Short-course & Hands-on Workshop: Artificial Intelligence Algorithms for Everyone

June 23 to June 26, 2025

Dr. R.M. (Rolando) Gonzales Martinez

university of groningen

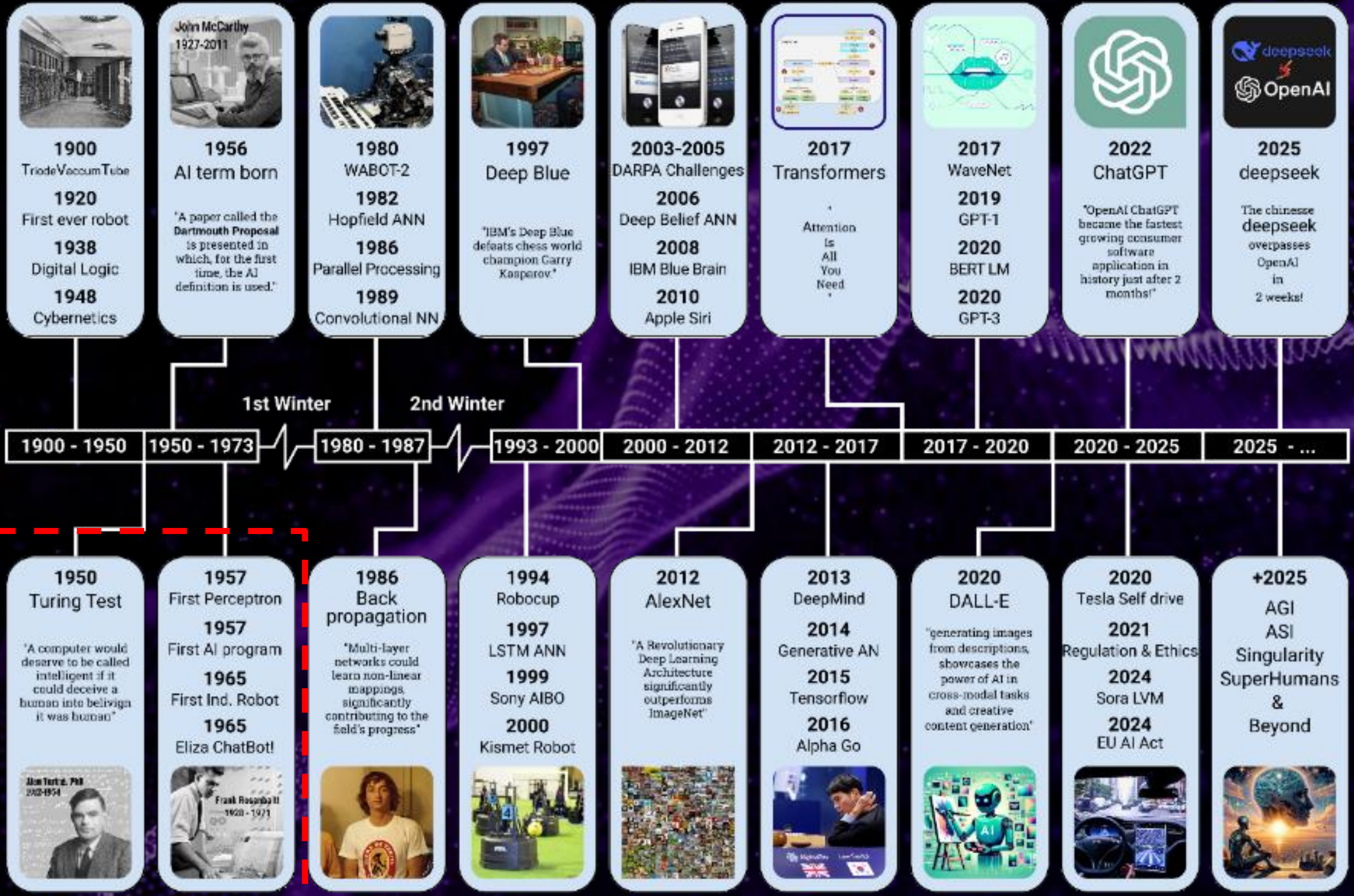
GitHub repository

https://github.com/rogon666/AI_workshop

- Presentations
- Databases
- Python scripts
- Additional lectures (papers, books)

June 23 (Monday)	**Introduction to artificial intelligence and Python programming (homogenization)** • The history of the spring and winters in artificial intelligence: from automatons to deep artificial neural networks and up to the singularity of human-level artificial intelligence • Introduction to Python programming of artificial intelligence algorithms, elastic nets and vibe coding
June 24 (Tuesday)	**Machine learning algorithms** • Machine learning algorithms for classification and regression: elastic nets, XGBoost, artificial neural networks, support vector machines, random forests • Time Series Forecasting : modeling and predicting time series data in practice with META's Prophet and long short-term memory models • Model evaluation, feature engineering, feature selection, and fine-tuning: metrics based on error minimization and the confusion matrix, Bayesian hypertuning
June 25 (Wednesday)	**Deep learning algorithms and recent trends in Artificial Intelligence** • Deep learning algorithms and Large Language Models (LLMs): the Transformers' architecture • Semantic and sentiment analysis of textual data with RoBERTa • Ethical considerations on the application of AI algorithms and mitigation strategies • Recent advances in AI: liquid neural networks, how to use DeepSeek without worrying about privacy issues, spatial machine learning, systematic reviews and literature reviews in 5-minutes with GPTs, quantum computing, liquid neural networks
June 26 (Thursday)	Presentations: • Using Mobile Phone Surveys (MPS) to measure child mortality during crisis periods (Kassoum Dianou) • Ways in which AI can support autonomous navigation in Space Missions (Duna Meya i Mendoza)

AI history and definition



1900 TriodeVaccumTube
1920 First ever robot
1938 Digital Logic
1948 Cybernetics

1956 AI term born
"A paper called the **Dartmouth Proposal** is presented in which, for the first time, the AI definition is used."

1980 WABOT-2
1982 Hopfield ANN
1986 Parallel Processing
1989 Convolutional NN

1997 Deep Blue
"IBM's Deep Blue defeats chess world champion Garry Kasparov."

2003-2005 DARPA Challenges
2006 Deep Belief ANN
2008 IBM Blue Brain
2010 Apple Siri

2017 Transformers
" Attention Is All You Need "

2017 WaveNet
2019 GPT-1
2020 BERT LM
2020 GPT-3

2022 ChatGPT
"OpenAI ChatGPT became the fastest growing consumer software application in history just after 2 months!"

2025 deepseek
The chinesse **deepseek** overpasses OpenAI in 2 weeks!

1900 - 1950 | 1950 - 1973 | 1st Winter | 1980 - 1987 | 2nd Winter | 1993 - 2000 | 2000 - 2012 | 2012 - 2017 | 2017 - 2020 | 2020 - 2025 | 2025 - ...

1950 Turing Test
"A computer would deserve to be called intelligent if it could deceive a human into belivign it was human"

1957 First Perceptron
1957 First AI program
1965 First Ind. Robot
1965 Eliza ChatBot!

1986 Back propagation
"Multi-layer networks could learn non-linear mappings, significantly contributing to the field's progress"

1994 Robocup
1997 LSTM ANN
1999 Sony AIBO
2000 Kismet Robot

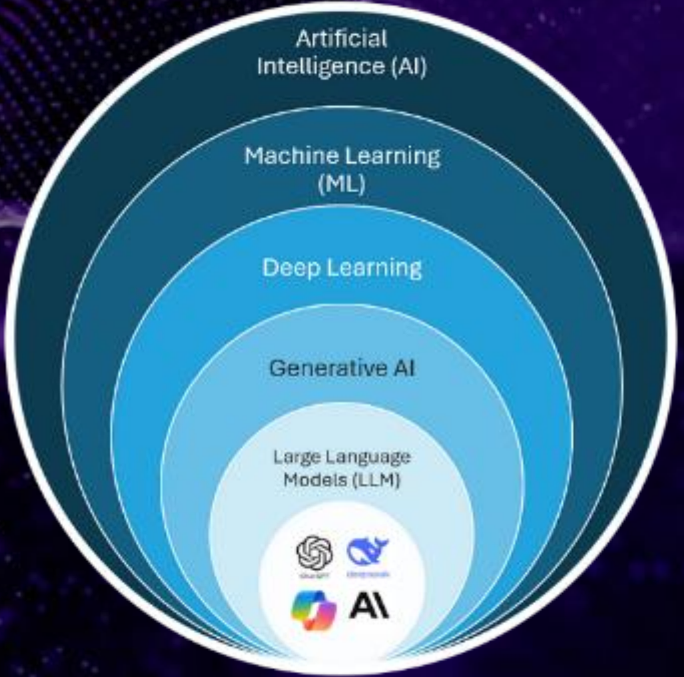
2012 AlexNet
"A Revolutionary Deep Learning Architecture significantly outperforms ImageNet"

2013 DeepMind
2014 Generative AN
2015 Tensorflow
2016 Alpha Go

2020 DALL-E
"generating images from descriptions, showcases the power of AI in cross-modal tasks and creative content generation"

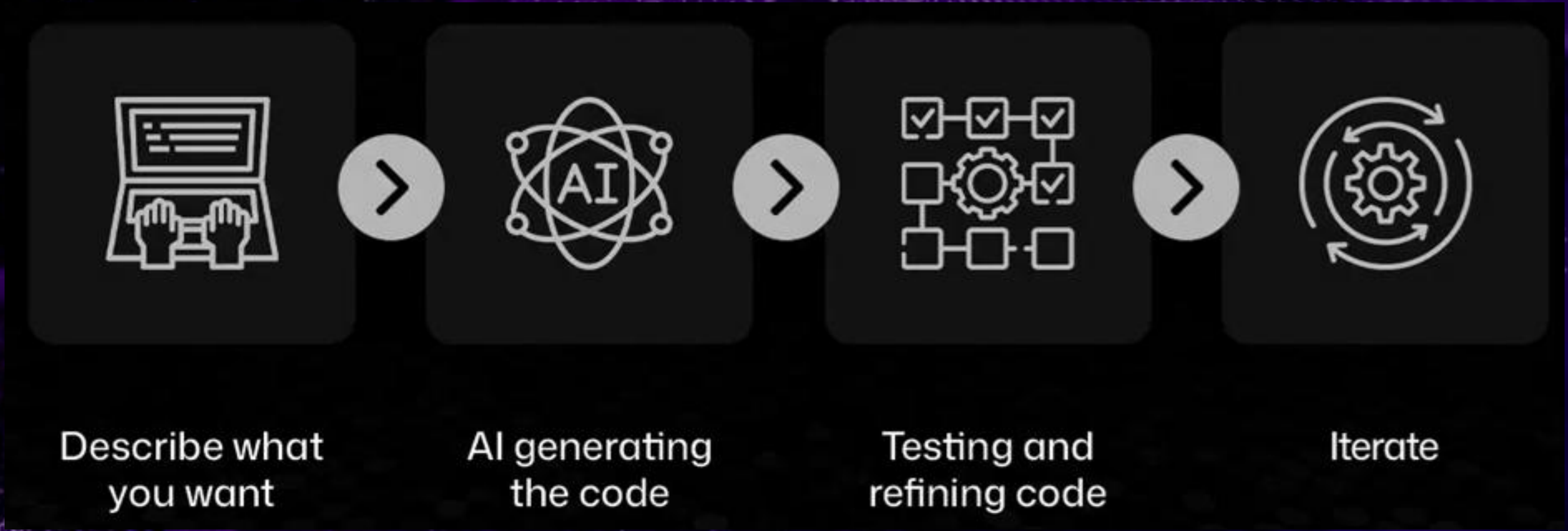
2020 Tesla Self drive
2021 Regulation & Ethics
2024 Sora LVM
2024 EU AI Act

+2025
AGI
ASI
Singularity
SuperHumans
&
Beyond



Vibe coding

How to Vibe Code in Python?



Elastic Nets

Elastic Nets are based on a regularization that aims to find a solution that minimizes the loss function with a combined penalty of **L1** (Lasso) and **L2** (Ridge).

$$\min_{\beta}\left\{\frac{1}{2N}\sum_{i=1}^{N}\left(y_i-\sum_{j=1}^{p}X_{ij}\beta_j\right)^2+\lambda_1\sum_{j=1}^{p}|\beta_j|+\lambda_2\sum_{j=1}^{p}\beta_j^2\right\}$$

Elastic Nets are useful in situations where the predictor variables are highly correlated or when the number of variables is greater than the number of observations.

Elastic Nets

Lasso (Least Absolute Shrinkage and Selection Operator, L1), Ridge (L2), and Elastic Nets are regularization techniques that help reduce overfitting and deal with multicollinearity and redundant variables.

$$\hat{\beta}_{\text{ridge}} = \arg\min_{\beta} \left(\sum_{i=1}^{m} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{n} \beta_j^2 \right)$$

$$\hat{\beta}_{\text{lasso}} = \arg\min_{\beta} \left(\sum_{i=1}^{m} (y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{n} |\beta_j| \right)$$

$$\hat{\beta}_{\text{elastic}} = \arg\min_{\beta} \left(\sum_{i=1}^{m} (y_i - \hat{y}_i)^2 + \lambda_1 \sum_{j=1}^{n} |\beta_j| + \lambda_2 \sum_{j=1}^{n} \beta_j^2 \right)$$

Elastic Nets

Lasso (Least Absolute Shrinkage and Selection Operator, L1), Ridge (L2), and Elastic Nets are regularization techniques that help reduce overfitting and deal with multicollinearity and redundant variables.

In `sklearn.linear_model.ElasticNet`,

$$\min_{w\in\mathbb{R}^p,\, b\in\mathbb{R}} \frac{1}{2n}\|y - Xw - b1\|_2^2 + \alpha\Big(\underbrace{\ell_1\text{-term}}_{l_1\_ratio\,\|w\|_1} + \underbrace{\ell_2\text{-term}}_{(1-l_1\_ratio)\,\frac{1}{2}\|w\|_2^2} \Big)$$

$\alpha > 0$ scales the overall penalty strength.

$l\_1\_ratio \in [0,1]$ interpolates between **pure Lasso** and **pure Ridge**.

Elastic Nets

Lasso (Least Absolute Shrinkage and Selection Operator, L1), Ridge (L2), and Elastic Nets are regularization techniques that help reduce overfitting and deal with multicollinearity and redundant variables.

