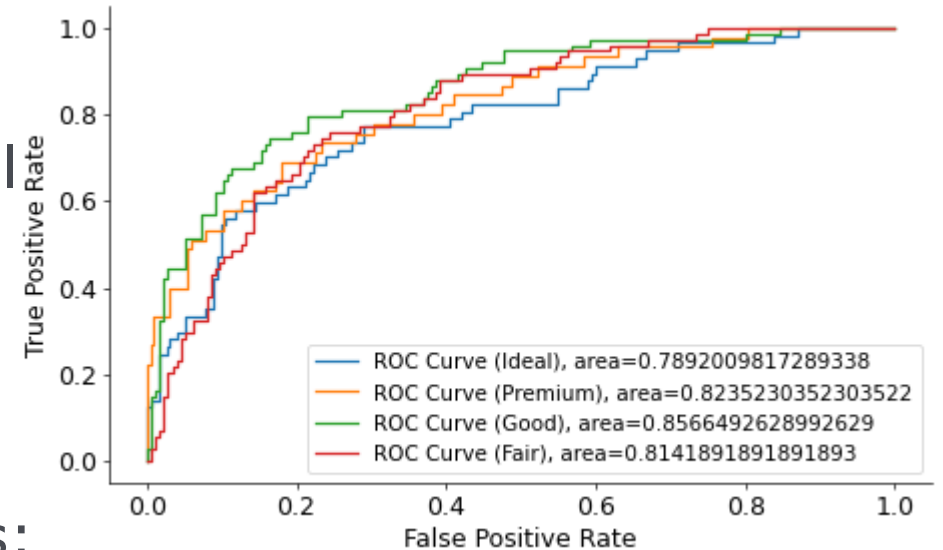
How can we
evaluate a
multi-class
model?

From Binary Classification to Multi-Class

- > Single Score:
 - > MCC / F1 / Accuracy / Precision / Recall
 - > Weighted
 - > Macro
 - > Micro
 - > Cohen Kappa Score – agreement score between the model and the true values:

$$k = \frac{p_0 - p_e}{1 - p_e}$$

- > Aggregated AUROC chart
- > Confusion Table

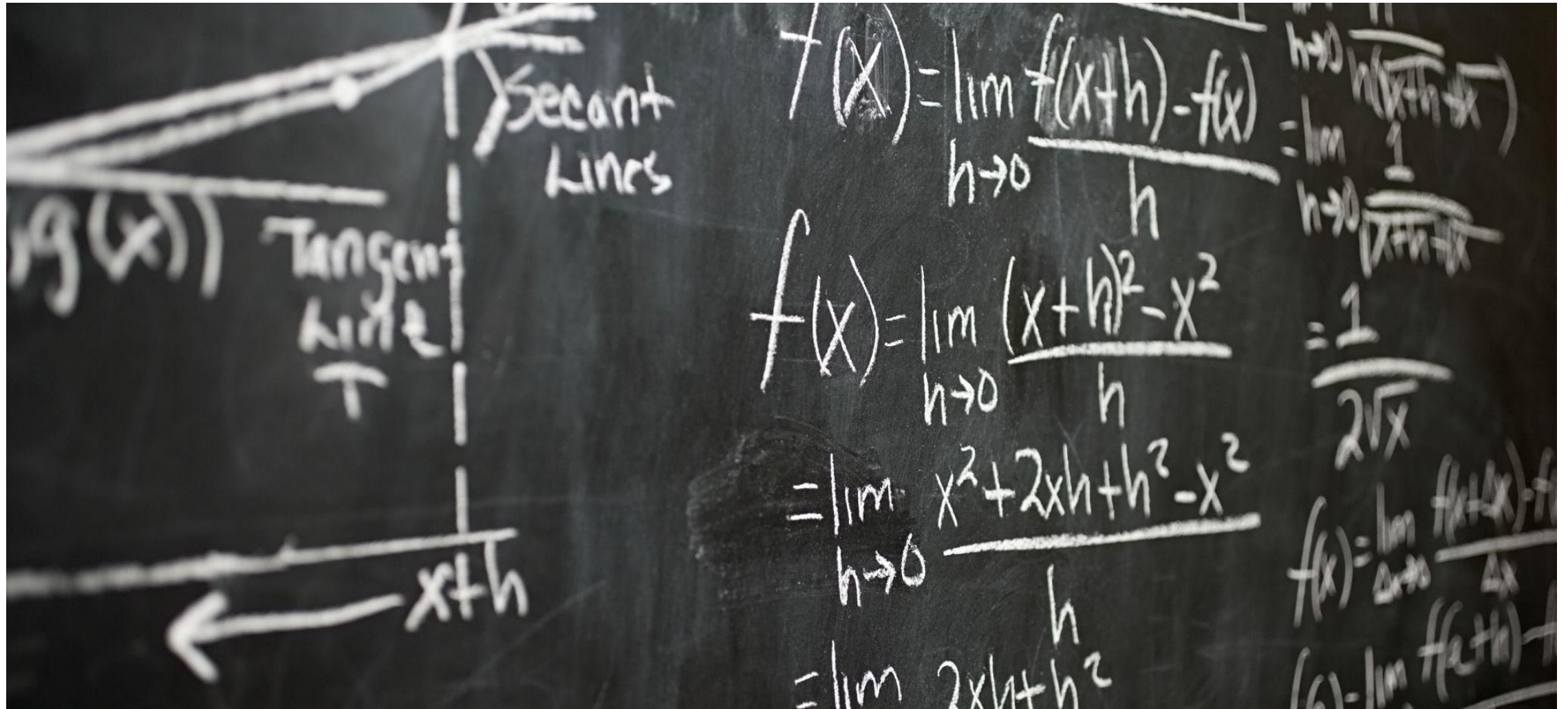


The Nightmare Before Christmas

- > [FH Campus Wien 22's: Spooky Author Identification | Kaggle](#)
- > Classify the authors of the sentences
- > Test-set is hidden!
- > Be creative with your feature functions!



Loss Functions



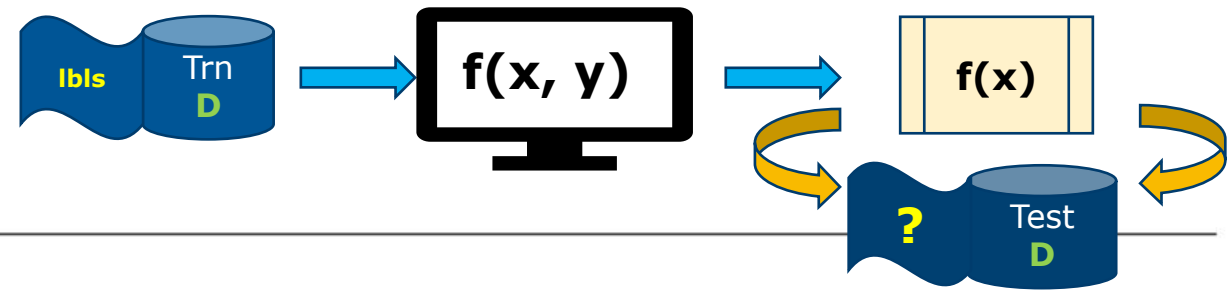
Loss Function

What was the loss function in the Decision Tree?

