# Implementation of XGBoost for a SUSY tau analysis

I. Slazyk

# Table of Contents

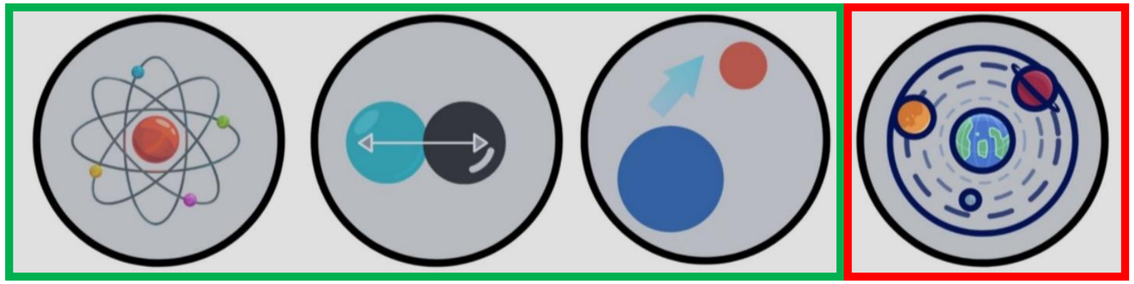
1

# Introduction

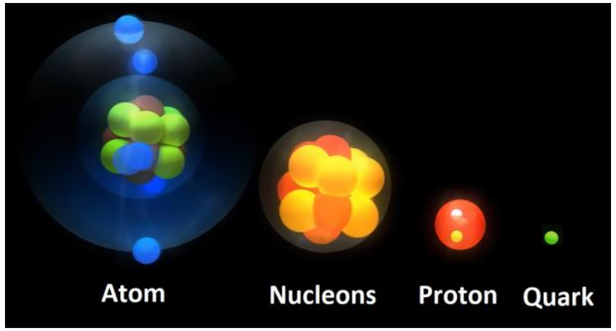
# Standard Model

**Standard Model (SM)** – set of mathematical principles that, with an experimental verification over time, resulted in a physics theory.

Describes the **fundamental forces** (three out of four):



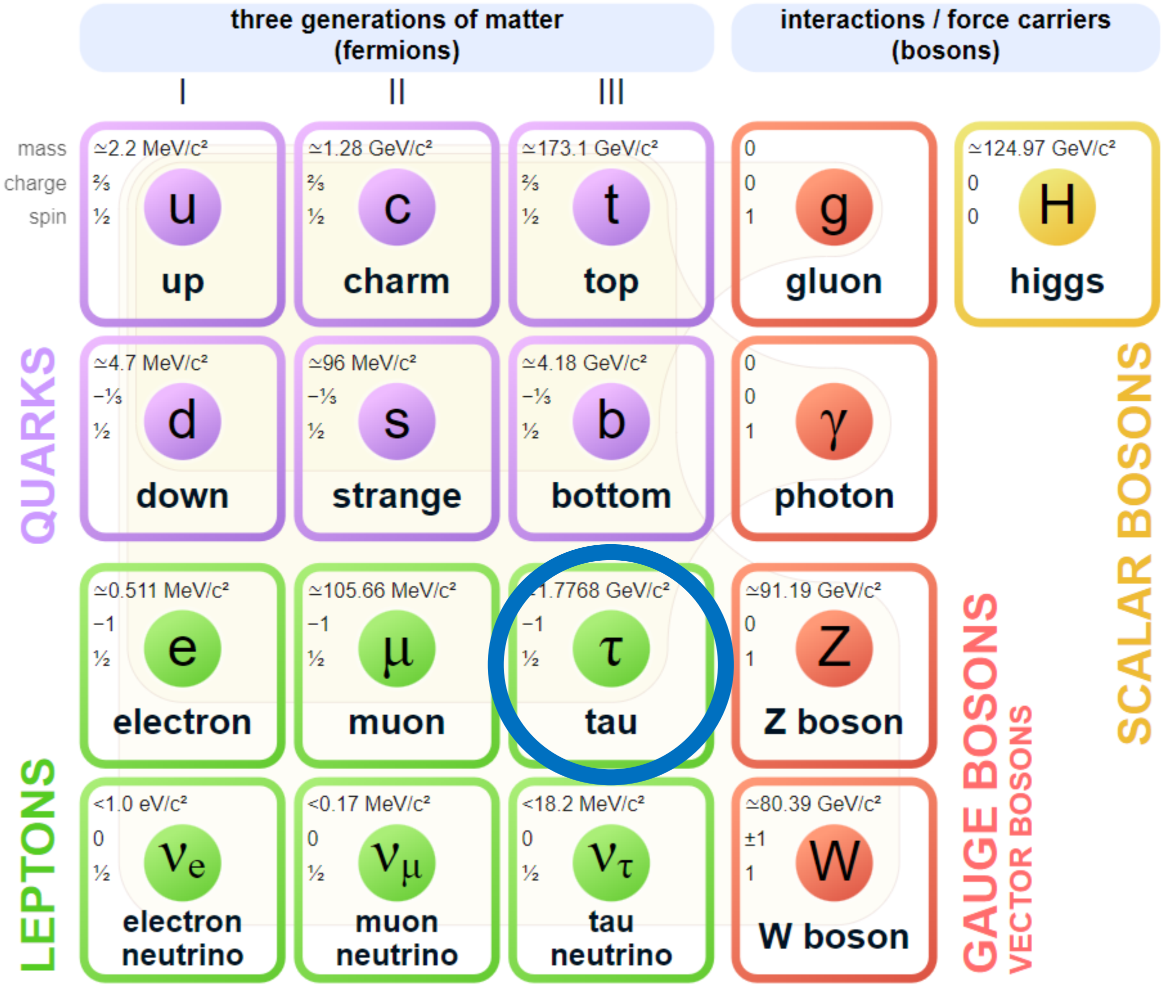
Electromagnetic Strong Weak Gravitational




Still, there are some missing pieces:

- gravity
- neutrino mass
- matter-antimatter asymmetry
- **dark matter**

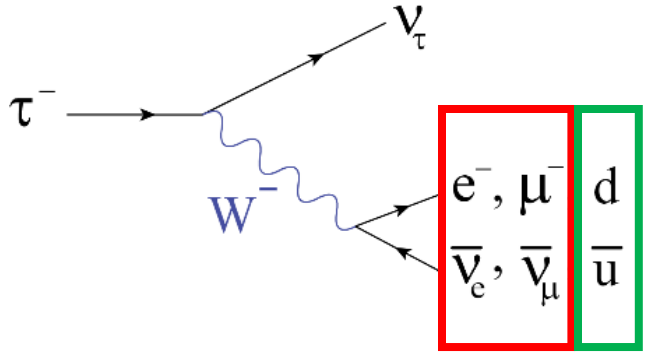
Describes the **elementary particles**:



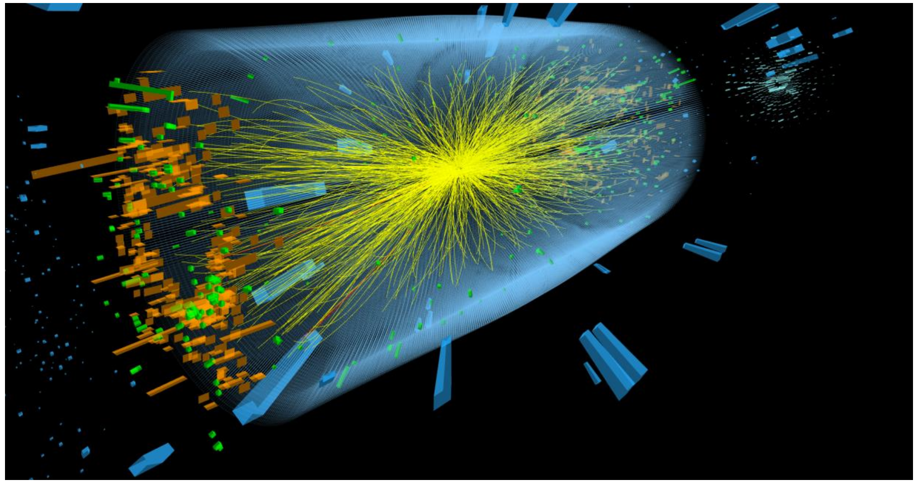
# Tau Lepton

**Tau ($\tau$) lepton** – an elementary particle being the third and last generation of the lepton family.


- Similar to its cousins (electron and muon) but much heavier
- The only lepton decaying leptonically and hadronically



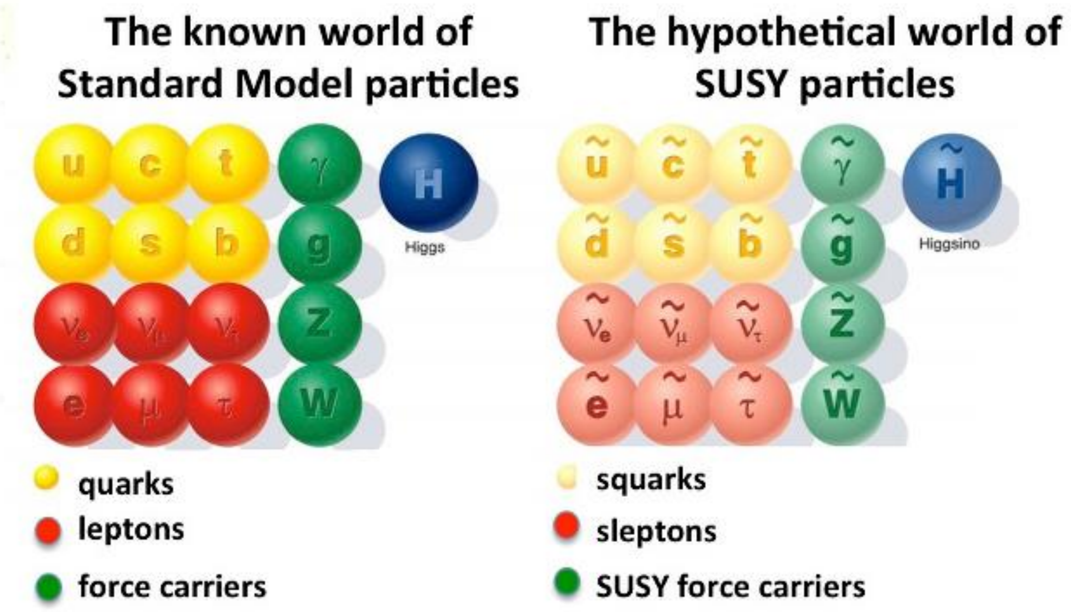

- Hard to detect



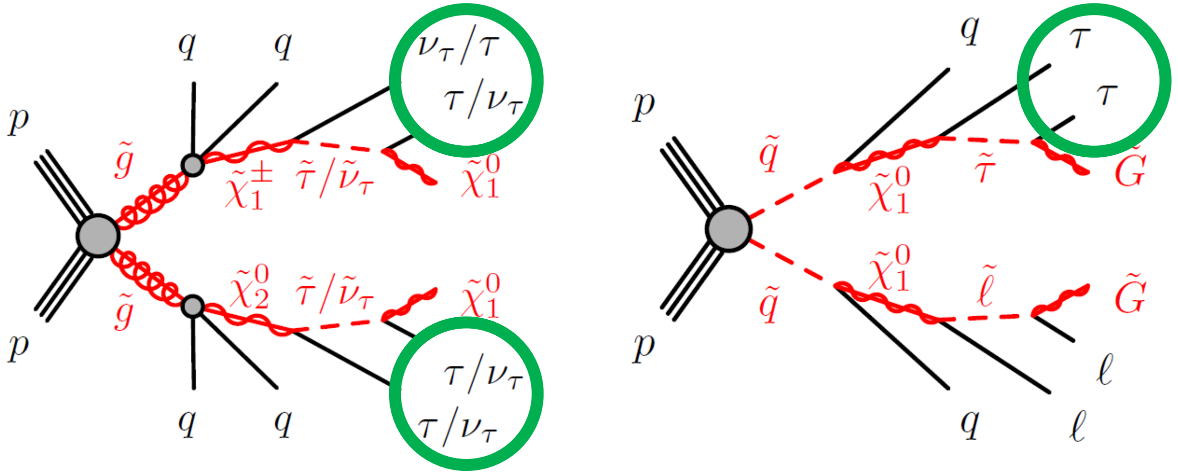
| Name | Tau / Tau Lepton / Tauon |
|---|---|
| Symbol | $\tau^-$ |
| Name | Tau |
| Electric Charge | $-1$ |
| Spin | $1/2$ |
| Lifetime | $2.903 \times 10^{-13}$ s |
| Mass | 1 776.86 MeV/c$^2$ |
| Flight distance | 87.11 $\mu$m |
| Composition | Elementary Particle |
| Type | Fermion |
| Family | Lepton |
| Generation | III |
| Interactions | Gravity, Electromagnetic, Weak |
| Discovered | 1975 |

# Supersymmetry

**Supersymmetry (SUSY)** – an extension of the SM that could provide solutions to some of the unsolved problems by introducing a symmetry between bosons and fermions resulting in a SUSY partner.



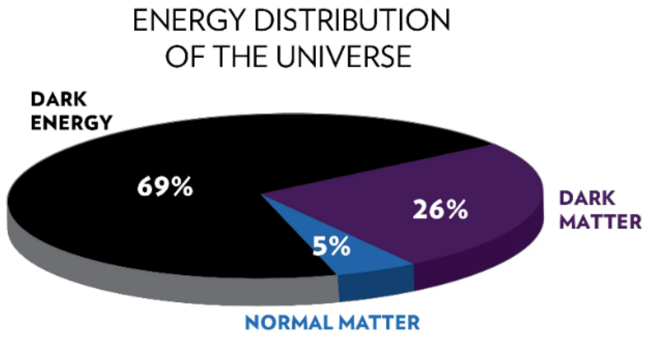

Tau sector is very interesting




Reference: Phys. Rev. D 99, 012009

SUSY could help better understand the Universe:

- Hierarchy problem with Higgs boson (low mass problem)
- Could unite strong force with electroweak force (unify 3 fundamental forces)
- Solution to dark matter



