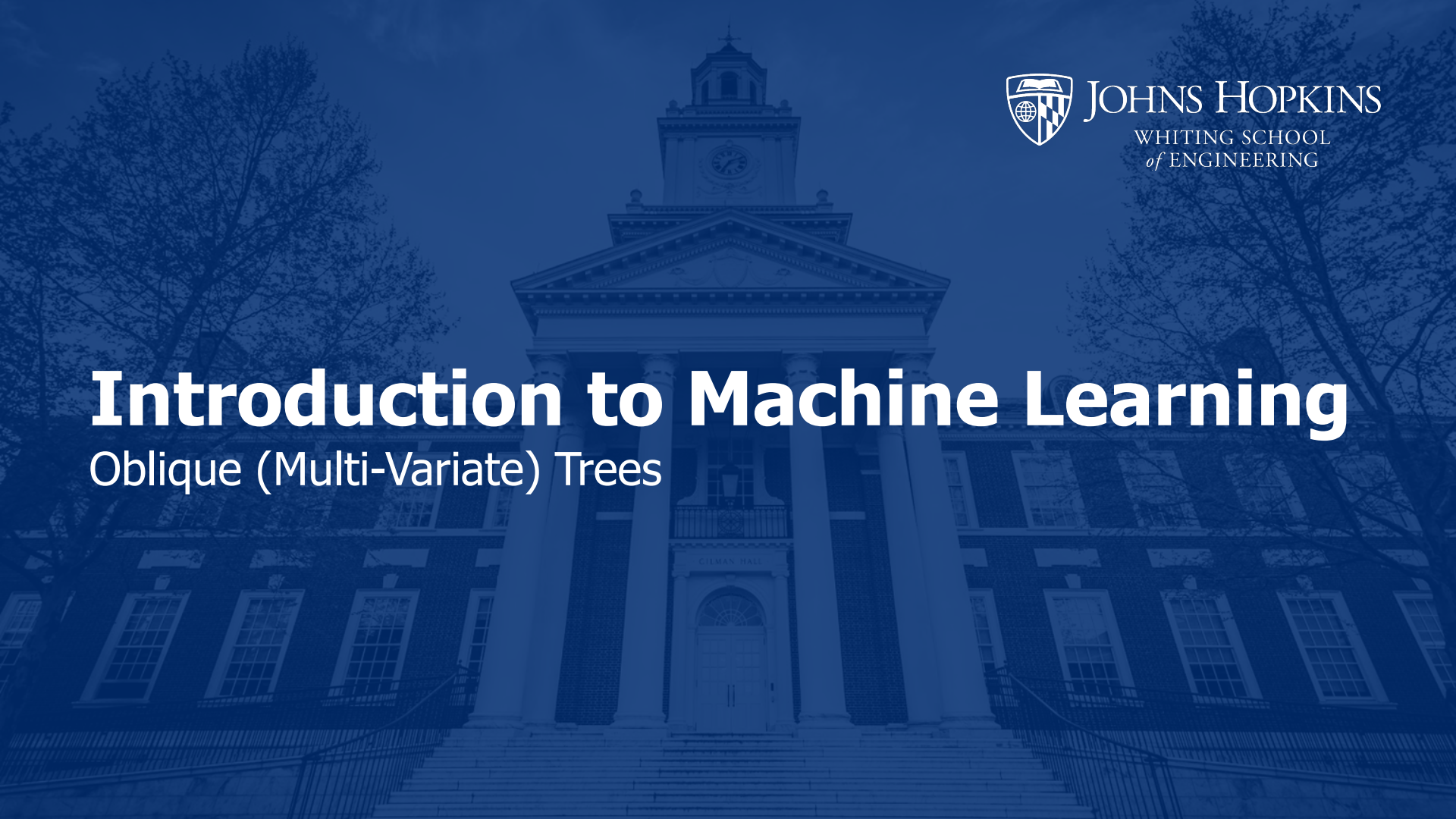


Introduction to Machine Learning

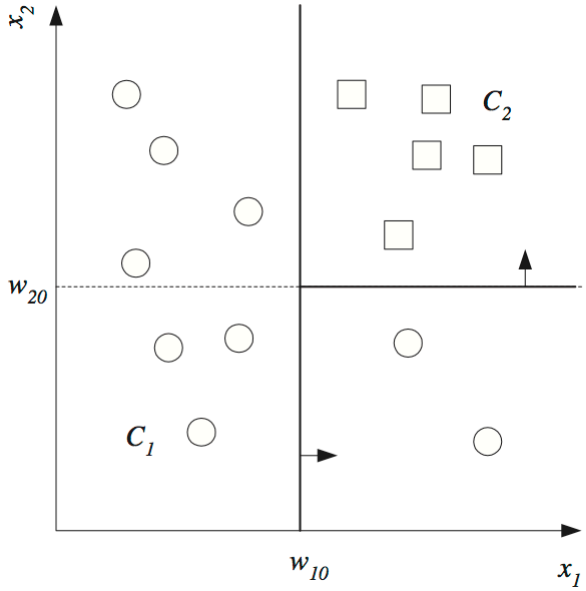
Oblique (Multi-Variate) Trees

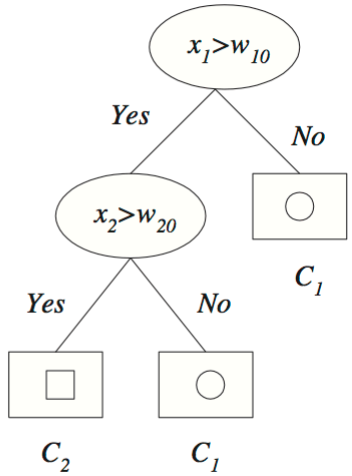
Preference Bias

- How we select the attributes in the tree
- Using information gain or MSE is a "greedy" choice
- Information gain incorporates the preference bias to reduce uncertainty/entropy in the data
- MSE incorporates the preference bias to reduce error
- Both tend to "prefer" shallow trees → supports generalization?

Representation Bias

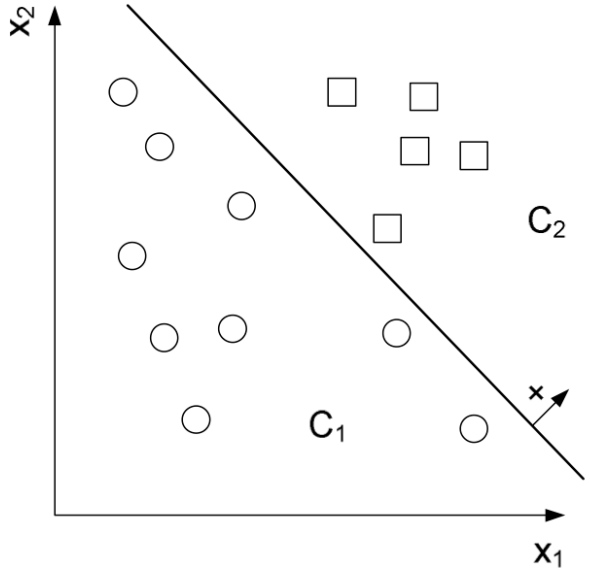
Most decision trees assume univariate splits (also called "axis parallel")

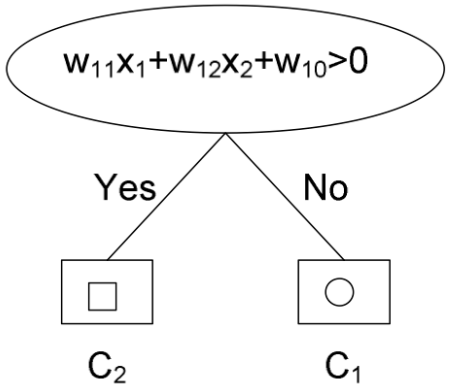




Oblique (Multi-variate) Trees

Replace each node with a split of the form $\alpha_0 + \sum_{i=1}^{d} \alpha_i x_i < \theta$





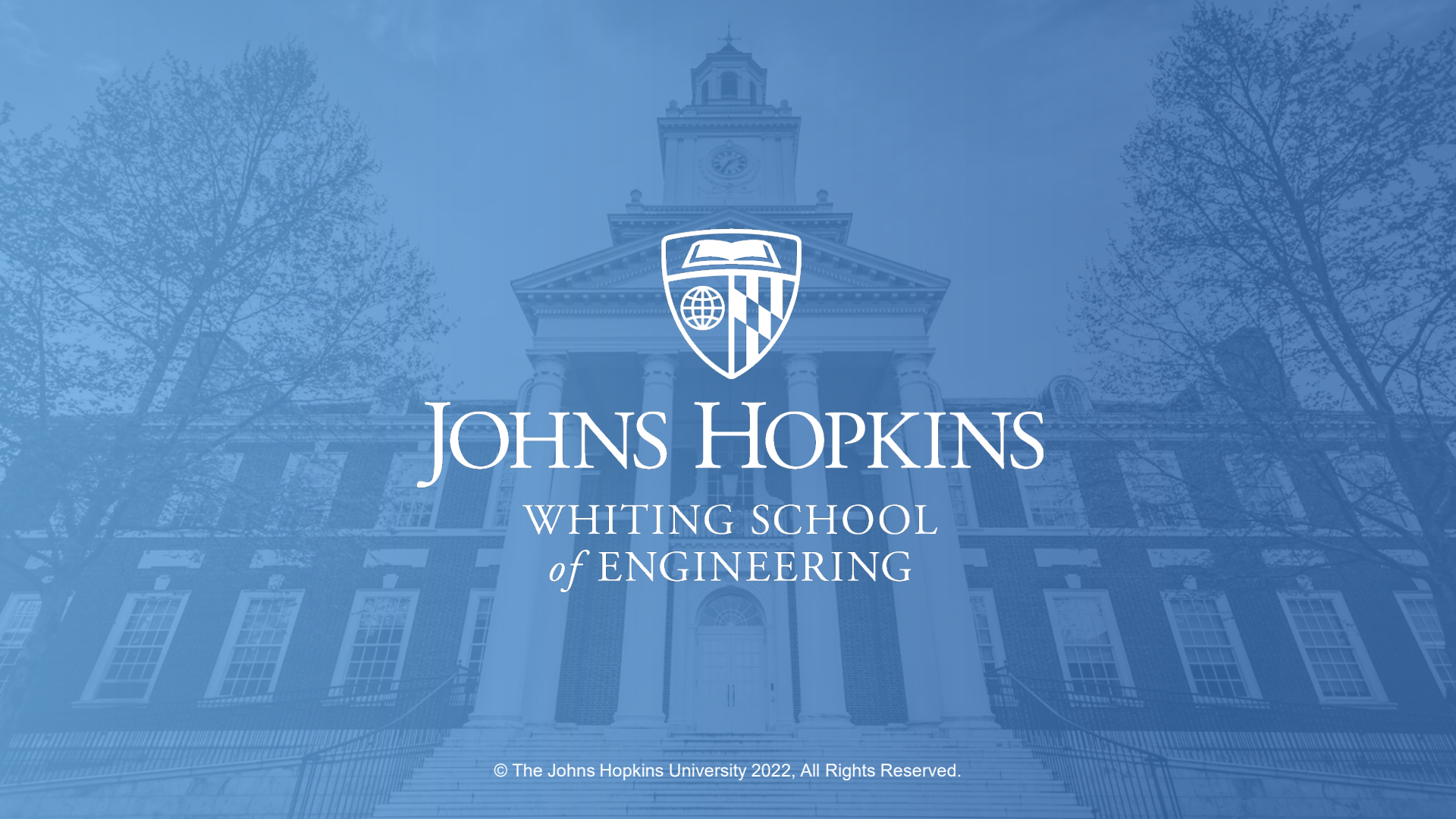
Oblique Classifier 1 (OC1)

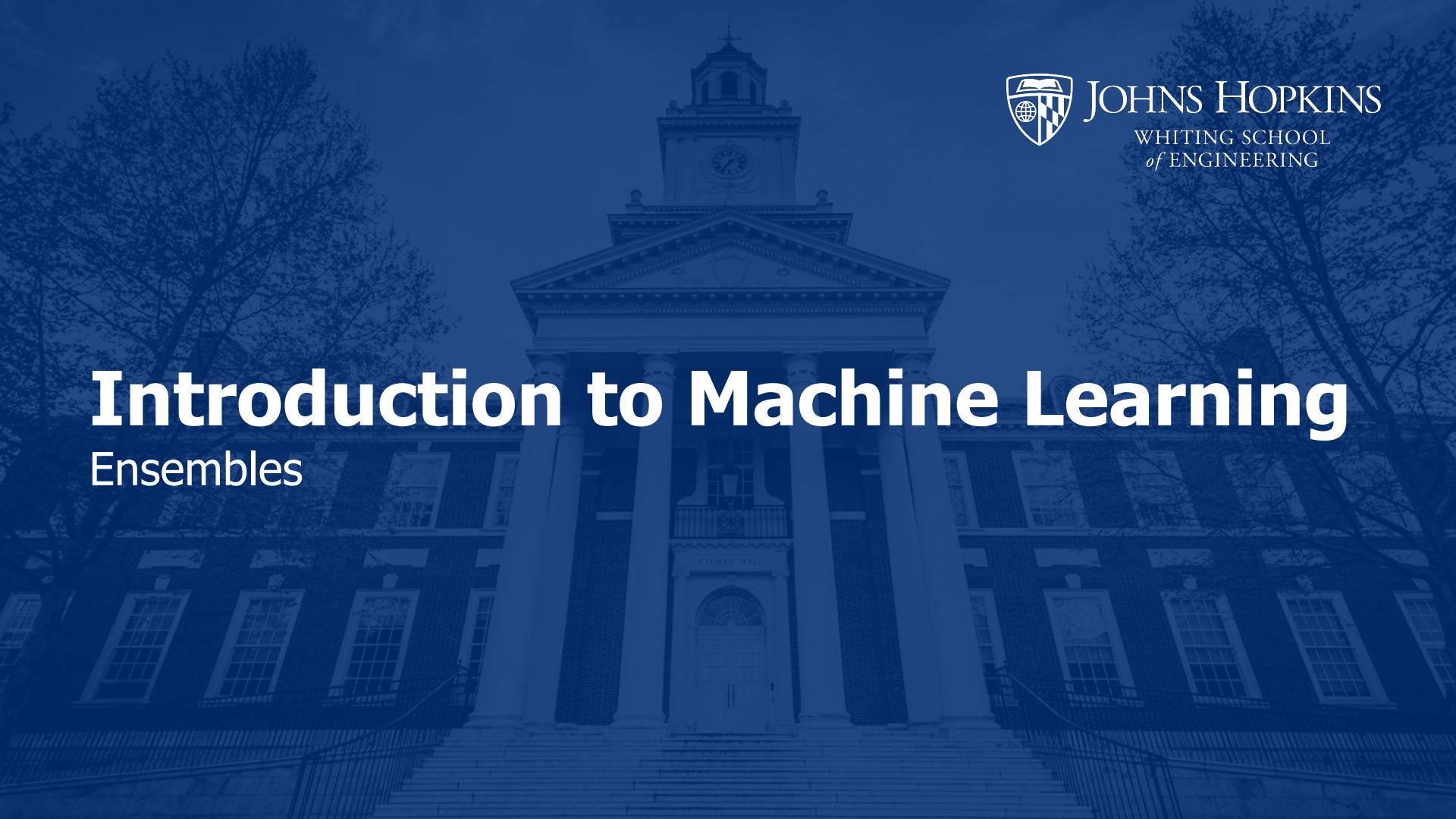
To find a split of a set of examples T:
- Find the best axis-parallel split of T. Let I be the impurity of this split.
- Repeat R times:
  - Choose a random hyperplane H.
  - (For the first iteration, initialize H to be the best axis-parallel split.)
  - Step 1: Until the impurity measure does not improve, do:
    - Perturb each of the coefficients of H in sequence.
  - Step 2: Repeat at most J times:
    - Choose a random direction and attempt to perturb H in that direction.
    - If this reduces the impurity of H, go to Step 1.
  - Let I_1 = the impurity of H. If $I_1 < I$, then set $I = I_1$.
- Output the split corresponding to I.