- > Performance measurements
model predictions vs truth values
- > How much of the truth did we lose by making predictions?
 - > **"Loss Function"** - $L(y_true, y_pred)$
- > Examples:
Squared Loss, Absolute Loss, Zero/One Loss
- > Capture the notion of what is *important* to learn
- > **We** decide which *Loss Function* to use!

Loss Function vs Evaluation Functions

	Loss Function	Evaluation Function
Examples:	Squared Loss, Absolute Loss, Zero-one Loss, Cross-Entropy Loss, KL-Divergence	Accuracy, F1-Score, MCC,
Data:	Training set	test-set (unseen data)
Stage	During training	After we obtain a model
Purpose:	Influence the learning algorithm process	Evaluate the model performance on unseen data
Direction:	Should be as low as possible	Should be high (closer to 1)



Expected Loss

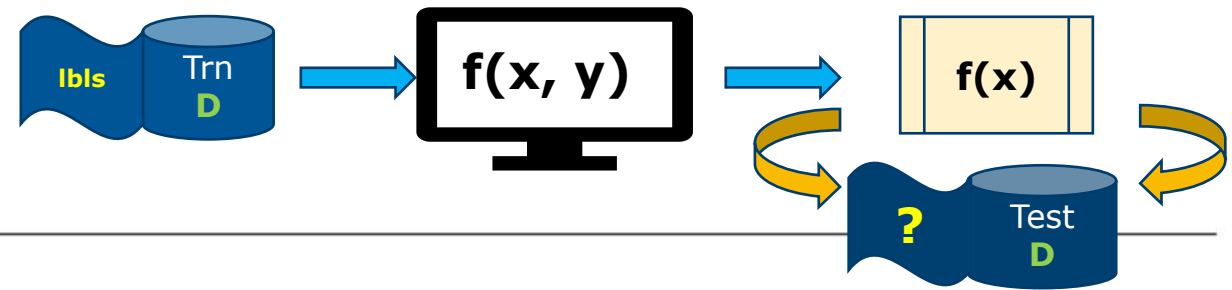
> The general **Expected Loss** of a model:

$$\epsilon \triangleq \mathbb{E}_{(x,y) \sim \mathcal{D}} [\ell(y, f(x))] = \sum_{(x,y)} \mathcal{D}(x,y) \ell(y, f(x))$$

- > We aspire to minimize the loss as much as possible.
- > But – we only have a **sample** of our Distribution (\mathbb{D})
 - > We don't really know what the true \mathbb{D} is...

Notes:

- Equivalent to the **estimated standard error**
- Loss is also known as **Risk**



Empirical Risk minimization

> The general **Expected Loss** of a model:

$$\epsilon \triangleq \mathbb{E}_{(x,y) \sim \mathcal{D}} [\ell(y, f(x))] = \sum_{(x,y)} \mathcal{D}(x,y) \ell(y, f(x))$$

> We aspire to minimize the loss of the training samples.

Training error: $\hat{\epsilon} \triangleq \frac{1}{N} \sum_{n=1}^N \ell(y_n, f(x_n))$

Training Error • • •

Is it possible to get no-training-error-at-all?

What happens if we minimize the training error to 0?

Yes. It is **overfitting**:

The model memorizes the information "by heart" - it won't generalize on new, unseen data.

Formally: the true objective is to minimize the *Expected Loss*, not the *Training Loss*

Machine Learning – A Formal Definition

Given:

- > Sample data pairs $(x, y) \in X \times Y$ taken from an unknown distribution \mathbb{D} , and
- > a loss function l

We want to learn a mapping function f :

$$f(x) = \operatorname{argmax} \mathbb{D}(x, y)$$

So that $l(f(x), y)$ has the lowest error (minimized risk)

Limits Of Learning



Machine Learning – A Formal Definition

Given:

- > A **limited** sample data pairs $(x, y) \in X \times Y$ taken from an unknown distribution \mathbb{D} , and
- > a loss function l

We want to learn a mapping function f :

$$f(x) = \operatorname{argmax}_{\mathbb{D}}(x, y)$$

So that $l(f(x), y)$ has the lowest error (minimized risk)

Is Limited Sample Enough?

- > **Law of Large Numbers**

- > Example: *al dente pasta*

- > *Formally:*

$$v_1, v_2, \dots, v_n \in V$$

- > V contains independent variables drawn from the same distribution

$$\hat{v} = \frac{1}{N} \sum_{n=1}^N v_n$$
$$N \rightarrow \infty \Rightarrow \hat{v} \rightarrow \mathbb{E}[v]$$

Problems Learning with a Limited Sample Set

- > Inductive Bias
- > Noisy Data
- > Underfitting
- > Overfitting

Inductive Bias

> A **preference** for a certain distinction over the other

Inductive Bias

Class A



Class B



A or B?

- > Fly vs Non-Fly?
- > Bird vs Non-Bird?



A or B?

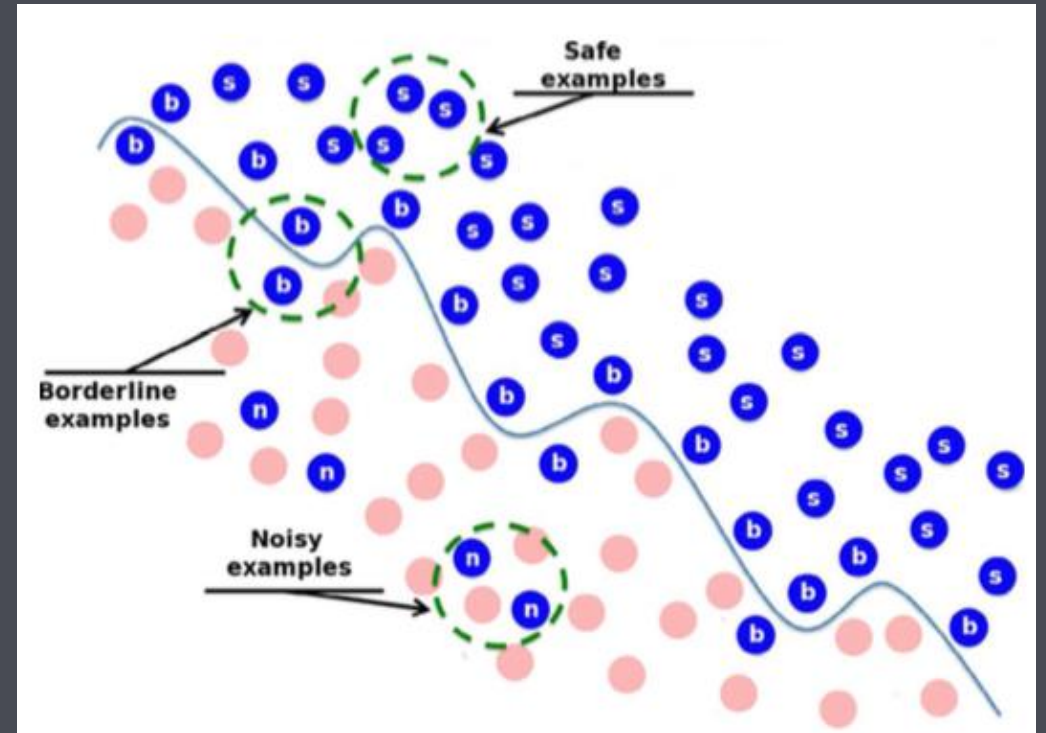
- > Fly vs Non-Fly? – 35%
- > Bird vs Non-Bird? – 65%

