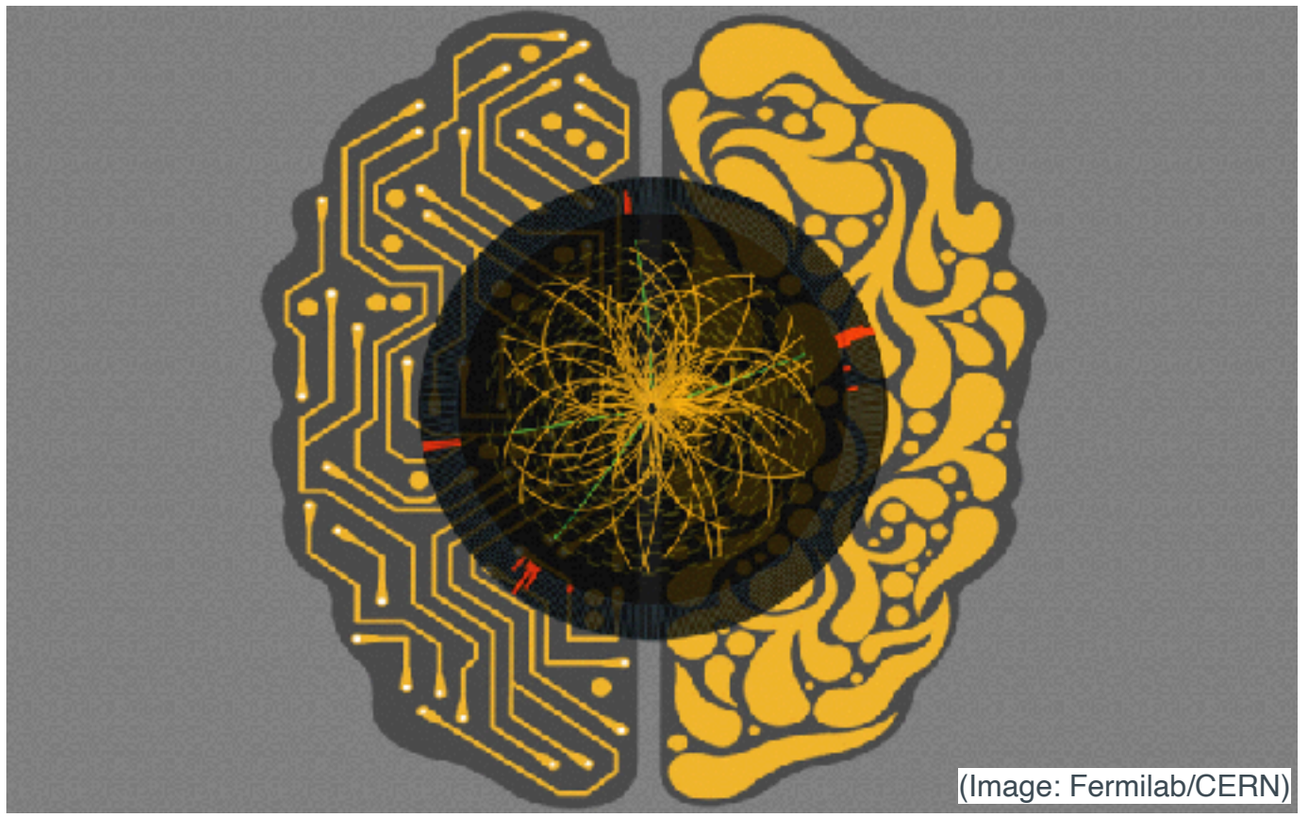
PHY 835: Collider Physics Phenomenology

Machine Learning in Fundamental Physics

Gary Shiu, UW-Madison



Lecture 9: Kernel Methods

Recap of Lecture 8

- Support Vector Machines: functional and geometric margins
- A simple example and animation:

  https://www.youtube.com/watch?v=5zRmhOUjjGY

- Optimal margin classifier
- Lagrange duality
- Kernel Methods

Outline for today

- Kernel methods
- Soft margins
- Ensemble Methods
    - Bagging
    - Boosting
    - Random Forest

Ref: Andrew Ng's Lecture Notes: https://sgfin.github.io/files/notes/CS229_Lecture_Notes.pdf and 1803.08823