2

# Tau+X Analysis
# Overview

# Previous Analysis

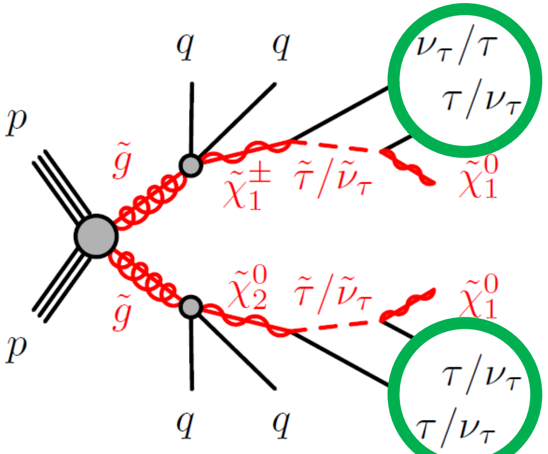
**Last publication:**

Search for squarks and gluinos in final states with hadronically decaying tau leptons, jets, and missing transverse momentum using $pp$ collisions at $\sqrt{s}$=13 TeV with the ATLAS detector.
[Phys. Rev. D 99, 012009]

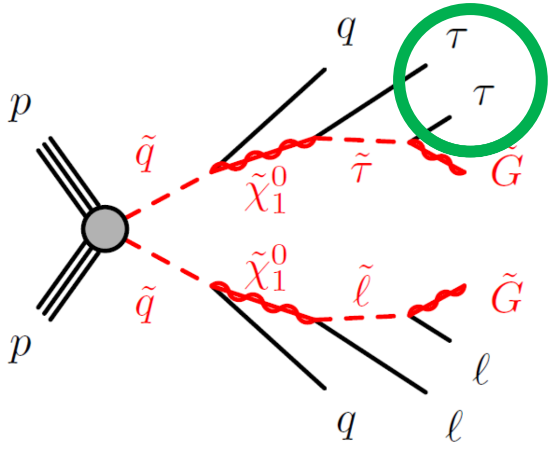
- 36 $\mathrm{fb}^{-1}$ (data: 15+16) in Release 20

**Signal models with $\tilde{\tau}_1$ NLSP:**

- Simplified model of gluino pairs
- Gauge-mediated supersymmetry breaking model




Simplified model of gluino pairs




Gauge-mediated supersymmetry breaking model

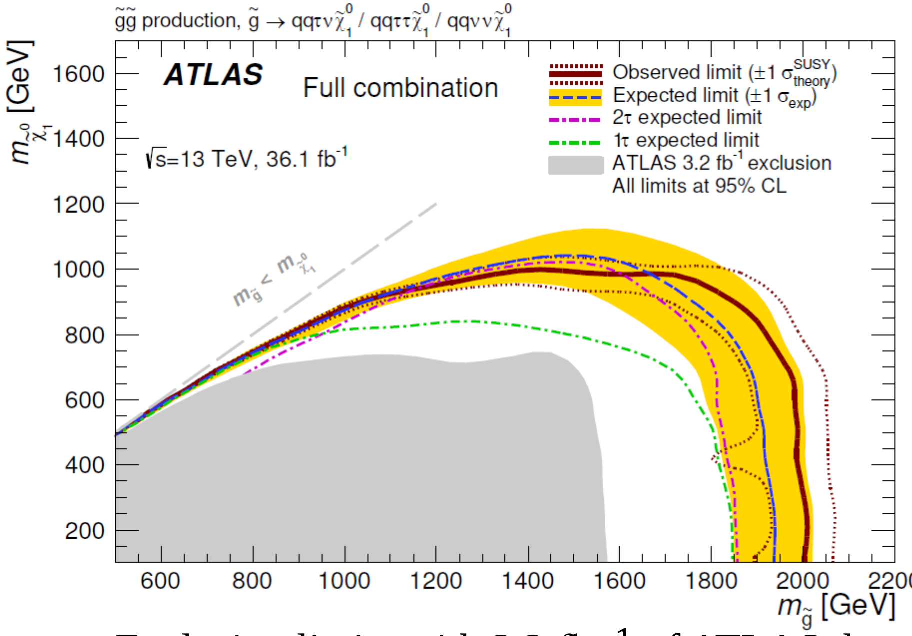
**Channels:**

- $1\tau$ + jets + MET
- $2\tau$ + jets + MET

**Main backgrounds:**

- $\mathbf{1\tau}$: $t\bar{t}$, $W \to \tau\nu$ + jets, $Z \to \nu\nu$ + jets
- $\mathbf{2\tau}$: $t\bar{t}$, $W \to \tau\nu$ + jets, $Z \to \tau\tau$ + jets

| | | $\tau$ Decay Mode | Branching Fraction (%) |
|---|---|---|---|
| Leptonic | | $\tau^{\pm} \to e^{\pm} + \bar{\nu}_e + \nu_\tau$ | $17.84 \pm 0.04$ |
| | | $\tau^{\pm} \to \mu^{\pm} + \bar{\nu}_\mu + \nu_\tau$ | $17.41 \pm 0.04$ |
| Hadronic | One-prong | $\tau^{\pm} \to \pi^{\pm} + (\geq 0\,\pi^0) + \nu_\tau$ | $49.46 \pm 0.10$ |
| | | $\tau^{\pm} \to \pi^{\pm} + \nu_\tau$ | $10.83 \pm 0.06$ |
| | | $\tau^{\pm} \to \rho^{\pm}(\to \pi^{\pm} + \pi^0) + \nu_\tau$ | $25.52 \pm 0.09$ |
| | | $\tau^{\pm} \to a_1(\to \pi^{\pm} + 2\pi^0) + \nu_\tau$ | $9.30 \pm 0.11$ |
| | | $\tau^{\pm} \to \pi^{\pm} + 3\pi^0 + \nu_\tau$ | $1.05 \pm 0.07$ |
| | | $\tau^{\pm} \to h^{\pm} + 4\pi^0 + \nu_\tau$ | $0.11 \pm 0.04$ |
| Hadronic | Three-prong | $\tau^{\pm} \to \pi^{\pm} + \pi^{\mp} + \pi^{\pm} + (\geq 0\pi^0) + \nu_\tau$ | $14.57 \pm 0.07$ |
| | | $\tau^{\pm} \to \pi^{\pm} + \pi^{\mp} + \pi^{\pm} + \nu_\tau$ | $8.99 \pm 0.06$ |
| | | $\tau^{\pm} \to \pi^{\pm} + \pi^{\mp} + \pi^{\pm} + \pi^0 + \nu_\tau$ | $2.70 \pm 0.08$ |




Exclusion limits with 3.2 $\mathrm{fb}^{-1}$ of ATLAS data

3

# Architecture Workflow

# General Workflow

**Analysis Interpreter**
**(Jupyter Notebook)**

Parameters → **Analysis Container** → **Analysis Processor** → **Analysis Plotter** ← Fields