(Human)
**Inductive
Bias**

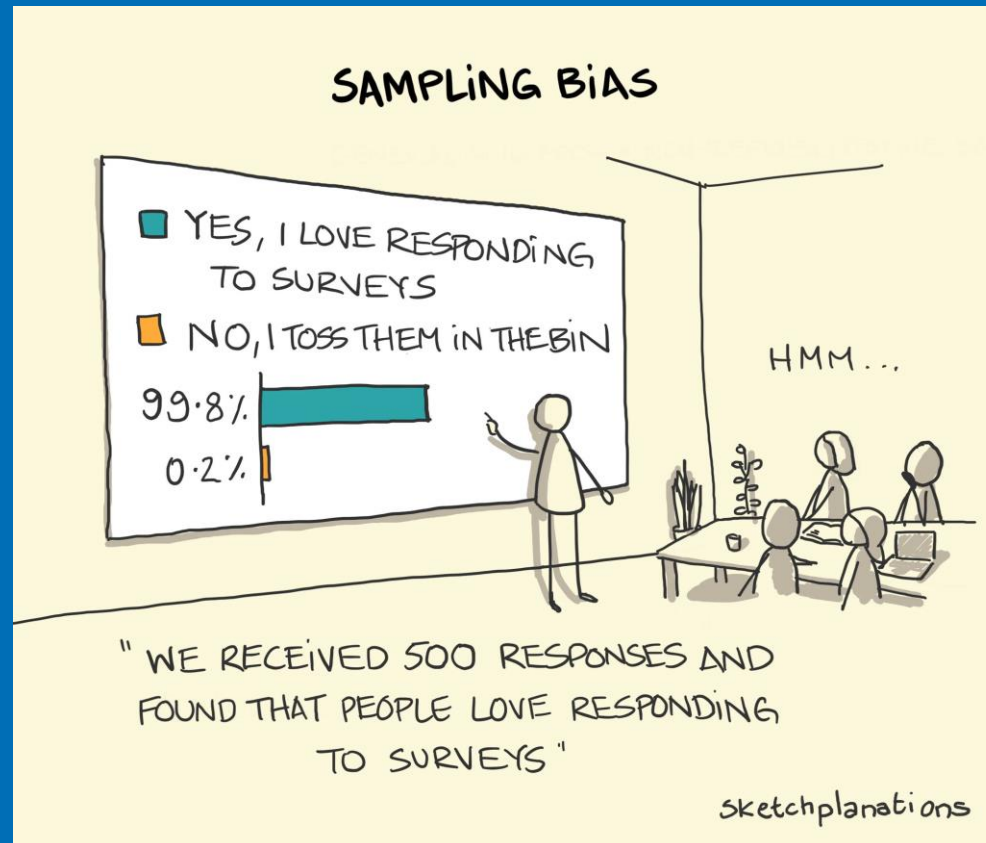


Noisy Data

- > Mislabeling
 - Medical Diagnosis
 - Inconclusive labels
- > Errors in the data
 - Typos
 - Missing points / Nulls
 - Redundant data

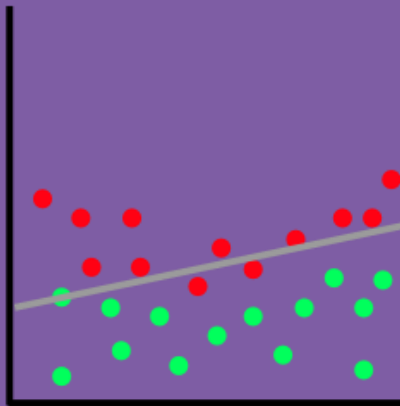


Sampling Bias

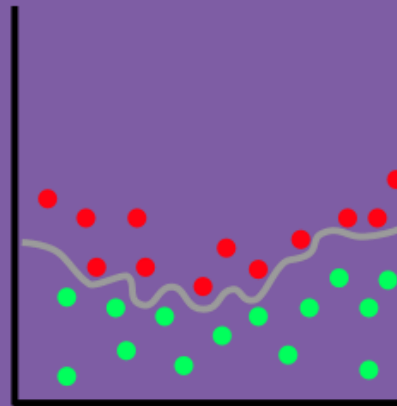


Underfitting

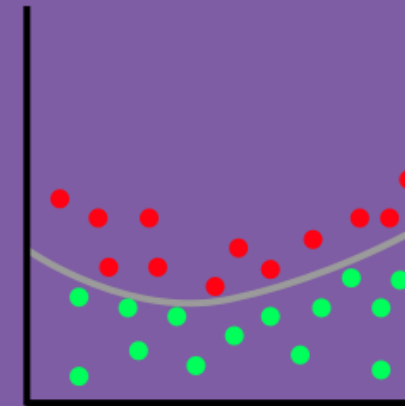
- > The model didn't learn (well) enough
- > Poor predictions
Careful: An imbalance dataset with 95% - 5%.
A naïve classifier achieves 95% accuracy.



Underfitting



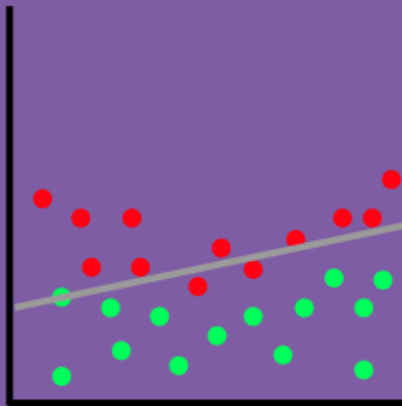
Overfitting



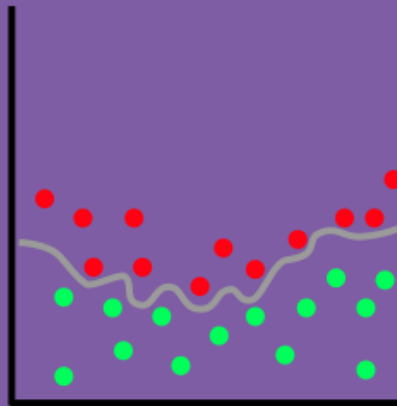
Balanced

Overfitting

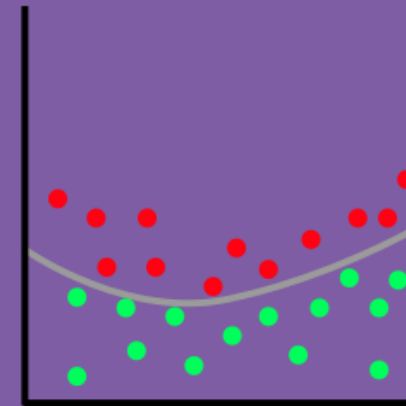
- > Fitting exactly to the training data
 - > Memorizing the learned data...
Hence failing on an understanding (generalization) exam
- > A Loss-score of nearly 0