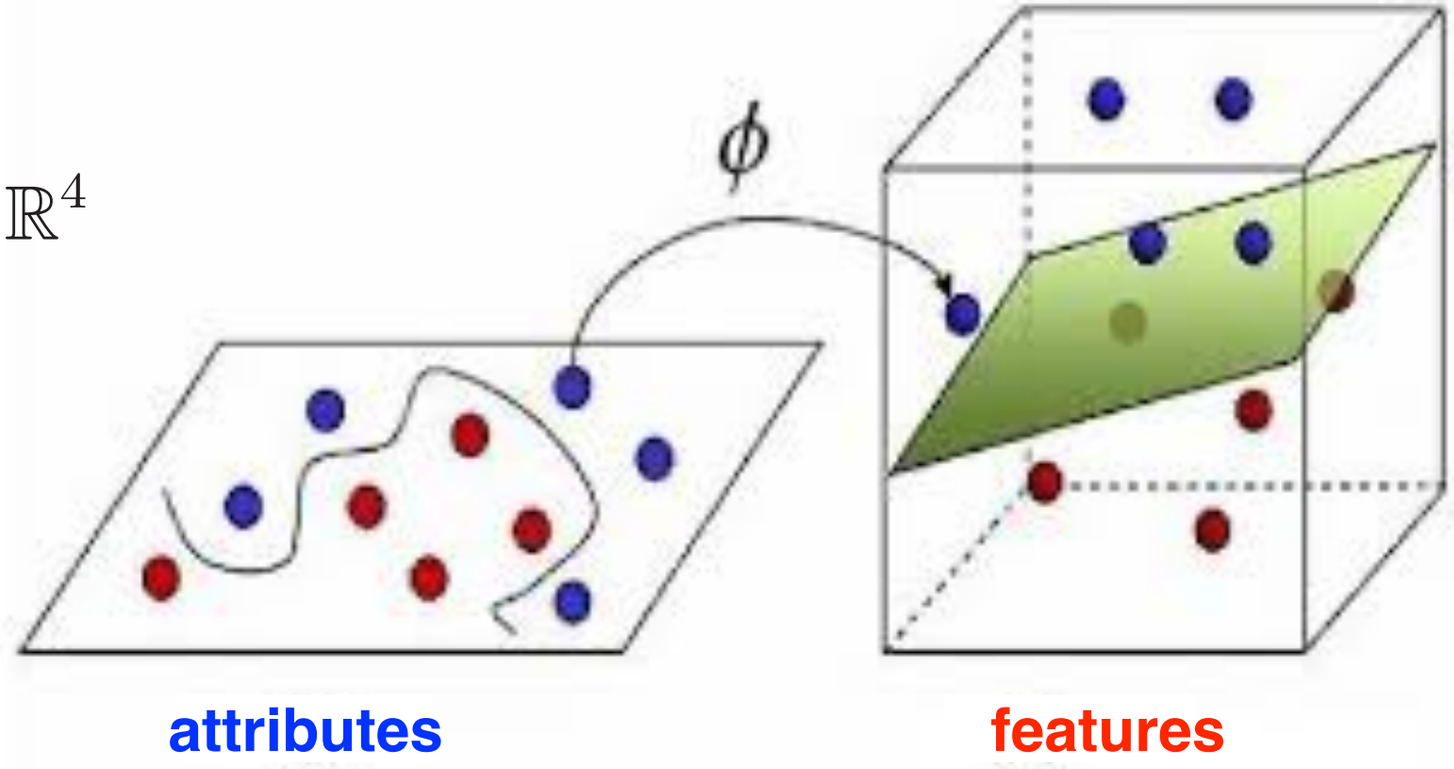
Feature Maps

- So far we have been considering linear separable data, what if the output is more accurately represented by a non-linear function e.g.

$$y = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x + \theta_0$$

- Consider the function $\phi : \mathbb{R} \to \mathbb{R}^4$ known as the feature map:

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix} \in \mathbb{R}^4$$



- y is a linear function over $\phi(x)$: $y = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x + \theta_0 = \theta^T \phi(x)$.

Kernels

- For linear functions, our algorithm for the optimal margin classifier requires computing $< x, z >$. This suggests that we should:

$$< x, z > \rightarrow K(x, z) = < \phi(x), \phi(z) > = \phi(x)^T \phi(z)$$

- The kernel $K(x, z)$ may be inexpensive to compute, even though it is expensive to calculate $\phi(x)$ (extremely high dimensional vector).