Backgrounds
Signal

Cuts
Data Processing
**Machine Learning**

**Histograms**

**Environments:**

- ROOT
- PyROOT
- **UpROOT**

**ROOT I/O:**

uproot

**Data Processing:**

Awkward Array

NumPy

pandas

**Data Visualisation:**

matplotlib

seaborn

**ML:**

scikit learn

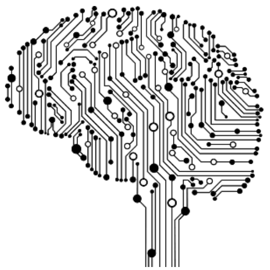
dmlc XGBoost

# Machine Learning Workflow

**ML workflow:**

data ~1M entries → UpROOT I/O (~100s) → jagged awkward-array → cutting (~0.5s) → cut jagged awkward-array → pre-processing (~10s) → rectangular awkward-array → Machine Learning

rectangular awkward-array → conversion (~20s) → rectangular pandas-array → Machine Learning



- **Algorithm**: **Boosted Decision Trees** (**BDTs**)
- **Framework**: **Extreme Gradient Boosting** (**XGBoost**)

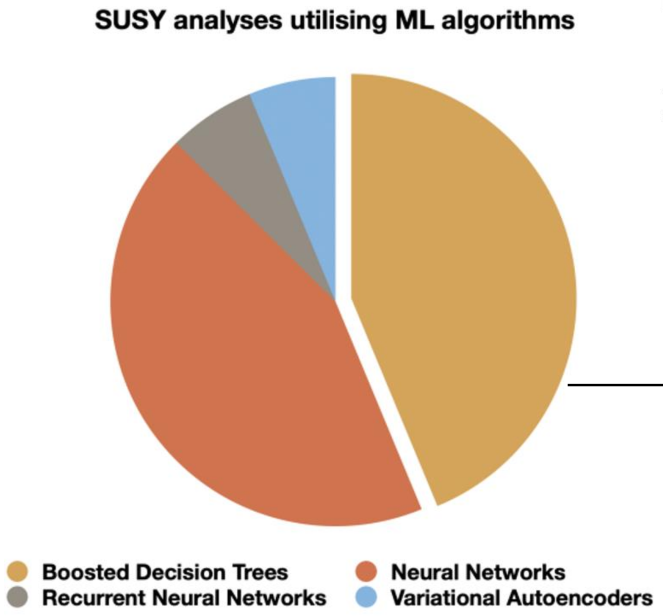
dmlc XGBoost

**Pipelines:**

- **Regression**
- **Binary-classification**
- **Multiclass-classification**

**Functionalities:**

- **Stopping function** preventing overfitting
- **Feature importance** plotter
- **Cross-validation**:
  - k-fold cross-validation
- **Hyperparameters** tuning:
  - exhaustive grid-search
  - randomized grid-search
- **GPU** computing

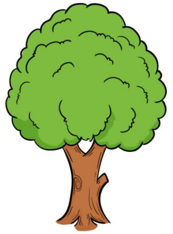



Source: ATLAS Machine Learning Forum

**XGBoost Execution Time:**

| | Fitting | Predicting |
|---|---|---|
| CPU: | ~76min | ~6.5min |
| GPU: | ~4min | ~5s |

Highly popular and widely recognized ML algorithm in the HEP community



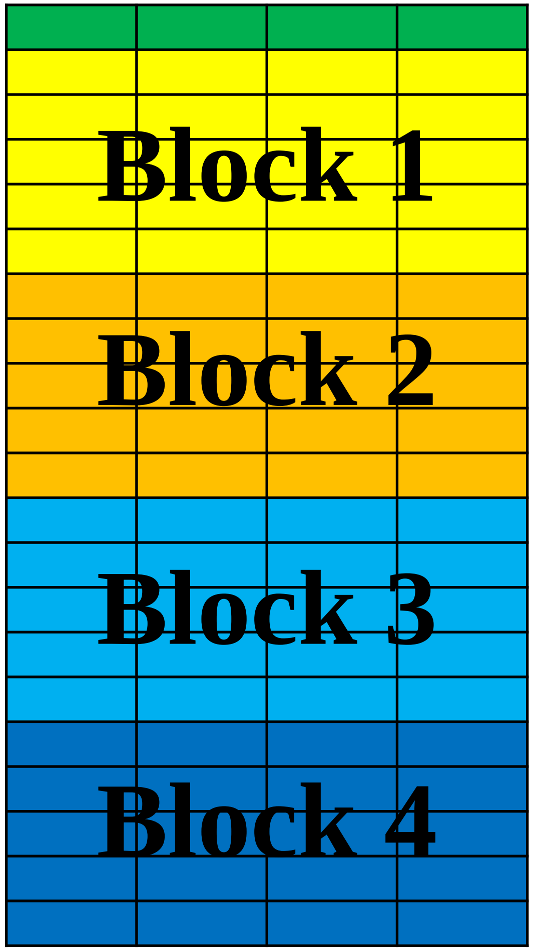
# XGBoost

**XGBoost** – external open-source library (framework) based on the Gradient Boosting. In comparison to the regular Gradient Boosting algorithm, the XGBoost increases speed and performance significantly.

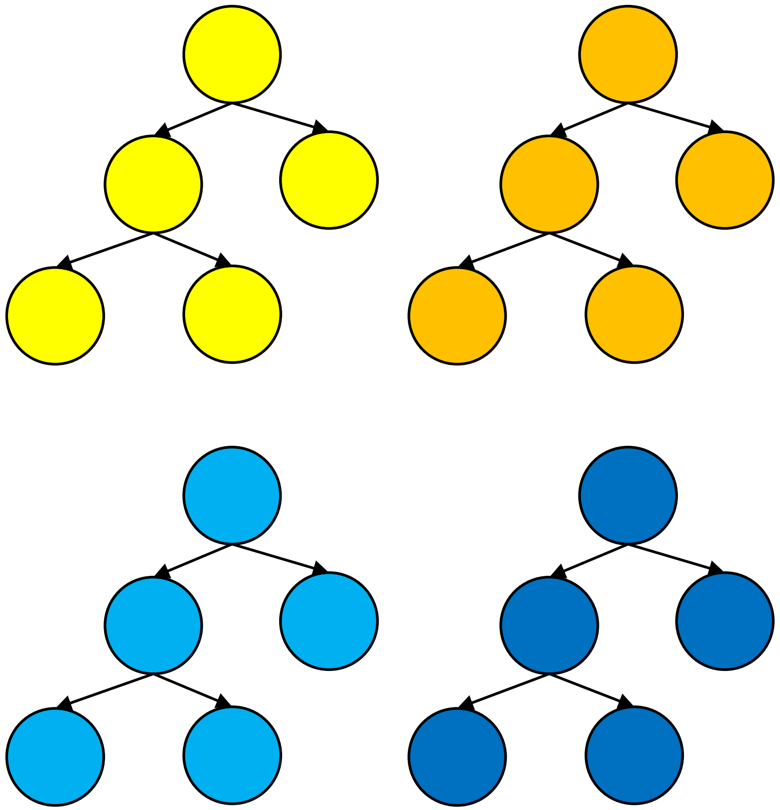
dmlc XGBoost

Major improvements:

- Parallelized tree building
- Tree pruning
- Efficient handling of missing data
- Regularization to prevent overfitting
- In-built cross-validation capability
- Hardware optimization


Parallelism



# 4 Machine Learning Challenges

# Padding

**Challenge:**

Kinematic data (mainly momenta and related quantities) is heavily nested / jagged. →

| entry | subentry |
|---|---|
| 0 | 0 |
| | 1 |
| | 2 |
| | 3 |
| 1 | 0 |
| | 1 |
| 2 | 0 |
| | 1 |
| | 2 |
| | 3 |
| | 4 |
| | 5 |
| 3 | 0 |
| | 1 |
| | 2 |
| | 3 |
| | 4 |
| | 5 |
| 4 | 0 |
| | 1 |
| | 2 |
| | 3 |
| | 4 |
| | 5 |
| 5 | 0 |
| | 1 |

Unfortunately, most of the machine learning algorithms do not work well with such arrays. → Rectangular format is preferred.