In `sklearn.linear_model.ElasticNet`,

$$\min_{w\in\mathbb{R}^p,\, b\in\mathbb{R}} \frac{1}{2n}\|y - Xw - b\mathbf{1}\|_2^2 + \alpha\underbrace{\Big(\underbrace{\ell_1\text{-term}}_{l_1\_ratio\,\|w\|_1} + \underbrace{\ell_2\text{-term}}_{(1-l_1\_ratio)\,\frac{1}{2}\|w\|_2^2} \Big)}_{\text{combined penalty}},$$

$$\|w\|_1 = \textstyle\sum_{j=1}^{p} |w_j|.$$

$$\|w\|_2^2 = \textstyle\sum_{j=1}^{p} w_j^2.$$

$$l_1\_ratio \in [0,1]$$

$l_1\_ratio = 1$, Lasso:

$$\alpha\left(1\,\|w\|_1 + 0\cdot\tfrac{1}{2}\|w\|_2^2\right) = \alpha\|w\|_1$$

$l_1\_ratio = 0$, Ridge:

$$\alpha\left(0\,\|w\|_1 + 1\cdot\tfrac{1}{2}\|w\|_2^2\right) = \frac{\alpha}{2}\|w\|_2^2$$

Laboratory: Python scripts

Python scripts:

- aiw_00_elizaAI.ipynb: AI-based psychological counseling system
- aiw_01_salaries.ipynb: Python script for OLS regression and Elastic Nets
- aiw_02_credit.ipynb: Python script for credit scoring
- aiw_03_creditlm.ipynb: Python script for credit scoring with machine learning logic

June 23 (Monday)	**Introduction to artificial intelligence and Python programming (homogenization)** • The history of the spring and winters in artificial intelligence: from automatons to deep artificial neural networks and up to the singularity of human-level artificial intelligence • Introduction to Python programming of artificial intelligence algorithms, elastic nets and vibe coding
June 24 (Tuesday)	**Machine learning algorithms** • Machine learning algorithms for classification and regression: elastic nets, XGBoost, artificial neural networks, support vector machines, random forests • Time Series Forecasting : modeling and predicting time series data in practice with META's Prophet and long short-term memory models • Model evaluation, feature engineering, feature selection, and fine-tuning: metrics based on error minimization and the confusion matrix, Bayesian hypertuning
June 25 (Wednesday)	**Deep learning algorithms and recent trends in Artificial Intelligence** • Deep learning algorithms and Large Language Models (LLMs): the Transformers' architecture • Semantic and sentiment analysis of textual data with RoBERTa • Ethical considerations on the application of AI algorithms and mitigation strategies • Recent advances in AI: liquid neural networks, how to use DeepSeek without worrying about privacy issues, spatial machine learning, systematic reviews and literature reviews in 5-minutes with GPTs, quantum computing, liquid neural networks
June 26 (Thursday)	Presentations: • Using Mobile Phone Surveys (MPS) to measure child mortality during crisis periods (Kassoum Dianou) • Ways in which AI can support autonomous navigation in Space Missions (Duna Meya i Mendoza)

MACHINE LEARNING ALGORITHMS

Machine learning in AI

Given an input space X (feature space) and an output space Y, machine learning is an optimization problem where the goal is to find a function $f: X \rightarrow Y$ that predicts the output $y \in Y$ given an input $x \in X$, with f^* being optimal in terms of **generalization** and **regularization**:

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \qquad x_i \in \mathcal{X} \qquad y_i \in \mathcal{Y}$$

$$f^* = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{(x,y)\sim P} \left[L(f(x), y)\right]$$

$$f^* = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i)$$

$$f^* = \arg\min_{f \in \mathcal{F}} \left[\frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i) + \lambda R(f)\right]$$

Machine learning in AI

Given a set of functions (models), M_1, M_2,..., Mk, it is necessary to consider **performance measures** for continuous and discrete variables when deciding on the best function (model):

$$M_1, M_2, \ldots, M_k$$

$$\hat{\beta}_{\text{train},i} = \arg\max_{\beta} \log \mathcal{L}(\beta; X_{\text{train},i}, y_{\text{train}})$$

$$\hat{y}_{\text{test},i} = g^{-1}(X_{\text{test},i}\hat{\beta}_{\text{train},i})$$

$$\text{MSE}_i = \frac{1}{n_{\text{test}}} \sum_{j=1}^{n_{\text{test}}} (y_{\text{test},j} - \hat{y}_{\text{test},ij})^2$$

$$R_i^2 = 1 - \frac{\sum_{j=1}^{n_{\text{test}}} (y_{\text{test},j} - \hat{y}_{\text{test},ij})^2}{\sum_{j=1}^{n_{\text{test}}} (y_{\text{test},j} - \bar{y}_{\text{test}})^2}$$

$$\text{AUC-ROC}_i = \text{AUC}(\text{Curva ROC}_i)$$

$$M_{\text{best}} = \arg\min_i \text{MSE}_i$$

$$M_{\text{best}} = \arg\max_i R_i^2$$

$$M_{\text{best}} = \arg\max_i \text{AUC-ROC}_i$$

Main types of learning (what was learning?)

- Supervised: minimizes loss on labeled data
- Semi-supervised: minimizes a combination of labeled and regularized unlabeled data
- Unsupervised: model trained with unlabeled data (e.g. clustering)

$$f^* = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i)$$

$$f^* = \arg\min_{f \in \mathcal{F}} \left[\frac{1}{|L|} \sum_{(x_i, y_i) \in \mathcal{D}_L} L(f(x_i), y_i) + \lambda R(f, \mathcal{D}_U) \right]$$

$$f^* = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} L(f(x_i))$$

Other types of learning

- Reinforcement: optimal policy $\pi*$ that maximizes expected cumulative discounted reward.
- Meta-learning: optimize a model for a specific task or to generalize across tasks
- Transfer: Trained ML model is reused and adapted to a new task

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)\right]$$

π: Policy (a mapping from states s_t to actions a_t)

$r(s_t, a_t)$: Reward received at time step t

$\gamma \in [0, 1)$: Discount factor (weights future rewards)

T: Time horizon

$\mathbb{E}[\cdot]$: Expectation over trajectories induced by the policy

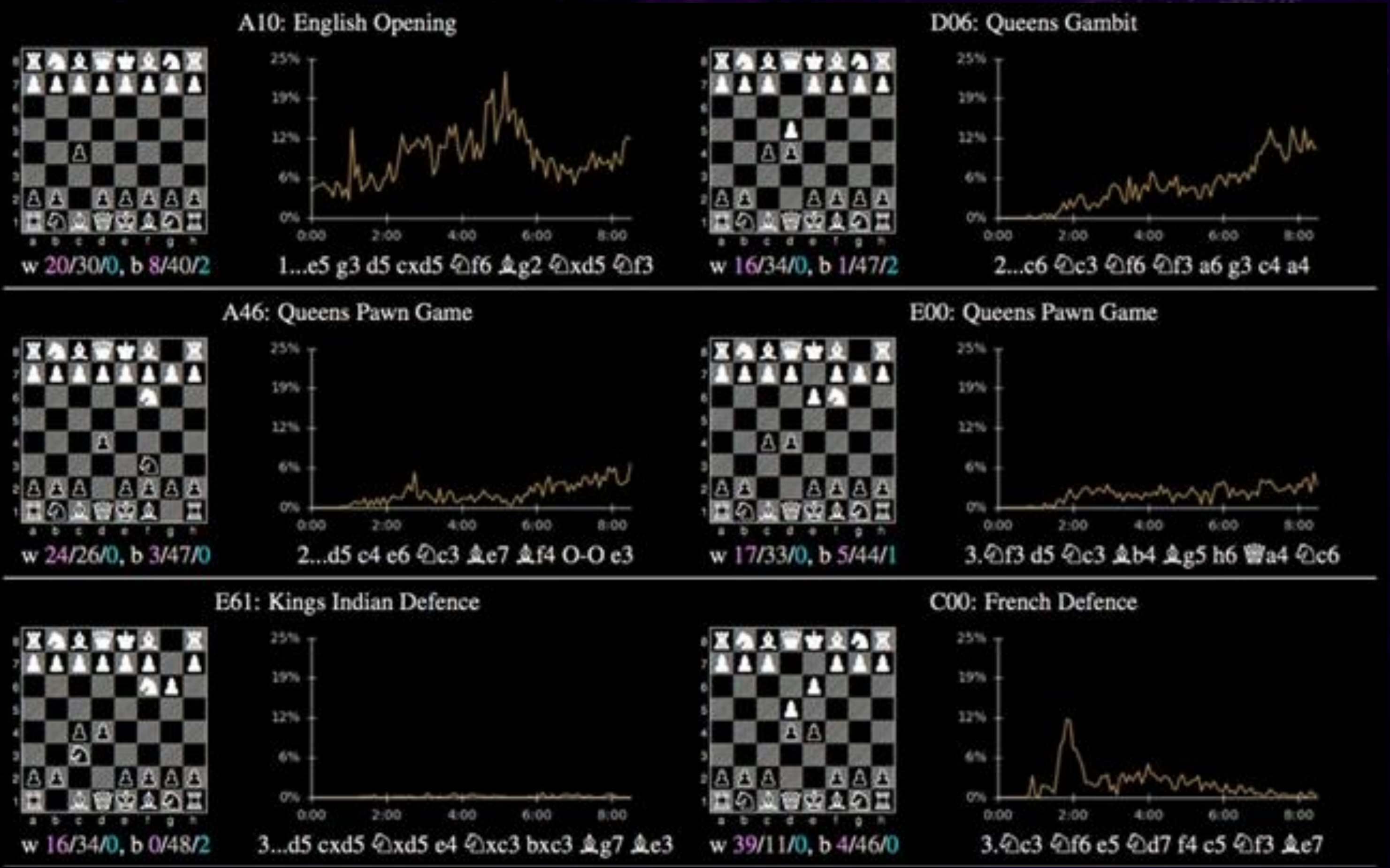
$$f_T^* = \arg\min_{f \in \mathcal{F}_T} \frac{1}{m} \sum_{j=1}^{m} L(f(x_j), y_j) \qquad f^* = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{T_i \sim p(T)} \left[\min_{\theta_i} L_{T_i}(f(\theta_i))\right]$$

$$f^* = \arg\min_{f \in \mathcal{F}} \left[\underbrace{\frac{1}{n_T} \sum_{i=1}^{n_T} L(f(x_i^T), y_i^T)}_{\text{Target Loss}} + \lambda \underbrace{\Omega(f; \mathcal{D}_S, \mathcal{T}_S)}_{\text{Transfer regularization}}\right]$$

Other types of learning

Reinforcement: optimal policy $\pi*$: Learning by interaction with an environment to maximize cumulative reward.

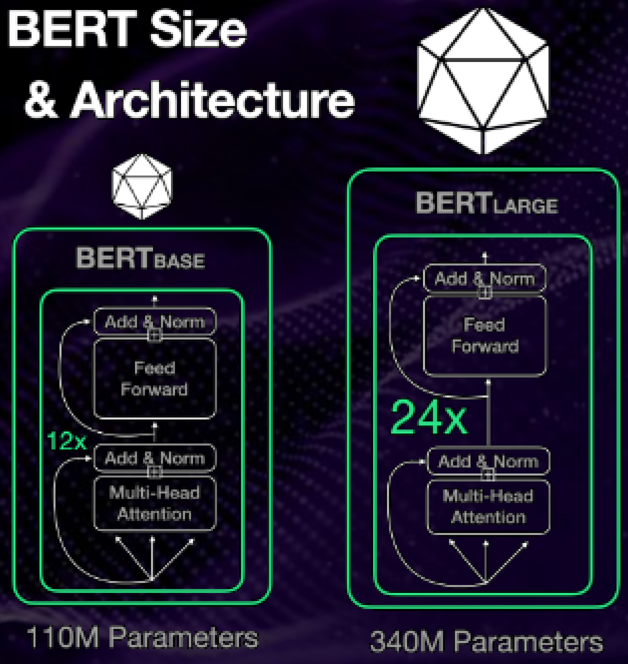
- AlphaZero (Deepmind)
- Boston dynamics

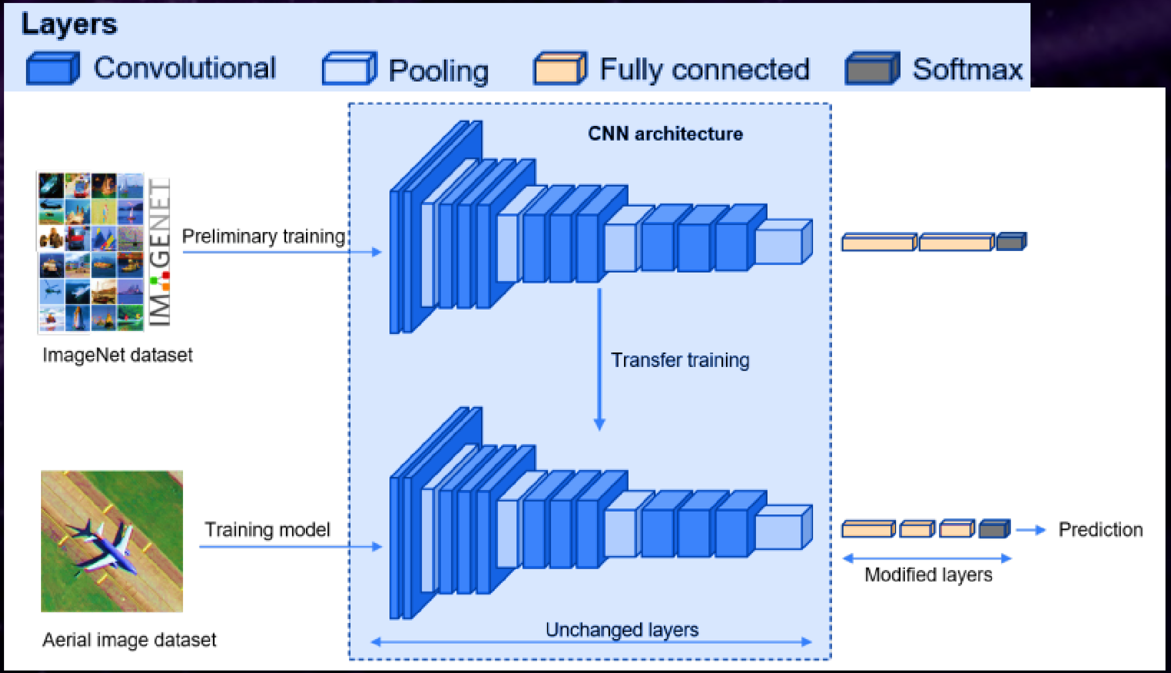


Other types of learning

Transfer: Leveraging knowledge from one domain or task to benefit another:

- BERT, GPTs transferred to other languages
- ImageNet-pretrained models transferred to remote sensing and geospatial AI.





Thapa, A., Horanont, T., Neupane, B., & Aryal, J. (2023). Deep learning for remote sensing image scene classification: A review and meta-analysis. *Remote Sensing*, *15*(19), 4804.

Sample partitions in AI

- **Train-Test Split:** The most basic and common method. The data is divided into two parts: a training set to train the model (estimate the model parameters) and a test set to evaluate the model's performance on unseen data.
- **Hold-Out with Validation:** Similar to the train-test split, but includes a third validation set:
    - **Training (Train):** To adjust the model parameters.
    - **Validation:** To adjust hyperparameters and prevent overfitting.
    - **Test:** For the final evaluation of the model.

Sample partitions in AI: K-fold cross-validation

- The sample is partitioned into k parts (folds).
- Conventional values: k = 5, k = 10, k = 20. Depends on data size and computational cost. Normally k =10
- In small samples, k = n (LOO partition).

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

$$D_1, D_2, \ldots, D_k$$

$$D_i \subset \mathcal{D} \text{ for all } i \text{ and } D_i \cap D_j = \emptyset \text{ for } i \neq j.$$

For each $i = 1, 2, \ldots, k$:

$$\mathcal{D}_{-i} = \mathcal{D} \setminus D_i = \bigcup_{j \neq i} D_j$$

Train $h(\cdot; \theta_i)$ using $\mathcal{D}_{-i}$.

$$E_i = \frac{1}{|D_i|} \sum_{(x_j, y_j) \in D_i} \ell(h(x_j; \theta_i), y_j)$$

$$E_{CV} = \frac{1}{k} \sum_{i=1}^{k} E_i$$

Sample partitions in AI: Stratified k-fold cross-validation

- Extends k-fold cross-validation so that each fold has approximately the same proportion of classes c = 1,2,...,C of the target as the full data set.
- Involves a stratification process (each category c is a stratum).
- Useful in data with an unbalanced target.

$$D = \{(x_i, y_i)\}_{i=1}^{n} \qquad c \in C$$

$$n_c = \sum_{i=1}^{n} \mathbb{I}(y_i = c) \qquad p(c) = \frac{n_c}{n}$$

$$S_i = \varnothing, \quad T_i = \varnothing \quad \text{para } i = 1, \ldots, k$$