- Using $K(x, z)$, we can get SVM to learn high dimensional feature space given by ϕ without having to explicitly find/represent $\phi(x)$.

- Suppose $x, z \in \mathbb{R}^n$ and the Kernel is given by:

$$K(x, z) = (x^T z)^2.$$

- What is the corresponding feature map $\phi(x)$?

Kernels

- We can write the Kernel as:

$$K(x,z) = \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_i z_i\right) = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j = \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j)$$

- The feature map (for n=3):

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix}$$

- Computing $\phi(x)$ takes $\mathcal{O}(n^2)$ time while computing $K(x, z)$ takes only $\mathcal{O}(n)$ time (linear in the dim. of the input attributes).
- For a huge input dimension (say ~ 1000), this is a significant speedup.

Kernels

- For a related Kernel:

$$\begin{aligned} K(x,z) &= (x^T z + c)^2 \\ &= \sum_{i,j=1}^{n} (x_i x_j)(z_i z_j) + \sum_{i=1}^{n} (\sqrt{2c}x_i)(\sqrt{2c}z_i) + c^2. \end{aligned}$$

- The corresponding feature map (for n=3):

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \\ \sqrt{2c}x_1 \\ \sqrt{2c}x_2 \\ \sqrt{2c}x_3 \\ c \end{bmatrix}$$

More generally $K(x,z) = (x^T z + c)^d$:

Entries of ϕ = monomials of the form $x_{i_1}x_{i_2}\ldots x_{i_d}$ up to degree d

- Computing $K(x,z)$ still takes only $\mathcal{O}(n)$ time, even for more general:

$$K(x,z) = (x^T z + c)^d$$

Kernels

- Intuitively, if $\phi(x)$ and $\phi(z)$ are close together, $K(x,z) = \phi(x)^T\phi(z)$ is large; conversely if they are far apart, $K(x,z)$ is small.
- $K(x,z)$ is a measure of how similar x and z are; consider e.g.,

$$K(x,z) = \exp\left(-\frac{||x-z||^2}{2\sigma^2}\right)$$

- Can this be a kernel? Yes, **Gaussian kernel.**
- In general, can we tell if some function $K(x,z)$ is a valid kernel?
- Given a finite set of m points, define an $m \times m$ **Kernel matrix**:

$$K_{ij} = K(x^{(i)}, x^{(j)})$$

- The kernel matrix should be **symmetric**, and **positive semi-definite.**

Kernels

- $K_{ij} = K_{ji}$ is straightforward whereas the positivity of the K matrix:

$$
\begin{aligned}
z^T K z &= \sum_i \sum_j z_i K_{ij} z_j \\
&= \sum_i \sum_j z_i \phi(x^{(i)})^T \phi(x^{(j)}) z_j \\
&= \sum_i \sum_j z_i \sum_k \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\
&= \sum_k \sum_i \sum_j z_i \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\
&= \sum_k \left(\sum_i z_i \phi_k(x^{(i)}) \right)^2 \\
&\geq 0.
\end{aligned}
$$

- Since z is arbitrary, the kernel matrix K is positive semi-definite.
- These two conditions turn out to be not only necessary but sufficient.