Underfitting



Overfitting



Balanced

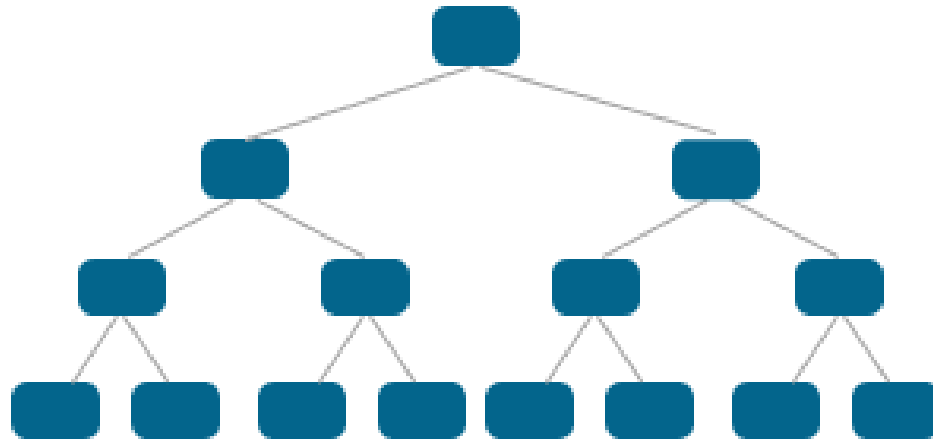
Modeling

- > Machine Learning – learns a formal *model* of our data
- > Using the model, we analyze what we can learn and what our inductive bias is.
- > Models contains **parameters**
- > **Training** = gradually adapting the model parameters to the data
- > *Example*: Decision-Trees update the nodes' yes/no questions
- > *Example*: Linear Regression updates the *slope* and the *intercept*

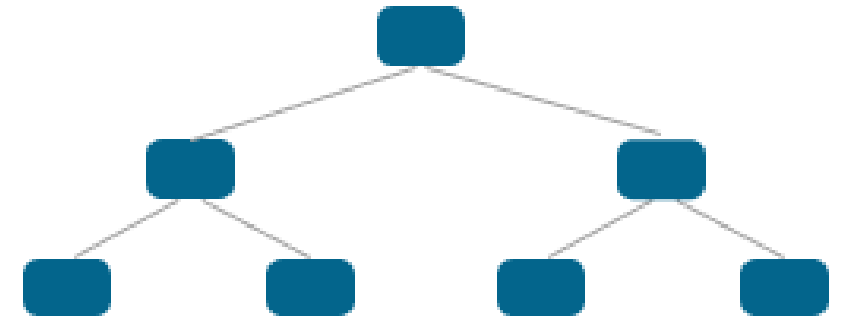
Loss Evaluation (Recap)

- > Overfitting can bring the loss to 0.
- > Solution: testing models on a **test-set**.
- > Simulates 'unseen' data
- > Must be prepared in advance
- > Different models should be compared using the same test-set.
- > Should *never* be used for training
- > *Best practice*: never even peek at it

High Variance / Bias in Decision Trees



High variance
Overfitting



High bias
Underfitting

Detecting High Variance / Bias

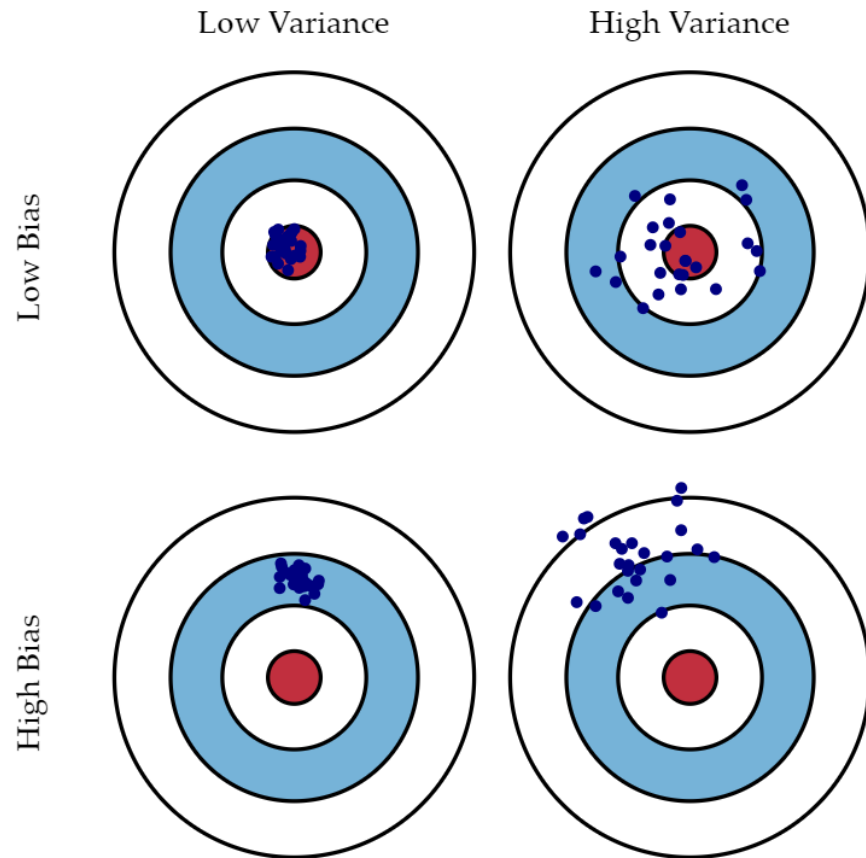
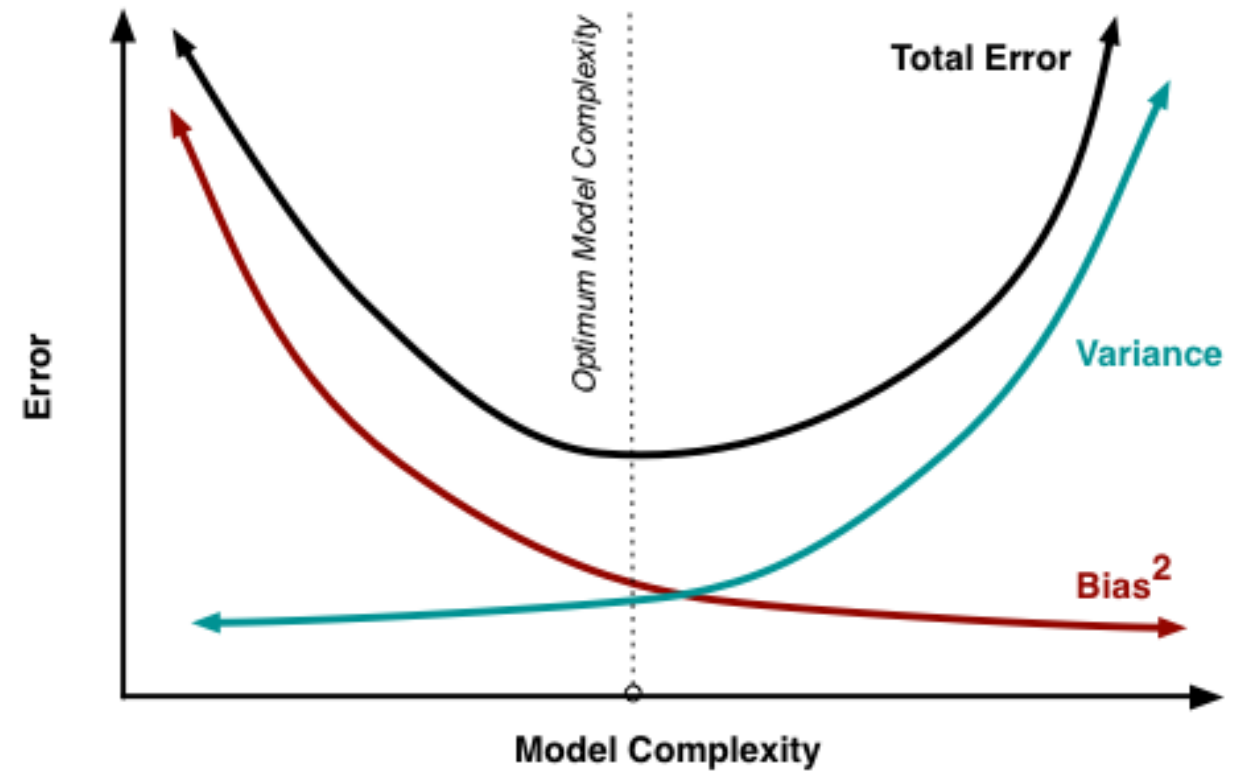