Perturbation

Perturb(H,m)
For $j = 1, \ldots, n$
Compute U_j **(Eq. 3.2)**
Sort $U_1, \ldots, U_n$ **in non-decreasing order.**
a'_m **= best univariate split of the sorted** U_j**s.**
H_1 **= result of substituting** a'_m **for** a_m **in** H**.**
If $(impurity(H_1) < impurity(H))$
{ $a_m = a'_m$; $P_{move} = P_{stag}$ }
Else if $(impurity(H) = impurity(H_1))$
{ $a_m = a'_m$ **with probability** P_{move}
$P_{move} = P_{move} - 0.1 * P_{stag}$ }

$$V_j = \alpha_0 + \sum_{i=1}^{d} \alpha_i x_i \qquad a_m > \frac{a_m x_{jm} - V_j}{x_{jm}}$$





Introduction to Machine Learning

Ensembles

Motivation

- It is difficult to determine the appropriate inductive bias for a learning method with respect to a target data set.
- Effect: Susceptibility to either overfitting (high bias/low variance) or underfitting (low bias/high variance).
- Approach:
  - Train multiple base learners (models)
  - Combine predictions from base learners
- Intended Result: Simultaneous reduction of bias and variance
- CAUTION!
  - The bias-variance tradeoff still applies
  - The ”No Free Lunch” Theorems still apply

Problem

- Given:
  - Training data set $\mathbf{X}$ for supervised learning.
  - $\mathbf{X}$ drawn from common instance space $\mathcal{X}$.
  - Collection of inductive learning algorithms.
- Hypotheses produced by applying inducers to $s(\mathbf{X})$

  $s : \mathbf{X} \rightarrow \mathbf{X}'$ (sampling, transformation, partitioning, etc.).
  - Consider hypotheses as $\mathbf{H}$ definitions of prediction algorithms ("based learners").
- Return: A new prediction algorithm (not necessarily $\in$ **H**) for $\mathbf{x} \in \mathbf{X}$ that combines outputs from collection of prediction algorithms.

Ensemble Methods

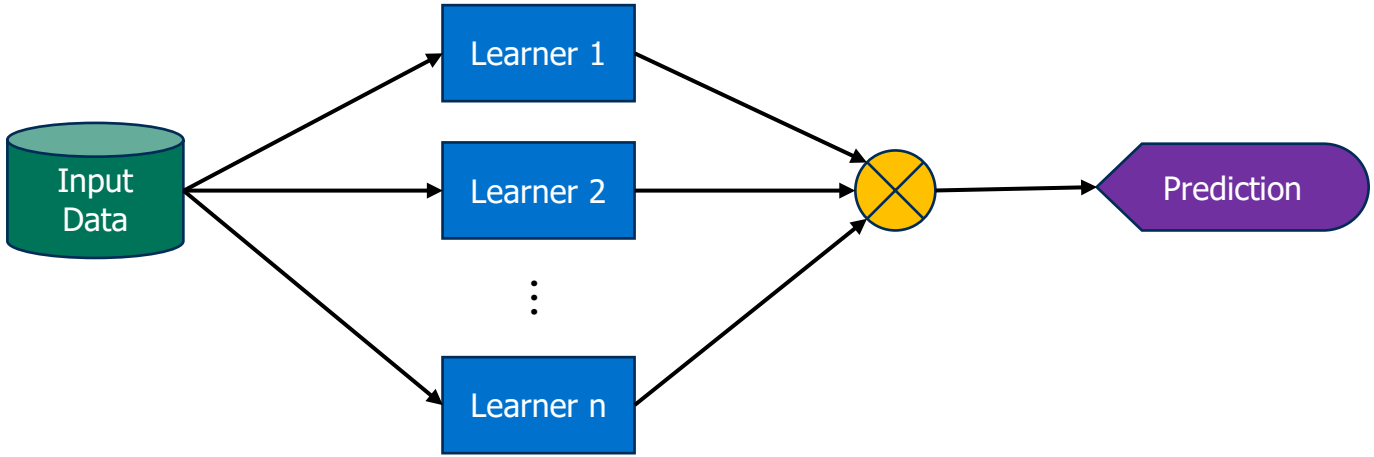
- Bagging
  - Short for "bootstrap aggregation"
  - Generate multiple sets of bootstrap samples and train on each set
  - Apply basic aggregation
- Boosting
  - A sequential process for training base learners
  - Each learner is trained based on the performance of previous learners
  - Apply a weighted aggregation scheme
- Stacking
  - Train different base learners (i.e., use different algorithms)
  - Train an aggregator to determine how best to combine

Critical Factors

- Accuracy
  - The expectation is that the base learners be trained to be as accurate as possible
  - Note that some ensemble methods only require "weak" learners, i.e., learners capable of performing only lightly better than chance
  - Accuracy comes from the aggregation process
- Diversity
  - The base learners need to have knowledge about different aspects of the task
  - Can be accomplished in a variety of ways
    - Diverse training sets
    - Reweighting of the same training set
    - Applying multiple methods with different inductive biases

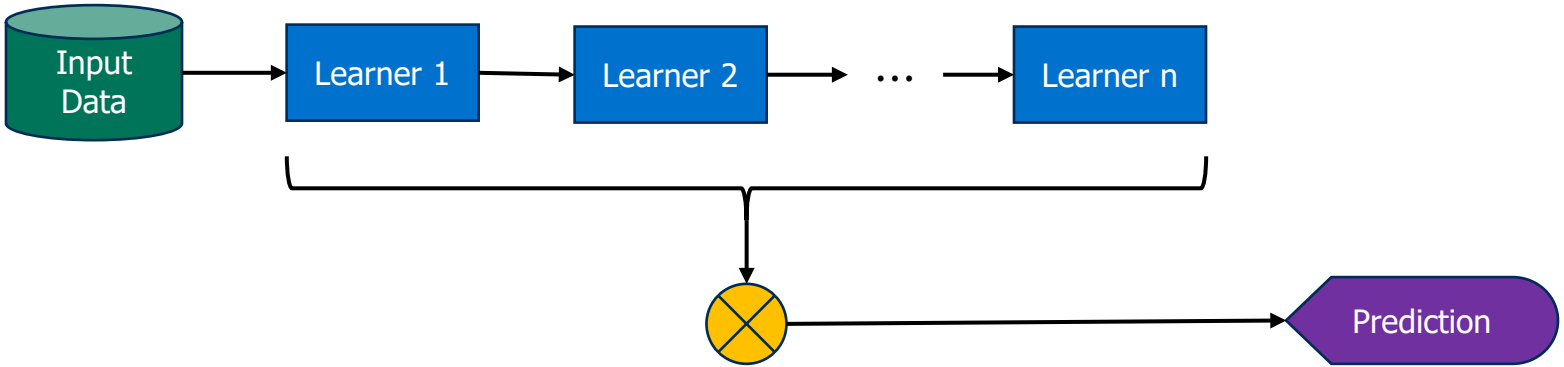
General Approaches: Parallel

- Train base learners independently
- Combine the results through simple aggregation



General Approaches: Sequential

- Train base learners iteratively
- Focus of training is correcting errors from previous learners



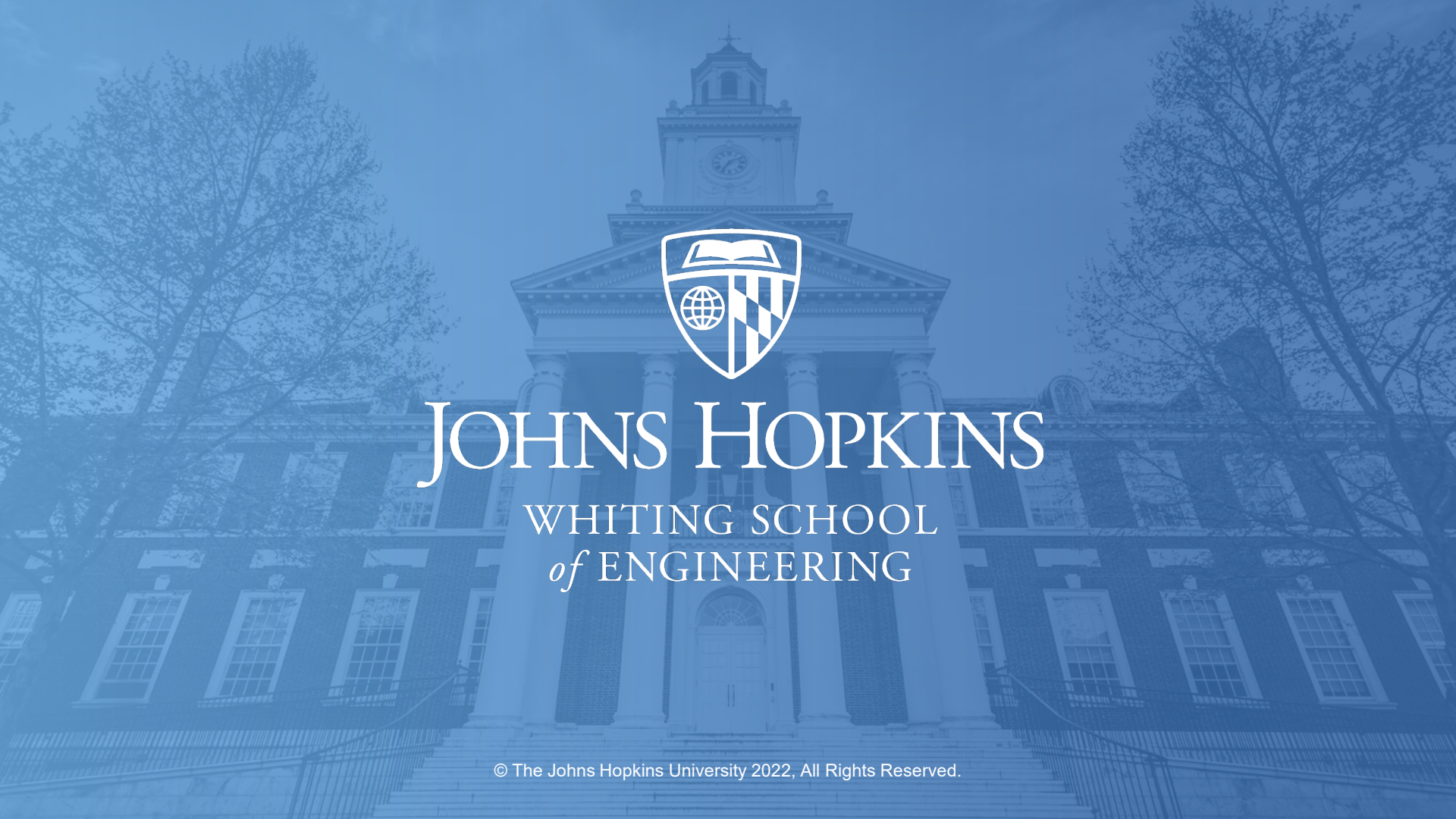
Diversity

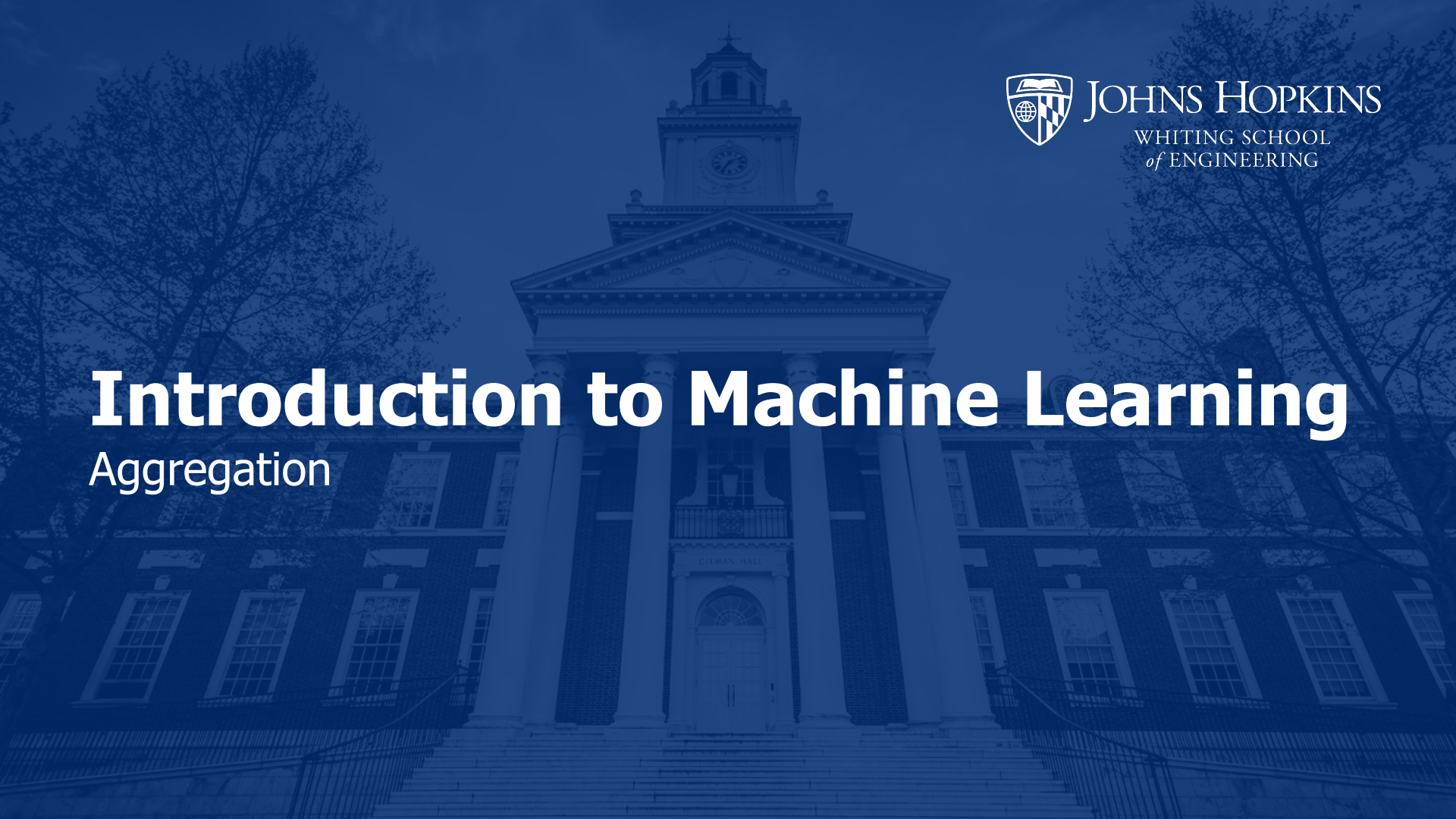
- The goal is to train a set of base learners negatively correlated with one another
- Ensemble diversity can be computed "pairwise" or "non-pairwise"
- Let $\mathcal{D}$ denote the ensemble of base learners
- A pairwise diversity measure (Yule's Q Statistic) – Want as close to –1 as possible
  - Let $y_{j,i}$ be an indicator of whether learner *i* recognizes data item $\mathbf{x}_j$ correctly, and let N^{ab} denote the number of times $y_{j,i} = a$ and $y_{j,k} = b$

$$Q_{i,k} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}} \quad \text{and} \quad \bar{Q} = \frac{1}{L(L-1)} \sum_{i=1}^{L-1} \sum_{k=i+1}^{L} Q_{i,k}$$

- A non-pairwise diversity measure (Entropy) – Want as close to +1 as possible
  - Let $\ell(\mathbf{x}_j)$ denote the number of learners that correctly classify $\mathbf{x}_j$

$$E(\mathcal{D}) = \frac{1}{N} \sum_{j=1}^{N} \frac{1}{(L - \lceil L/2 \rceil)} \min\{\ell(\mathbf{x}_j), L - \ell(\mathbf{x}_j)\}$$





Introduction to Machine Learning

Aggregation

Combining Learners

- In ensemble learning, the results of predictions from the base learners need to be combined to generate a final prediction.
- Such combiners are often referred to as "aggregators."
- Combining/aggregating predictions varies by task (classification/regression).
- The type of ensemble generally dictations the method of aggregation:
  - Bagging/boosting: Apply static combination rule after training
  - Stacking: Train an aggregation model based on performance of base learners

Majority Vote

- Arguably one of the "simplest" methods.
- Applied specifically to classification problems.
- Given a set of base learners $\mathcal{D}$, let $d_{t,c} \in \{0, 1\}$ denote whether classifier *t* indicated class *c*.
- Generate prediction as follows:

$$\text{class} = \text{argmax}_{c \in \mathcal{C}} \sum_{t=1}^{|\mathcal{D}|} d_{t,c}$$

Weighted Majority Vote

- Adjust majority vote based on performance of base learners.
- Applied specifically to classification problems.
- Given a set of base learners $\mathcal{D}$, let $d_{t,c} \in \{0, 1\}$ denote whether classifier *t* indicated class *c*.
- Let $w_t = L_t / \sum_{t'=1}^{|\mathcal{D}|} L'_t$ where L_t is the estimated accuracy of learner *t*.
- Generate prediction as follows:

$$\text{class} = \text{argmax}_{c \in \mathcal{C}} \sum_{t=1}^{|\mathcal{D}|} w_t d_{t,c}$$