To make datasets ML-friendly, we are using UpROOT and Awkward Array as processing tools.

**Current solution:**

Setting up a certain threshold for a number of entries each event can take and padding with zeros.

**Example:**

threshold = 3

$$\text{event1} = [\overbrace{400, 200, 150, \mathbf{100}}^{\text{jet\_pt}}]$$

event2 = [500, 300, 200]

event3 = [100, 300]

event4 = [350]

$\xrightarrow{\text{padding}}$

| 0 | 1 | 2 |
|---|---|---|
| 400 | 200 | 150 |
| 500 | 300 | 200 |
| 100 | 300 | **0** |
| 350 | **0** | **0** |

$\xrightarrow{\text{feature engineering}}$

| jet_pt_0 | jet_pt_1 | jet_pt_2 |
|---|---|---|
| 400 | 200 | 150 |
| 500 | 300 | 300 |
| 100 | 300 | 0 |
| 350 | 0 | 0 |

**Disadvantages:**

- loss of information →
- possible bias introduced from padding →
    - study the effect from padding

# Negative Weights

**Challenge:**

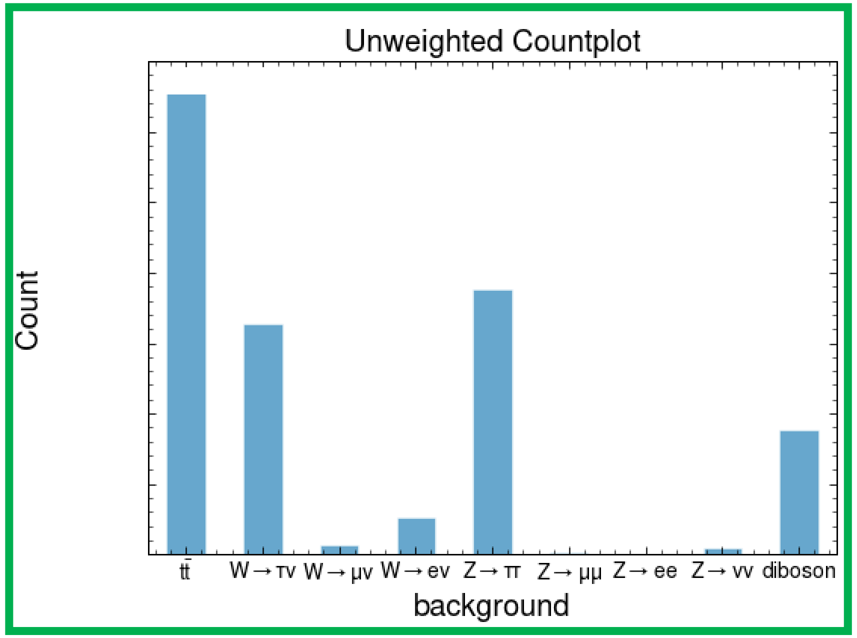
There are events with negative weights.

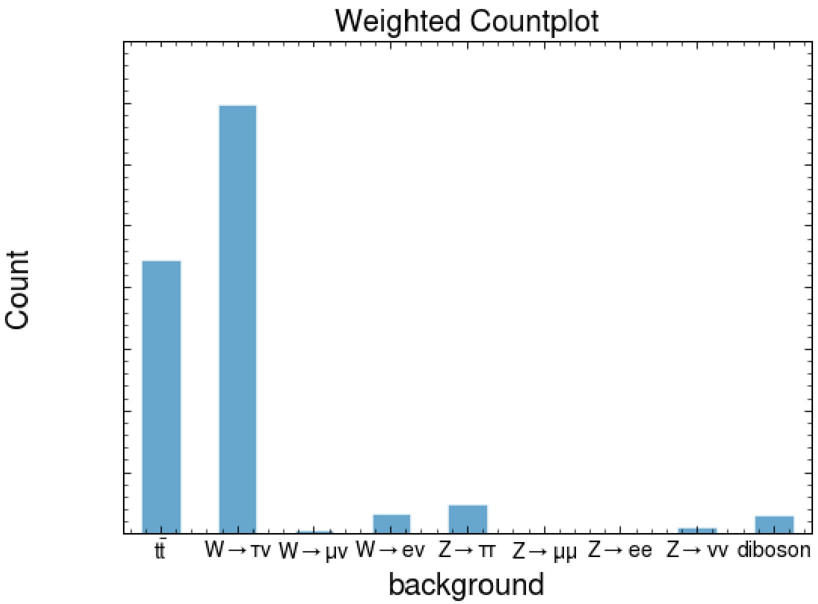
Unfortunately, during the training process XGBoost ignores such events. → Only positive weights are allowed.

Many ML tools are not thoroughly tested with respect to negative weights.

**Current solution:**

Training with events that have positive weights and evaluating using all events.



# 5 Summary

# Summary

- HVL and UiB interested in the continuation of search for squark & gluino → tau(s) + jets + MET (reference: SUSY-2016-30)

- The “baseline” analysis script has been initialized

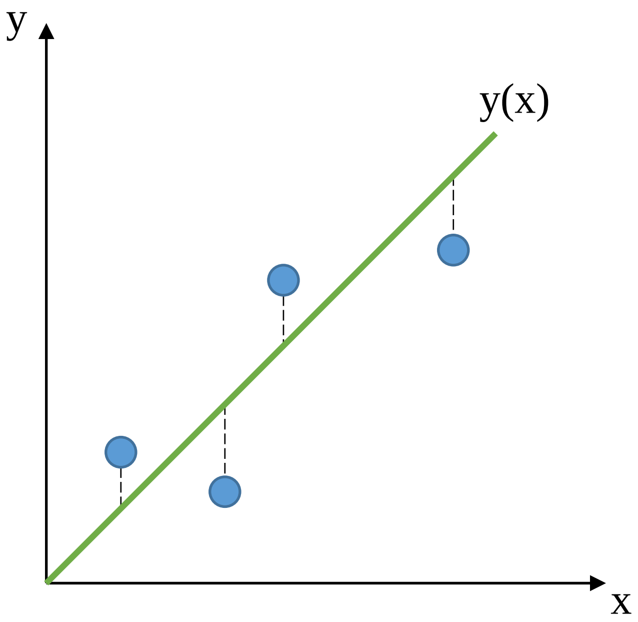
- Implementation of ML techniques:
    - BDTs (XGBoost) – in progress
    - Neural Networks – soon to be started
    - Other ML algorithms will also be considered

- Any comments and ideas are welcome
    - Methods used in classification problems in data analysis of HEP particle collisions when facing challenges with jagged arrays
        - Are there other possibilities?
    - Rectangularization
        - Padding: zeroes, mean / average value / large negative numbers – how does it influence a model?
        - When discarding less energetic particles (jets) – how much do we lose in the predictive power?
    - Negative weights
        - How to deal with events with negative weights during the training process?