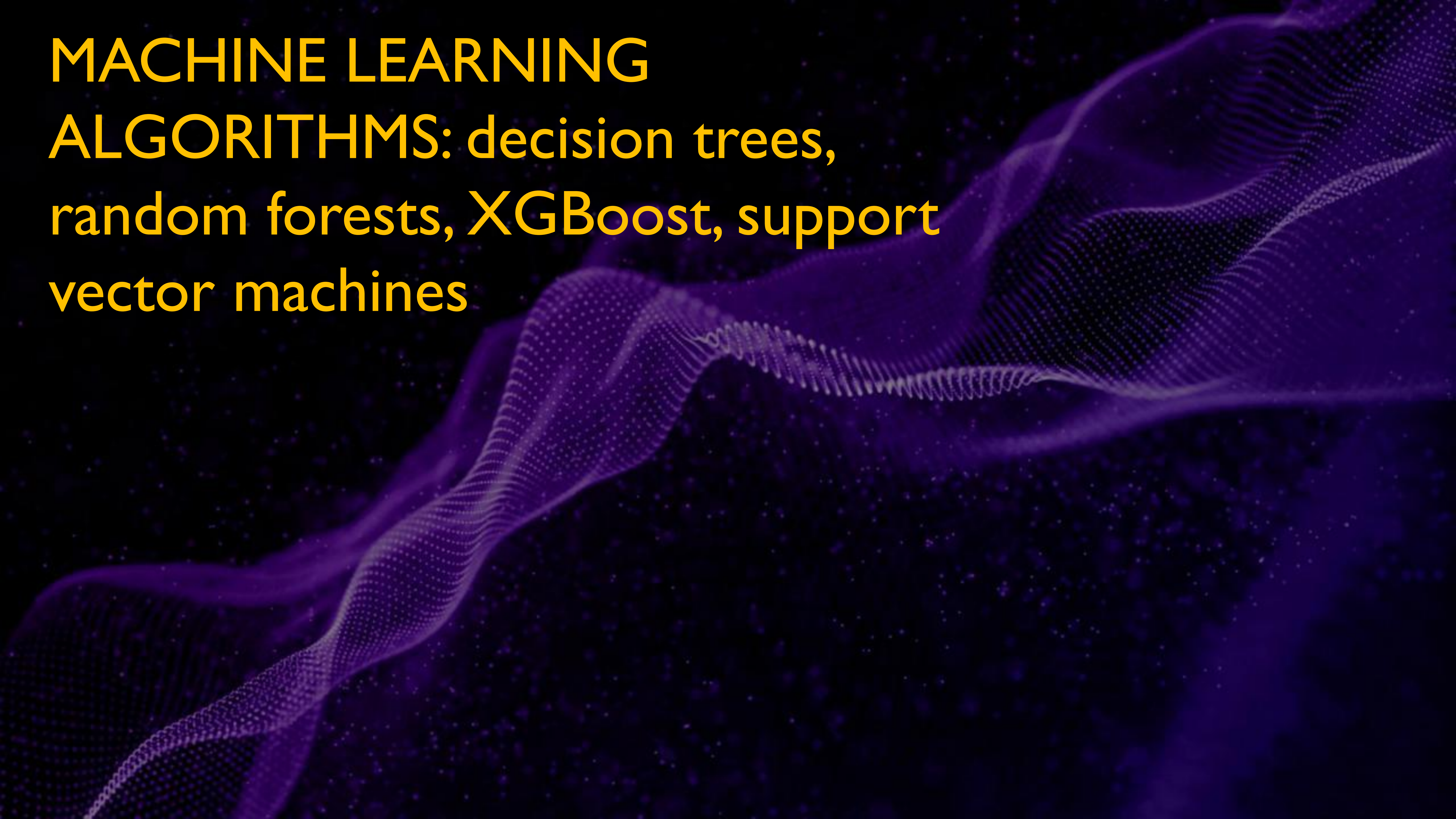
$$D_c = \{(x_i, y_i) \mid y_i = c, i = 1, \ldots, n\}$$

$$\tilde{D}_c = \{\tilde{D}_{c,1}, \tilde{D}_{c,2}, \ldots, \tilde{D}_{c,k}\}$$

$$T_i = T_i \cup \tilde{D}_{c,i}$$

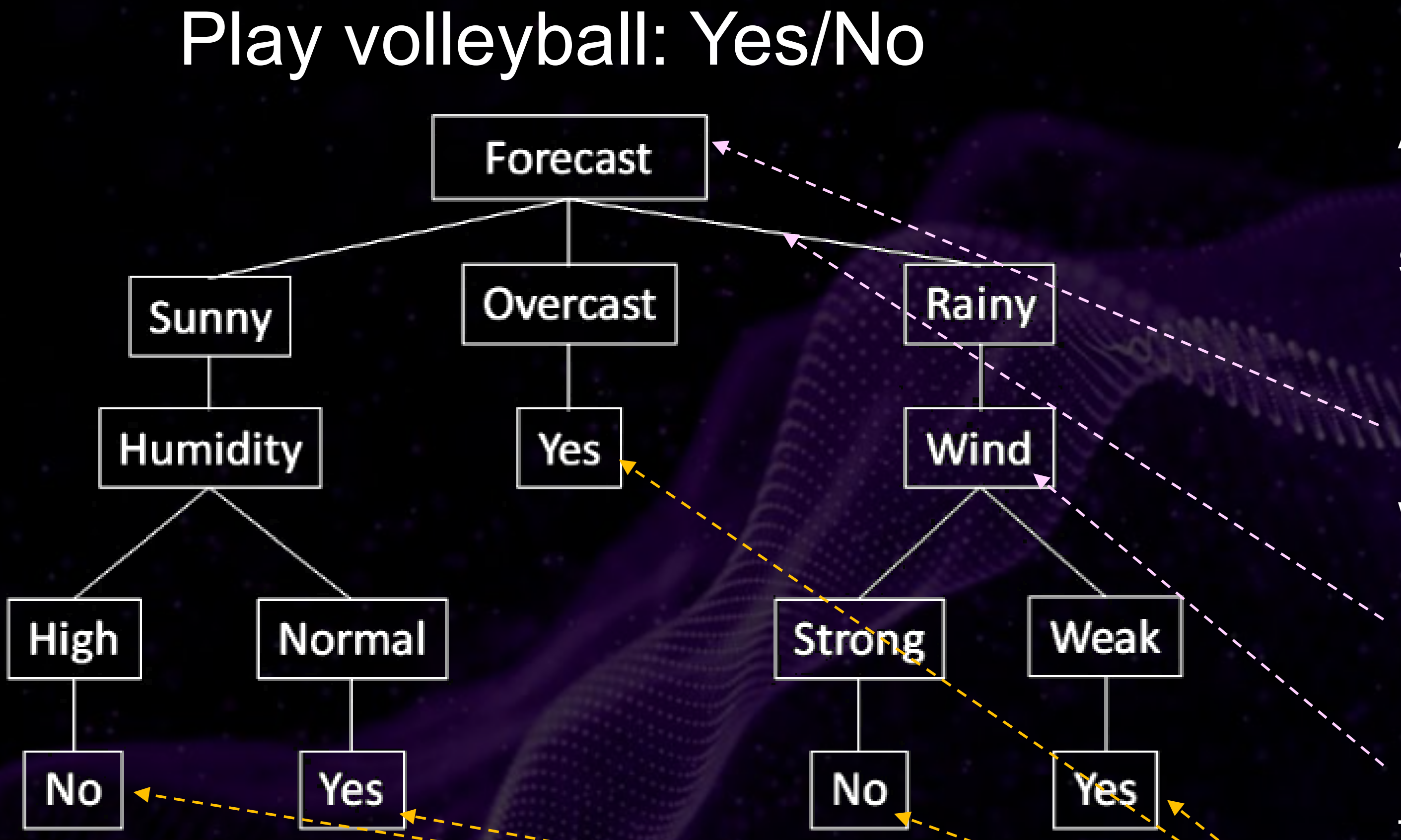
$$S_i = S_i \cup \left(\bigcup_{j \neq i} \tilde{D}_{c,j} \right)$$

$$\{(S_i, T_i)\}_{i=1}^{k}$$



MACHINE LEARNING ALGORITHMS: decision trees, random forests, XGBoost, support vector machines

Decision Trees and Random Forests



A decision tree is an algorithm with hierarchical structure in which a series of sequential decisions are made. It has the following structure:

Root: The top node of the tree where the data splitting begins.

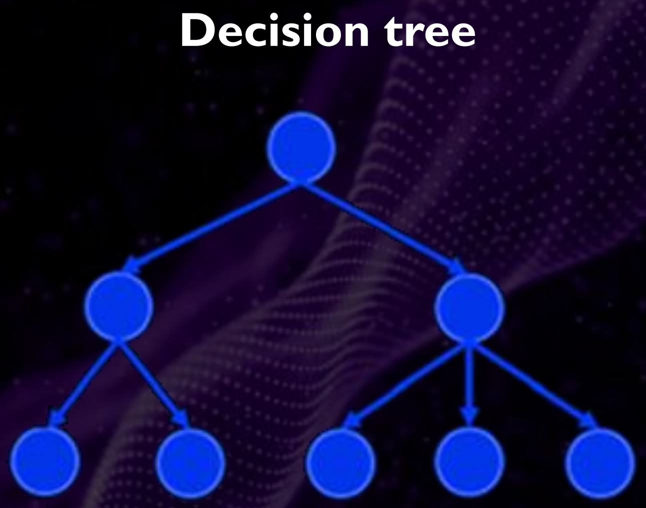
Branches: Represent the splits.

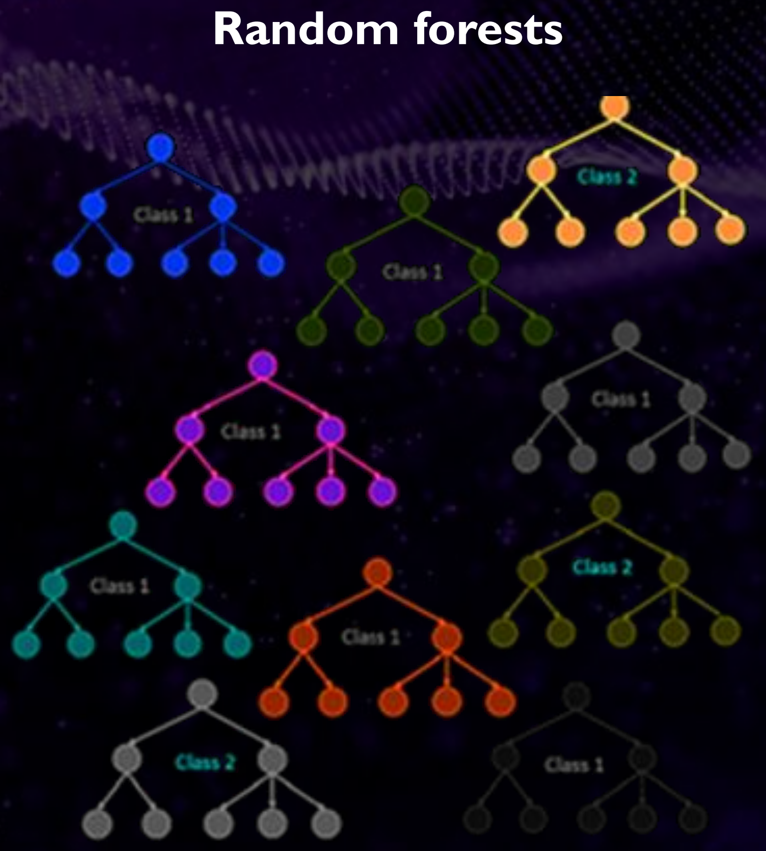
Nodes: Represent splitting options for further divisions.

Leaves: Represent the final decisions or classifications.

Decision Trees and Random Forests

Random forests are supervised learning algorithms that combine the output of multiple decision trees. Why random?





$$\mathcal{D}^{(b)} = \{(x_i, y_i)\}_{i=1}^{n},$$

$$m \ll p$$

Decision Trees and Random Forests: Algorithm

1. **Selection of the Feature and Partition Threshold:**

   - **For each feature:** The algorithm evaluates all possible split points (thresholds).
   - **For each possible threshold:** The samples are divided into two groups: one where the feature is less than or equal to the threshold, and another where it is greater.
   - **Gini calculation:** The Gini index is calculated for the resulting branches from the split.

2. **Choosing the Best Partition Threshold:**

   - **Purity comparison:** The Gini indices of the possible splits are compared. The goal is to minimize impurity (Gini < 0.5) in the child nodes.
   - **Threshold selection:** The threshold that results in the greatest reduction in the Gini index (impurity reduction) is selected. This means choosing the threshold that creates the purest possible child nodes.

Decision Trees and Random Forests: Algorithm

3. **Node Splitting:**

   - Once the optimal threshold is selected, the node is split into two branches: one for instances that meet the condition (i.e., where the feature value is less than or equal to the threshold) and another for those that do not (where the value is greater than the threshold).

4. **Process Repetition:**

   - This process is repeated recursively for each child node until a stopping criterion is met (e.g., when all nodes are pure, or when impurity cannot be significantly reduced).

Decision Trees and Random Forests

The initial node of the decision tree is chosen by evaluating all features and possible thresholds, selecting the combination that minimizes the impurity (pushing the class distribution of the target in each node as far toward one class as possible) of the resulting child nodes.

Partitioning the Data

D_{izq}: subset of data where the chosen feature $x_i \leq \text{threshold}$

D_{der}: subset where $x_i > \text{threshold}$

Impurity of Children After a Split

$$I_{\text{children}}(\text{threshold}) = \frac{|D_{\text{izq}}|}{|D|} \cdot I(D_{\text{izq}}) + \frac{|D_{\text{der}}|}{|D|} \cdot I(D_{\text{der}})$$

$$x^*, \text{threshold}^* = \arg \min_{x_i, \text{threshold}} I_{\text{children}}(\text{threshold})$$

Decision Trees and Random Forests

- **Gini** measures the proportion of classes in a particular node.
- Gini is a measure of purity used in the construction of decision trees.

It is used to evaluate the quality of a split or partition at the tree's nodes. The goal of the Gini index is to determine how mixed the data is.

$$Gini = 1 - (p_1^2 + p_2^2)$$

The Gini index ranges between 0 and 0.5:

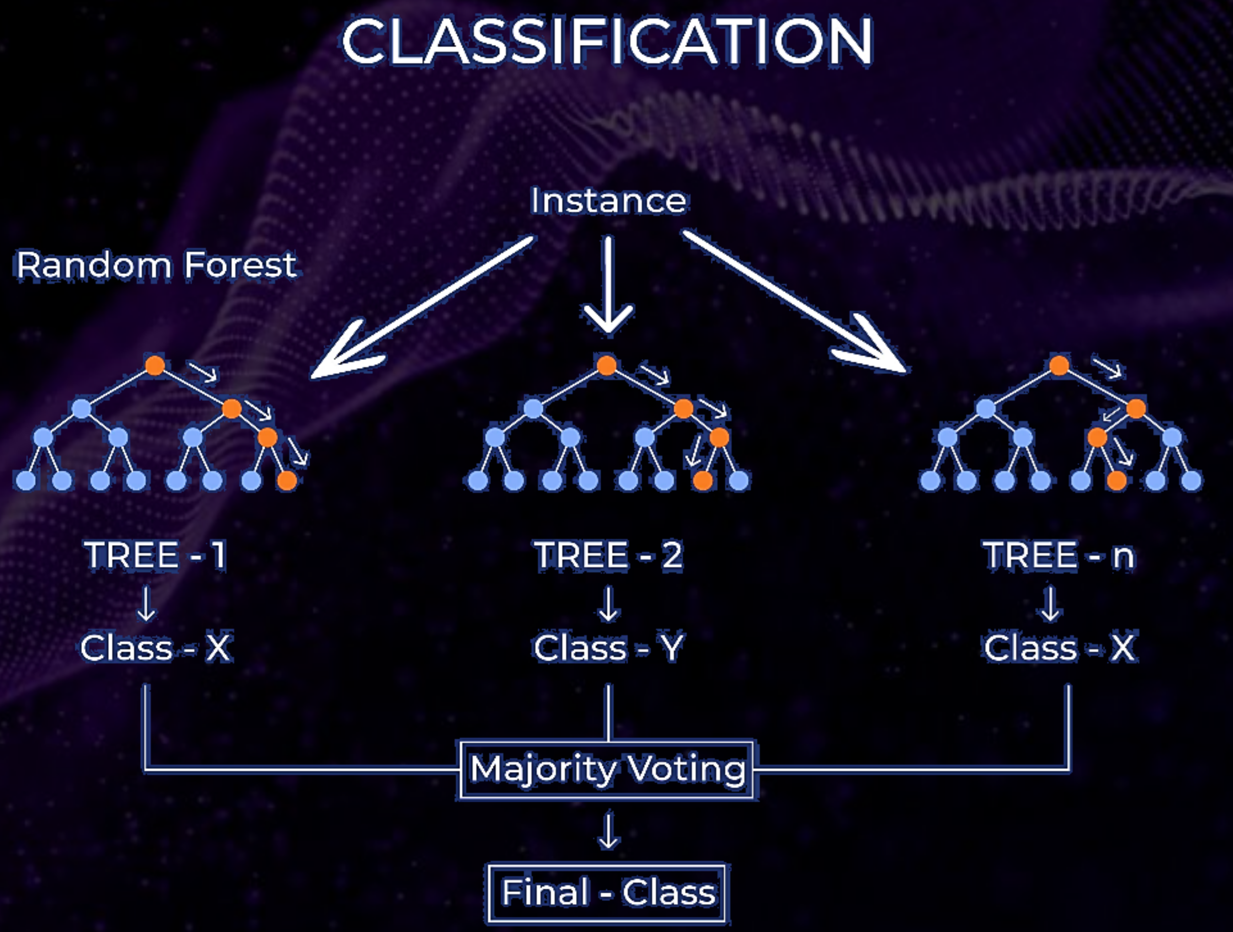
Gini = 0:
Extreme case indicates that all instances in the node belong to a single class, meaning the node is pure. Ideal situation.