Weighted Majority Vote

Algorithm 12.1 Weighted Majority Ensemble

```
1:  function CombineWeightedMajority(𝒟, ℒ)
2:      n ← |ℒ|                                   ▷ the number of models in ℒ
3:      m ← |𝒟|                                   ▷ the number of examples in the training set
4:      for i ← 1, n do
5:          ℓ_i ← Train(ℒ_i, 𝒟)
6:          w_i ← 1                               ▷ initialize weight to unity
7:      end for
8:      for j ← 1, m do
9:          q_0 ← 0; q_1 ← 0
10:         for i ← 1, n do
11:             if ℓ_i(x_j) = 0 then
12:                 q_0 ← q_0 + w_i               ▷ vote for class 0
13:             end if
14:             if ℓ_i(x_j) = 1 then
15:                 q_1 ← q_1 + w_i               ▷ vote for class 1
16:             end if
17:         end for
18:         ℓ_i(x_j) ← (q_0 > q_1) ? 0 : ((q_0 < q_1) ? 1 : rand(0, 1))
19:         for i ← 1, n do
20:             if ℓ_i(x_j) ≠ class'(x_j) then    ▷ class'(x_j) is the true class
21:                 w_i ← βw_i                    ▷ β ∈ (0, 1) is a penalty factor
22:             else
23:                 w_i ← w_i/β
24:             end if
25:         end for
26:     end for
27:     return MakePredictor(ℒ, w)
28: end function
```

Simple Aggregation – Regression

- The process for aggregation in regression is similar to that of classification.
- Replace vote with finding the (weighted) mean (or median).
- Let $\hat{y}_t$ denote the prediction/response from the tth base learner
- Let $w_t = (1/L_t) / \sum_{t'=1}^{|\mathcal{D}|} (1/L_{t'})$ where L_t is the estimated error of learner *t*.
- For unweighted:

$$\text{resp} = \frac{1}{|\mathcal{D}|} \sum_{t=1}^{|\mathcal{D}|} \hat{y}_t$$

- For weighted:

$$resp = \sum_{t=1}^{|\mathcal{D}|} w_t \hat{y}_t$$

Ensemble Selection

- Conventional wisdom indicates as the number of base learners increases, accuracy and resistance to overfitting should improve.
- As the number of base learners increases, so does computational complexity.
- As the number of base learners increase, diversity can start to decrease, thus *reducing* the performance (accuracy) of the ensemble.
- This suggests there might be an "optimal" subset of base learners to be used in a given ensemble.

Ensemble Selection

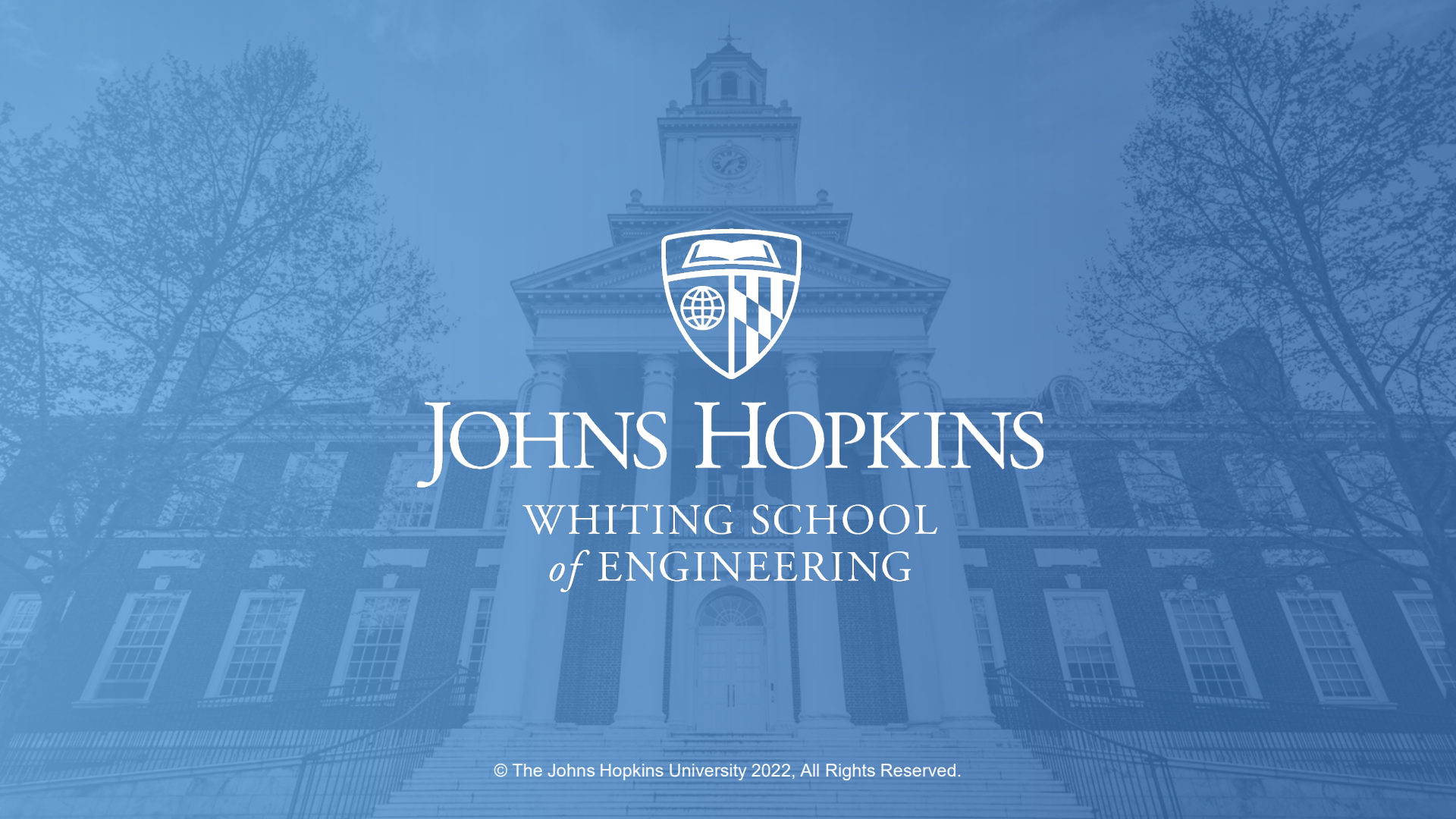
- Selecting the base learners to make up an ensemble is a "subset selection" problem.
- Approaches to subset selection are also applied in "feature" selection (addressed in the module on dimensionality reduction).
- Common approaches:
  - Stepwise selection:
    - Forward
    - Backward
    - Local search
  - Wrapper-based
  - Filter-based

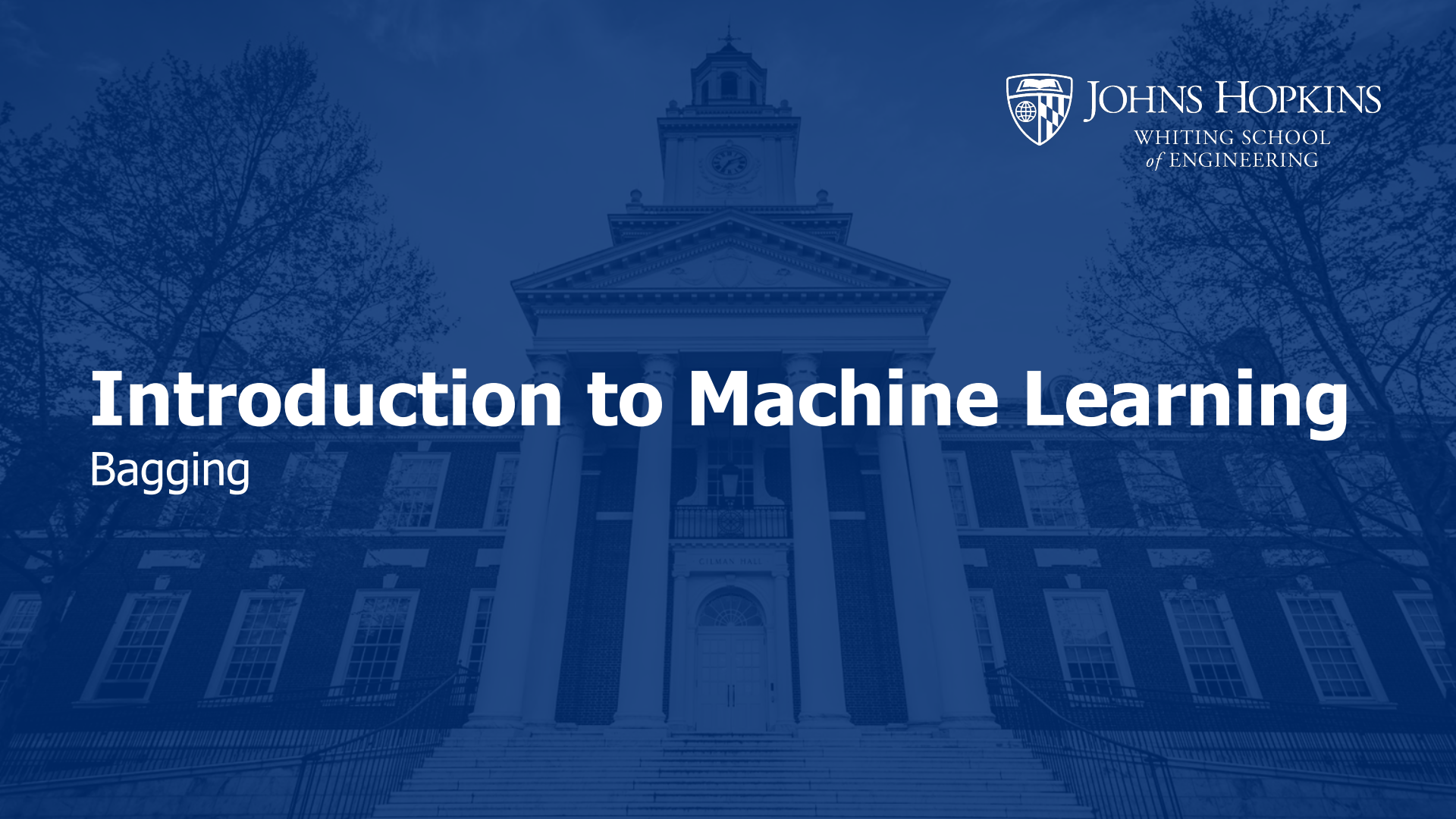
Forward Model Selection

A wrapper-based approach:

1. Begin with an empty ensemble
2. Select the base learner from a library of base learners
   a. Approach 1: based on individual model performance on a validation set
   b. Approach 2: based on maximizing ensemble performance on a validation set
3. Repeat step 2 as long as ensemble performance continues to improve, or until all models in the library have been examined
4. Return the selected subset of based learners

Assumes the models can be ordered based on performance, but still requires evaluation of the ensemble as it is generated; this is what makes it a wrapper method.





Introduction to Machine Learning

Bagging

Bootstrap Aggregation

- Perhaps one of the most basic methods for constructing ensembles.
- Has strong statistical properties from which uncertainty quantification can be performed.
- Referred to as "bagging" for short.
- Results in a set of "diverse" base learners due to the fact they are trained on different subsets of the training data.

The Bagging Process

- Generate *b* different bootstrap training sets, $\mathbf{X}_1, \ldots, \mathbf{X}_b$ each of size $|\mathbf{X}|$
- Each data set $\mathbf{X}_i$ is generated by sampling from $\mathbf{X}$ *with replacement* until $|\mathbf{X}|$ examples are selected.
- Train *b* different base learners using the appropriate data set.

  In other words, train base learner $\mathcal{D}_i$ using $\mathbf{X}_i$ as the training set.
- The combiner is (weighted) majority or (weighted) mean/median.
- **REMARK:** Bagging is optimized when the base learners exhibit high variance and low bias, such as a decision tree.

Random Forests

- Breiman defines a random forest as follows.

  Def: A *random forest* is a classifier consisting of a collection of tree-structured classifiers $\{h(\mathbf{x}, \Theta_k), k = 1, \ldots, m\}$ where $\{\Theta_k\}$ are independent, identically distributed random vectors and each tree casts a unit vote for the most popular class at input $\mathbf{x}$.

- This can be adapted for regression using averaging instead of voting.
- A common variant on this method is to also select a random subset of features from which to generate the trees.
- **REMARK:** Several cases have been demonstrated where random forests show competitive performance on tasks often ”solved” via deep learning.

Random Forests

Algorithm 12.3 The Random Forest Algorithm

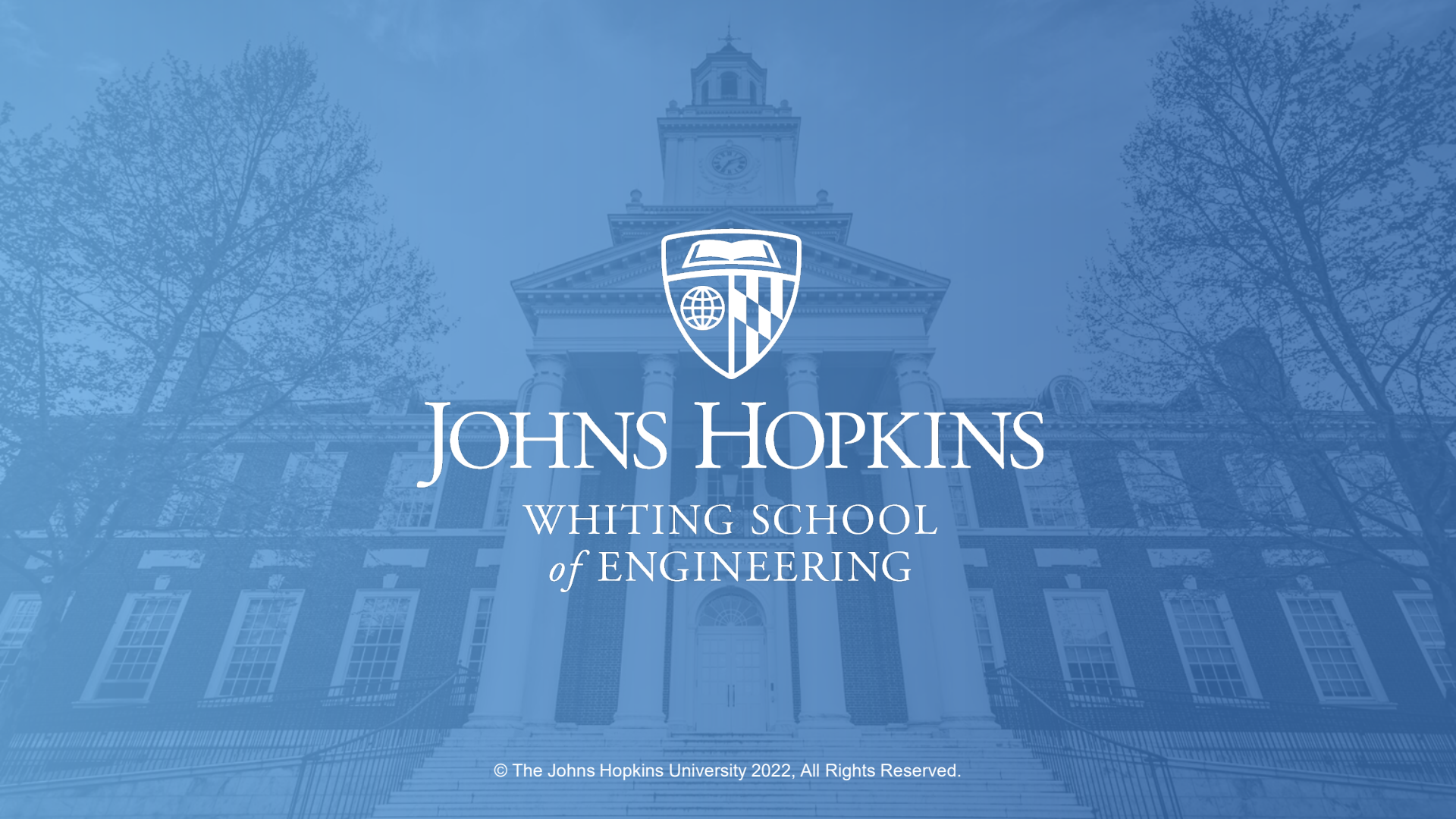
```
1: function RandomForest(D, k, n)
2:     // D is the training set.
3:     // k is the number of available attributes
4:     // n is the number of classifiers to be constructed
5:     L ← ∅
6:     for i ← 1 to n do
7:         D_i ← RandSubset(D, k)
8:         ℓ_i ← DTSubsetLearn(D_i)
9:         L ← L ∪ ℓ_i
10:    end for
11:    return L
12: end function
```