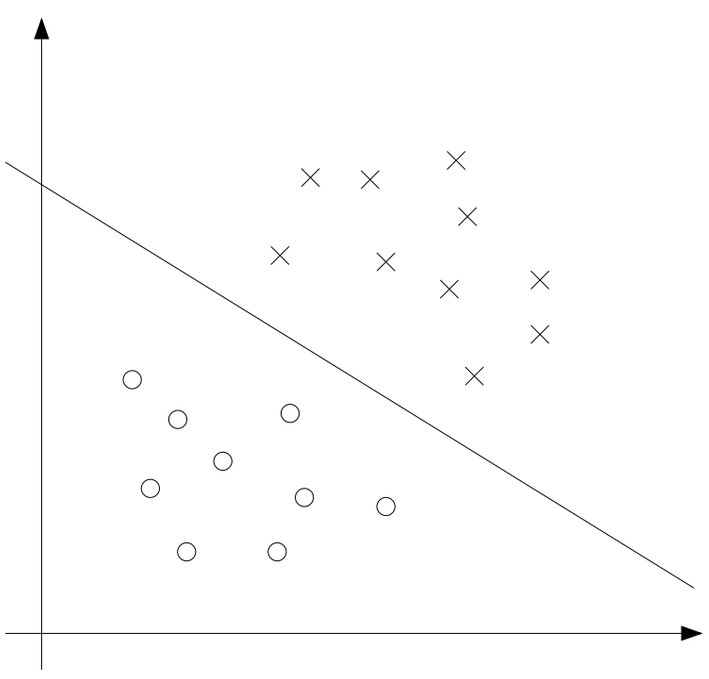
Mercer's Theorem

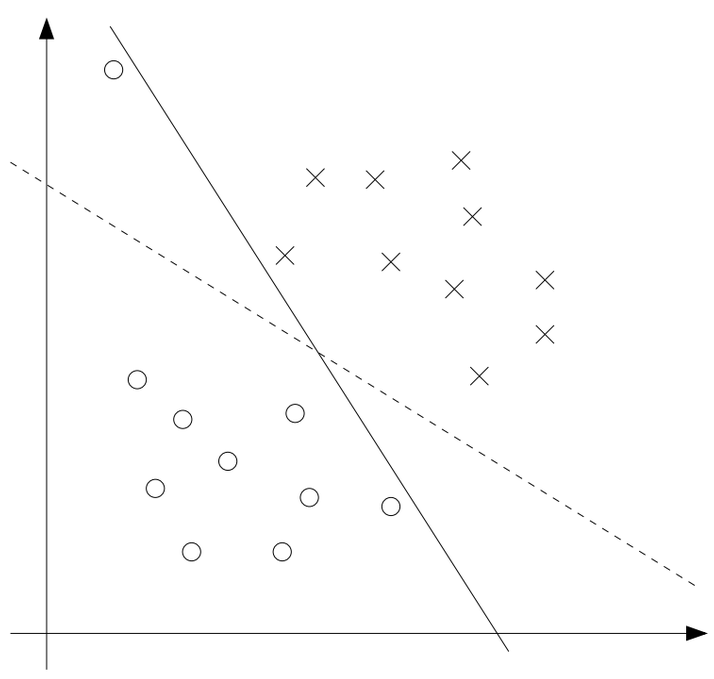
Theorem (Mercer). Let $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ be given. Then for K to be a valid (Mercer) kernel, it is necessary and sufficient that for any $\{x^{(1)}, \ldots, x^{(m)}\}$, $(m < \infty)$, the corresponding kernel matrix is symmetric positive semi-definite.

- Check the validity of a kernel without constructing the feature map.
- The kernel trick has wider applications than SVM, e.g.
  - In classifying digits (MNIST database), a simple polynomial kernel or the Gaussian kernel gives extremely good performance.
  - In classifying strings of English alphabets with length k, we have a 26^k dim. feature space. Kernel calculations take $\mathcal{O}(26)$ time.
- Kernel trick: replacing $< x, z >$ by $K(x, z)$ turns the algorithm to work in higher dim. feature space.

SVM: Soft Margins

- Earlier discussion (w/o kernel) assumed data is linearly separable.
- Mapping data to a higher dim. feature space generally increases the chance for the data to be separable, but there is no guarantee.
- Also, choice of separating hyperplane is susceptible to outliners.





SVM: Soft Margins

- Optimization with a soft margin (and penalty):

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \ldots, m \\ & \xi_i \geq 0, \quad i = 1, \ldots, m. \end{aligned}$$

- Here we introduced an L_1 regularization; we allow the margin to be less than 1 by paying a cost controlled by C.
- This SVM implementation is in the package sklearn.
- We can construct the Lagrangian:

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2}w^T w + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i\left[y^{(i)}(x^T w + b) - 1 + \xi_i\right] - \sum_{i=1}^{m} r_i\xi_i.$$

- Like before, we can turn this primal problem into its dual problem.

SVM: Soft Margins

- The dual optimization problem:

$$\begin{aligned} \max_\alpha \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0, \end{aligned}$$

- As before, we can express w in terms of α_i and make predictions:

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}. \qquad w^T x + b = \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b.$$

- Some differences:
  - Introducing an L_1 regularization turns $0 \leq \alpha_i$ to $0 \leq \alpha_i \leq C$
  - The solution of b^* is modified.
  - The KKT dual complementarity condition:

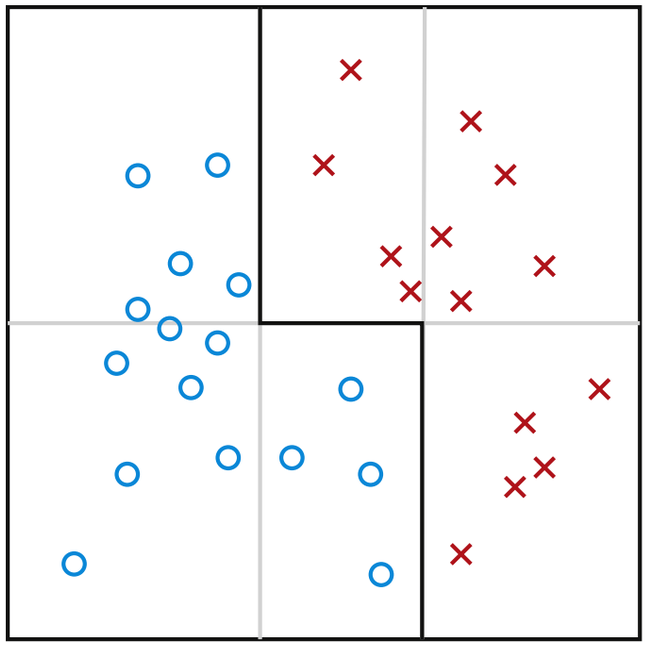
$$\begin{aligned} \alpha_i = 0 \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \geq 1 \\ \alpha_i = C \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) \leq 1 \\ 0 < \alpha_i < C \quad &\Rightarrow \quad y^{(i)}(w^T x^{(i)} + b) = 1. \end{aligned}$$

Ensemble Methods

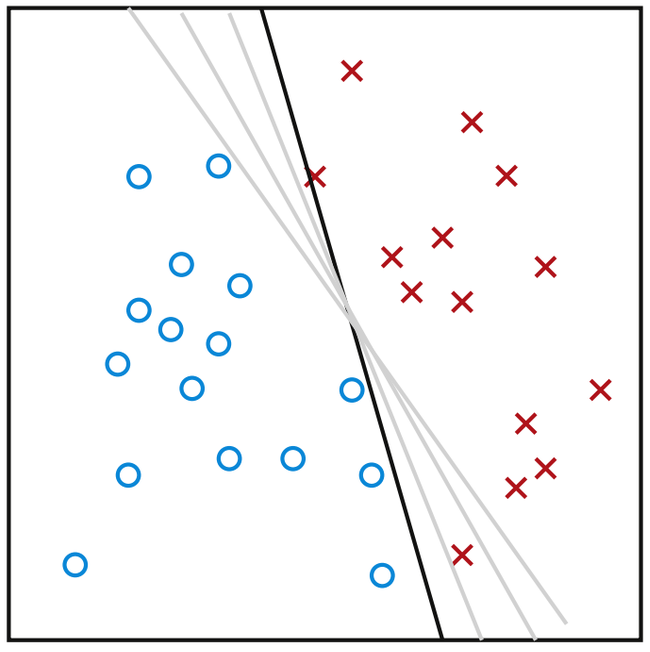
Two heads are better than one, or「三個臭皮匠，勝過一個諸葛亮」

Why Ensembles?

- **Statistical:** Multiple minima with same performance (training set too small). Choosing average reduces risk of wrong hypothesis choice.
- **Computational:** get stuck in local minima; results (e.g. decision tree structure + classification) vary strongly depending on training set.
- **Representational:** more expressive than single predictor, e.g.,



Aggregating different linear hypotheses



Linear perceptron hypothesis

Bagging

- **BAGG=Bootstrap AGGregation (Breiman 1996)**
- Partition your dataset: $\mathscr{L} \to \{\mathscr{L}_1, \ldots, \mathscr{L}_M\}$.
- Each $\mathscr{L}_i$ is large enough to learn a predictor $g_{\mathscr{L}_i}$.
- For continuous predictors, take average:

$$\hat{g}^A_{\mathcal{L}}(\mathbf{x}) = \frac{1}{M}\sum_{i=1}^{M} g_{\mathcal{L}_i}(\mathbf{x}).$$