Gini close to 0.5 (binary case):
Indicates that the instances in the node are evenly distributed among the classes, meaning the node is impure.

$$0 \leq G(m) \leq 1 - \frac{1}{K}$$

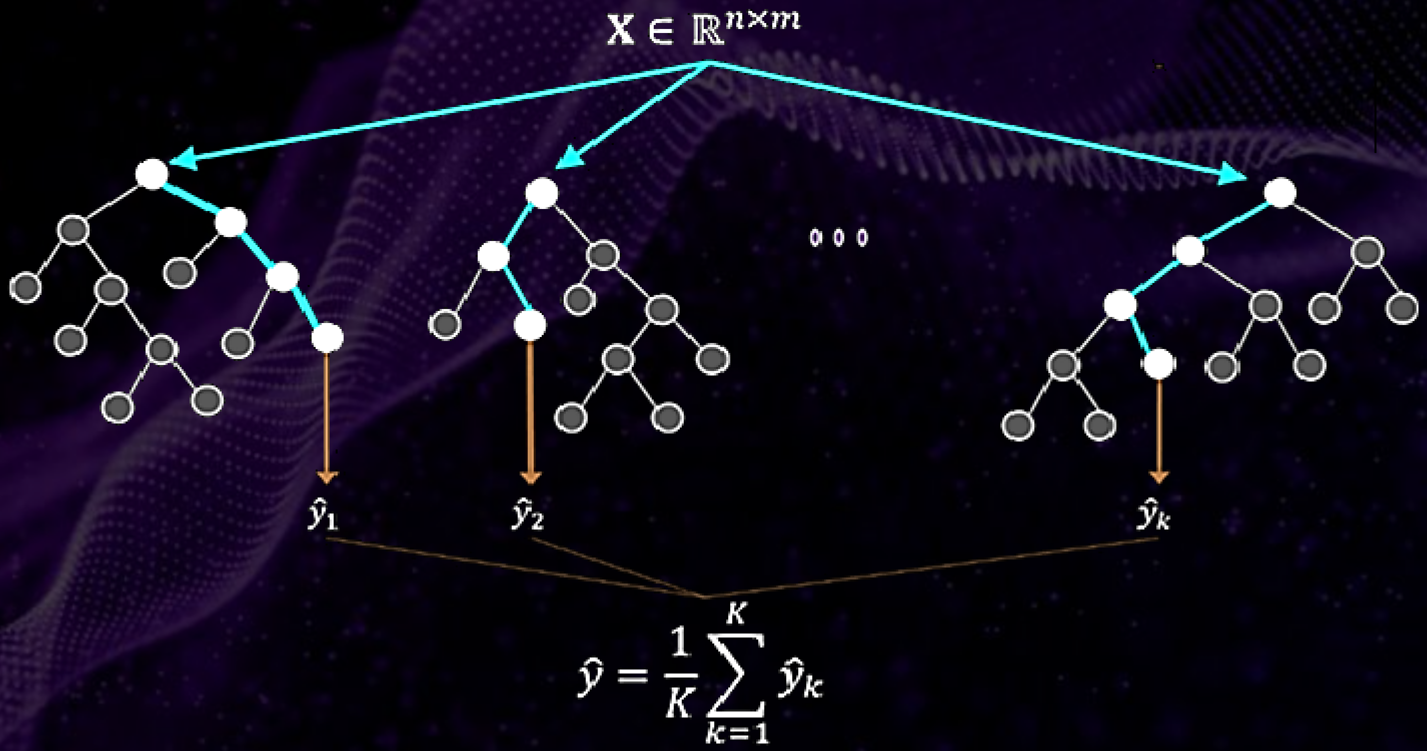
Random Forests

The final classification is the result of aggregating the individual classifications of the individual decision trees. In the discrete case, the final classification is the result of aggregating the individual classifications of the individual decision trees.



Random Forests

In the case of continuous data, the final result of the random forest is the average of the individual predictions of each tree.


$\mathbf{X} \in \mathbb{R}^{n \times m}$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_k$
$\hat{y} = \frac{1}{K}\sum_{k=1}^{K} \hat{y}_k$

Random Forests

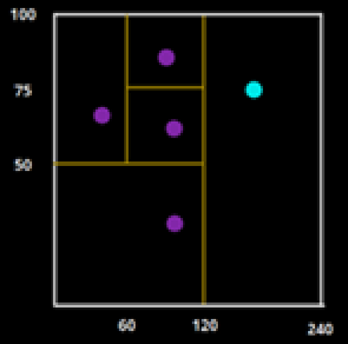
Advantages:

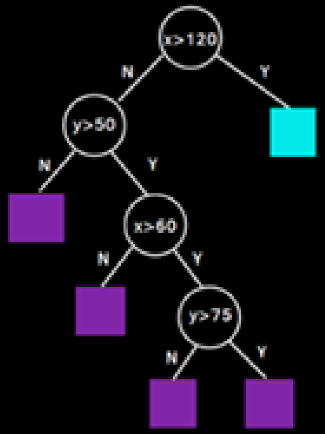
- It can be applied to both categorical and continuous inputs and outputs.
- It is parallelizable.
- It is robust to outliers.
- It works well with imbalanced data.
- The greater the number of trees, the more accurate the result, but overfitting must be monitored.
- It can generalize better due to the use of multiple decision trees.

Disadvantages:

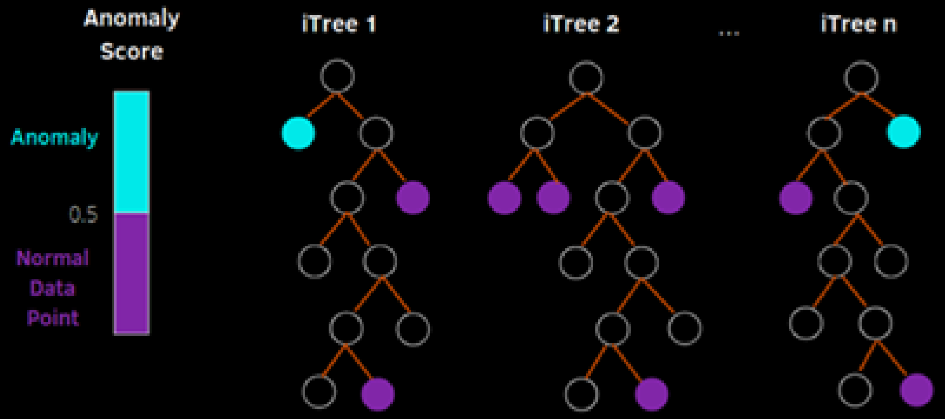
- **Interpretability**: Random forest models are not easily interpretable.
- For very large datasets, the size of the trees can consume a lot of memory.
- They can tend to overfit, so it is necessary to tune the hyperparameters.

Isolation Forests





Anomaly Score



```
Algorithm 1 Isolation Forest 1D with Contamination Threshold
Require: One-dimensional vector $x \in \mathbb{R}^n$, number of trees $T$,
    subsample size $\psi$, contamination rate $\pi$
Ensure: Binary anomaly labels for each $x_i \in x$
 1: Initialize scores $\ell(x_i) = 0$ for all $x_i \in x$
 2: for $t = 1$ to $T$ do
 3:     Draw a random subsample $x^{(t)} \subset x$, size $\min(n, \psi)$
 4:     for each $x_i \in x$ do
 5:         Compute path length $h_t(x_i) \leftarrow \text{Isolate}(x^{(t)}, x_i)$
 6:         Accumulate $\ell(x_i) \leftarrow \ell(x_i) + h_t(x_i)$
 7:     end for
 8: end for
 9: Normalize path lengths: $\ell(x_i) \leftarrow \ell(x_i)/T$
10: Compute expected path length $c(\psi)$
11: Compute anomaly scores: $s(x_i) = 2^{-\frac{\ell(x_i)}{c(\psi)}}$

12: // Contamination thresholding step
13: Sort scores $s(x_i)$ in descending order
14: Let $\tau$ be the $100 \cdot (1 - \pi)$-th percentile of the scores
15: for each $x_i \in x$ do
16:     if $s(x_i) > \tau$ then
17:         Label $x_i$ as an outlier
18:     else
19:         Label $x_i$ as an inlier
20:     end if
21: end for
22: return Anomaly labels for each $x_i$
```

Isolation Forests

If $\mathbb{E}[h(x)] \ll c(\psi)$, then $s(x) \approx 1$: strong anomaly

If $\mathbb{E}[h(x)] \approx c(\psi)$, then $s(x) \approx 0.5$: borderline

If $\mathbb{E}[h(x)] \gg c(\psi)$, then $s(x) \approx 0$: normal point

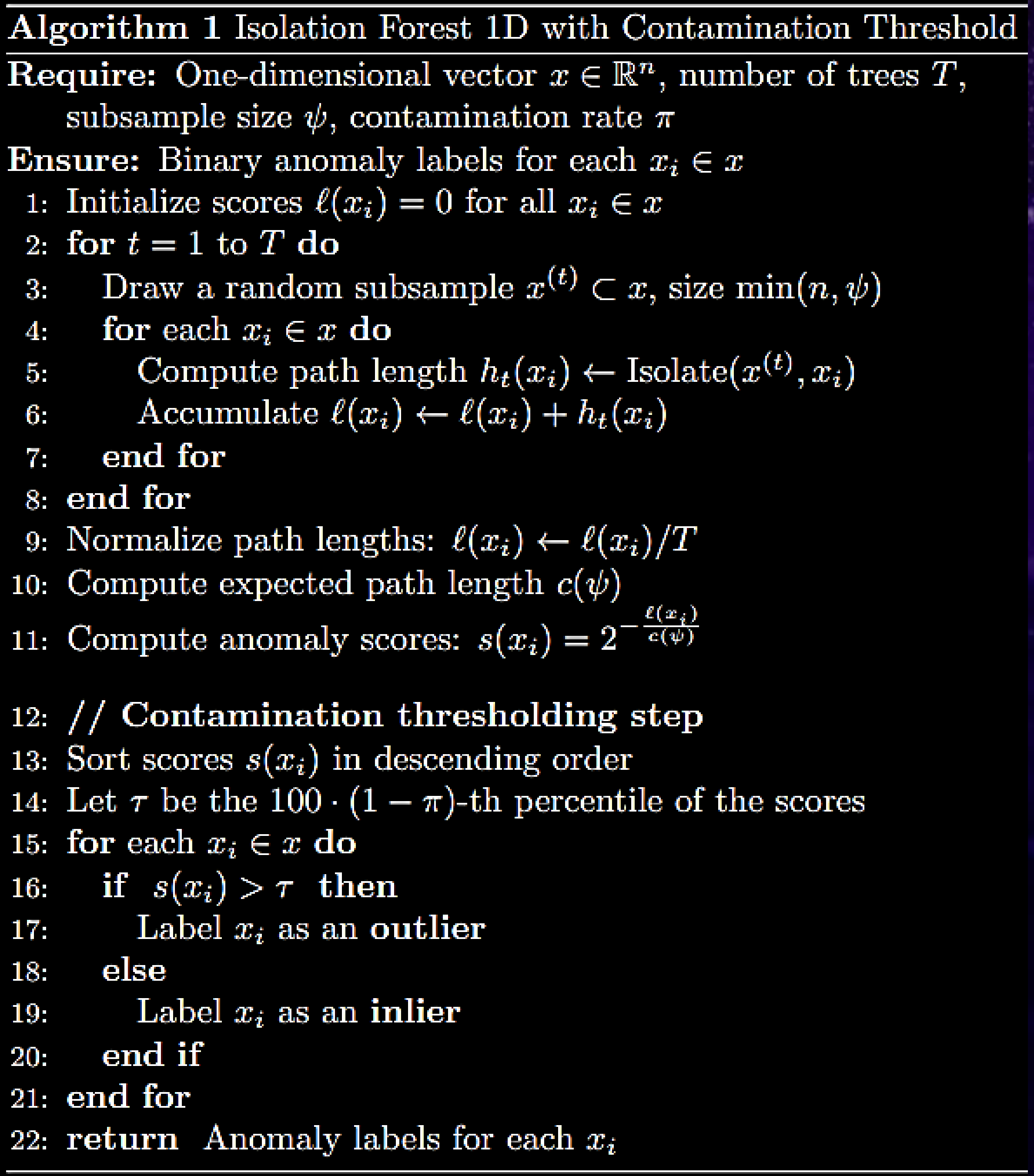
$$s(x) = 2^{-\frac{\mathbb{E}[h(x)]}{c(n)}}$$

$$\mathbb{E}[h(x)] = \frac{1}{T}\sum_{t=1}^{T} h_t(x)$$

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n}$$

$$H(i) \approx \ln(i) + \gamma$$

$\gamma \approx 0.5772$ Euler–Mascheroni constant



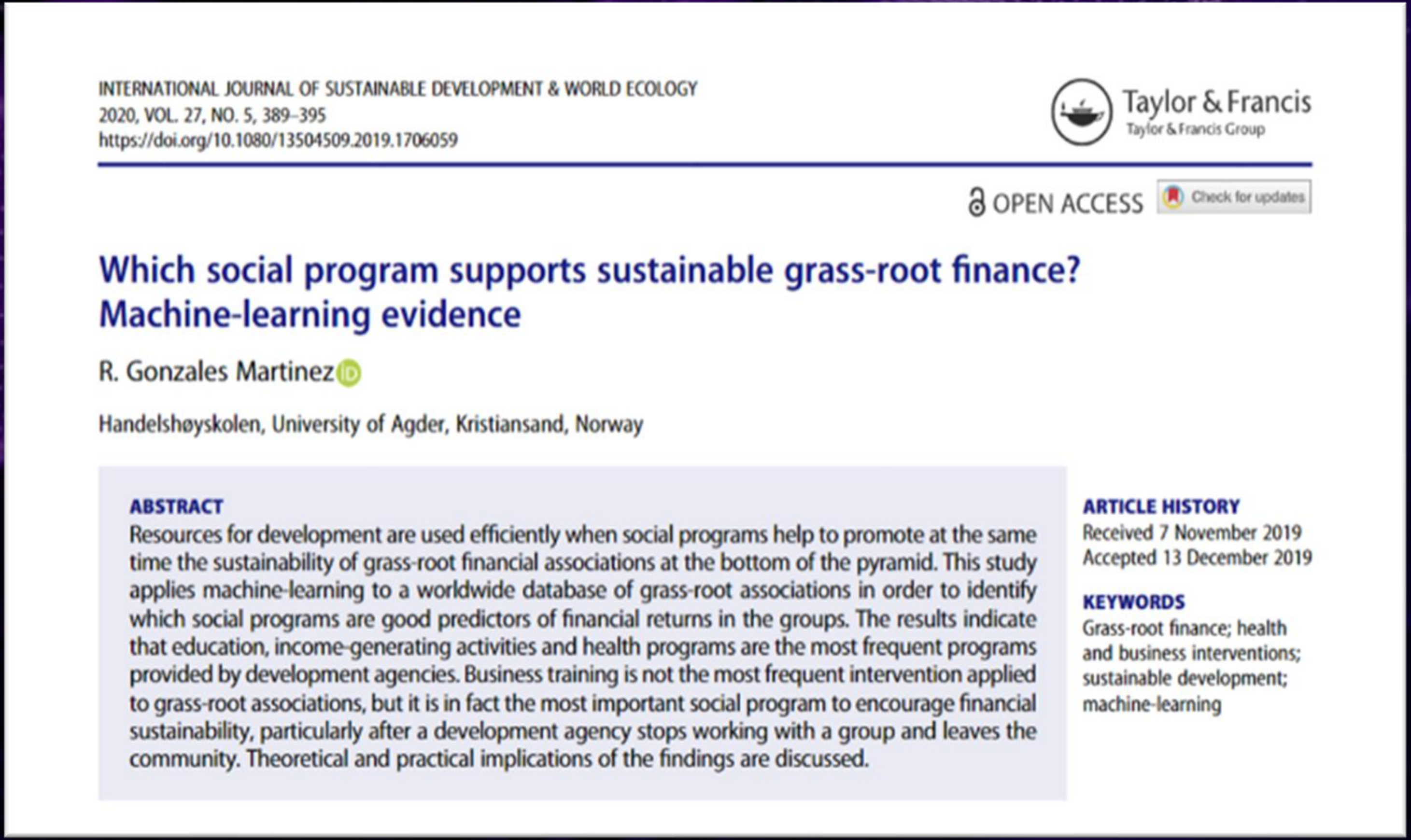
Algorithm 1 Isolation Forest 1D with Contamination Threshold

```
Require: One-dimensional vector $x \in \mathbb{R}^n$, number of trees $T$,
    subsample size $\psi$, contamination rate $\pi$
Ensure: Binary anomaly labels for each $x_i \in x$
 1: Initialize scores $\ell(x_i) = 0$ for all $x_i \in x$
 2: for $t = 1$ to $T$ do
 3:     Draw a random subsample $x^{(t)} \subset x$, size $\min(n, \psi)$
 4:     for each $x_i \in x$ do
 5:         Compute path length $h_t(x_i) \leftarrow \text{Isolate}(x^{(t)}, x_i)$
 6:         Accumulate $\ell(x_i) \leftarrow \ell(x_i) + h_t(x_i)$
 7:     end for
 8: end for
 9: Normalize path lengths: $\ell(x_i) \leftarrow \ell(x_i)/T$
10: Compute expected path length $c(\psi)$
11: Compute anomaly scores: $s(x_i) = 2^{-\frac{\ell(x_i)}{c(\psi)}}$

12: // Contamination thresholding step
13: Sort scores $s(x_i)$ in descending order
14: Let $\tau$ be the $100 \cdot (1 - \pi)$-th percentile of the scores
15: for each $x_i \in x$ do
16:     if $s(x_i) > \tau$ then
17:         Label $x_i$ as an outlier
18:     else
19:         Label $x_i$ as an inlier
20:     end if
21: end for
22: return Anomaly labels for each $x_i$
```