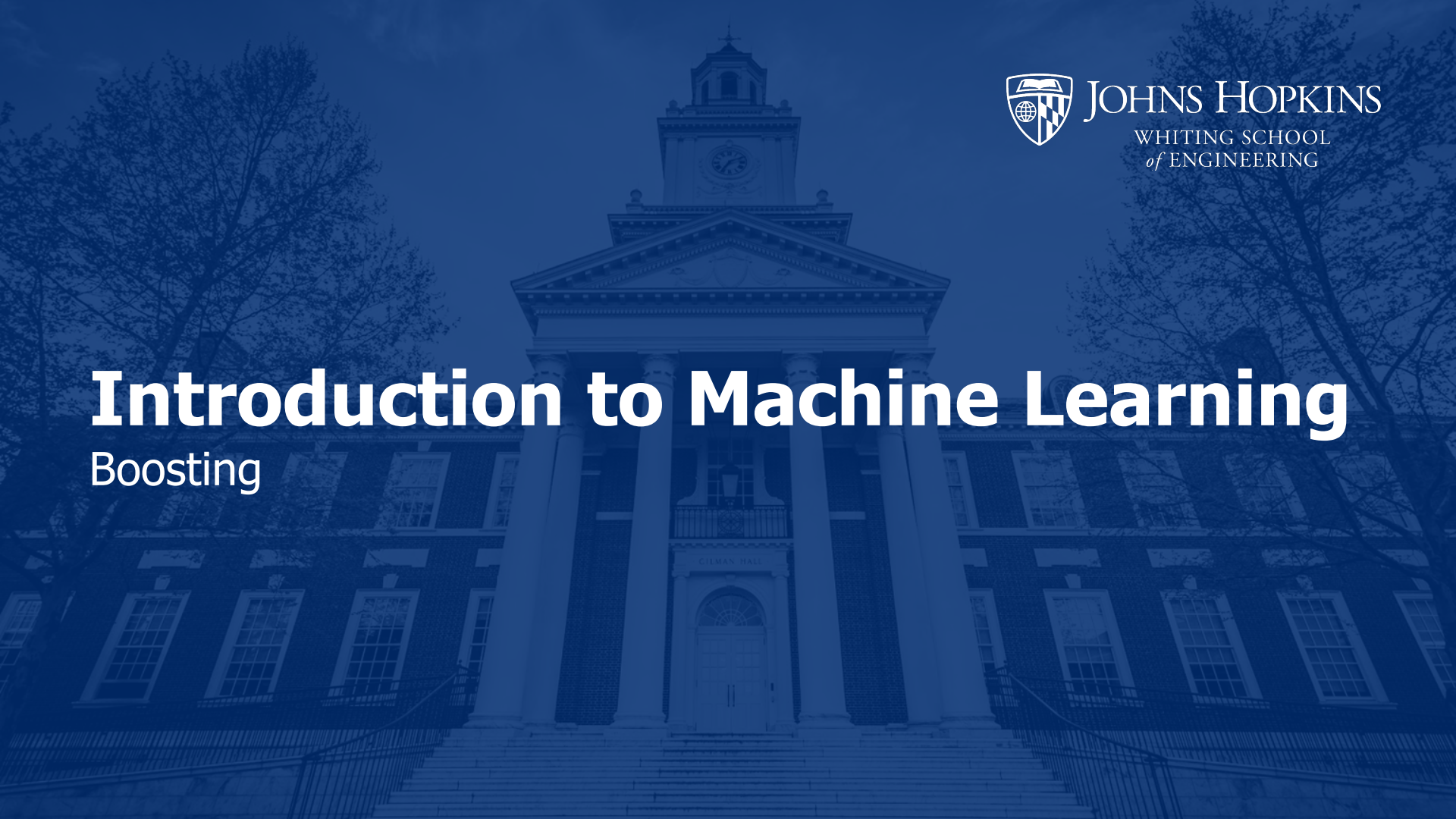
Properties of Random Forests

- The trees in a RF are simpler and therefore easier to construct.
- RF ensembles have been shown to be highly accurate and resistant to overfitting (ensembles naturally regularize).
- Individual trees are considered to be relatively easy to interpret (although, full ensembles are not).
- RF ensembles provide a level of interpretability through the ability to do variable importance analysis.
- Technically, RF ensembles do not need cross-validation – can use out-of-bag samples for generalization assessment. Often, we still cross-validate.
- RF ensembles can also be used for clustering by counting the number of times pairs of points appear in the same leaves across the set of trees.
- There exist many variants.

Possible Variants

- Individual trees have all features available and grow trees with optimal splits fully on bootstrapped samples.
- Individual trees have a random subset of features available and grow trees with optimal splits on bootstrapped samples.
- Individual trees are grown with random splits.
- Only “stumps” are used as base learners where each stump is generated based on information gain.
- Only “random stumps” are used as base learners.
- Oblique (multi-variate) splits are allowed when generating the trees.





Introduction to Machine Learning

Boosting

Boosting

- Invented independently by Freund (1990) and Shapire (1990).
- A sequential method of constructing ensembles by re-weighting the training data based on prior performance.
- As a sequential (iterative) process, stops generating base learners either when a target number (tunable) are generated or when performance of the ensemble has plateaued.
- Assumes weak learners, consisting of high bias and low variance.
- Tends to work poorly with noisy data since it tends to over-weight the noisy examples.

Basic Boosting

The ”original” boosting algorithm only built an ensemble of three base learners.

1. For training set $\mathbf{X}$, where $|\mathbf{X}| = n$ sample $n_1 < n$ examples $\mathbf{X}_1 \subset \mathbf{X}$ without replacement.
2. Train base learner ℓ_1 on this sample.
3. Sample a second set of $n_2 < n$ examples $\mathbf{X}_2 \subset \mathbf{X}$ such that
   a. The first $n_2/2$ examples correspond to the misclassified examples from $\mathbf{X}_1$ when using ℓ_1
   b. The remaining $n_2/2$ are drawn from $\mathbf{X}$ without replacement but correctly classified by ℓ_1
4. Train based learner ℓ_1 from $\mathbf{X}_2$
5. Apply ℓ_1 and ℓ_2 to $\mathbf{X}$ and extract all examples where they disagree to yield $\mathbf{X}_3 \subset \mathbf{X}$
6. Train ℓ_3 on $\mathbf{X}_3$
7. Use ensemble via majority vote.

Adaptive Boosting

Algorithm 12.2 Discrete AdaBoost Algorithm

function AdaBoost($\mathcal{D}$, $\mathcal{L}_{weak}$, L)
  // $\mathcal{D}$ is the training set.
  // $\mathcal{L}_{weak}$ is a weak learning algorithm
  // L is the number of classifiers to be constructed
  $\mathbf{w}^1 \leftarrow [1 \cdots 1]$ ▷ $\mathbf{w}$ has $1/|\mathcal{D}|$ component values; the distribution is uniform.
  for $i \leftarrow 1, L$ **do**
    $\mathbf{p}^i \leftarrow \frac{\mathbf{w}^i}{\sum_{j=1}^{|\mathbf{D}|} w_j^i}$ ▷ Derives sampling distribution from weights
    $\ell_i \leftarrow Train(\mathcal{L}_{weak}, \mathcal{D}, \mathbf{p}^i)$ ▷ Train after sampling using $\mathbf{p}^i$
    $error_i \leftarrow \sum_{j=1}^{|\mathcal{D}|} p_j^i |\ell_i(\mathbf{x}_j) - y_j|$
    $\beta_i = error_i/(1 - error_i)$
    for $j \leftarrow 1, |\mathcal{D}|$ **do**
      $w_j^{i+1} \leftarrow w_j^i \beta_i^{1-|\ell_i(\mathbf{x}_j)-y_j|}$
    end for
  end for
  return $\ell_1, \ldots, \ell_L, \beta_1, \ldots, \beta_L$
end function

- Often referred to as "AdaBoost."
- Often uses decision trees (or decision stumps) as base learners, but this is not required.
- Algorithm to the left is also referred to as the AdaBoost.M1 algorithm.
- Choose class as

$$\text{class} = \text{argmax}_c \sum_{i=1}^{L} \left(\log \frac{1}{\beta_i} \right) \ell_i^c (\mathbf{x}_j)$$

Advanced Boosting

- More advanced boosting algorithms have been developed in recent years.
- Examples of prominent algorithms include:
  - **Gradient Boosting** – Train learners correlated with negative gradient of the loss function with respect to the entire ensemble.
  - **XGBoost** – Gradient boosting with regularization added to the loss function.
  - **LightGBM** – Gradient boosting with one-sided sampling (reduces training examples needed) and feature bundling (reduces features to be evaluated).
  - **CatBoost** – Omits the sample being used to train when estimating the gradient, similar to a "jackknife" sample.
- To focus on common characteristics, we will discuss basic gradient boosting.

Gradient Boosting

- Core Idea: Develop base learners/models that are correlated highly with the gradient of the loss function for the entire ensemble.
- The goal is to learn an approximation $\hat{F}(\mathbf{x})$ of an underlying target function $F(\mathbf{x})$ by minimizing loss function $\mathcal{L}(y, F(\mathbf{x}))$ and constructing a weighted sum of functions.

$$\hat{F}_m(\mathbf{x}) = \hat{F}_{m-1}(\mathbf{x}) + \rho_m f_m(\mathbf{x})$$

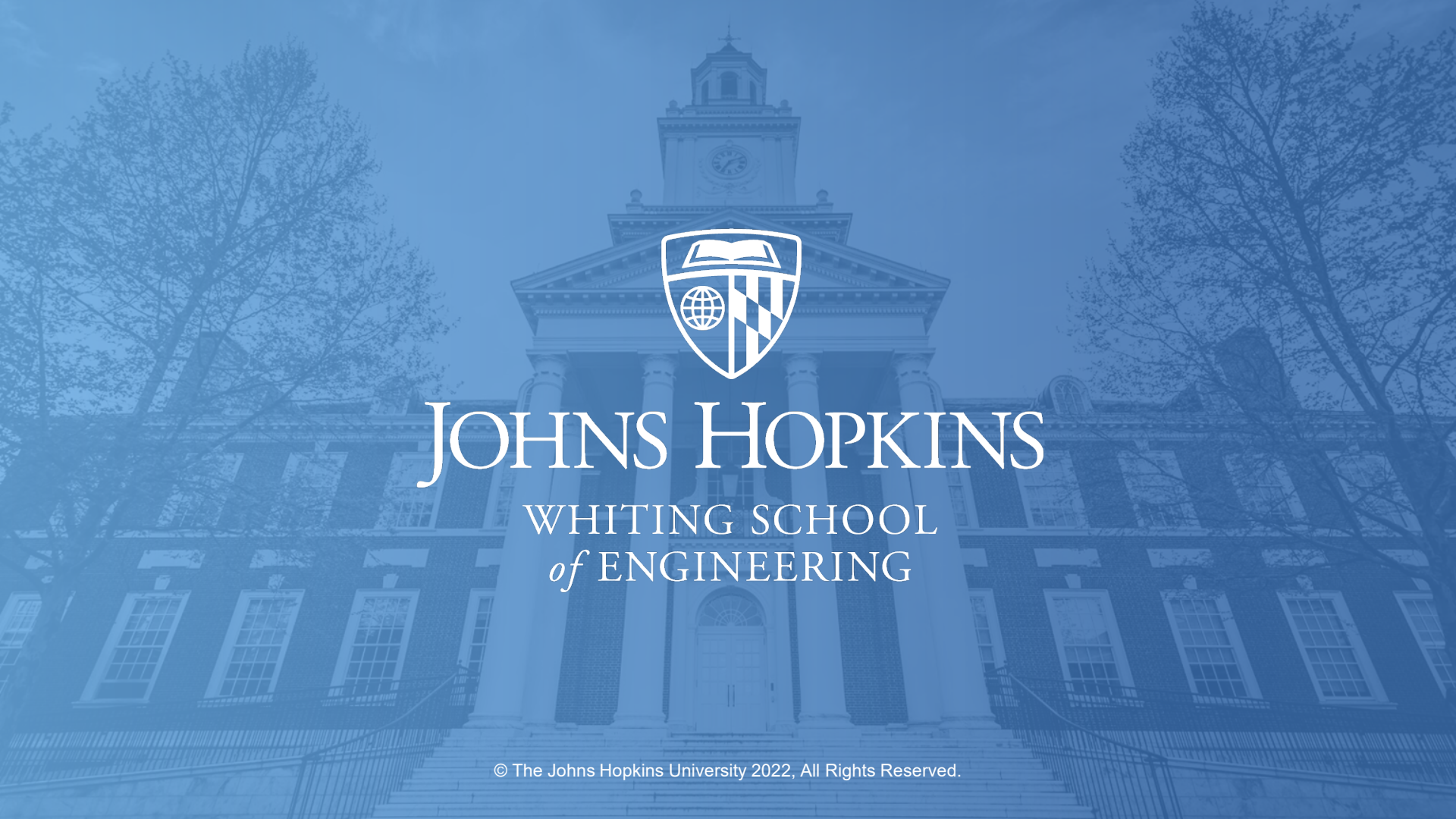
  where ρ_m is the weight of the mth function, $f_m(\mathbf{x})$ Here, we assume that $\hat{F}_0(x) = \text{argmin}_f \mathcal{L}(y, F(\mathbf{x}))$
- Each successive base learner finds $\rho_m f_m(\mathbf{x}) = \text{argmin}_{\rho,f} \sum_{i=1}^{n} \mathcal{L}\left(y_i, \hat{F}_{m-1}\left(\mathbf{x}_i\right) + \rho f\left(\mathbf{x}_i\right)\right)$
- Significantly, the first learner trains on the original data, and subsequent learners train on the *residuals* of the ensemble so far.

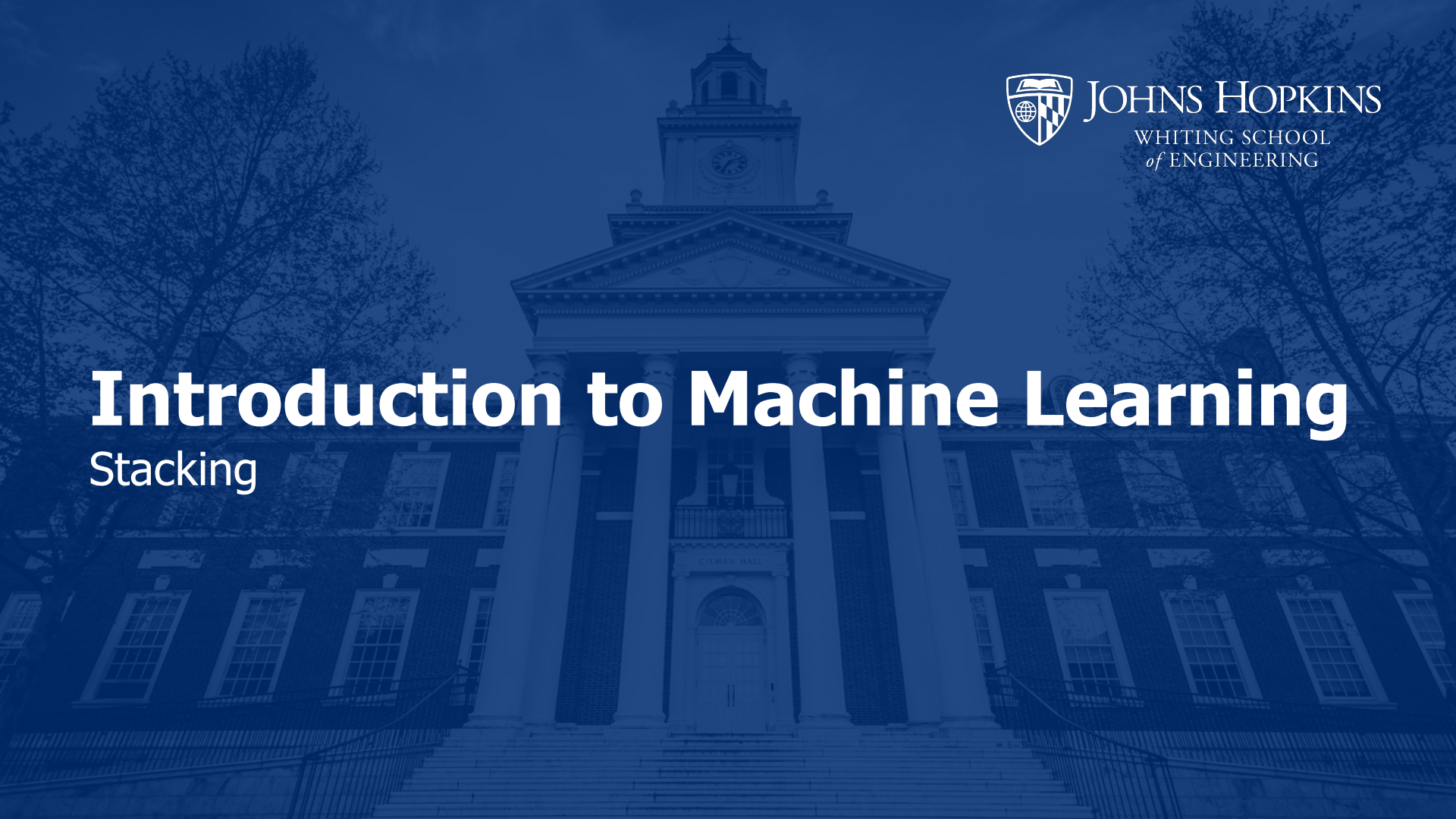
$$r_{mi} = \left.\frac{\partial \mathcal{L}\left(y_i, \hat{F}(\mathbf{x})\right)}{\partial \hat{F}(\mathbf{x})}\right|_{\hat{F}(\mathbf{x})=\hat{F}_{m-1}(\mathbf{x})}$$