Fig. 1 Graphical illustration of bias and variance.



[Understanding the Bias-Variance Tradeoff \(fortmann-roe.com\)](http://fortmann-roe.com)

So, what can we do?



So, what can we do?

- > Collect more (labeled) data
- > Hyperparameters Tuning
- > Better models

So, what can we do?

- > Collect more (labeled) data
 - > Not always possible
 - > Expensive (requires time / labor)
- > Hyperparameters Tuning
- > Better models

So, what can we do?

- > Collect more (labeled) data
- > **Hyperparameters Tuning**
- > Better models

Hyperparameters

- > Knobs to adjust the learning process, to minimize the error
- > Tunes the inductive bias of the learning algorithm
- > *Example:* `max_depth`, `criterion` (`DecisionTreeClassifier`)
- > *Example:* `penalty` (`LogisticRegression`)

Hyperparameters Tuning

How can we find the optimal values that yield the best model for the given data?

Hyperparameter tuning vs. model training

Hyperparameter
tuning



Best
hyperparameters

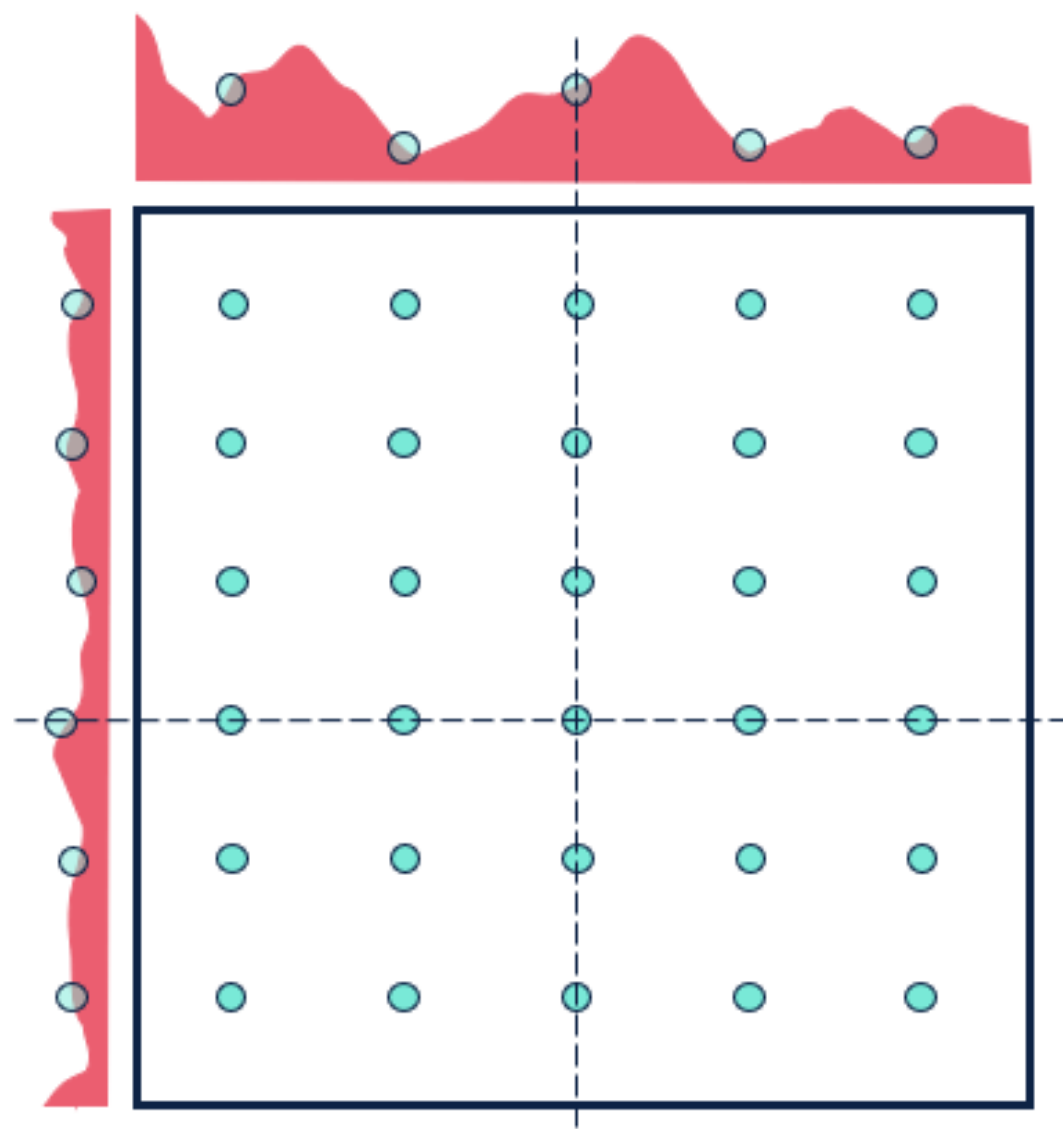
Model
training



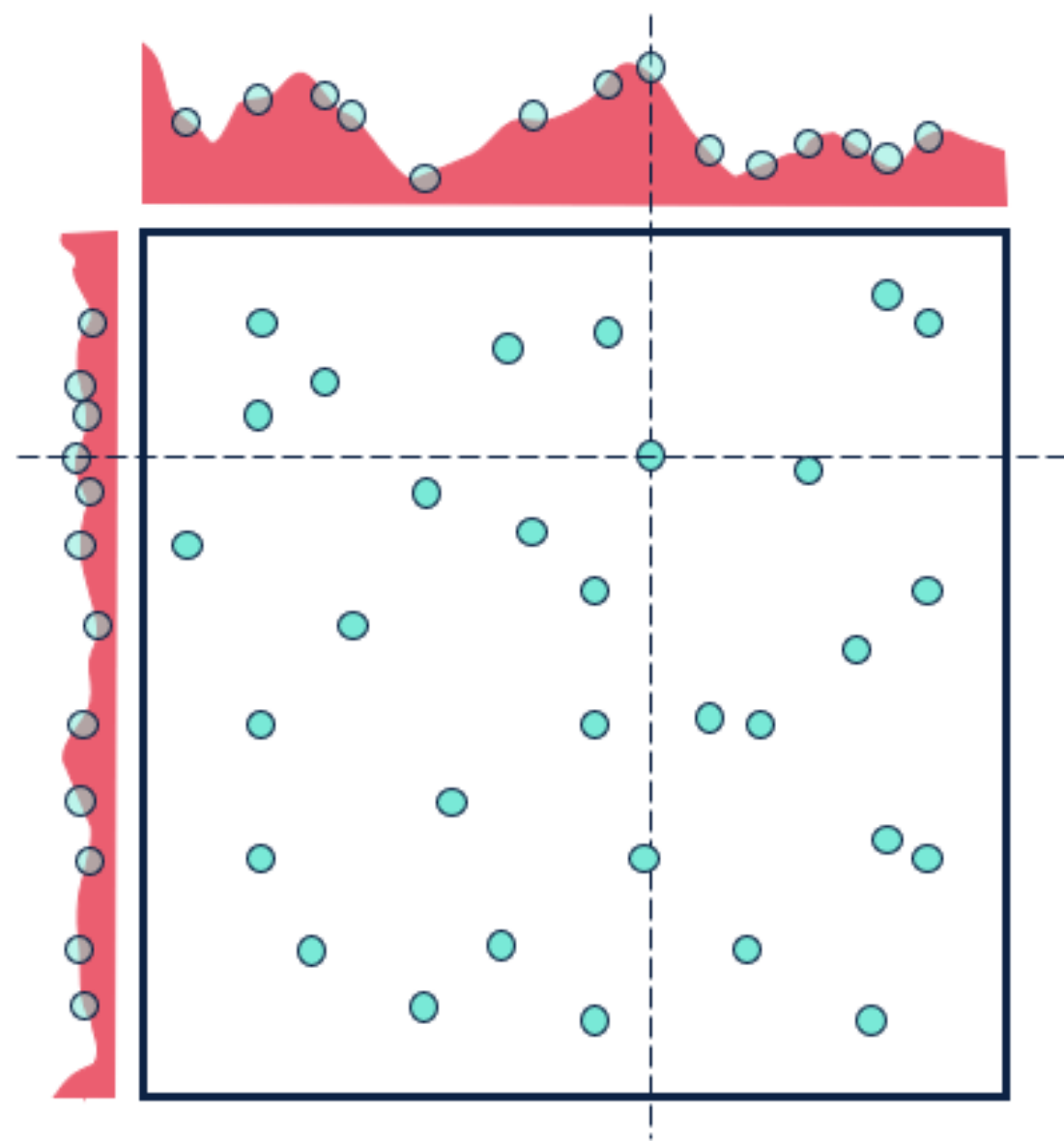
Best model
parameters

Hyperparameters Tuning

- > Manual Search
- > Random Search
- > Grid Search
- > Automated with statistical models:
 - Bayesian Optimization
 - Genetic Algorithms



Grid Search



Random Search

Existing Tools

- > Optuna
- > HyperOpt
- > Ray Tune
- > HiPlot (Facebook)
- > NNI (Microsoft)
- > Scikit-learn ([3.2. Tuning the hyper-parameters of an estimator — scikit-learn 1.1.3 documentation](#))
- > ... and many more.

The Process of Hyperparameters Tuning



Any problems with
this approach?

Tuning Hyperparam

1

Test them using the
Training-set

- Measure the training error (*loss*) and correct accordingly

2

Test them with the **Test-set**

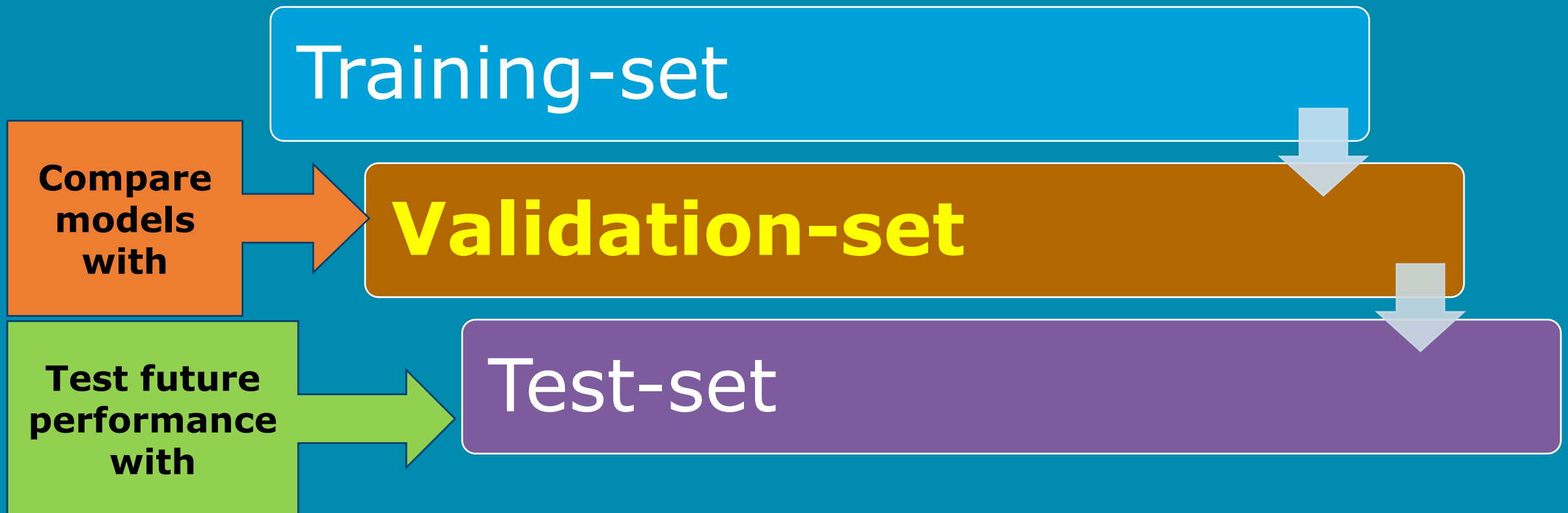
- Measure the test error (*evaluation*) and correct accordingly.