Decision Trees and Random Forests: Example

- Random forests are used to identify which social programs promote better financial performance and improved social conditions.
- The predictions from the decision trees are combined into a final prediction of financial outcomes.
- The combined result is the "random forest"



INTERNATIONAL JOURNAL OF SUSTAINABLE DEVELOPMENT & WORLD ECOLOGY
2020, VOL. 27, NO. 5, 389–395
https://doi.org/10.1080/13504509.2019.1706059

Taylor & Francis
Taylor & Francis Group

OPEN ACCESS Check for updates

Which social program supports sustainable grass-root finance? Machine-learning evidence

R. Gonzales Martinez

Handelshøyskolen, University of Agder, Kristiansand, Norway

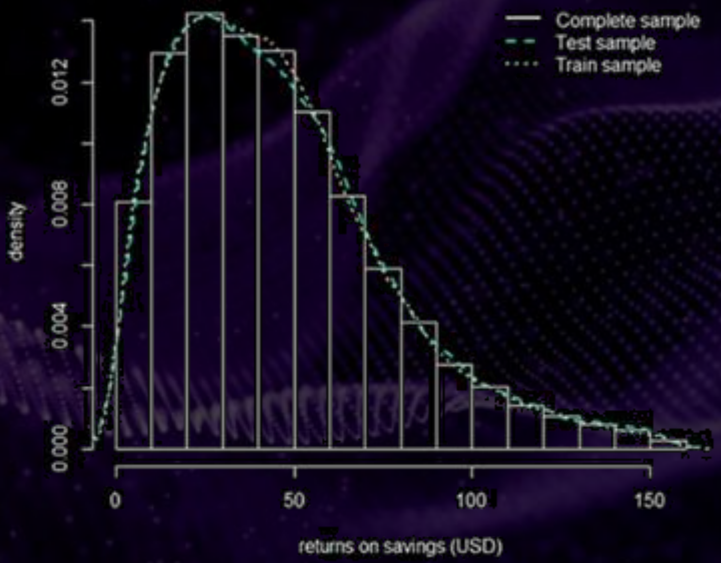
ABSTRACT
Resources for development are used efficiently when social programs help to promote at the same time the sustainability of grass-root financial associations at the bottom of the pyramid. This study applies machine-learning to a worldwide database of grass-root associations in order to identify which social programs are good predictors of financial returns in the groups. The results indicate that education, income-generating activities and health programs are the most frequent programs provided by development agencies. Business training is not the most frequent intervention applied to grass-root associations, but it is in fact the most important social program to encourage financial sustainability, particularly after a development agency stops working with a group and leaves the community. Theoretical and practical implications of the findings are discussed.

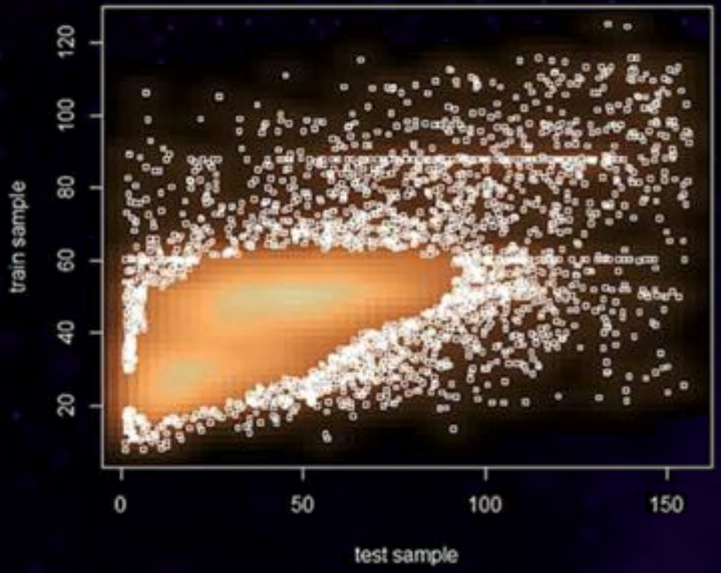
ARTICLE HISTORY
Received 7 November 2019
Accepted 13 December 2019

KEYWORDS
Grass-root finance; health and business interventions; sustainable development; machine-learning

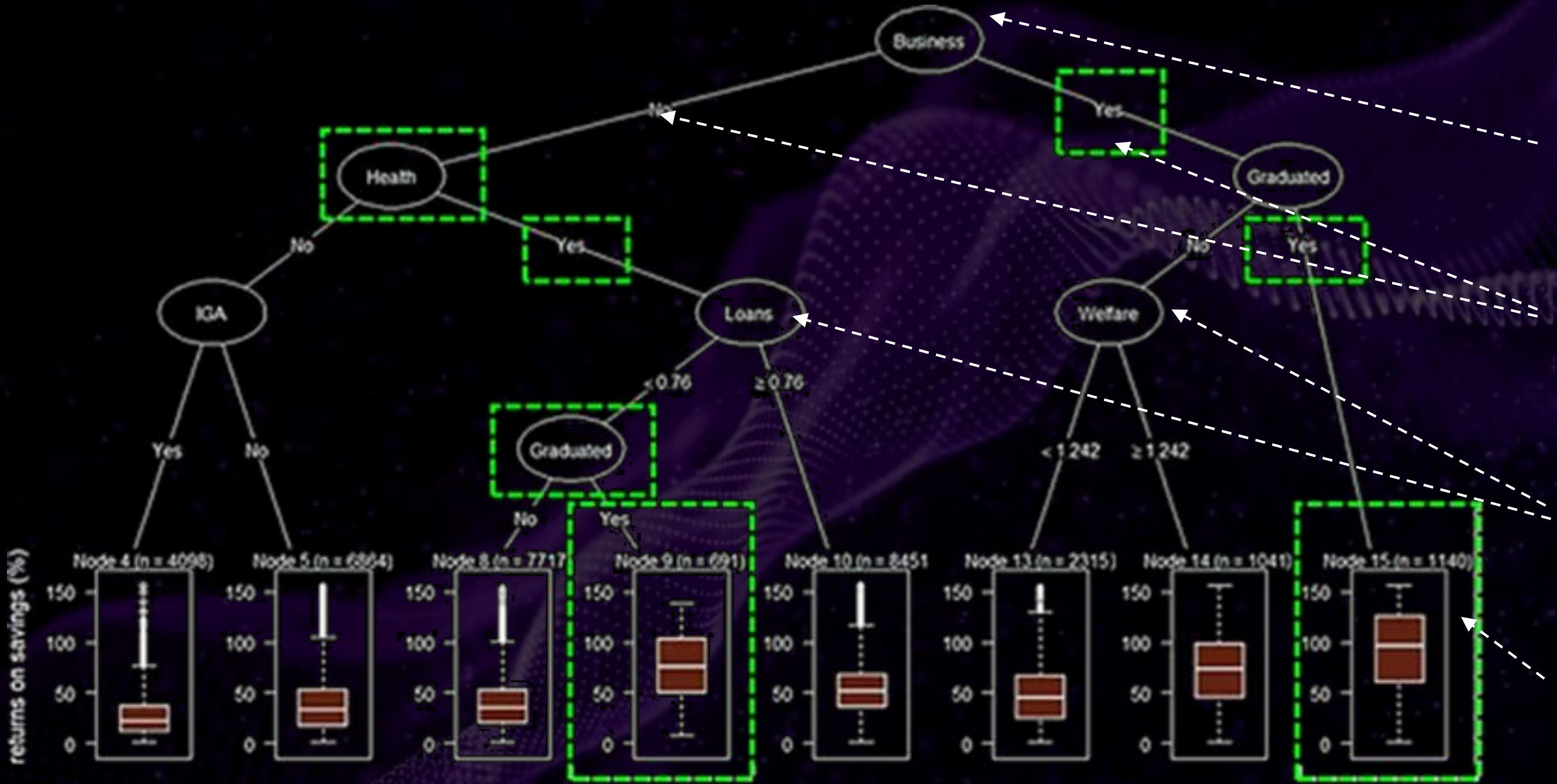
Decision Trees and Random Forests: Example

- Cross-validation: the database was randomly divided into a training set and a validation set.
- 65% of the data in the training set.
- The root mean square error (RMSE) and mean absolute percentage error (MAPE) were evaluated.





Decision Trees and Random Forests: Example



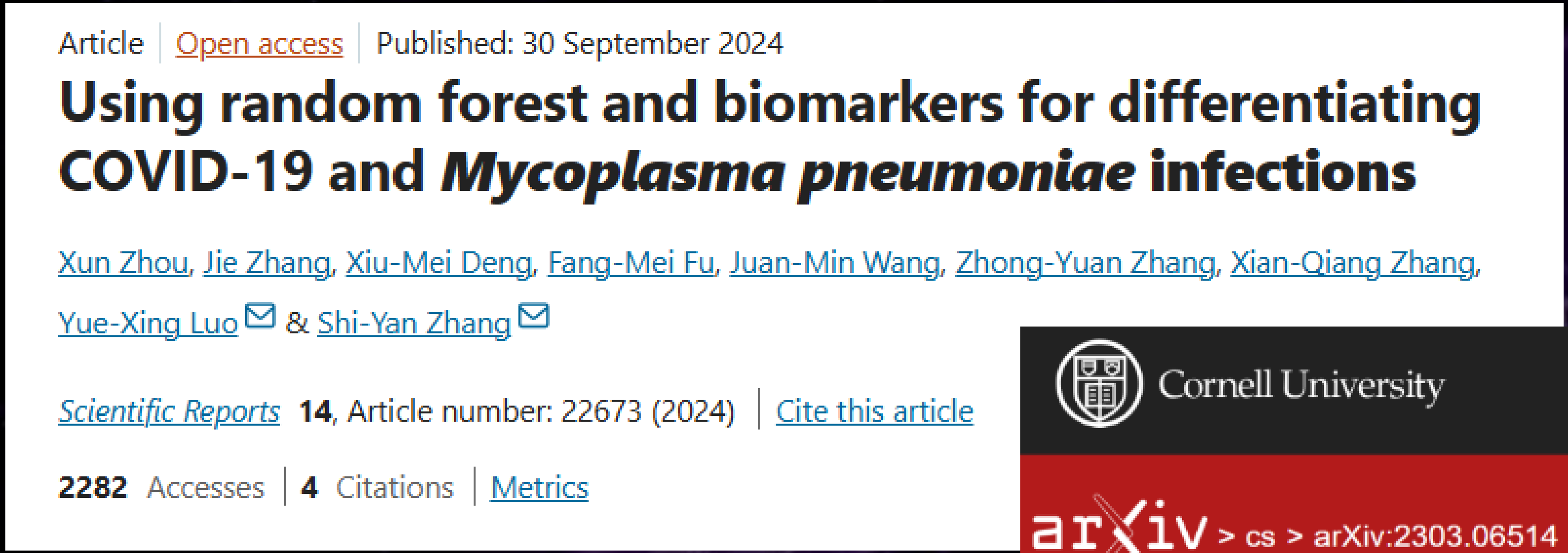
Root: The top node of the tree where data splitting begins.

Branches: Represent the divisions.

Nodes: Represent splitting options for further divisions.

Leaves: Represent the final decisions or classifications.

Decision Trees and Random Forests: Example



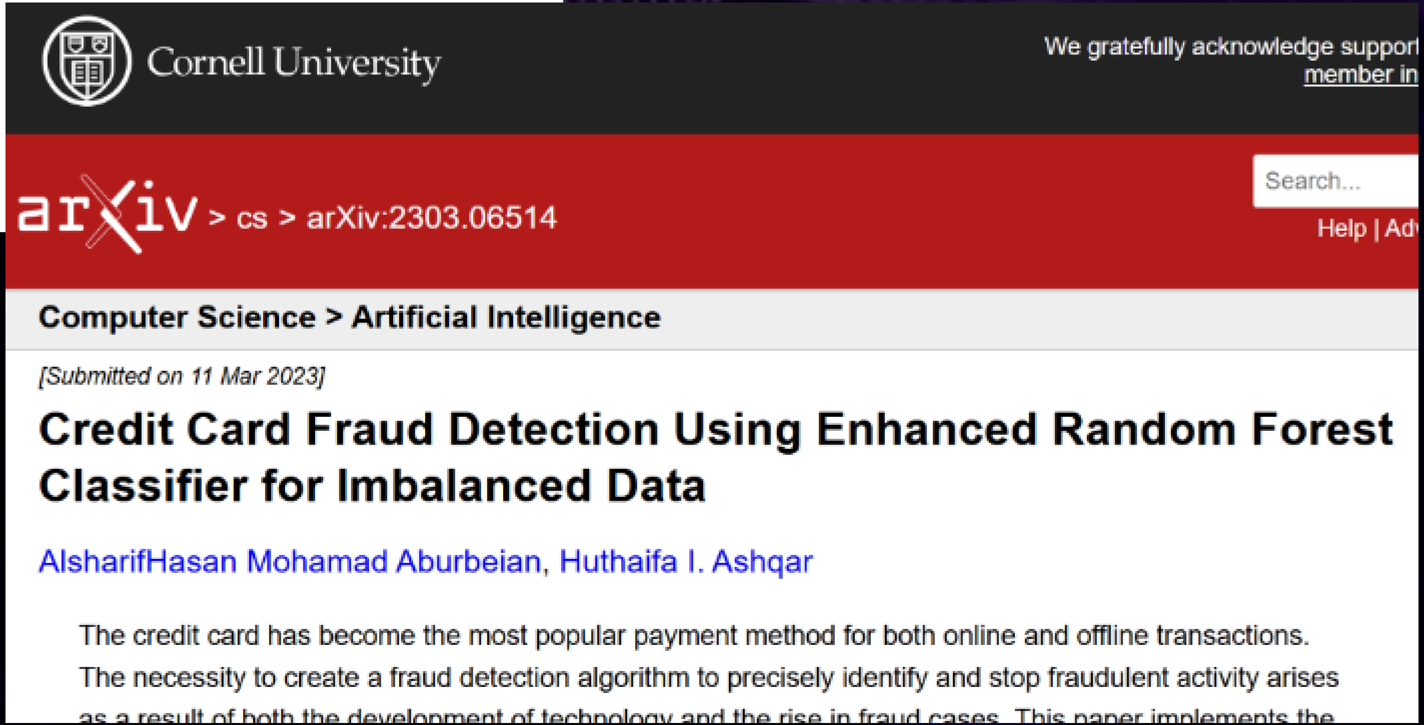
Article | Open access | Published: 30 September 2024

Using random forest and biomarkers for differentiating COVID-19 and *Mycoplasma pneumoniae* infections

Xun Zhou, Jie Zhang, Xiu-Mei Deng, Fang-Mei Fu, Juan-Min Wang, Zhong-Yuan Zhang, Xian-Qiang Zhang, Yue-Xing Luo & Shi-Yan Zhang

Scientific Reports **14**, Article number: 22673 (2024) | Cite this article

2282 Accesses | **4** Citations | Metrics



Cornell University

We gratefully acknowledge support
member in

arXiv > cs > arXiv:2303.06514

Search...