Gradient Boosting

Algorithm 1.1 Gradient Boosting

function GRADBOOST($\mathbf{X}, \mathcal{L}, T$)
  Initialize with $\hat{F}_0(\mathbf{X}) = \underset{f}{\operatorname{argmin}} \sum_{\mathbf{x}_i \in \mathbf{X}} \mathcal{L}(y_i, f(\mathbf{x}_i))$
  for $m = 1, \ldots, M$ **do**
    for $\mathbf{x}_i \in \mathbf{X}$ **do**
      Calculate $r_{mi} = \left. \frac{\partial \mathcal{L}(y_i, \hat{F}(\mathbf{x}_i))}{\partial \hat{F}(\mathbf{x}_i)} \right|_{\hat{F}(\mathbf{X}) = \hat{F}_{m-1}(\mathbf{X})}$
    end for
    Train $f_m(\mathbf{X})$ using $\mathcal{D}_m = \{\mathbf{x}_i, r_{mi}\}_{\mathbf{x}_i \in \mathbf{X}}$
    $\rho_m f_m(\mathbf{X}) = \underset{\rho, f}{\operatorname{argmin}} \sum_{\mathbf{x}_i \in \mathbf{X}} \mathcal{L}(y_i, \hat{F}_{m-1}(\mathbf{x}_i))$
    $\hat{F}_m(\mathbf{X}) = \hat{F}_{m-1}(\mathbf{X}) + \rho_m f_m(\mathbf{X})$
  end for
  return $\hat{F}_m(\mathbf{X})$
end function



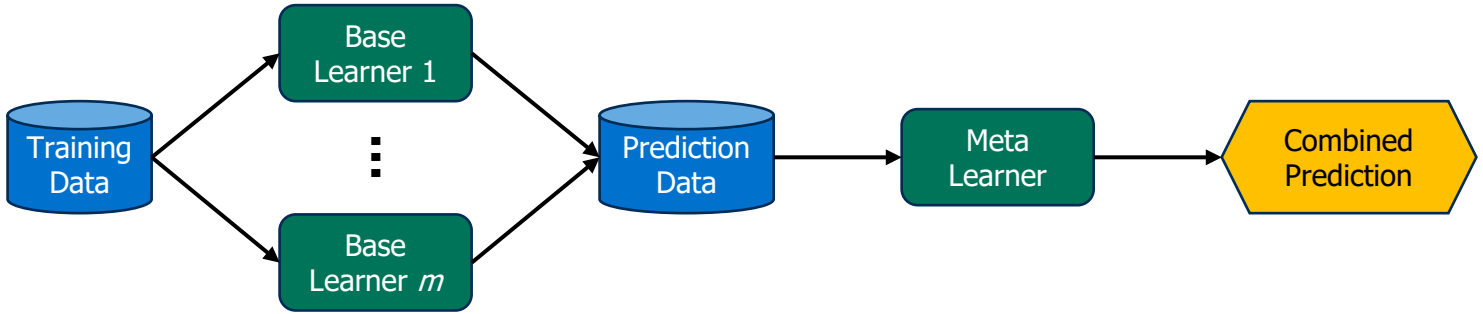


Introduction to Machine Learning

Stacking

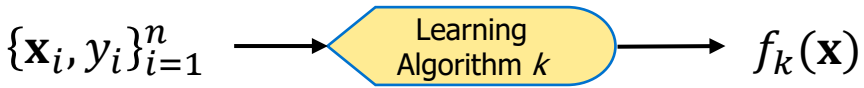
Stacking Introduction

- So far, all of the methods examined have assumed the models in the ensemble are of the same type.
- Stacking is a method that allows different types of models to be combined.
- The primary advantage is that stacking allows models with different inductive biases to contribute to the overall prediction based on their individual strengths.
- In stacking, a meta-learner is used to combine the predictions.

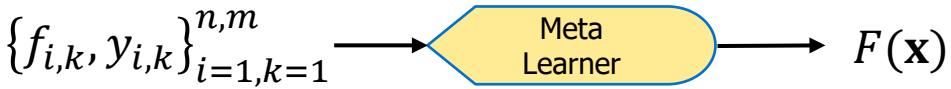


Learning for Stacking

- Based learners are referred to as "Level 0" models.
- The meta-learner result in a "Level 1" model.
- The purpose of the Level 1 model is to combine predictions.
- The data set for the Level 1 model consists of (predictions, target) pairs.
- Level 0:



- Level 1:



Stacking

Algorithm 1.1 Stacking

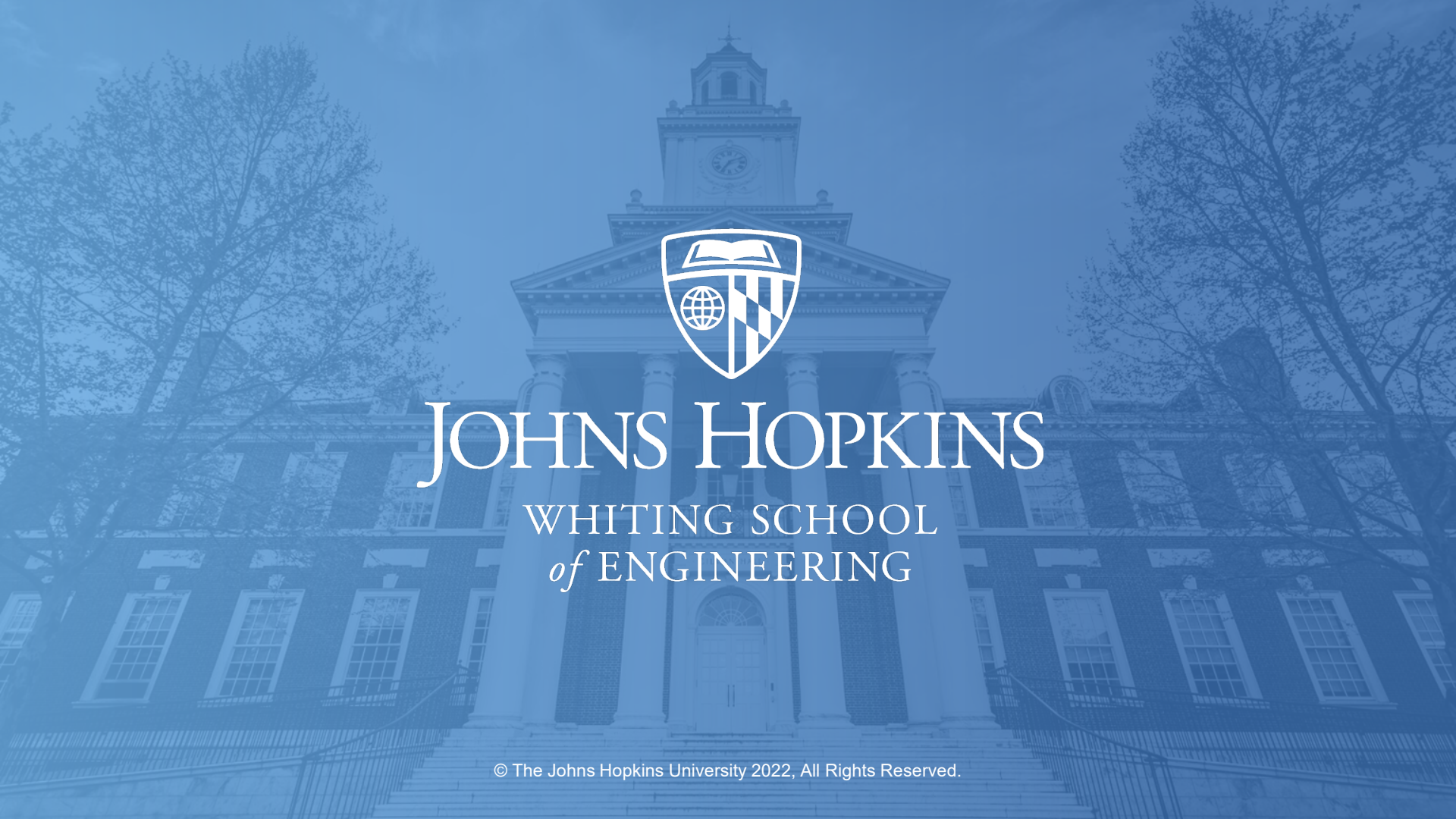
```
1: function STACK(X, T)
2:     for k = 1, ..., m do
3:         Train f_k(X) using X
4:     end for
5:     H = ∅
6:     for k = 1, ..., m do
7:         f(x_i) = []
8:         for i = 1, ..., |X| do
9:             f(x_i) ← f(x_i)||f_k(x_i)
10:        end for
11:        H ← H ∪ {(f(x_i), y_i)}
12:    end for
13:    Train F(X) using H
14:    return f_1(X), ..., f_m(X), F(X)
15: end function
```

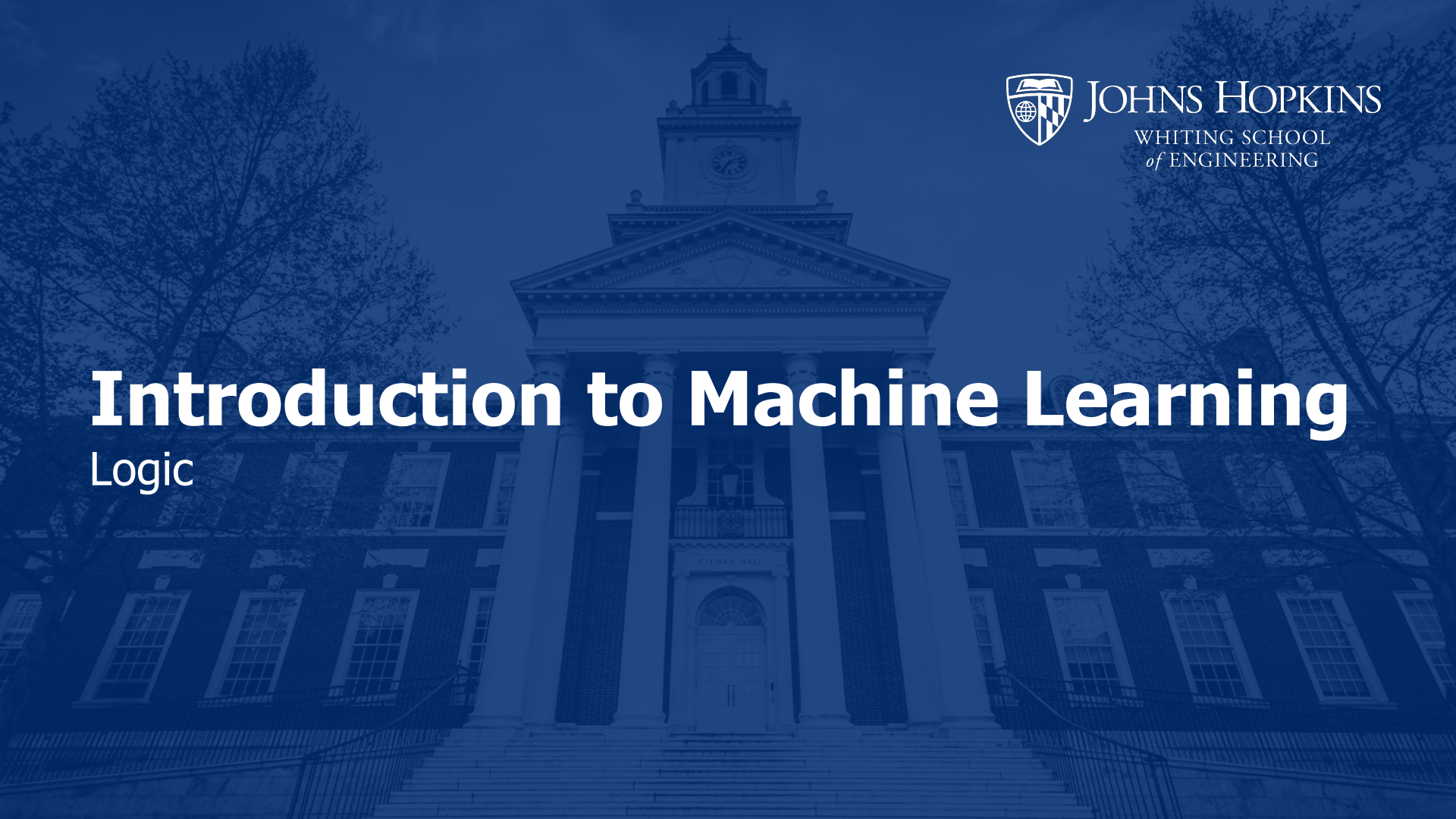
Types of Models

- Different types of models are used for the level 0 models (e.g., SVM, decision tree, neural network).
- For classification problems, the meta-model can be anything but is often logistic regression.
- If linear classifier insufficient, can move to a more sophisticated model.
- For regression problems, the meta-model can be anything but is often linear regression.
- Similar to classification, if linear regression insufficient, can move to a more sophisticated model.

Overfitting

- The conventional wisdom is that all ensemble methods are robust with respect to overfitting.
- Regularization is based upon diversity among members of the ensemble.
- Bagging – Diversity comes from the individual data sets.
- Boosting – Diversity comes from re-weighting the data and then weighting the base learners.
- Gradient Boosting – Diversity comes from focusing on combining predictions focused on the residuals.
- Stacking – Diversity comes from the different inductive biases of the base learners.





Introduction to Machine Learning

Logic

Logic Motivation

- One of the original problem areas in AI was mathematical theorem proving: Logic Theorist, GPS.
- First complete inference procedure was computational.
- Early on many researchers realized that it was essential to have a "formal" language for talking about knowledge.
- Logic seems like the obvious language.
- Logic and logical inference are generally viewed as essential to symbolic AI.
- Logic is also something that, compared to models like neural networks, tends to be "easy" to understand, thus supporting *explainable* methods in machine learning.

Propositional Logic

- The simplest formal logic.
- Largely based on set theory.
- Forms the foundation for most other logics.
- Will present syntax and semantics.

Syntax of Propositional Logic

- The constants T and F are sentences.
- A propositional symbol (e.g., *P*, *Q*, *R*) is a sentence.
- If α is a sentence and *β* is a sentence, then the following are sentences.

$$(\alpha)$$

$$\neg\alpha$$

$$\alpha \vee \beta$$

Propositional Logic Semantics

- Logical constants T and F have fixed interpretations (i.e., True and False respectively).
- If sentence α is True, then so is (α).
- If sentence α if False, then $\neg\alpha$ is True.
- If either α or β is True, then $\alpha \vee \beta$ is True.