Never Use The Test-Data for Training!

WILL LEAD TO
INDUCTIVE BIAS



Solution 1: Separate the data into 3 sets

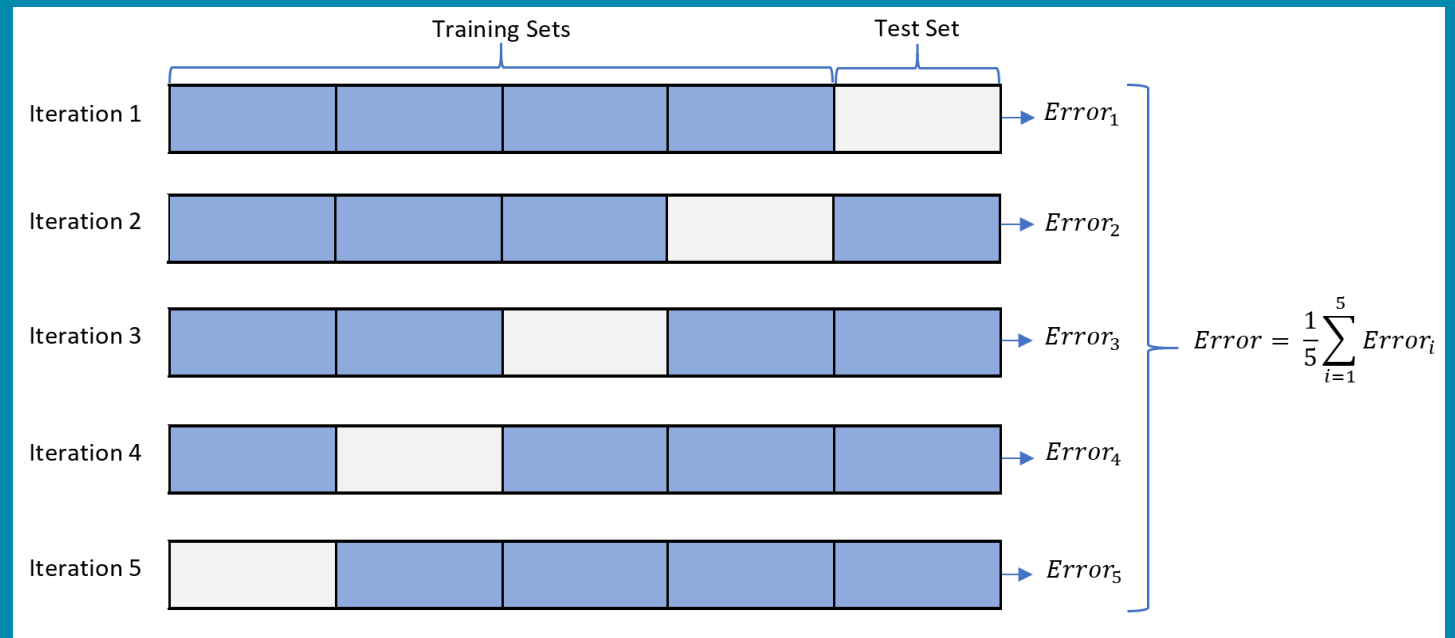


Solution 1: Hyperparameters tuning

1. Split the data into Train/Validation/Test sets
(e.g. 70% / 10% / 20%)
2. For each hyperparameters combination:
 - a) Train a model using the training-set
 - b) Compute the error rate (loss) on the validation-set
3. Choose the model with the lowest error (loss)
4. Evaluate future performance on the test-data

Solution 2: Cross Validation

- > Divide the data into K parts
- > Loop **k** times:
 - Train on all parts except for the k_{th} part
 - Test on the k_{th}
- > Average the test result



Take-Aways

- > A **sample** of a population is **prone** to **errors**
- > Have awareness of the **biases** and errors
- > Be careful with dirty & **noisy** data, **over/underfitting**
- > Follow experiment best-practices guidelines:
 - > **Never touch the test data**

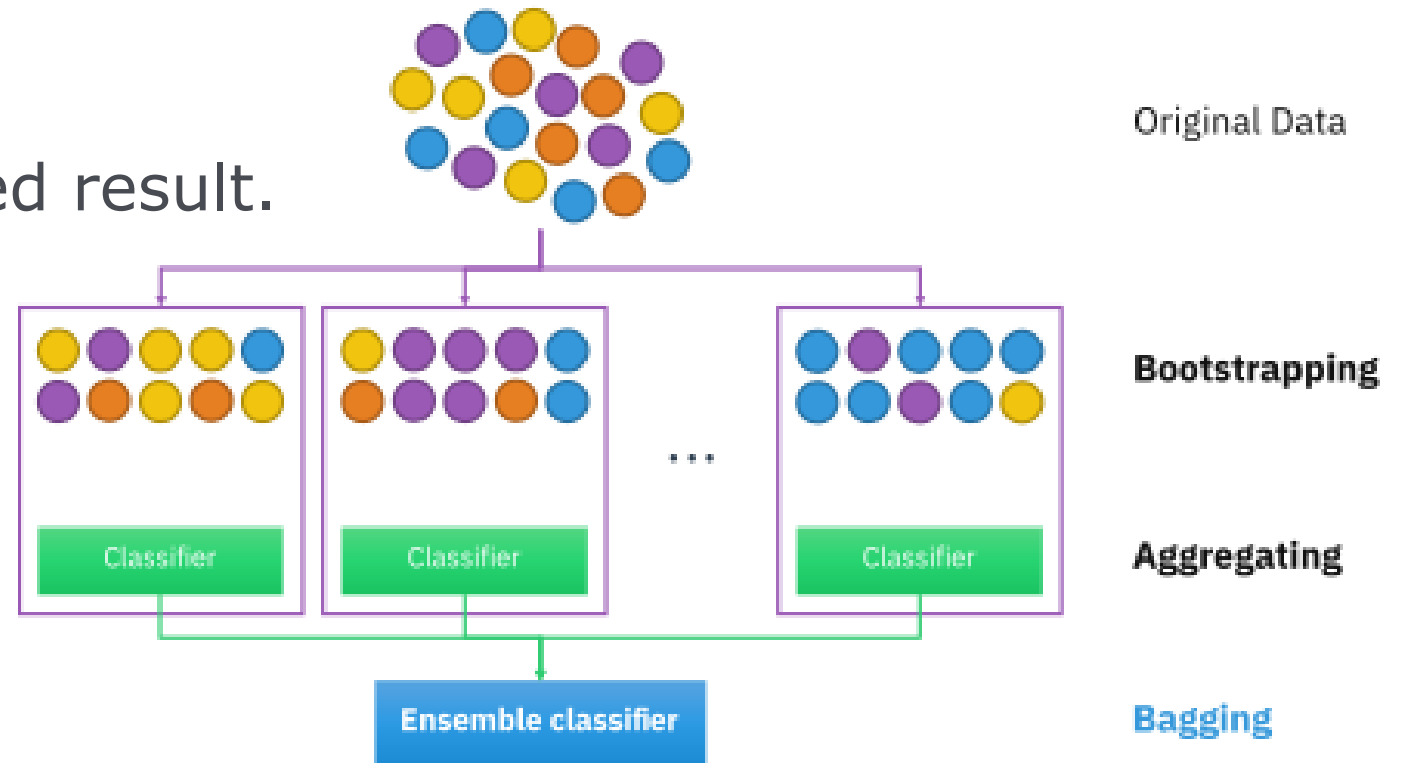
So, what can we do?