Help | Adv

Computer Science > Artificial Intelligence

[Submitted on 11 Mar 2023]

Credit Card Fraud Detection Using Enhanced Random Forest Classifier for Imbalanced Data

AlsharifHasan Mohamad Aburbeian, Huthaifa I. Ashqar

The credit card has become the most popular payment method for both online and offline transactions. The necessity to create a fraud detection algorithm to precisely identify and stop fraudulent activity arises as a result of both the development of technology and the rise in fraud cases. This paper implements the

XGBoost (Extreme Gradient Boosting)

- **XGBoost** aims to minimize an objective function that combines the loss function with a regularization term across $t = 1,2, \ldots, T$ trees in a model to obtain estimates of Θ (parameters and hyperparameters, such as leaf weights and penalties).
- A **second-order Taylor expansion** is applied to approximate the loss function, using a gradient g (partial derivative with respect to the prediction) and a Hessian (second partial derivative with respect to the prediction).

$$\mathrm{Obj}(\Theta) = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{t=1}^{T} \Omega(f_t),$$

$$L(y_i, \hat{y}_i^{(t)}) \approx L(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \tfrac{1}{2} h_i [f_t(x_i)]^2,$$

$$g_i = \left.\frac{\partial L(y_i, \hat{y})}{\partial \hat{y}}\right|_{\hat{y}=\hat{y}_i^{(t-1)}}, \quad h_i = \left.\frac{\partial^2 L(y_i, \hat{y})}{\partial \hat{y}^2}\right|_{\hat{y}=\hat{y}_i^{(t-1)}},$$

$$\mathrm{Obj}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \tfrac{1}{2} h_i [f_t(x_i)]^2 \right] + \Omega(f_t).$$

XGBoost (Extreme Gradient Boosting)

- In the regularization function: T is the number of trees, w is the weight assigned to the leaves, γ penalizes the depth of the tree and the penalty λ prevents the leaves from growing too much.
- Each tree node is split to maximizes gain.
- In the predictions η is the learning rate that scales the contribution of each new tree to avoid overfitting.

$$\Omega(f_t) = \gamma T + \tfrac{1}{2}\lambda \textstyle\sum_{j=1}^{T} w_j^2$$

$$\text{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

G_L, H_L: sums of gradients/hessians in the left child

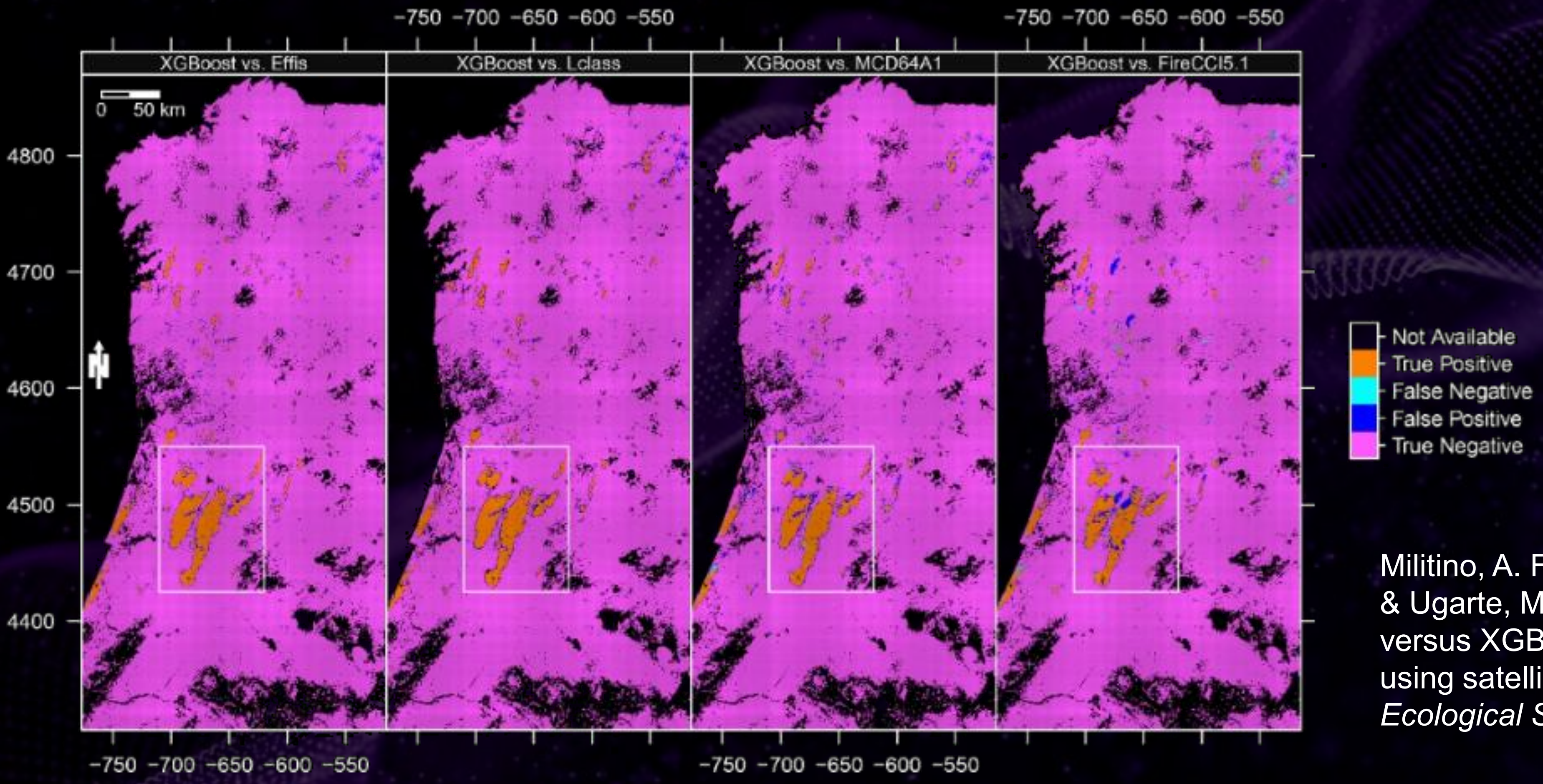
G_R, H_R: same for right child

γ: regularization term (minimum gain required to split)

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + \eta f_t(x_i)$$

XGBoost (Extreme Gradient Boosting)

From: Logistic regression versus XGBoost for detecting burned areas using satellite images



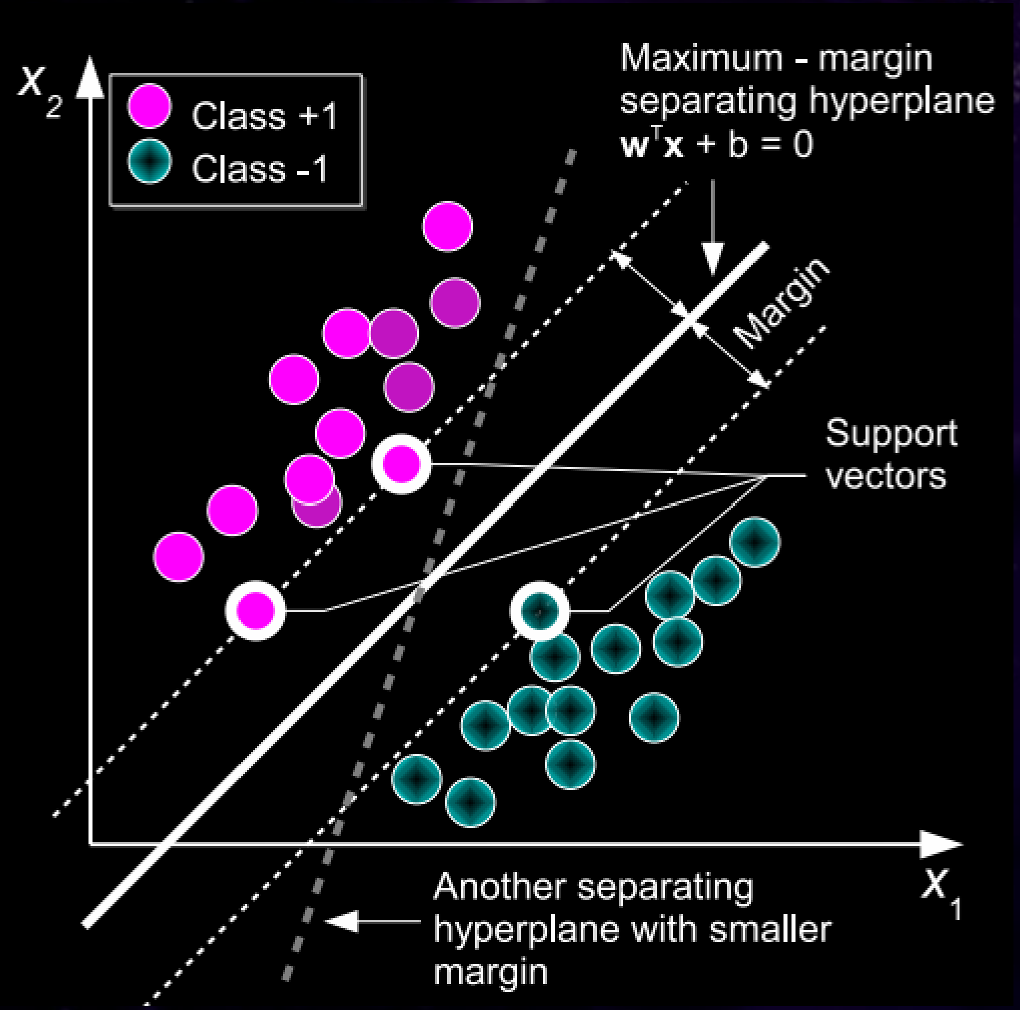
Militino, A. F., Goyena, H., Pérez-Goya, U., & Ugarte, M. D. (2024). Logistic regression versus XGBoost for detecting burned areas using satellite images. *Environmental and Ecological Statistics*, *31*(1), 57-77.

Support Vector Machine

- Instead of simply finding a line that separates the classes, SVM seeks the line that not only separates them but does so in such a way that the distance from the line to the closest points (from both classes) is as wide as possible.

- The margin is the band around the (hyper)plane within which no data point is located (ideally).

- The support vectors are the points on the margin that define the position of the (hyper)plane.


X_2
Class +1
Class -1
Maximum - margin separating hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$
Margin
Support vectors
Another separating hyperplane with smaller margin
X_1

Support Vector Machine

Formally:

- The objective of SVM is to maximize the margin, subject to constraints.
- The constraints are incorporated into the objective function using Lagrange multipliers for n-data, with Karush-Kuhn-Tucker (KKT) conditions.
- The objective function has a dual form, subject to constraints, and is solved using numerical methods.

$$\{(x_i, y_i)\}_{i=1}^{n}, \quad \text{where } x_i \in \mathbb{R}^d, \quad y_i \in \{-1, 1\}$$

Hyperplane equation: $\mathbf{w}\ \mathbf{x} + b = 0$

Margin (distance to the closest point): $\dfrac{1}{\|\mathbf{w}\|}$

Objective function (to minimize): $\min\limits_{\mathbf{w},b} \quad \dfrac{1}{2}\|\mathbf{w}\|^2$

Constraint (for all i): $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad \text{for all } i$

Support Vector Machine

Formally:

- The objective of SVM is to maximize the margin, subject to constraints.
- The constraints are incorporated into the objective function using Lagrange multipliers with Karush-Kuhn-Tucker (KKT) conditions (for n-data points)
- The objective function has a dual form, subject to constraints, and is solved using numerical methods.

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i(\mathbf{w} \ \mathbf{x}_i + b) - 1\right]$$

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \Rightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$\alpha_i \left[y_i(\mathbf{w} \ \mathbf{x}_i + b) - 1\right] = 0 \qquad \alpha_i \geq 0$$

$$\max_{\alpha} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \quad \alpha_i \geq 0$$

Support Vector Machine

- The values of α define which data points become support vectors; they are used to calculate the margin, the bias, and to classify the data into classes.

- In problems that are not linearly separable, the "kernel trick" is used: a non-linear kernel function replaces the optimization function in the dual form.

- In problems with some degree of class overlap, a penalty term controlled by the hyperparameter C is added.

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$\text{Margin} = \frac{1}{\|\mathbf{w}\|}$$

$$b = y_i - \mathbf{w}\,\mathbf{x}_i$$

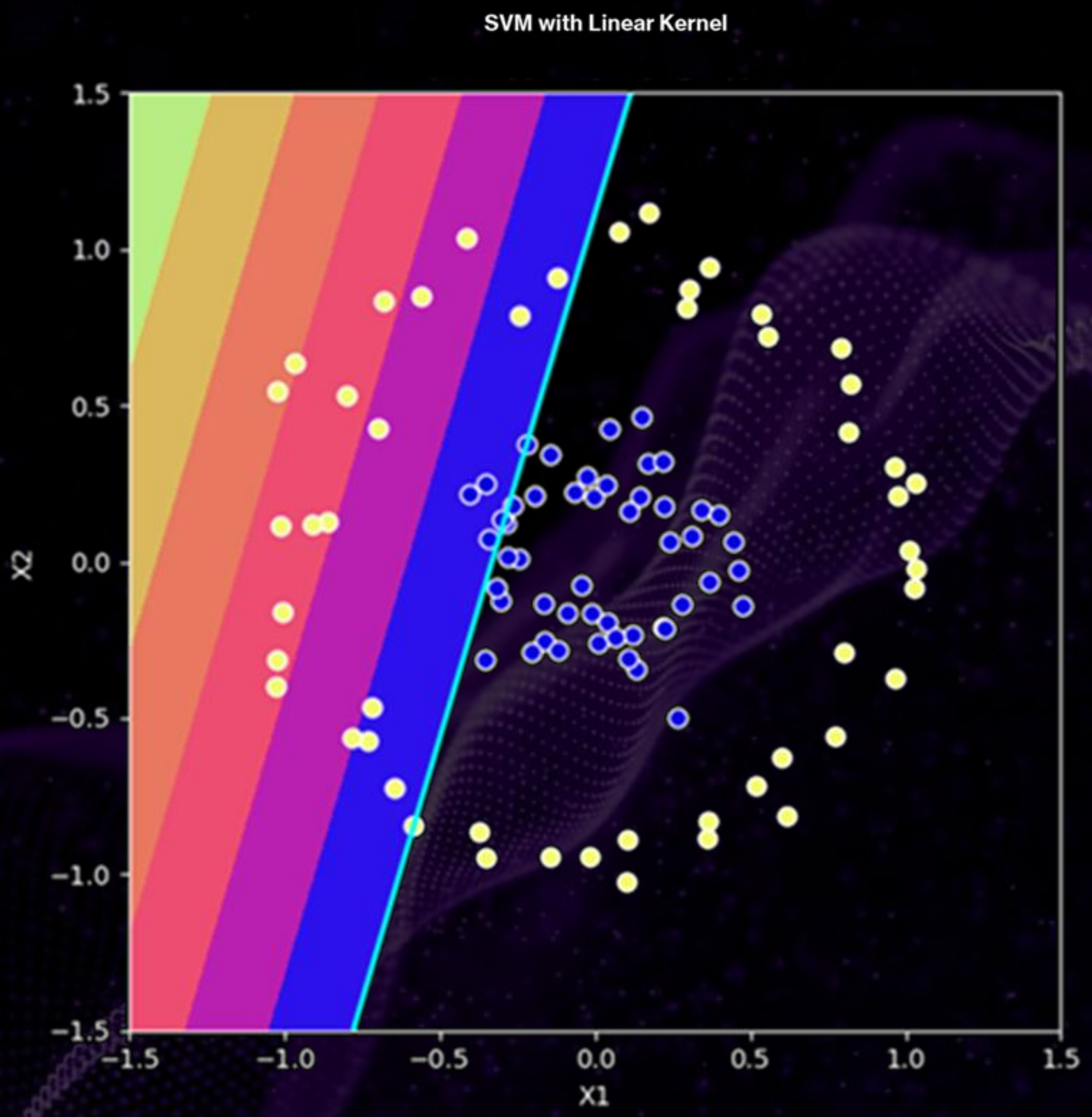
$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + b\right)$$