Truth Tables

Negation

α	$\neg \alpha$
T	F
F	T

Disjunction

α	β	$\alpha \vee \beta$
T	T	T
T	F	T
F	T	T
F	F	F

More Truth Tables

Conjunction

α	β	$\alpha \wedge \beta$
T	T	T
T	F	F
F	T	F
F	F	F

$\neg(\neg\alpha \vee \neg\beta)$

Implication

α	β	$\alpha \Rightarrow \beta$
T	T	T
T	F	F
F	T	T
F	F	T

$\neg\alpha \vee \beta$

Equivalence

α	β	$\alpha \Leftrightarrow \beta$
T	T	T
T	F	F
F	T	F
F	F	T

$\neg(\neg(\neg\alpha \vee \beta) \vee \neg(\neg\beta \vee \alpha))$

Propositional Inference

- Double-Negation: $[\neg\neg\alpha] \vdash \alpha$
- AND-Elim: $[\alpha_1 \wedge \cdots \wedge \alpha_n] \vdash \alpha_i$
- AND-Intro: $[\alpha_1, \ldots, \alpha_n] \vdash [\alpha_1 \wedge \cdots \wedge \alpha_n]$
- OR-Intro: $[\alpha_i] \vdash [\alpha_1 \vee \cdots \vee \alpha_i \vee \cdots \vee \alpha_n]$
- Modus Ponens: $[\alpha \Rightarrow \beta, \alpha] \vdash \beta$
- Modus Tollens: $[\alpha \Rightarrow \beta, \neg\beta] \vdash \neg\alpha$
- Unit Resolution: $[\alpha \vee \beta, \neg\beta] \vdash \alpha$
- Resolution: $[\alpha \vee \beta, \neg\beta \vee \delta] \vdash \alpha \vee \delta$
- Subsumption: $[(\alpha \wedge \beta) \vee \beta] \vdash \beta$

Normal Forms

- **Conjunctive Normal Form (CNF)**
  - A conjunction of disjunctions (clauses) of literals.

    $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
- **Disjunctive Normal Form (DNF)**
  - A disjunction of conjunctions (terms) of literals.

    $(A \wedge B) \vee (A \wedge \neg C) \vee (A \wedge \neg D) \vee (\neg B \wedge \neg C) \vee (\neg B \wedge \neg D)$
- **Horn Form**
  - A conjunction of Horn clauses (clauses with ≤ 1 positive literal)

    $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$
  - Often written as a set of implications (i.e., implicative normal form)

    $B \Rightarrow A$ and $(C \wedge D) \Rightarrow B$

First-Order Logic

- Also called Predicate Calculus
- Extend language beyond propositions
  - Constants (individuals in the world)—*KingJohn*, 2, *MSU,* …
  - Variables (e.g., *x*, *y*, *row*, *col*)
  - Functions (map individuals to individuals)—*Sqrt*, *LeftLegOf*, …
  - Predicates (map individuals to truth values)—*Brother*, >, ==, …
  - Connectives ($\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$)
  - Quantifiers (universal "$\forall$" and existential "$\exists$")
  - Equality (i.e., =)

Quantifiers

- Universals are like conjunction
  - $\forall x P(x)$ means P holds for all values x
  - Usually used with implication
- Existentials are like disjunction
  - $\exists x Q(x)$ means Q holds for some value x
  - ∃Usually used with conjunction
- Switching the order of ***like*** quantifiers does not change meaning.
- Switching the order of ***different*** quantifiers does.

Inference Rules in FOL

- PL inference rules apply in FOL as well.
- Universal Elim: If $\forall x P(x)$ is true, then $P(x)$ is true for any constant *c* in the domain. (Like AND Elimination)
- Existential Intro: If $P(c)$ is true, then $\exists x P(x)$ is true. (Like OR Introduction)
- Existential Elim: If $\exists x P(x)$ is true, then $P(c)$ is true for some constant *c* not appearing in any other sentence (Skolem constant).

Inference Rules in FOL

- Generalized Modus Ponens
  - Ex. From $P(c), Q(c), \text{and } \forall x(P(x) \wedge Q(x)) \Rightarrow R(x)$ derive $R(c)$
  - Let $\text{subst}(\theta, \alpha)$ denote substitutions resulting from applying some substitution list θ to sentence α.
  - Given atomic sentences $P_1, \ldots, P_n$ and implication $(Q_1, \wedge \cdots \wedge Q_n) \rightarrow R$ and $\text{subst}(\theta, P_i) = \text{subst}(\theta, Q_i)$ derive $\text{subst}(\theta, R)$
- GMP requires the ability to find substitutions allowing "facts" to match left-hand-sides of rules.
- This matching process is called "unification."

Resolution Theorem Proving

- Proof by contradiction
- Unification used to match terms in clauses
- Procedure:
  - Negate the theorem to be proven
  - Add to axiom set
  - Convert all axioms to clause form
  - Resolve clauses until either the empty clause is produced or no resolvable clauses remain
  - Empty clause proves theorem (contradiction)

Short Example

- Given
  - Whoever can read is literate.
  - Dolphins are not literate.
  - Some dolphins are intelligent.
- Prove
  - Some who are intelligent cannot read.

Convert to FOL

- Given

$$\forall x[\text{Read}(x) \Rightarrow \text{Literate}(x)]$$
$$\forall x[\text{Dolphin}(x) \Rightarrow \neg \text{Literate}(x)]$$
$$\exists x[\text{Dolphin}(x) \wedge \text{Intelligent}(x)]$$

- Prove

$$\exists x[\text{Intelligent}(x) \wedge \neg \text{Read}(x)]$$

Convert to Clause Form

- Axioms

  $\neg \text{Read}(x) \vee \text{Literate}(x)$

  $\neg \text{Dolphin}\ (y) \vee \neg \text{Literate}\ (y)$

  $\text{Dolphin}\ (A)$ {A is a Skolem constant)

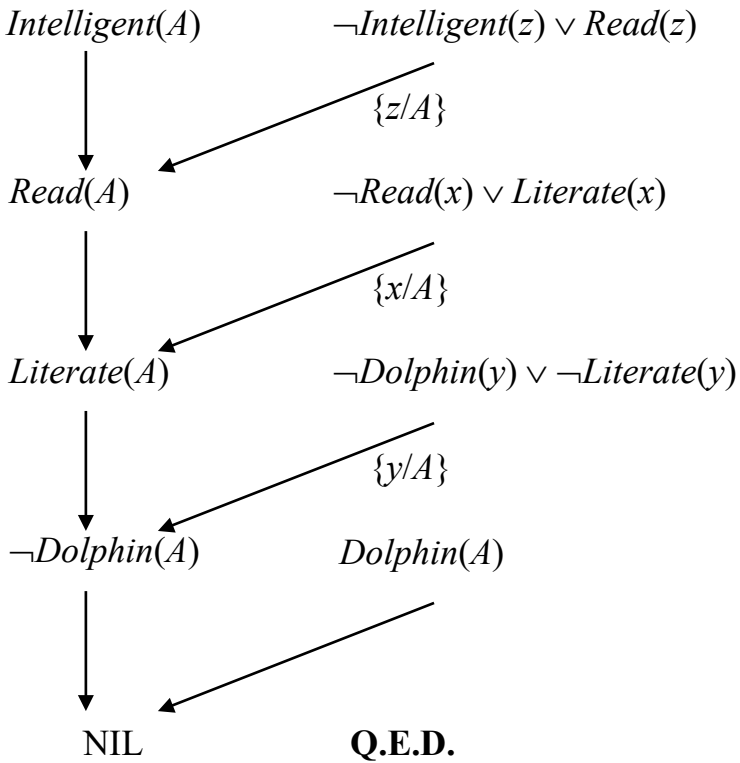
  $\text{Intelligent}\ (A)$

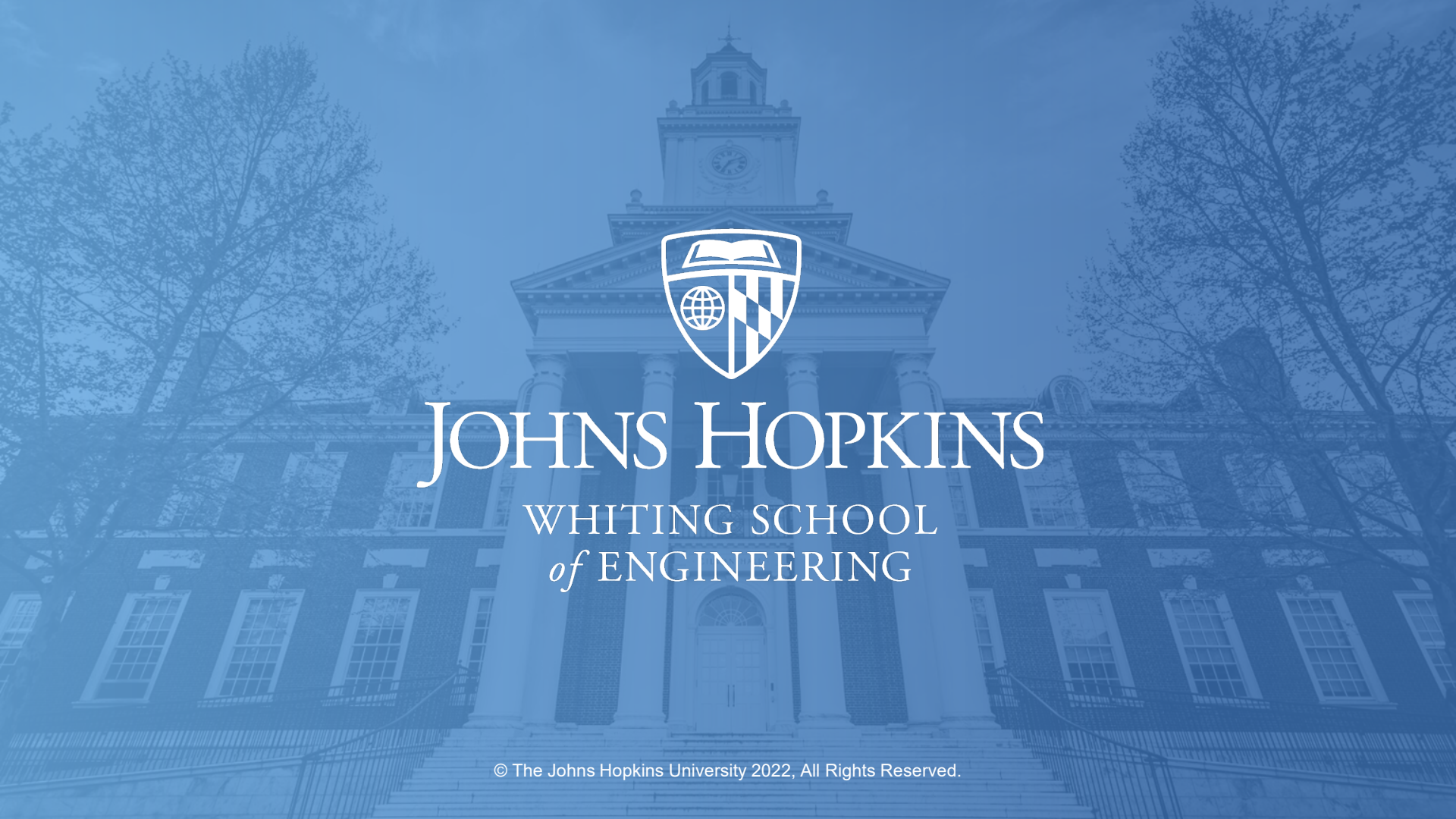
- Negated Theorem

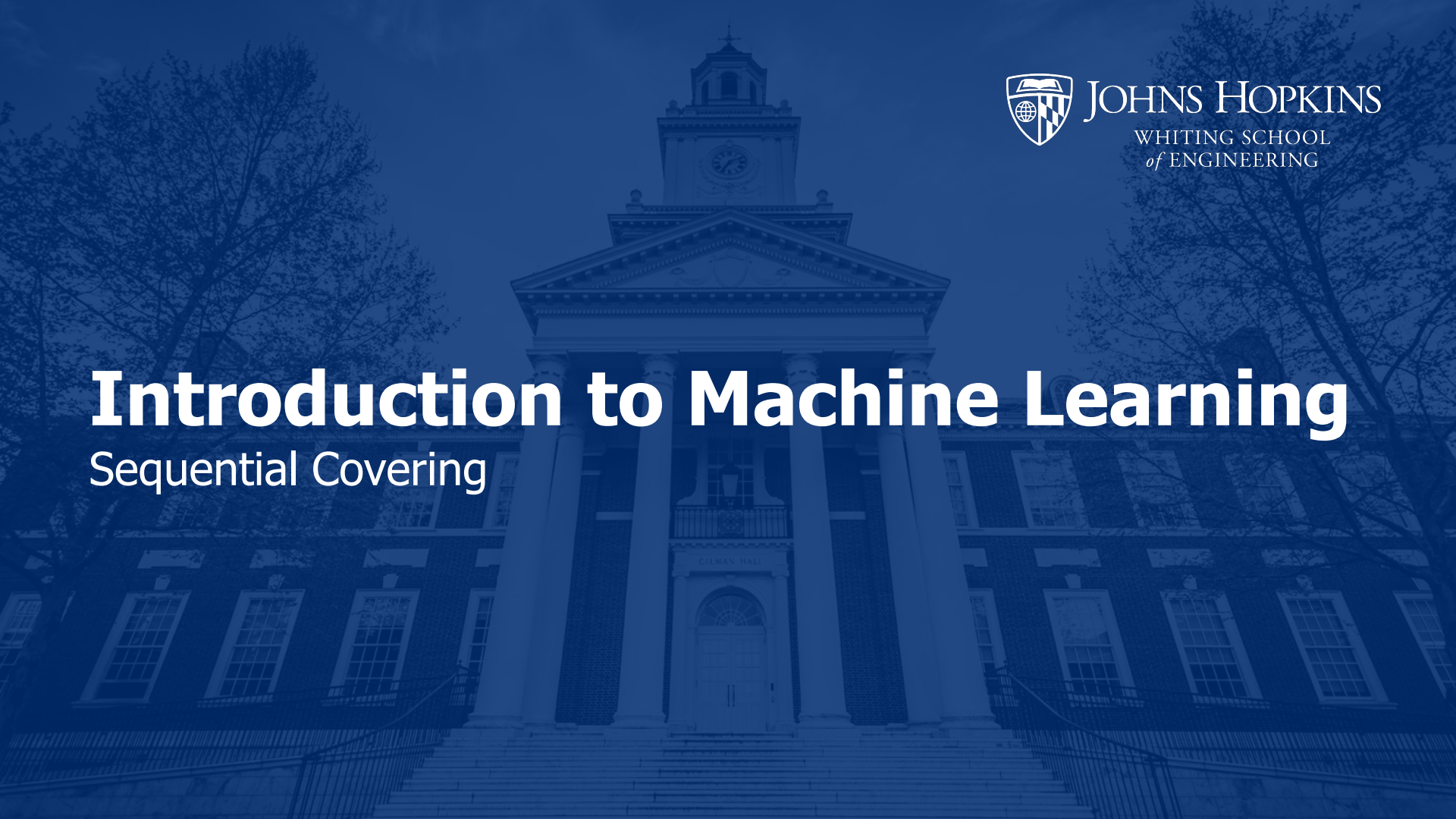
  $\neg \exists x[\ \text{Intelligent}\ (x) \wedge \neg \text{Read}(x)]$ which yields in clause form

  $\neg\ \text{Intelligent}\ (z) \vee \text{Read}(z)$

Resolution Proof Tree







Introduction to Machine Learning

Sequential Covering

Covering

- A rule "covers" a subset of examples in $\mathcal{D}$ if the left-hand side of the rule matches those examples.
- We learn a set of rules one rule at a time by finding rules that "cover" the data.