# Thank you for your attention!

# Backup Slides

# Bias & Variance

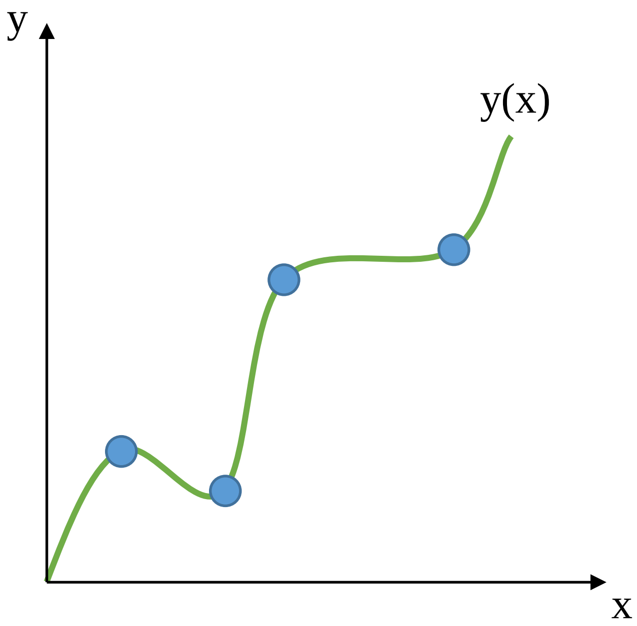


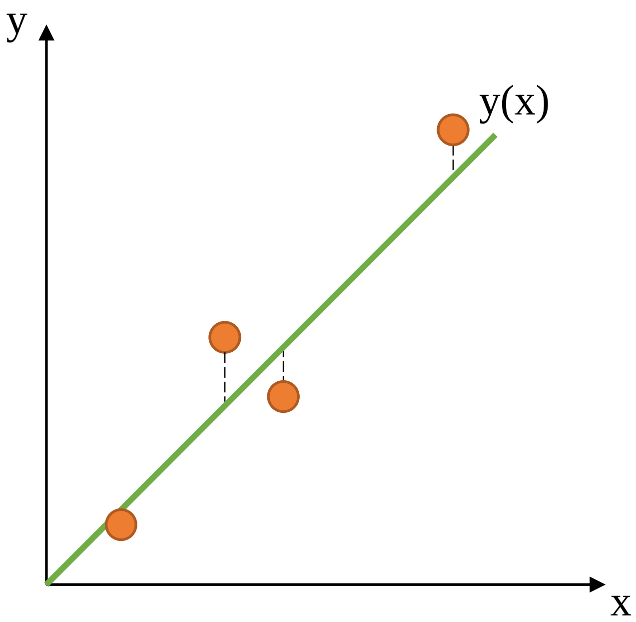

**Bias** → error rate
of a training data

High Bias
↕
Underfitting

**Variance** → error rate
of a testing data

High Variance
↕
Overfitting



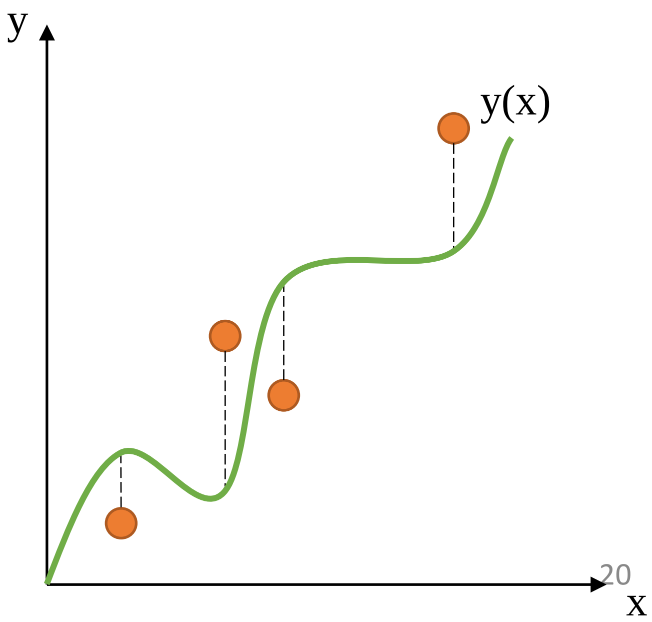




Methods for finding the sweet spot:

- Pruning
- **Bagging**
- **Boosting**

$\left.\right\}$ **Ensembles**

- Regularization
    - Lasso regression (L1)
    - Ridge regression (L2)



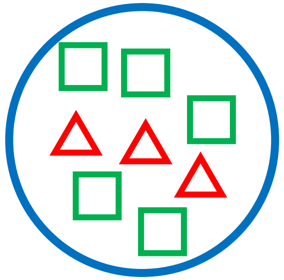
# Decision Tree

Tree-based algorithms are commonly used for supervised machine learning problems.

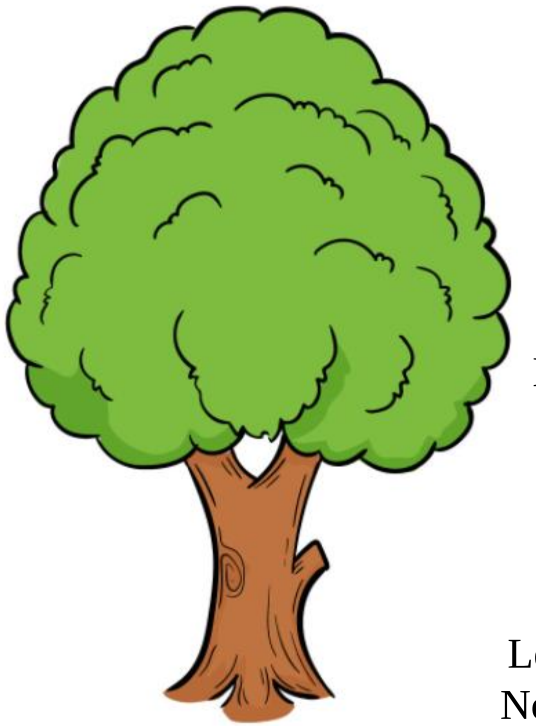
**Decision tree** → is a representation of a decision-making process. In general, a decision tree asks a question and then classifies the data based on the answer. The classification can be either for discrete or numerical values.