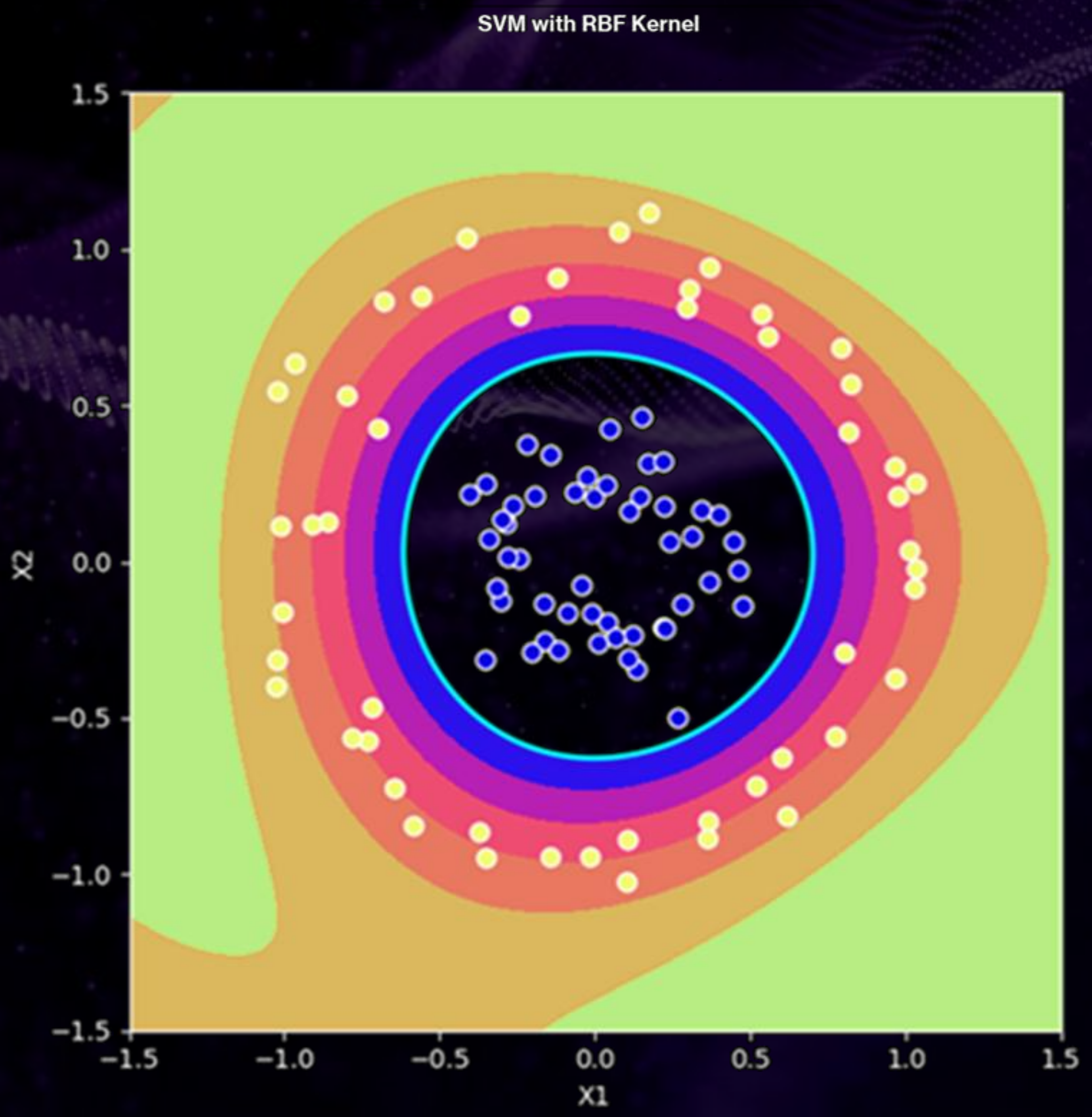
$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$$

$$\min_{\mathbf{w}, b, \xi} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \xi_i$$

Support Vector Machine





Soft Support Vector Machines (Soft SVM)

The hyperparameter **C** controls the trade-off between maximizing the margin and minimizing classification errors:

High C:

- The margin will be narrower.
- Better classification.
- Could result in overfitting.

Low C:

- The margin will be wider.
- Some data points will be misclassified or fall within the margin.
- A low C introduces more tolerance to errors and can improve the model's ability to generalize to unseen data.
- Could result in underfitting if C is too low.

EVALUATION METRICS of machine learning models

Continuous target variables

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

$$\mathrm{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

$$\mathrm{EVS} = 1 - \frac{\mathrm{Var}(y - \hat{y})}{\mathrm{Var}(y)}$$

$$\mathrm{MBD} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)$$

Sign shows direction (positive → overestimation, negative → underestimation).

Confusion matrix

In classification problems (of discrete variables), metrics based on the **confusion matrix** can be used to compare and select machine learning models

	Positive Prediction	**Negative Prediction**
Positive Class	True Positive (TP)	False Negative (FN)
Negative Class	False Positive (FP)	True Negative (TN)

Metrics based on the confusion matrix

Accuracy is defined as the proportion of correct predictions (both positive and negative) out of the total predictions made.

In the case of imbalanced data, other measures such as the F1 score and sensitivity (recall) are more appropriate

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

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

Metrics based on the confusion matrix

- Precision (Positive Predictive Value): The proportion of correctly classified positive examples among all classified as positive.

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

- Sensitivity (Recall, True Positive Rate, TPR): The proportion of positive examples correctly identified among all examples that are actually positive.

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

- Specificity (Specificity or True Negative Rate, TNR): The proportion of negative examples correctly identified among all examples that are actually negative.

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Metrics based on the confusion matrix

- **Negative Predictive Value:** The proportion of negatives among all examples that were classified as negative.
- **F1 Score:** The harmonic mean of precision and sensitivity.
- **False Positive Rate:** The proportion of negative examples that were incorrectly classified as positive.

$$\text{Negative Predictive Value} = \frac{TN}{TN + FN}$$

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}}$$

$$\text{False Positive Rate} = \frac{FP}{FP + TN}$$

Metrics based on the confusion matrix

Balanced accuracy, prevalence (percentage of positive cases), detection prevalence (cases predicted as positive), detection rate (cases correctly predicted as positive)

$$\text{Balanced Accuracy} = \frac{\text{Sensitivity} + \text{Specificity}}{2}$$

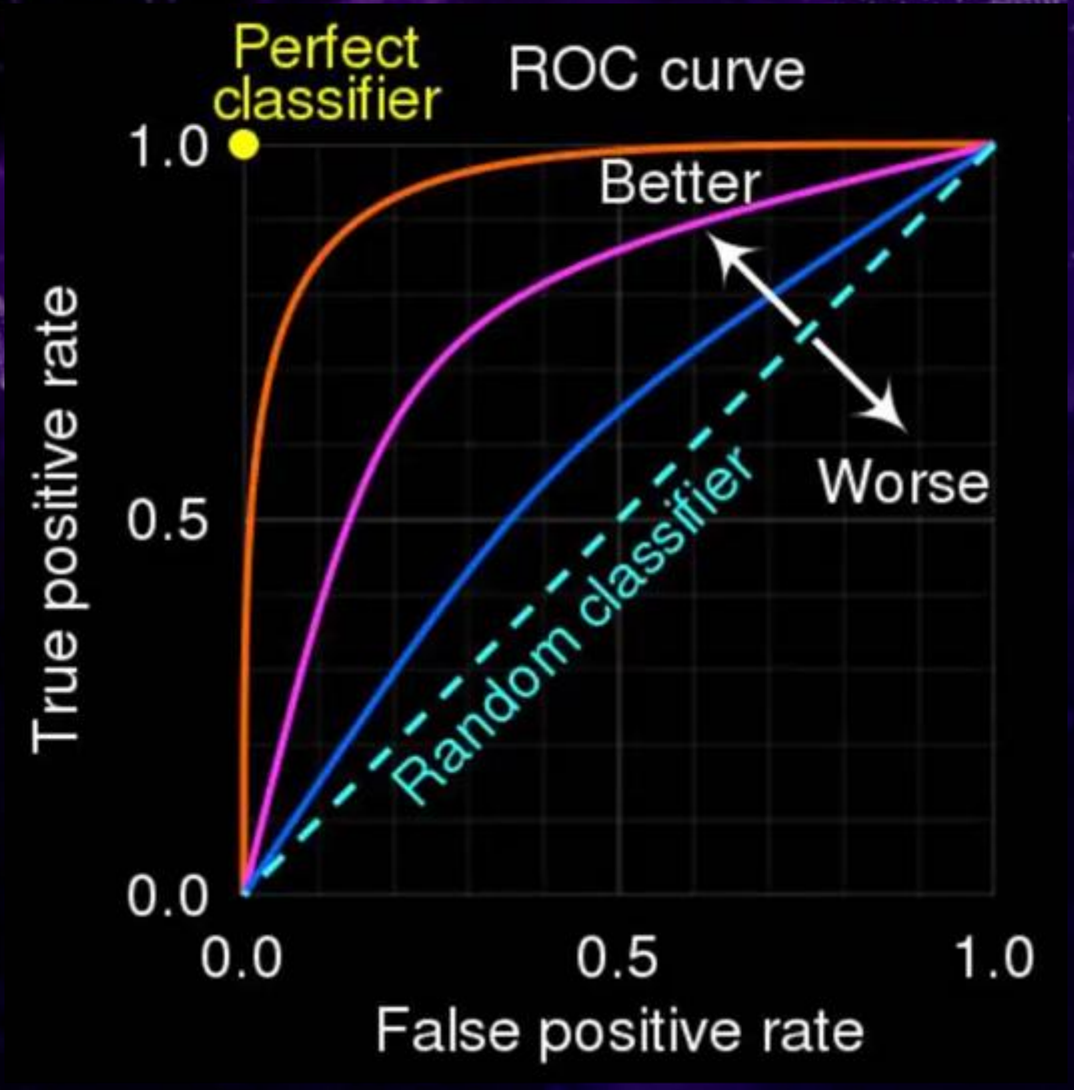
$$\text{Prevalence} = \frac{TP + FN}{TP + TN + FP + FN}$$

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

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

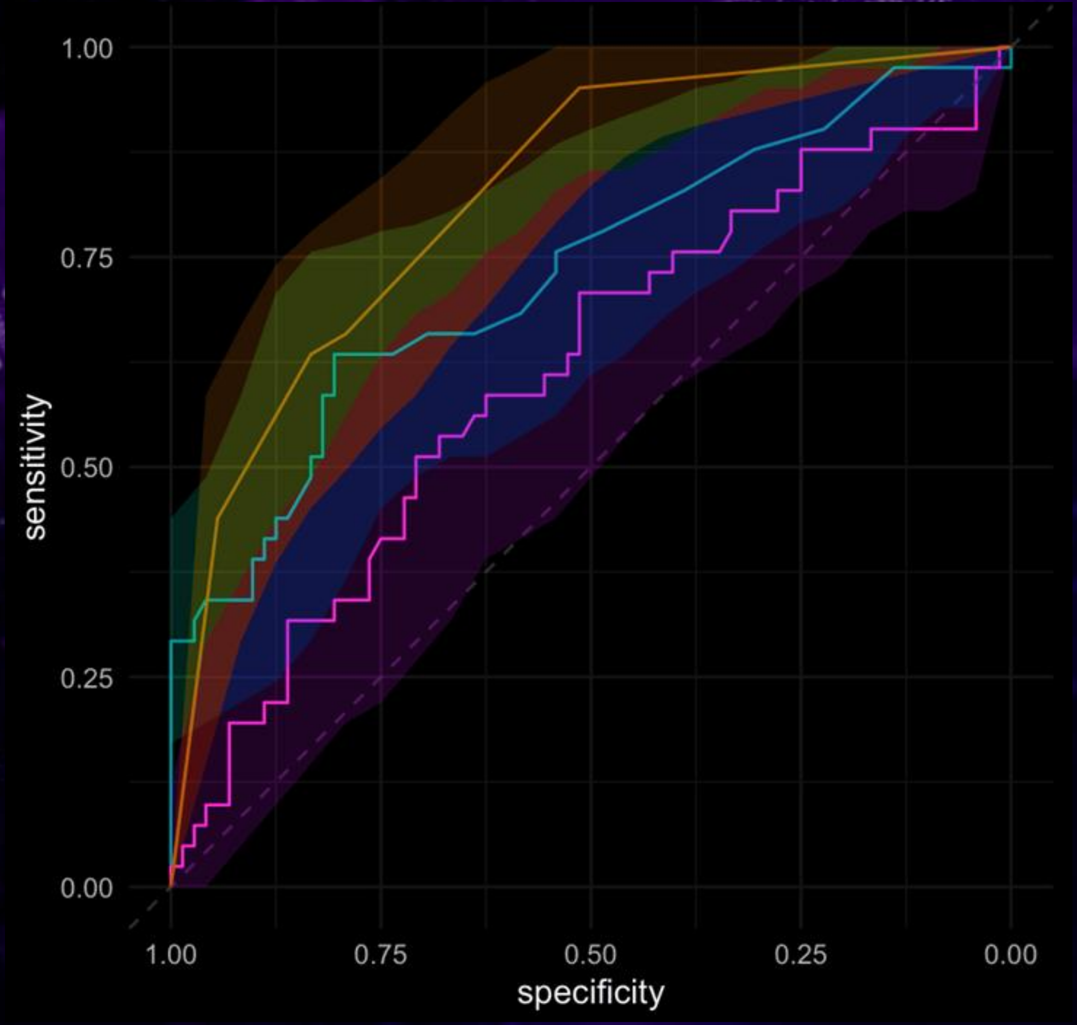
ROC (Receiver Operating Characteristic) curve and validation of ML models

- ROC: graphical representation of the ability of a classification model to discriminate between classes.
- It shows the relationship between the True Positive Rate (TPR) and the False Positive Rate (FPR) for different cutoff thresholds of the predicted probability:
  - X-axis (False Positive Rate, FPR): proportion of negatives that are incorrectly classified as positive.
  - Y-axis (true positive rate, TPR or Sensitivity): Proportion of positives correctly identified by the model.



ROC (Receiver Operating Characteristic) curve and validation of ML models

- The performance of an ML model depends on the partitioning of the data set used for training.
- Cross-validation and hyper-tuning techniques allow to evaluate and control the stability and generalization of the model to give a robust estimate of the discriminative/predictive ability of ML models.



FEATURE SELECTION ALGORITHMS: Boruta

Boruta

- The Boruta algorithm identifies the relevant features in a dataset by creating shadow features.
- For each original feature, Boruta compares its importance with the highest importance observed among the shadow features.
- Boruta performs a hypothesis test to determine if the importance of the original feature is significantly higher than the maximum importance of the shadows (null hypothesis: the feature is irrelevant compared to noise).

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}, \quad x_i \in \mathbb{R}^p, \quad y_i \in \mathbb{R} \text{ (or } \{0,1\} \text{ for classification)}$$

$$X = [X_1, X_2, \ldots, X_p] \in \mathbb{R}^{n \times p}$$

$$X'_j = \text{Permutation}(X_j) \qquad X' = \{X'_1, X'_2, \ldots, X'_k\}$$

$$X_{\text{ext}} = X \cup X' = \{X_1, \ldots, X_k, X'_1, \ldots, X'_k\}$$

$$I_{\text{shadow}}^{\max} = \max\{I(X'_1), I(X'_2), \ldots, I(X'_k)\}$$

$$H_0 : I(X_j) \leq I_{\text{shadow}}^{\max}$$

$$p_j = \frac{\#\{\text{permutations where } I(X_j) \leq I_{\text{shadow}}^{\max}\}}{T}$$

If $p_j < \alpha$ (e.g., 0.01), we reject H_0 and declare X_j important.