Algorithm 9.1 Sequential Covering

```
function SequentialCovering(D)
    partition D into Pos and Neg
    repeat
        Rule ← LearnOne(Pos)
        LearnedRules ← LearnedRules ∪ {Rule}
        Pos ← Pos − {p ∈ Pos | covered(p, Rule)}
    until Perf(Rule, D) > θ
    LearnedRules ← Sort(LearnedRules, Perf([], D))
    return LearnedRules
end function
```

One Rule

Algorithm 9.2 Learn One Rule

```
1: function LearnOne(D)
2:     bestHyp ← most general hypothesis
3:     candHyps ← {bestHyp}
4:     while candHyps ≠ ∅ do                ▷ Generate next more specific candidate hypotheses
5:         allConstraints ← set of all (a = v)          ▷ v is a value of a occurring in D
6:         newCandHyps ←
7:         for all h ∈ candHyp and c ∈ allConstraints do
8:             specialize h by adding c
9:         end for
10:        remove duplicate, inconsistent, or non-max-specific h ∈ newCandHyps
11:        // Update bestHyp only if performance improves
12:        for all h ∈ newCandHyps do
13:            if Perf(h, D) > Perf(bestHyp, D) then
14:                bestHyp ← h
15:            end if
16:        end for
17:        // Update candHyps with the best from the candidates
18:        candHyps ← up to the k best members of newCandHyps
19:    end while
20:    return newRule of the form "IF bestHyp THEN prediction"
21:    // prediction is most frequent value of targets among matching examples
22: end function
```

FOIL

Prolog style:

$$P(x_1, x_2, \ldots, x_k) \leftarrow \ell_1, \ldots, \ell_n$$

$$Q(v_1, \ldots, v_r)$$

$$\text{Equal}(x_j, x_k)$$

Bindings

- Positive
- Negative

Algorithm 9.3 First Order Inductive Learning (FOIL)

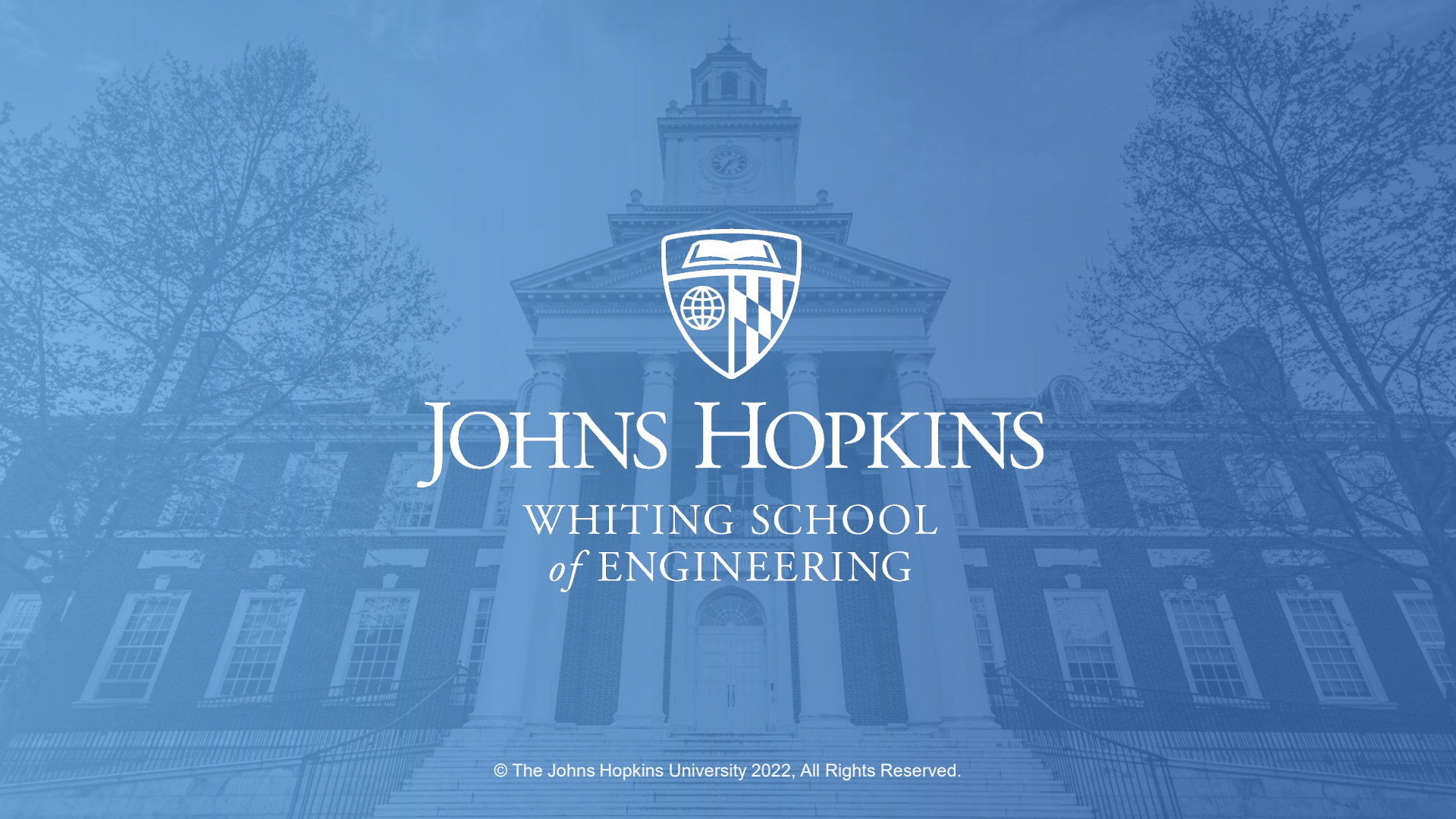
```
function FOIL(T, P, D)
    // T = target
    // P = predicates
    // D = examples
    Pos ← {x ∈ D | x is a positive example}
    Neg ← {x ∈ D | x is a negative example}
    while Pos ≠ ∅ do
        newRule ← most general rule
        newRuleNeg ← Neg
        while newRuleNeg ≠ ∅ do
            candLit ← GenCandidates(P)
            bestLit ← arg max_{ℓ ∈ candLit} FOILGain(ℓ, newRule)
            add bestLit to newRule preconditions (i.e., antecedent)
            newRuleNeg ← subset of newRuleNeg satisfying preconditions
        end while
        learnedRules ← learnedRules ∪ {newRule}
        Pos ← Pos − {p ∈ Pos | covered(p, Rule)}
    end while
    return learnedRule
end function
```

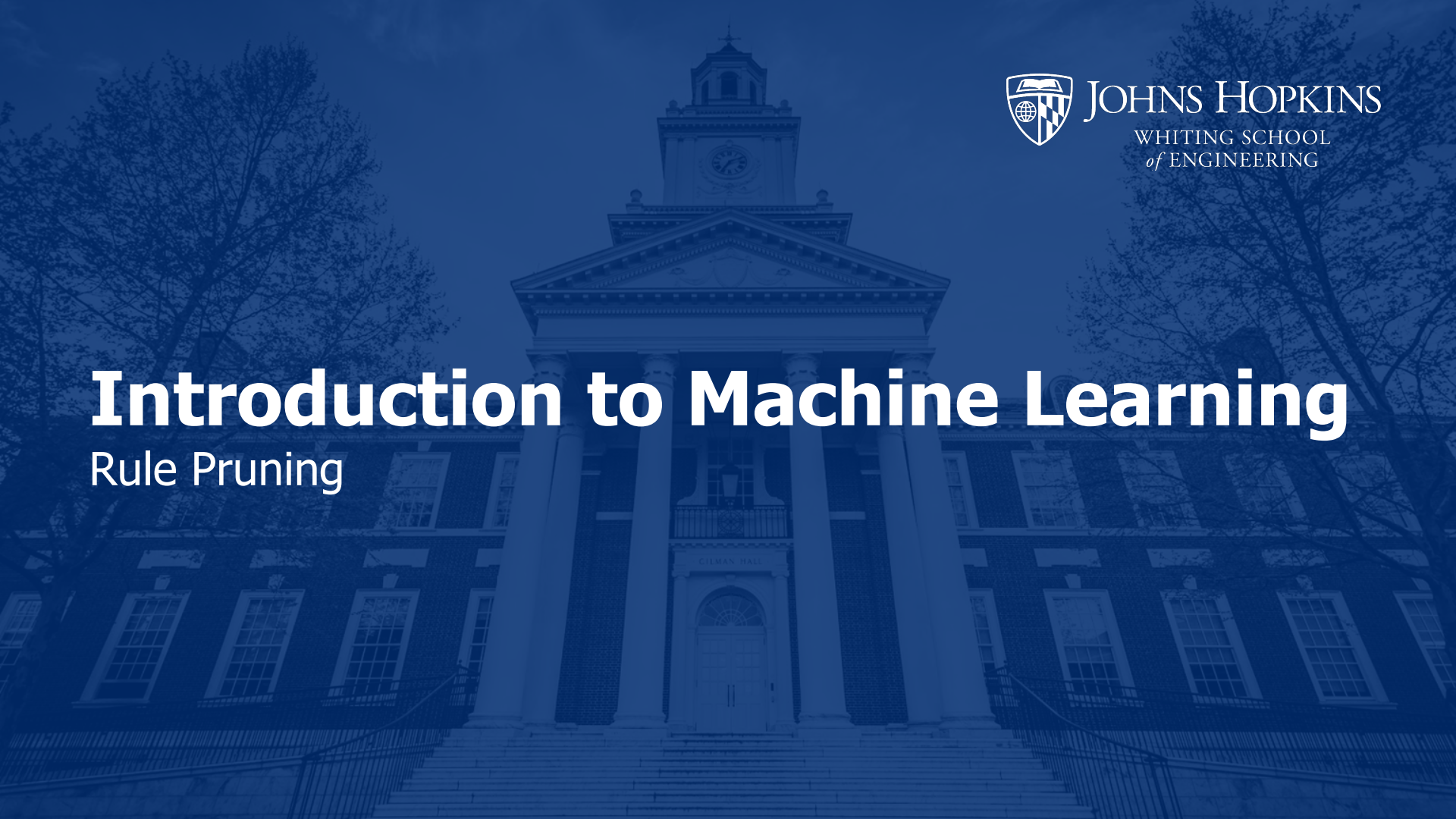
Selecting Predicates

- Similar to decision trees, select based on "information gain."

$$\text{FOILGain}\,(\ell,\ \text{Rule}\,) = t\left(\lg\frac{p_1}{p_1+n_1} - \lg\frac{p_0}{p_0+n_0}\right)$$

p_0 : # positive bindings in *Rule*
n_0 : : # negative bindings in *Rule*
p_1 : # positive bindings in *Rule + l*
n_1 : # positive bindings in *Rule + l*





Introduction to Machine Learning

Rule Pruning

Pruning

- Two approaches to pruning
  - Prune whole rules
  - Prune predicates within rules

$$X \wedge \cancel{Y} \wedge Z \Rightarrow c$$

- Note that pruning predicates *subsumes* pruning rules
- Sort the rules by coverage
- Resulting pruning strategy, similar to decision trees is

  Incremental Reduced Error Pruning (IREP)

Pruning

Algorithm 9.4 Incremental Reduced Error Pruning (IREP)

```
1: function IREP(Pos, Neg)
2:     ruleSet ← {}
3:     while Pos ≠ ∅ do
4:         SPLIT(Pos, Neg) into (GrowPos, GrowNeg) and (PrunePos, PruneNeg)
5:         Rule ← GrowRule(GrowPos, GrowNeg)
6:         Rule ← PruneRule(PrunePos, PruneNeg)
7:         if Perf(Rule, (PrunePos, PrunNeg)) < θ then    ▷ Best rule worse than chance
8:             return ruleSet
9:         else
10:            RuleSet ← RuleSet + Rule
11:            (Pos, Neg) ← (Pos, Neg)− Covered(Rule)
12:        end if
13:    end while
14:    return ruleSet
15: end function
```

- Grow using 2/3 of the data; Prune using 1/3 of the data

The PruneRule Function

- Let $P = |\ \text{PrunePos}\ |$
- Let $N = |\ \text{PruneNeg}\ |$
- For the current rule, let *p* be the number of examples in *PrunePos* covered by the rule, and *n* be the number of examples in *PruneNeg* covered by the rule.
- Define

$$v(\ \text{Rule}\ ,\ \text{PrunePos}, \text{PruneNeg}\ ) = \frac{p + N - n}{P + N}$$

- Choose the predicate the prune that maximizes v.

RIPPER

- "Repeated Incremental Pruning to Produce Error Reduction"

- Let $\text{RuleSet} = \{\text{Rule}_1, \ldots, \text{Rule}_k\}$
- Two options:
  1. Replace Rule: Total error
  2. Revise Rule: Add new predicates
- Make decisions based on Minimum Description Length

$$\mathcal{L}(\mathcal{D}) = \min_{H \in \mathcal{H}} [\mathcal{L}(H) + \mathcal{L}(\mathcal{D} \mid H)]$$

Minimum Description Length

- An information theoretic approach

$$
\begin{aligned}
H_{map} &= \underset{H \in \mathcal{H}}{\operatorname{argmax}} P(\mathcal{D}|H)P(H) \\
&= \underset{H \in \mathcal{H}}{\operatorname{argmax}} \log P(\mathcal{D}|H) + \log P(H) \\
&= \underset{H \in \mathcal{H}}{\operatorname{argmin}} -\log P(\mathcal{D}|H) - \log P(H)
\end{aligned}
$$

- Note: $\mathcal{L}(\mathcal{D} \mid H) = -\log P(\mathcal{D} \mid H)$ and $\mathcal{L}(H) = -\log P(H)$
- So

$$
H_{\text{map}} = \underset{H \in \mathcal{H}}{\operatorname{argmin}} \mathcal{L}(\mathcal{D}) = \underset{H \in \mathcal{H}}{\operatorname{argmin}} \mathcal{L}(\mathcal{D} \mid H) + \mathcal{L}(H)
$$

RIPPER Algorithm

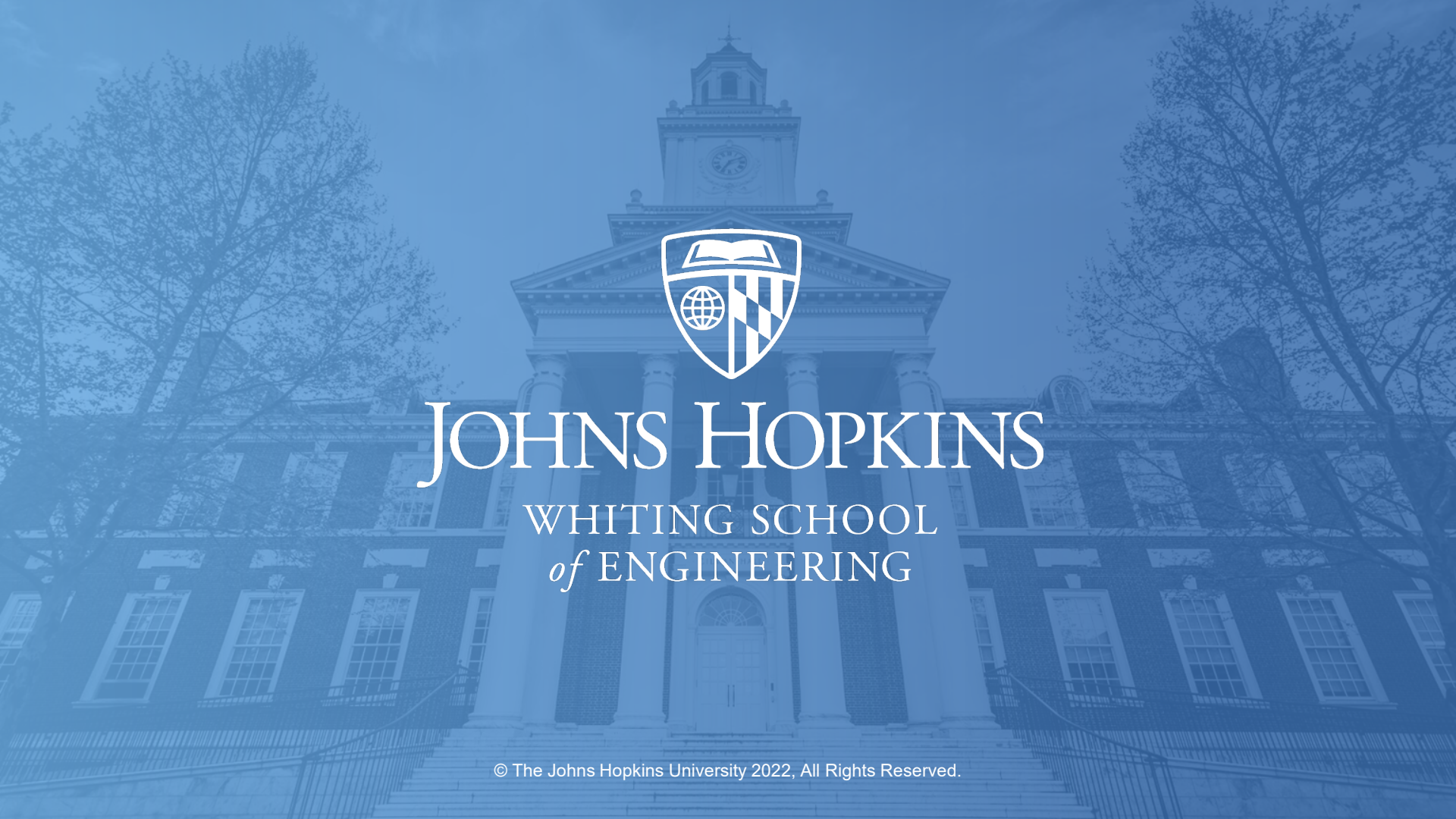
```
1: function RIPPER(Pos, Neg)
2:     ruleSet ← BuildRuleSet(Pos,Neg)
3:     repeat
4:         ruleSet ← OptimizeRuleSet(ruleSet,Pos,Neg)
5:     until k times
6:     return ruleSet
7: end function
```