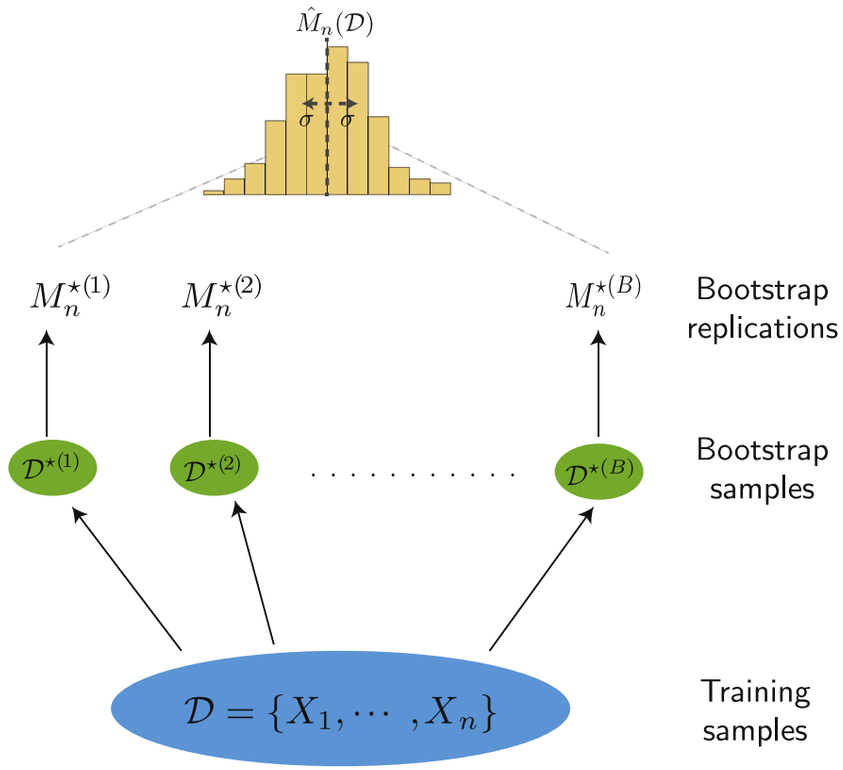
- For classification, implement majority rule:

$$\hat{g}^A_{\mathcal{L}}(\mathbf{x}) = \arg\max_j \sum_{i=1}^{M} I[g_{\mathcal{L}_i}(\mathbf{x}) = j], \qquad I[g_{\mathscr{L}_i}(x) = j] = \text{indicator function} = 1 \text{ if } g_{\mathscr{L}_i}(x) = j$$

- BAGGing significantly reduces the variance w/o increasing the bias.

Empirical Bootstrapping

- **Problem of BAGGing:** requires a lot of data in each partition.
- Way around: empirical bootstrapping



$$\hat{g}_{\mathcal{L}}^{BS}(\mathbf{x}) = \frac{1}{M}\sum_{i=1}^{M} g_{\mathcal{L}_i^{BS}}(\mathbf{x}).$$

can share data points

- **Price:** increase in bias because we are recycling points.
- Where useful? When faced with unstable learning algorithms (prediction error dominated by variance).

Boosting

- **Idea: Give different weights to the predictors.**

$$g_A(\mathbf{x}) = \sum_{K=1}^{M} \alpha_k g_k(\mathbf{x}), \qquad \sum_k \alpha_k = 1.$$

- Initialize equal weights $\alpha_k = 1/M \; \forall$ points and adjust at each step.
- **AdaBoost [Freud, Schapire, 1995]:** Consider a classifier for $y \in \{+1, -1\}$

- **Initialize** $w_{t=1}(\boldsymbol{x}_n) = 1/N, n = 1, \ldots, N.$
- **For** $t = 1 \cdots, T$(desired termination step), **do**:

  (1) Select a hypothesis $g_t \in \mathcal{H}$ that minimizes the weighted error

$$\epsilon_t = \sum_{i=1}^{N} w_t(\boldsymbol{x}_i) \mathbb{1}(g_t(\boldsymbol{x}_i) \neq y_i)$$

  (2) Let $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$, update the weight for each data $\boldsymbol{x}_n$ by

$$w_{t+1}(\boldsymbol{x}_n) \leftarrow w_t(\boldsymbol{x}_n) \frac{\exp[-\alpha_t y_n g_t(\boldsymbol{x}_n)]}{Z_t},$$

  where $Z_t = \sum_{n=1}^{N} w_t(\boldsymbol{x}_n) e^{-\alpha_t y_n g_t(\boldsymbol{x}_n)}$ ensures all weights add up to unity.

- **Output** $g_A(\boldsymbol{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t g_t(\boldsymbol{x})\right)$

Correctly predicted points have $y_n g_t(x_n) = +1$

$\epsilon_t \sim 0.5$ (as good as guess): do nothing

$\epsilon_t > 0.5$ (large error): weights enhanced for correctly predicted points

$\epsilon_t < 0.5$ (small error): enhance weights for wrongly predicted points.