Boruta

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}, \quad x_i \in \mathbb{R}^p, \quad y_i \in \mathbb{R} \text{ (or } \{0,1\} \text{ for classification)}$$

$$X = [X_1, X_2, \ldots, X_p] \in \mathbb{R}^{n\times p}$$

$$X'_j = \text{Permutation}(X_j) \qquad X' = \{X'_1, X'_2, \ldots, X'_k\}$$

$$X_{\text{ext}} = X \cup X' = \{X_1, \ldots, X_k, X'_1, \ldots, X'_k\}$$

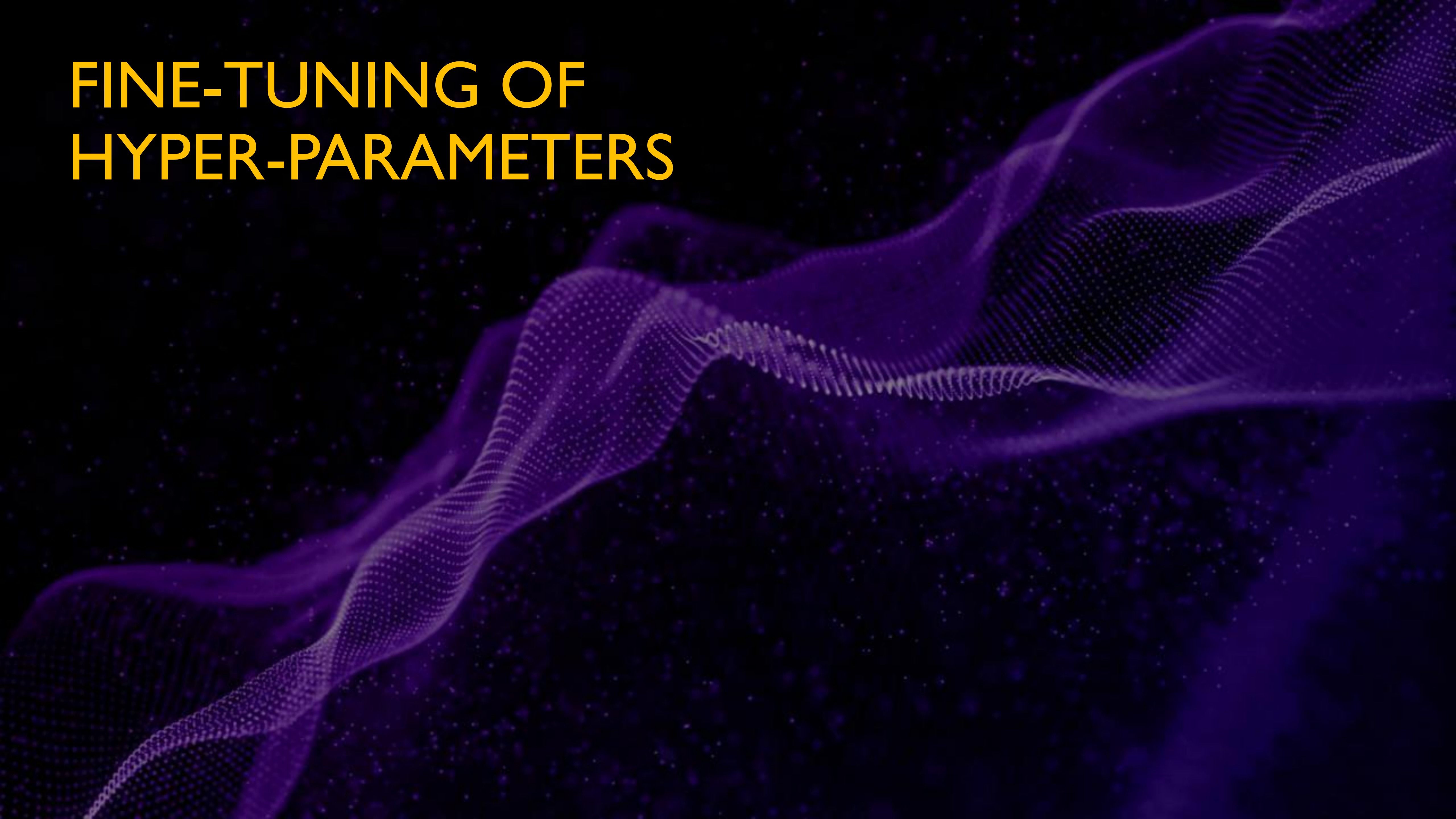
$$I_{\text{shadow}}^{\max} = \max\{I(X'_1), I(X'_2), \ldots, I(X'_k)\}$$

$$H_0 : I(X_j) \le I_{\text{shadow}}^{\max}$$

$$p_j = \frac{\#\{\text{permutations where } I(X_j) \le I_{\text{shadow}}^{\max}\}}{T}$$

If $p_j < \alpha$ (e.g., 0.01), we reject H_0 and declare X_j important.

FINE-TUNING OF HYPER-PARAMETERS



GridSearchCV: Cross-validation with grid search

- GridSearchCV is used to optimize the hyperparameters of a machine learning model.
- GridSearchCV performs an exhaustive search through a specified set of hyperparameter values to find the best combination that maximizes the predictive performance of the model.

$$D = \{(x_i, y_i)\}_{i=1}^{n}$$

$$M(\theta) \qquad \theta = (\theta_1, \theta_2, \ldots, \theta_m)$$

$$\mathcal{L}(M, D)$$

$$\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_m$$

$$\mathcal{S} = \{\theta_1^{(1)}, \theta_1^{(2)}, \ldots\} \times \{\theta_2^{(1)}, \theta_2^{(2)}, \ldots\} \times \cdots \times \{\theta_m^{(1)}, \theta_m^{(2)}, \ldots\}$$

$$D_1, D_2, \ldots, D_k$$

$$\theta = (\theta_1, \theta_2, \ldots, \theta_m) \in \mathcal{S}$$

$$\text{CV\_score}(\theta) = 0$$

$$D_{\text{train}}^{(i)} = D \setminus D_i \qquad D_{\text{val}}^{(i)} = D_i$$

$$\text{score}_i(\theta) = \mathcal{L}(M(\theta), D_{\text{val}}^{(i)})$$

$$\text{CV\_score}(\theta) \mathrel{+}= \frac{1}{k}\text{score}_i(\theta)$$

$$\theta^* = \arg\min_{\theta \in \mathcal{S}} \text{CV\_score}(\theta)$$

GridSearchCV: Cross-validation with grid search

- S is the set of hyperparameter values (the set of all possible combinations of hyperparameters to be evaluated).
- S represents the grid where each dimension corresponds to a different hyperparameter and is obtained as the Cartesian product of the S-sets of the m hyperparameters to be evaluated.

$$D = \{(x_i, y_i)\}_{i=1}^{n}$$

$$M(\theta) \qquad \theta = (\theta_1, \theta_2, \ldots, \theta_m)$$

$$\mathcal{L}(M, D)$$

$$\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_m$$

$$\mathcal{S} = \{\theta_1^{(1)}, \theta_1^{(2)}, \ldots\} \times \{\theta_2^{(1)}, \theta_2^{(2)}, \ldots\} \times \cdots \times \{\theta_m^{(1)}, \theta_m^{(2)}, \ldots\}$$

$$D_1, D_2, \ldots, D_k$$

$$\theta = (\theta_1, \theta_2, \ldots, \theta_m) \in \mathcal{S}$$

$$\text{CV\_score}(\theta) = 0$$

$$D_{\text{train}}^{(i)} = D \setminus D_i \qquad D_{\text{val}}^{(i)} = D_i$$

$$\text{score}_i(\theta) = \mathcal{L}(M(\theta), D_{\text{val}}^{(i)})$$

$$\text{CV\_score}(\theta) \mathrel{+}= \frac{1}{k}\text{score}_i(\theta)$$

$$\theta^* = \arg\min_{\theta \in \mathcal{S}} \text{CV\_score}(\theta)$$

GridSearchCV: Cross-validation with grid search

- Cross-validation divides the dataset into k subsets (folds). Then, the model is trained on k-1 of these subsets and validated on the remaining subset.
- This process is repeated k times, changing the validation subset each time, to ensure that every observation is used for both training and validation.

$$D = \{(x_i, y_i)\}_{i=1}^{n}$$

$$M(\theta) \qquad \theta = (\theta_1, \theta_2, \ldots, \theta_m)$$

$$\mathcal{L}(M, D)$$

$$\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_m$$

$$\mathcal{S} = \{\theta_1^{(1)}, \theta_1^{(2)}, \ldots\} \times \{\theta_2^{(1)}, \theta_2^{(2)}, \ldots\} \times \cdots \times \{\theta_m^{(1)}, \theta_m^{(2)}, \ldots\}$$

$$D_1, D_2, \ldots, D_k$$

$$\theta = (\theta_1, \theta_2, \ldots, \theta_m) \in \mathcal{S}$$

$$\text{CV\_score}(\theta) = 0$$

$$D_{\text{train}}^{(i)} = D \setminus D_i \qquad D_{\text{val}}^{(i)} = D_i$$

$$\text{score}_i(\theta) = \mathcal{L}(M(\theta), D_{\text{val}}^{(i)})$$

$$\text{CV\_score}(\theta) \mathrel{+}= \frac{1}{k}\text{score}_i(\theta)$$

$$\theta^* = \arg\min_{\theta \in \mathcal{S}} \text{CV\_score}(\theta)$$

GridSearchCV: Cross-validation with grid search

- The model M is estimated on each training partition of the sample, and predictions are made on the test sample.
- Performance metrics (scores) of the model are obtained on the test samples and averaged to calculate the mean cross-validation score across the k-folds.
- The algorithm returns the best hyperparameters and the optimal model trained on the entire dataset D.

$$D = \{(x_i, y_i)\}_{i=1}^{n}$$

$$M(\theta) \qquad \theta = (\theta_1, \theta_2, \ldots, \theta_m)$$

$$\mathcal{L}(M, D)$$

$$\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_m$$

$$\mathcal{S} = \{\theta_1^{(1)}, \theta_1^{(2)}, \ldots\} \times \{\theta_2^{(1)}, \theta_2^{(2)}, \ldots\} \times \cdots \times \{\theta_m^{(1)}, \theta_m^{(2)}, \ldots\}$$

$$D_1, D_2, \ldots, D_k$$

$$\theta = (\theta_1, \theta_2, \ldots, \theta_m) \in \mathcal{S}$$

$$\text{CV\_score}(\theta) = 0$$

$$D_{\text{train}}^{(i)} = D \setminus D_i \qquad D_{\text{val}}^{(i)} = D_i$$

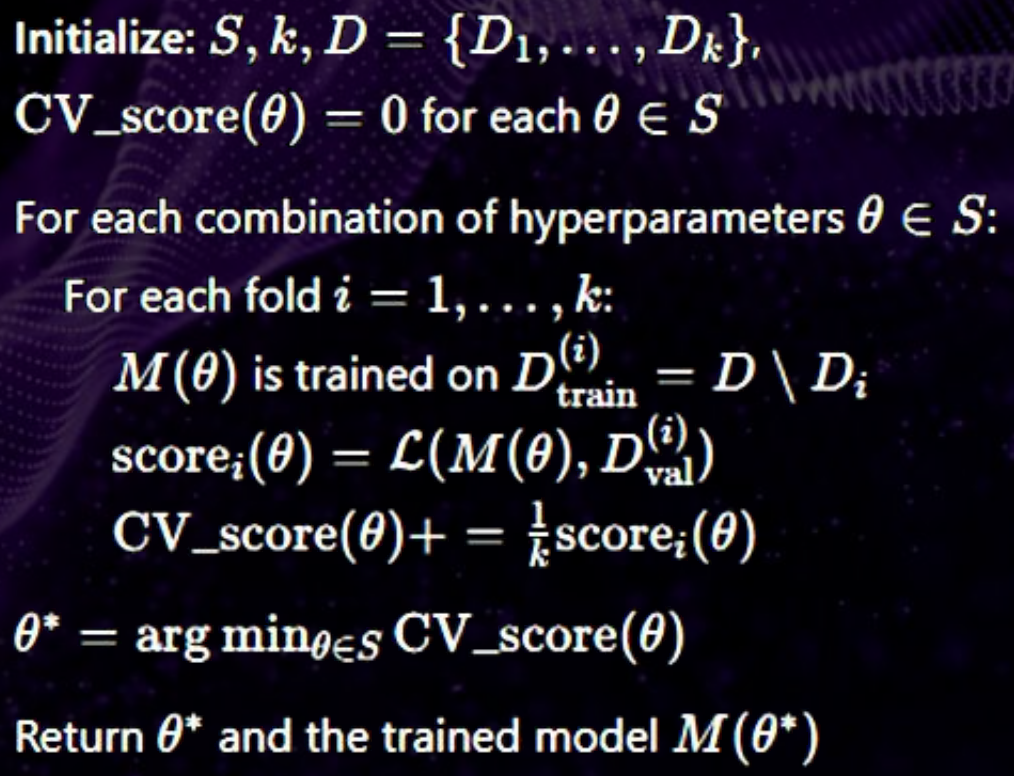
$$\text{score}_i(\theta) = \mathcal{L}(M(\theta), D_{\text{val}}^{(i)})$$

$$\text{CV\_score}(\theta) \mathrel{+}= \frac{1}{k}\text{score}_i(\theta)$$

$$\theta^* = \arg\min_{\theta \in \mathcal{S}} \text{CV\_score}(\theta)$$

GridSearchCV: Cross-validation with grid search

The algorithm returns the best hyperparameters and the optimal model trained on the entire dataset D.



```
Initialize: $S, k, D = \{D_1, \ldots, D_k\}$,
$\mathrm{CV\_score}(\theta) = 0$ for each $\theta \in S$

For each combination of hyperparameters $\theta \in S$:
    For each fold $i = 1, \ldots, k$:
        $M(\theta)$ is trained on $D_{\mathrm{train}}^{(i)} = D \setminus D_i$
        $\mathrm{score}_i(\theta) = \mathcal{L}(M(\theta), D_{\mathrm{val}}^{(i)})$
        $\mathrm{CV\_score}(\theta) + = \frac{1}{k}\mathrm{score}_i(\theta)$

$\theta^* = \arg\min_{\theta \in S} \mathrm{CV\_score}(\theta)$

Return $\theta^*$ and the trained model $M(\theta^*)$
```

RandomizedSearchCV: Cross-validation and hypertuning with non-exhaustive search

Given a Cartesian space H formed by the values of the n-hyperparameters h:

- Randomly select j=1,2,...,m-samples (m-iterations) of hyperparameter values and form tuples.
- Train ML models with the tuples on a training sample in k-partitions (cv).
- Calculate j-performance metrics ℓ on the validation sample.Select the h values that maximize (or minimize) ℓ.
- Fit the model with the complete sample (refit).

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{m}$$

$$\mathcal{H} = \mathcal{H}_1 \times \mathcal{H}_2 \times \cdots \times \mathcal{H}_n$$

$$h^{(j)} = (h_1^{(j)}, h_2^{(j)}, \ldots, h_n^{(j)})$$

$$M(h^{(j)}) \qquad \mathcal{D}_{\text{train}} \subseteq \mathcal{D}$$

$$\ell^{(j)} = \mathcal{L}(M(h^{(j)}), \mathcal{D}_{\text{val}})$$

$$h^* = \arg \max_{j \in \{1,2,\ldots,k\}} \ell^{(j)}$$

$$M(h^*)$$

BayesSearchCV: Cross-validation and hyperparameter tuning with Gaussian processes

Given a **discrete or continuous** space H formed by the values of the n-hyperparameters h:

- A **probabilistic** (surrogate) function is calculated to approximate the objective performance function.
- Values of h are chosen that optimize the surrogate function and iteratively maximize the acquisition function α(h).
- The acquisition function balances:
  - ***Exploration*** of regions in the hyperparameter space.
  - ***Exploitation*** (concentration) in regions with the best hyperparameters (where the difference between surrogates is positive).

$$f(\mathbf{h}) \qquad \{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{n_0}\}$$

$$\mathbf{h} = (h_1, h_2, \ldots, h_n)$$

$$\mathbf{h}^* = \arg\min_{\mathbf{h} \in \mathcal{H}} f(\mathbf{h})$$

$$f(\mathbf{h}^*)$$

$$\alpha(\mathbf{h}) = \mathbb{E}\left[\max(0, f(\mathbf{h}) - f(\mathbf{h}_{\text{best}}))\right]$$

$$\mathcal{GP}(\mu(\mathbf{h}), k(\mathbf{h}, \mathbf{h}'))$$

$$\mu(\mathbf{h}) = \mathbb{E}[f(\mathbf{h})]$$

$$k(\mathbf{h}, \mathbf{h}') = \text{Cov}(f(\mathbf{h}), f(\mathbf{h}'))$$

$$f(\mathbf{h}) \sim \mathcal{N}(\mu(\mathbf{h}), \sigma^2(\mathbf{h}))$$

PRACTICAL APPLICATIONS credit scoring with machine learning

Python scripts for credit scoring with AI algorithms:

- aiw_creditML01.ipynb: Support vector machines vs. logistic model
- aiw_creditML02.ipynb: Comparing credit scoring models based on XGBoost, SVMs, random forests and a logistic model
- aiw_creditML03.ipynb: SVMs with k-fold cross-validation
- aiw_creditML04.ipynb: Feature selection with Boruta
- aiw_creditML05.ipynb: SVM hyper-parameter fine-tuning with GridSearchCV, RandomizedSearchCV, BayesSearchCV