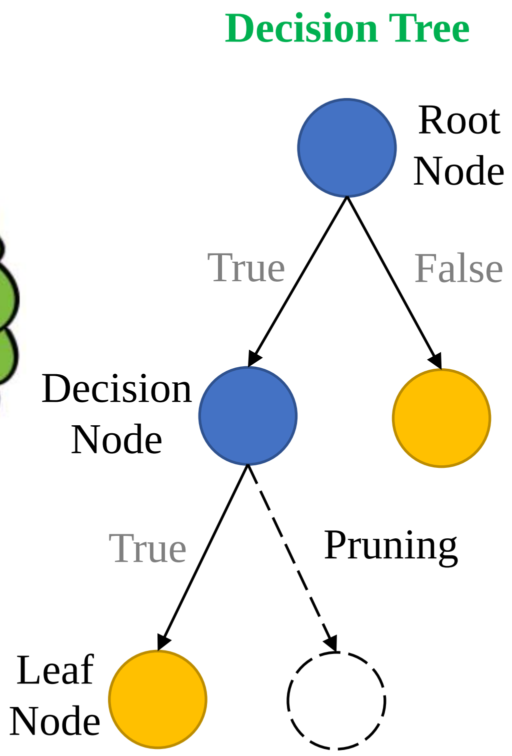
Several methods to quantify impurity:

- **Gini Impurity** → $1 - \sum_{i=1}^{n} (p_i)^2$
- Entropy/Information Gain
- Chi-Square



$$1 - \left(\left(\frac{5}{8}\right)^2 + \left(\frac{3}{8}\right)^2\right)=0.46875$$






Pros:

- Easy to visualize and understand
- Little to none data preparation
- Universal (classification & regression)

Cons:

- Often inaccurate
- Prone to overfitting → high variance
- Unstable → small change in data can lead to a big change in structure

# Ensembles

**Ensemble** learning → is a model that makes predictions based on a number of different models. By combining individual models (weak/base learners), the ensemble model tends to be more flexible (less bias) and less data-sensitive (less variance).

Two most popular ensemble methods are:

- **bagging** – training a lot of individual models in **parallel** way. Each model learns independently from each other.
  - Bagging (Bootstrap Aggregating)
  - Random Forest
- **boosting** – training a lot of individual models in a **sequential** way. Each model learns from mistakes made by the previous model.
  - AdaBoost (Adaptive Boosting)
  - Gradient Boosting
  - XGBoost (Extreme Gradient Boosting)
  - LightGBM, Catboost, …

Pros:

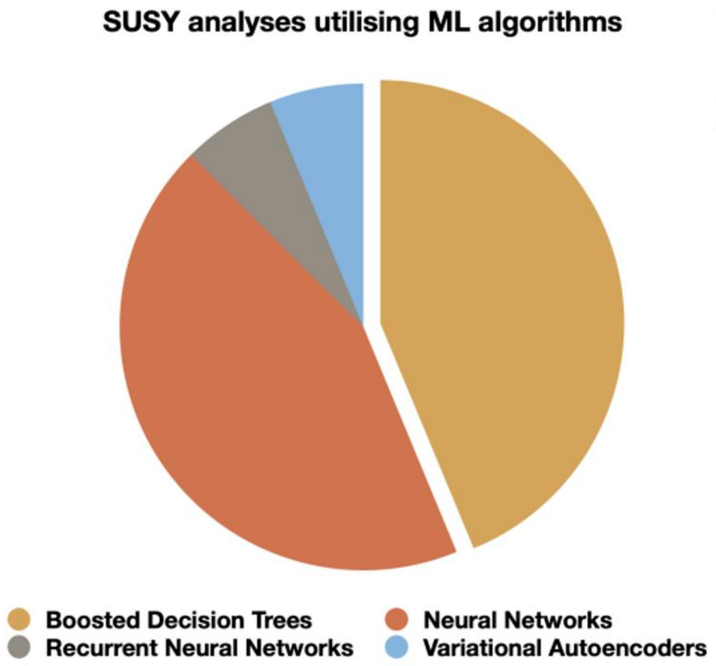
- Perform much better than single individual models
- Bias/variance tradeoff
- Unlikely to underfit/overfit

Cons:

- Less interpretable
- Computationally expensive

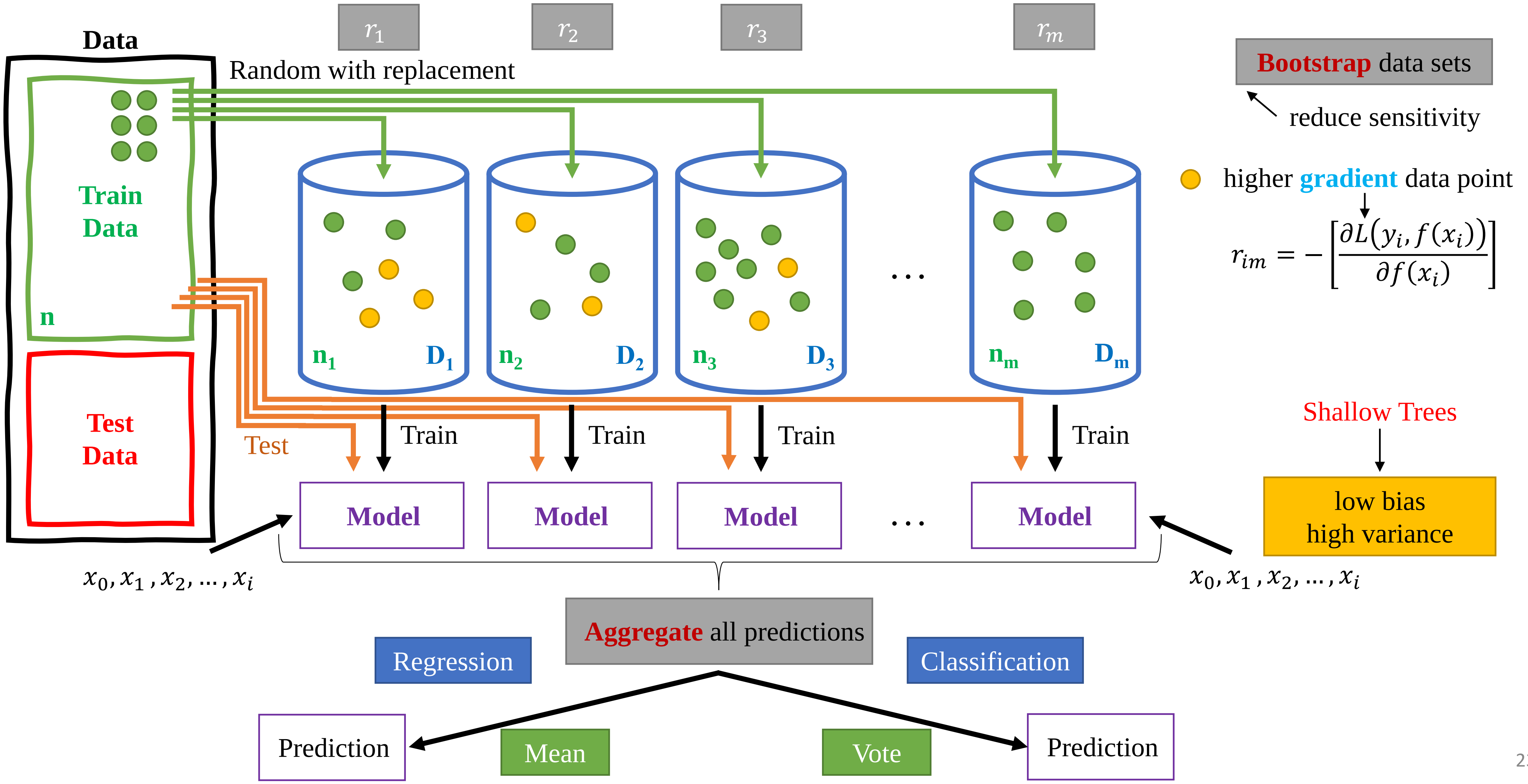
Bagged/Boosted Decision Tree (BDT)

- Highly popular and widely recognized algorithm in the High Energy Physics community
- Used to classify physics processes
- Used to define analysis regions




Source: ATLAS Machine Learning Forum

# Gradient Boosting



# Confusion Matrix

**Accuracy Score**

- Number of correct predictions over all predictions.