Boosting

- **Various variants of boosted algorithms:** gradient boosted trees, XGBoost (extreme gradient boosting), ….

- You are encouraged to explore how these variants work by reading their documentations and experimenting with them:

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

https://xgboost.readthedocs.io/en/latest/index.html

Random Forest

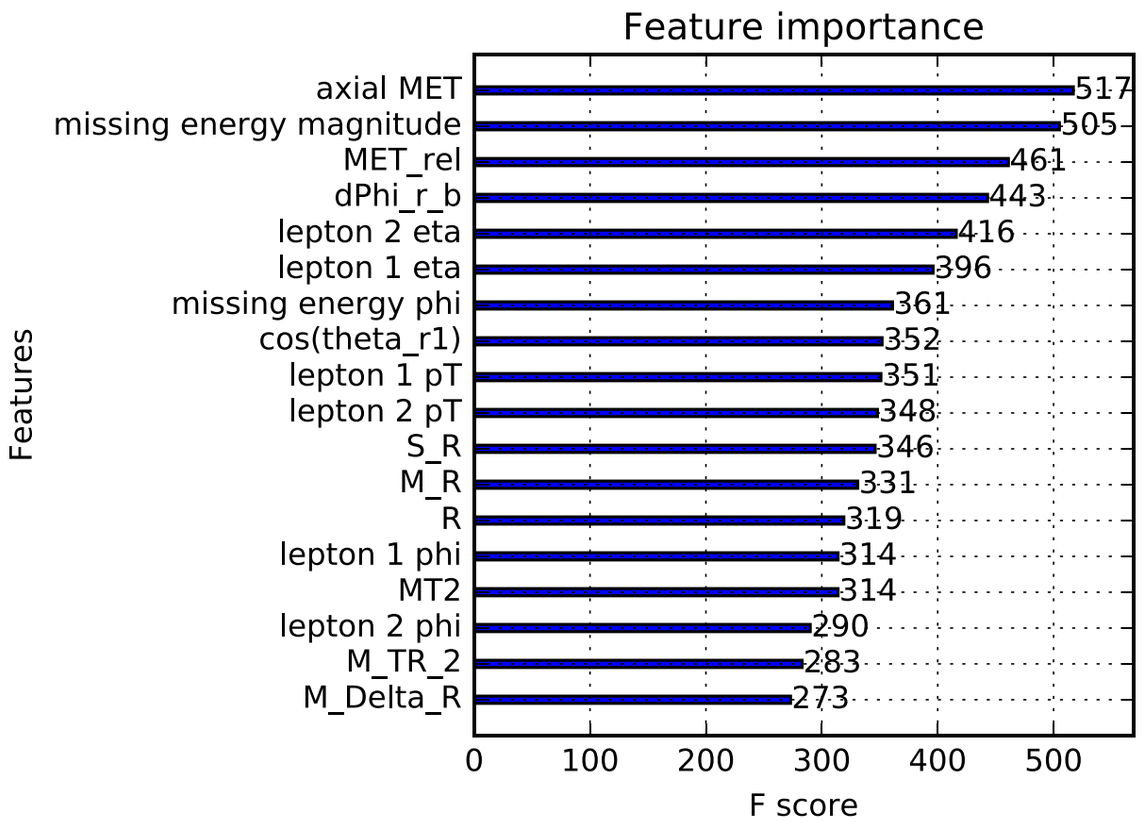
- Multiple Decision Trees used
- Use different subsets of data for fitting a tree (bagging)
- Use different subsets of features (Outlook, Humidity) to fit the data:
- Trying to tackle high-variance in decision trees
- Reduces correlations between decision trees
- Many variants and applications; recent applications include classifying non-Higgsable gauge groups in F-theory:

https://arxiv.org/pdf/1804.07296.pdf

SUSY Dataset

- Use XGBoost to classify Monte-Carlo simulations into SUSY vs SM. We can compare performance with logistic regression done earlier.

- Feature score to rank significance of features:



https://physics.bu.edu/~pankajm/ML-Notebooks/HTML/NB10_CVIII-XGboost_susy.html

Summary

- Kernel methods
- Soft margins
- Ensemble Methods
  - Bagging
  - Boosting
  - Random Forest