- > Collect more (labeled) data
- > Hyperparameters Tuning
- > **Better models**

Random Forests

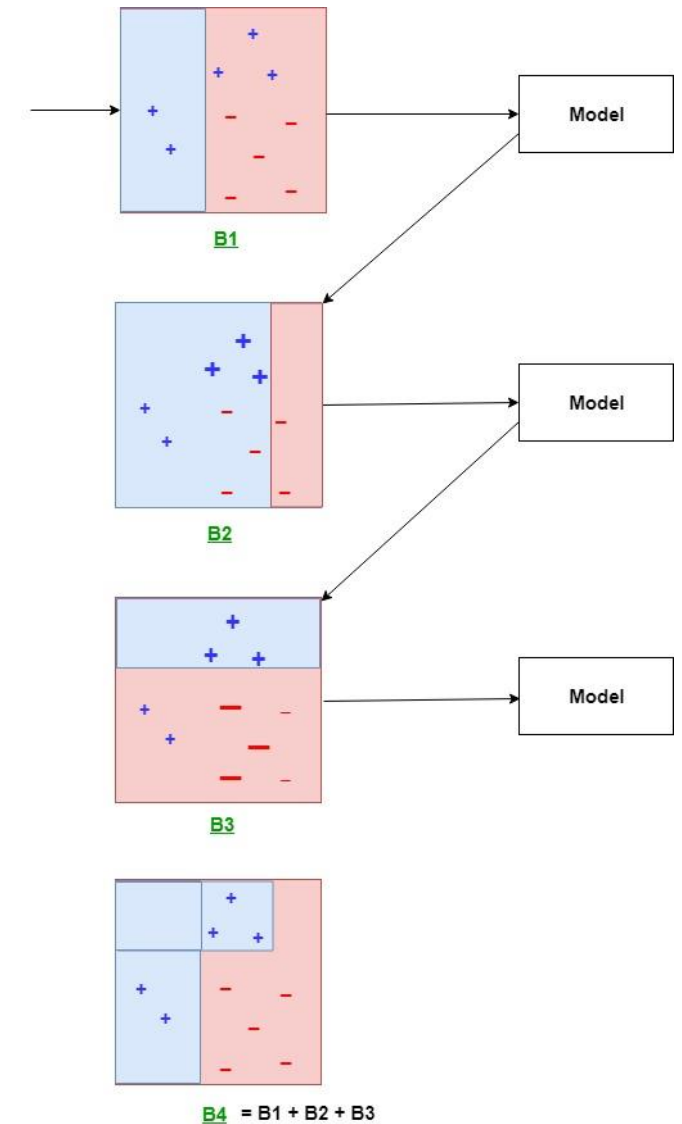
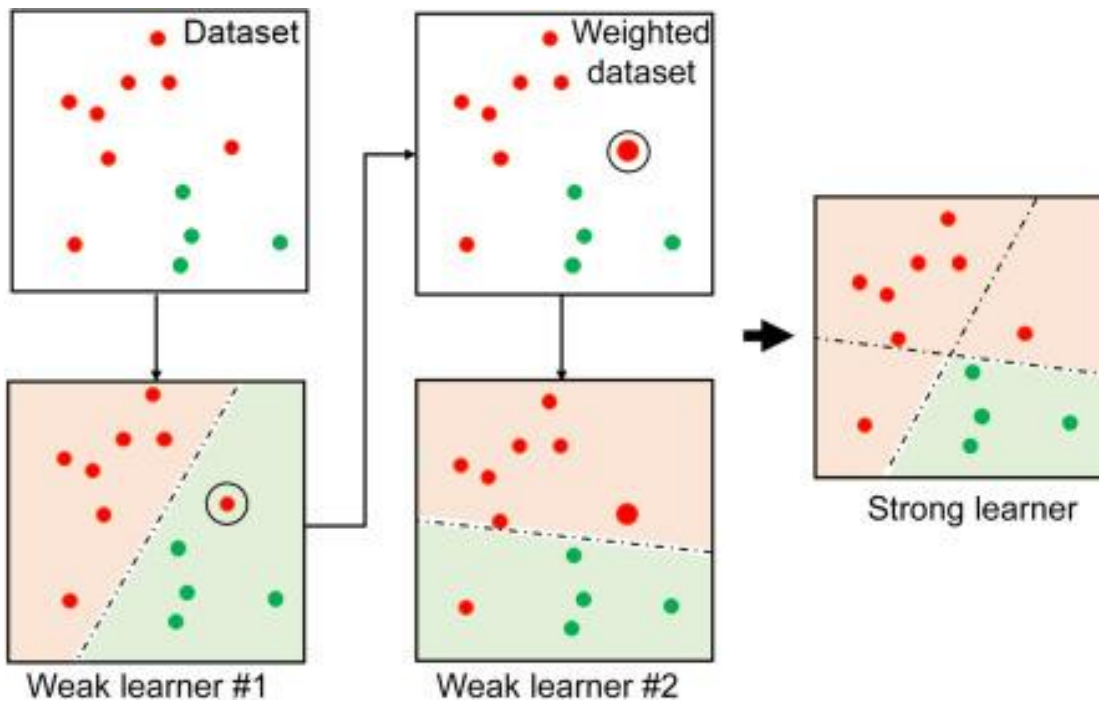
- > **Bootstrap Aggregating (bagging)**
Training n classifiers on subsets of the data
(randomly selected, with replacement)

For prediction,
choose the most voted result.



Random Forests

- > Boosting (e.g., AdaBoost)
Ensemble a combination of weak models



Random Forests

- > One of the strongest tools in Data Science
 - > XGBoost
 - > LightGBM
- > Works well on small(er) amounts of data
- > Can do both: classification & regression