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

**Precision Score**

- Number of correct positive predictions over all positive predictions.

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

**Recall Score (Sensitivity)**

- Number of correct positive predictions over the actual positives.

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

**F1 Score**

- A weighted harmonic mean of precision & recall.

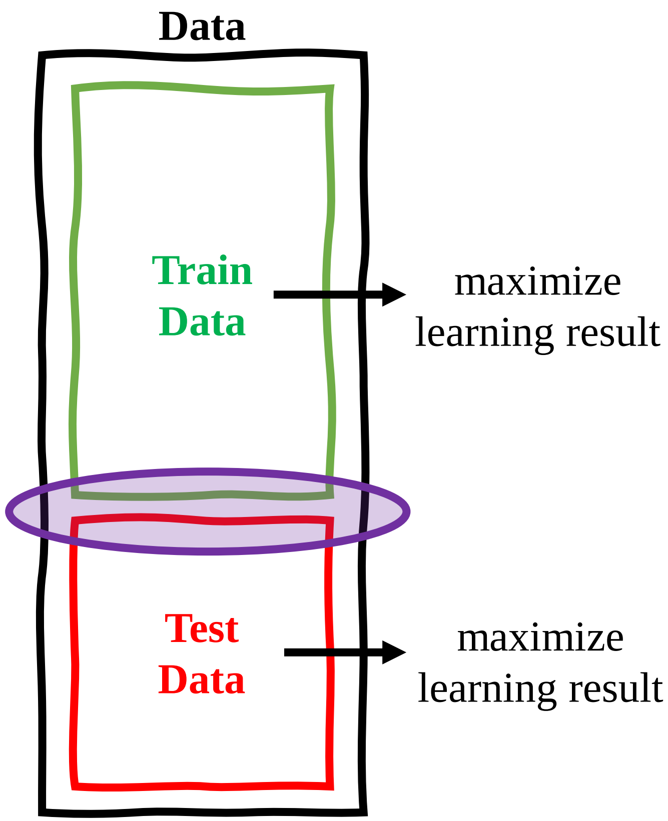
$$2 \cdot \frac{Recall \cdot Precision}{Recall + Precision}$$

| | | Predicted Class | | |
|---|---|---|---|---|
| | | **Positive** | **Negative** | |
| **Actual Class** | **Positive** | True Positive (TP) | False Negative (FN)<br>**Type II Error** | **Sensitivity**<br>$\frac{TP}{(TP + FN)}$ |
| | **Negative** | False Positive (FP)<br>**Type I Error** | True Negative (TN) | **Specificity**<br>$\frac{TN}{(TN + FP)}$ |
| | | **Precision**<br>$\frac{TP}{(TP + FP)}$ | **Negative Predictive Value**<br>$\frac{TN}{(TN + FN)}$ | **Accuracy**<br>$\frac{TP + TN}{(TP + TN + FP + FN)}$ |

# K-Fold Cross-Validation

**Cross-Validation** is a statistical method used to estimate the skill of machine learning models.

## Train/Test Split




## 10-Fold Cross-Validation

Running $k$ separate learning experiments

- Testing set → 1
- Training set → $k - 1$

Average test results from $k$ experiments


Takes more time but assessment of the learning algorithm is more accurate.
All the data is used for learning and training.

**Average** all results

# Hyperparameters Tuning

Hyperparameters are parameters that are not directly learnt within estimators. They are usually passed as arguments to the constructor of the estimator classes.

**Grid-Search** is a tuning technique that attempts to compute the optimum values of hyperparameters.

```
parameters_distribution = {
    'n_estimators': [25, 50, 75, 100, 150, 200],
    'learning_rate': [0.01, 0.05, 0.1, 0.15, 0.2, 0.3],
    'max_depth': [3, 4, 5, 6, 7, 10, 15],
    'min_child_weight': [1, 5, 10],
    'gamma': [0.5, 1, 1.5, 2, 5, 10, 100],
    'subsample': [0.3, 0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'reg_alpha': [0, 0.1, 0.5, 1, 10, 25, 50, 100],
    'reg_lambda': [0, 0.1, 0.5, 1, 10, 25, 50, 100],
}
```

- **Exhaustive Grid-Search**

computes the grid for all the parameter combinations
computing time can take hours or even days

- **Randomized Grid-Search**

computes the grid for random number of parameters
computing time decreases significantly

**General parameters** – guide the overall functioning

| | | |
|---|---|---|
| • **booster** | gbtree | type of model to run at each iteration |
| • **nthread** | all | number of threads to use in parallel processing |

**Booster parameters** – guide of the individual booster at each step

| | | |
|---|---|---|
| • **n_estimators** | automatically found | number of classifiers |
| • **eta** | 0.05 – 0.3 | learning rate |
| • **max_depth** | 3 – 10 | the maximum depth of a tree |
| • **min_child_weight** | 1 | defines the minimum sum of weights in a child |
| • **gamma** | 0 | specifies minimum loss reduction to make a split |
| • **subsample** | 0.5 – 1 | defines random fraction of observations for each tree |
| • **colsample_bytree** | 0.5 – 1 | defines random fraction of columns for each tree |
| • colsample_bylevel | 1 | defines random fraction of columns for each split in each level |
| • max_delta_step | 0 | tree's weight estimation |
| • **lambda** | 1 | L2 regularization term on weights |
| • **alpha** | 0 | L1 regularization term on weights |
| • tree_method | auto | tree construction algorithm |
| • scale_pos_weight | 1 | balance of positive and negative weights |

**Learning task parameters** – guide the optimization performance

| | | |
|---|---|---|
| • **objective** | reg / log / multilog | defines loss function to be minimized |
| • **eval_metric** | rmse / error / merror | the metric to be used for validation data |

# ROC & PR Curves (Binary-Classification)

**ROC Curve** – **R**eceiver **O**perating **C**haracteristic Curve
**PR Curve** – **P**recision-**R**ecall Curve

**ROC and PR Curves**
**Both are used to:**

- explain model goodness of fit
- identify the correct threshold to map probabilities value to the actual classes

**Used when:**

- ROC - there is a balanced class distribution
- PR - there is an imbalanced class distribution

**Metrics:**

- ROC - Area Under Curve (AUC)
    - $AUC = \int_0^1 TPR \, d(FPR)$

where $TPR$ is True Positive Rate and $FPR$ is False Positive Rate

- PR - Average Precision (AP)
    - $AP = \sum_n (R_n - R_{n-1}) P_n$

where $R_n$ and $P_n$ are the precision and recall at the $n_{th}$ threshold