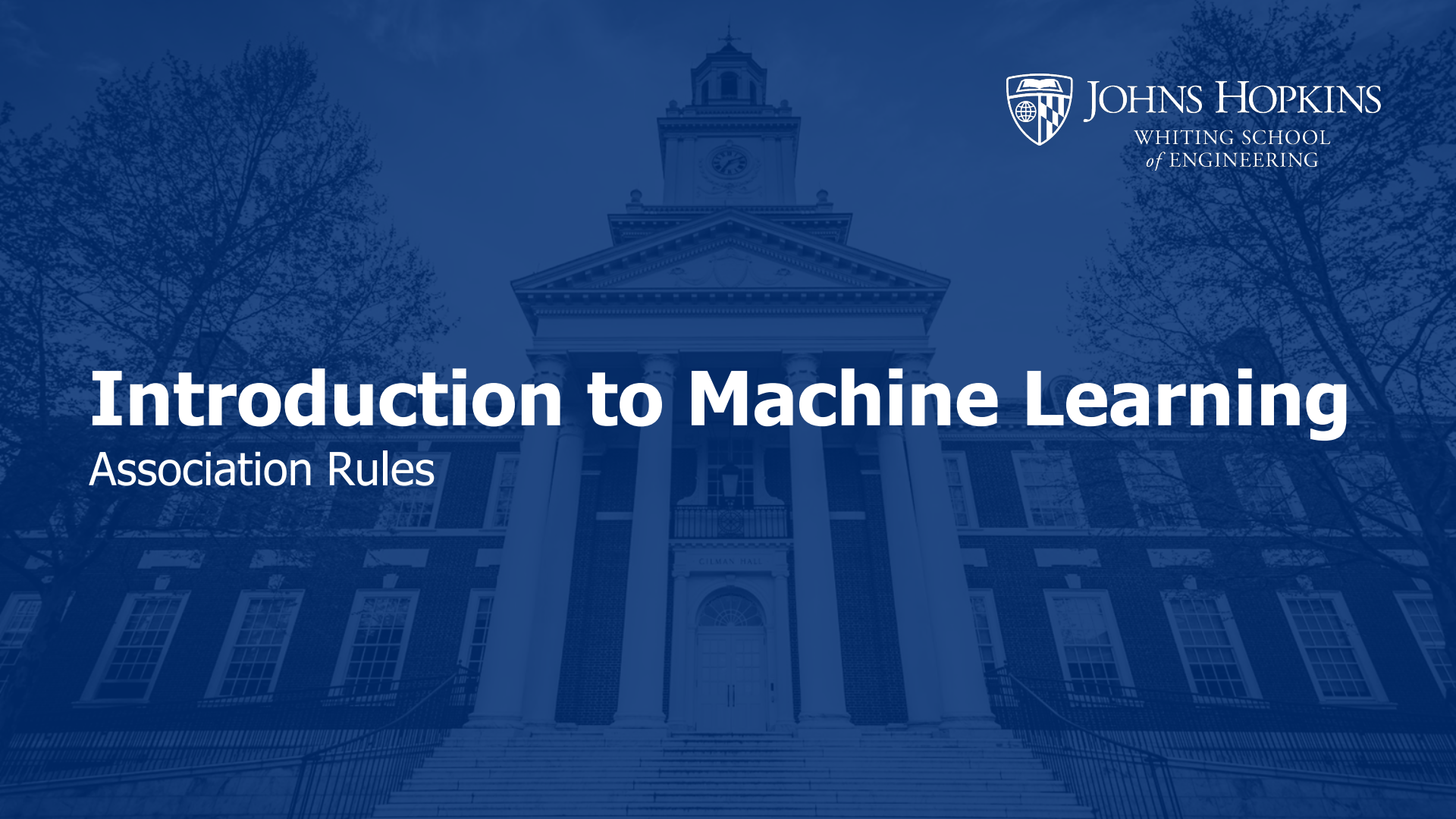
Building the Rule Set

```
function BuildRuleSet(Pos, Neg)
    ruleSet ← {}
    DL ← DescriptionLength(ruleSet, Pos, Neg)
    while Pos ≠ ∅ do
        SPLIT(Pos, Neg) into (GrowPos, GrowNeg) and (PrunePos, PruneNeg)
        Rule ← GrowRule(GrowPos, GrowNeg)
        Rule ← PruneRule(Rule, PrunePos, PruneNeg)
        RuleSet ← RuleSet + Rule
        if DescriptionLength(ruleSet, Pos, Neg) > DL + 64 then
            for each rule R in ruleSet do
                if DescriptionLength(ruleSet, −{R}, Pos, Neg) < DL then
                    DL ←DescriptionLength(ruleSet, Pos, Neg)
                end if
            end for
            return ruleSet
        end if
    end while
end function
```

Optimizing the Rule Set

```
26: function OptimizeRuleSet(ruleSet, Pos, Neg)
27:     for each rule R in ruleSet do
28:         Delete R from ruleSet
29:         UPos ← examples in Pos not covered by ruleSet
30:         UNeg ← examples in Neg not covered by ruleSet
31:         Split (UPos, UNeg) into (GrowPos, GrowNeg) and (PrunePos, PruneNeg)
32:         RepRule ← GrowRule(GrowPos, GrowNeg)
33:         RepRule ← PruneRule(RepRule, PrunePos, PruneNeg)
34:         RevRule ← GrowRule(GrowPos, GrowNeg, R)
35:         RevRule ← PruneRule(RepRule, PrunePos, PruneNeg)
36:         Choose better of RepRule and RevRule and add to ruleSet
37:     end for
38:     return ruleSet
39: end function
```





Introduction to Machine Learning

Association Rules

Transaction Data

- Assume a binary representation
  - Rows correspond to data items (transactions)
  - Columns correspond to features (items sold)
  - Let positive be denoted "1" and negative be denoted "0"
- Let $I = \{i_1, i_2, \ldots, i_d\}$ be a set of items
- Let $T = \{t_1, \ldots, t_n\}$ be a set of transactions
- Itemset: A collection of zero or more items
- k-itemset: A collection of exactly k items

Itemset Metrics

- Itemset Support: The number of transactions containing the itemset

$$\sigma(X) = |\{t_i : X \subseteq t_i \wedge t_i \in T\}|$$

- Rule Support:

$$\sigma(X \Rightarrow Y) = \frac{\sigma(X \cup Y)}{n} = P(X, Y)$$

- Rule Confidence:

$$c(X \Rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)} = P(Y \mid X)$$

An Example

- Suppose you go to the story and buy beer, diapers, and milk

  Itemset:$\{beer, diapers, milk\}$

- Possible association rules:

$$\{Beer\} \Rightarrow \{Diapers, Milk\}$$
$$\{Diapers\} \Rightarrow \{Beer, Milk\}$$
$$\{Milk\} \Rightarrow \{Beer, Diapers\}$$
$$\{Beer, Diapers\} \Rightarrow \{Milk\}$$
$$\{Beer, Milk\} \Rightarrow \{Diapers\}$$
$$\{Diapers, Milk\} \Rightarrow \{Beer\}$$

- Find frequent itemsets: $\sigma(X) > \text{minsup}$
- Use frequent itemsets to find high-confidence rules: $c(X \Rightarrow Y) > \text{minconf}$

Apriori Principle

- **Theorem:** If an itemset is frequent, then all of its subsets must also be frequent. Conversely, if an itemset is infrequent, then all of its supersets must also be infrequent.

Known as the anti-monotone property

- **Definition:** Let I be a set of items, and let $J = 2^I$ be the power set of I.

  A measure *f* is ***monotone*** if $\forall X, Y \in J(X \subseteq Y) \Rightarrow f(X) \leq f(Y)$

  Similarly, a measure *g* is said to be ***anti-monotone*** if

  $\forall X, Y \in J(X \subseteq Y) \Rightarrow g(X) \geq g(Y)$

Frequent Itemset Generation

Algorithm 1 Frequent Itemset Generation

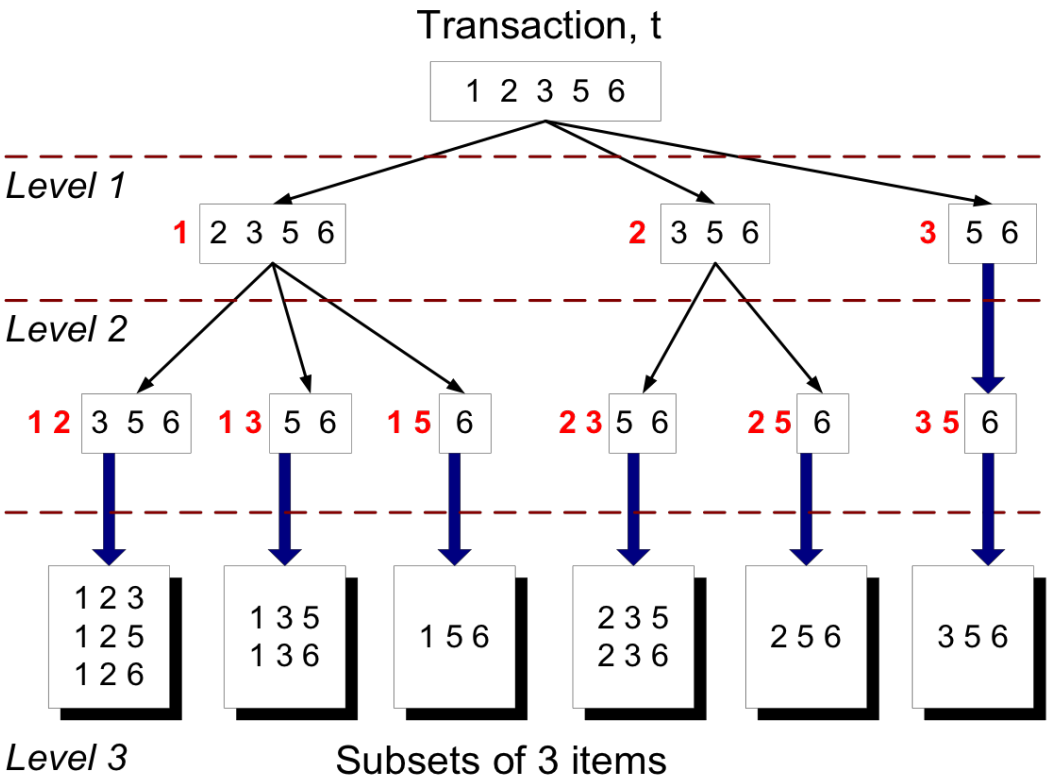
$k \leftarrow 1$
$F_k \leftarrow \{i \in I \wedge \sigma(\{i\}) \geq n \times minsup\}$ // Finds all 1-itemsets
repeat
  $k \leftarrow k + 1$
  $C_k \leftarrow$ apriori-gen(F_{k-1}) // Generates candidate itemsets
  for each transaction $t \in T$ do
    $C_t \leftarrow$ subset(C_k, t) // Identifies all candidates belonging to t
    for each candidate itemset $c \in C_t$ do
      $\sigma(c) \leftarrow \sigma(c) + 1$ // Increment the support count
    end for
  end for
  $F_k \leftarrow \{c | c \in C_k \wedge \sigma(c) \geq n \times minsup\}$ // Extracts frequent k-itemsets.
until $F_k = \emptyset$
Result $\leftarrow \bigcup F_k$

Apriori Generation

Algorithm 2 Apriori-Gen(F_{k-1})

$C_k \leftarrow \emptyset$
for each itemset $f_1 \in F_{k-1}$ do
  for each itemset $f_2 \in F_{k-1}$ do
    if $(f_1[1] = f_2[1]) \wedge \cdots \wedge (f_1[k-1] = f_2[k-1])$ then
      $c \leftarrow$ merge(f_1, f_2) // merge the two itemsets
      if containsInfrequent(c, F_{k-1}) then
        remove c // Prunes infrequent candidate
      else
        $C_k \leftarrow C_k \cup \{c\}$
      end if
    end if
  end for
end for
return C_k

Example


Transaction, t
1 2 3 5 6
Level 1
1 2 3 5 6
2 3 5 6
3 5 6
Level 2
1 2 3 5 6
1 3 5 6
1 5 6
2 3 5 6
2 5 6
3 5 6
1 2 3
1 2 5
1 2 6
1 3 5
1 3 6
1 5 6
2 3 5
2 3 6
2 5 6
3 5 6
Level 3
Subsets of 3 items

Finding Association Rules

- For itemset *Y*, a rule is a partition of *Y* into disjoint sets *X* and $Y - X$
- Considering rule of the form $X \Rightarrow Y - X$
- Want confidence $c(X \Rightarrow Y - X) > \text{minconf}$
- Note that

$$c(X \Rightarrow Y - X) = \frac{\sigma(X \cup Y - X)}{\sigma(X)} = \frac{\sigma(Y)}{\sigma(X)}$$

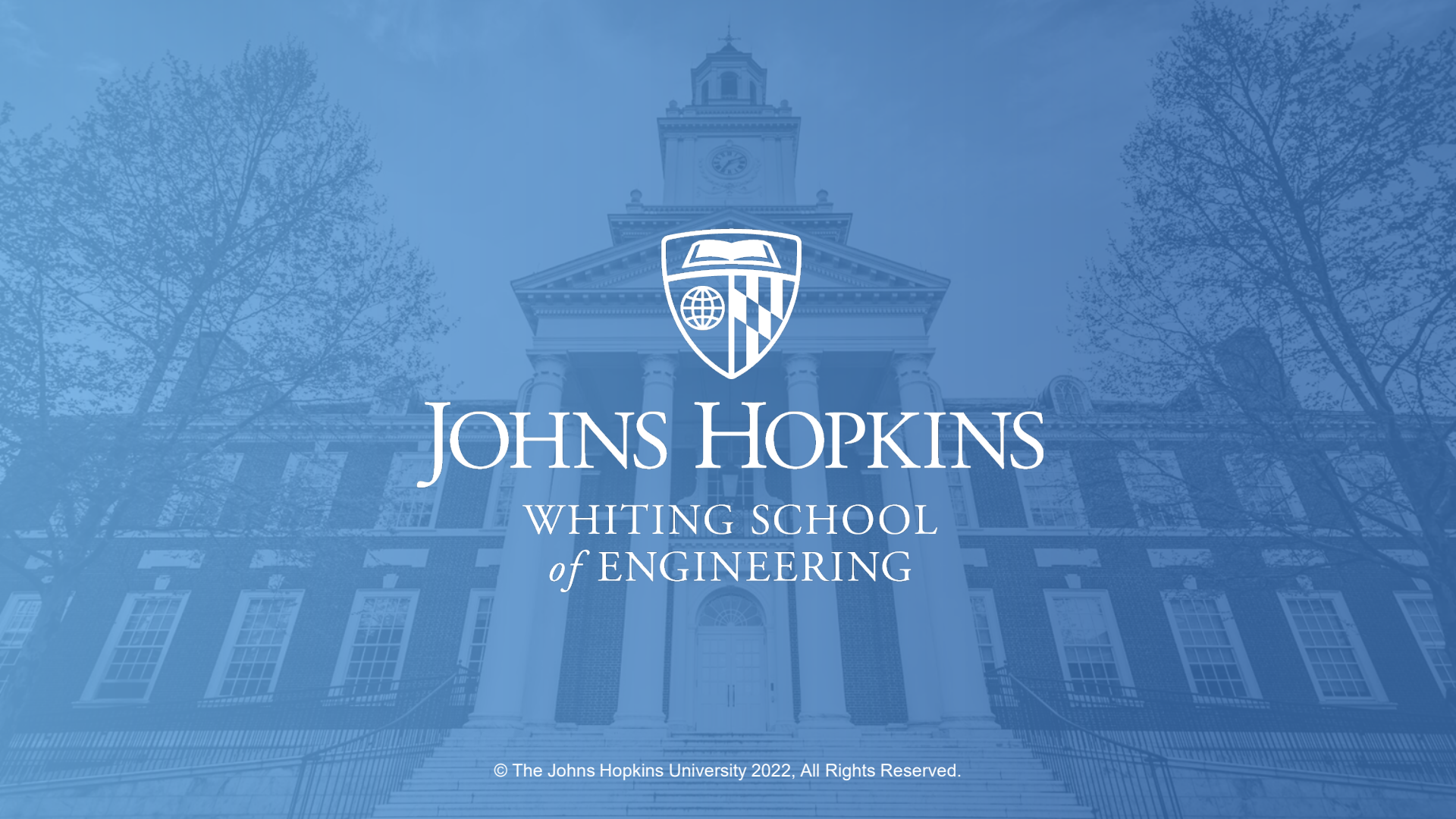
- **Theorem:** If a rule $X \Rightarrow Y - X$ does not satisfy the confidence threshold, then any rule $X' \Rightarrow Y - X'$ where $X' \subset X$ must not satisfy the confidence threshold either.

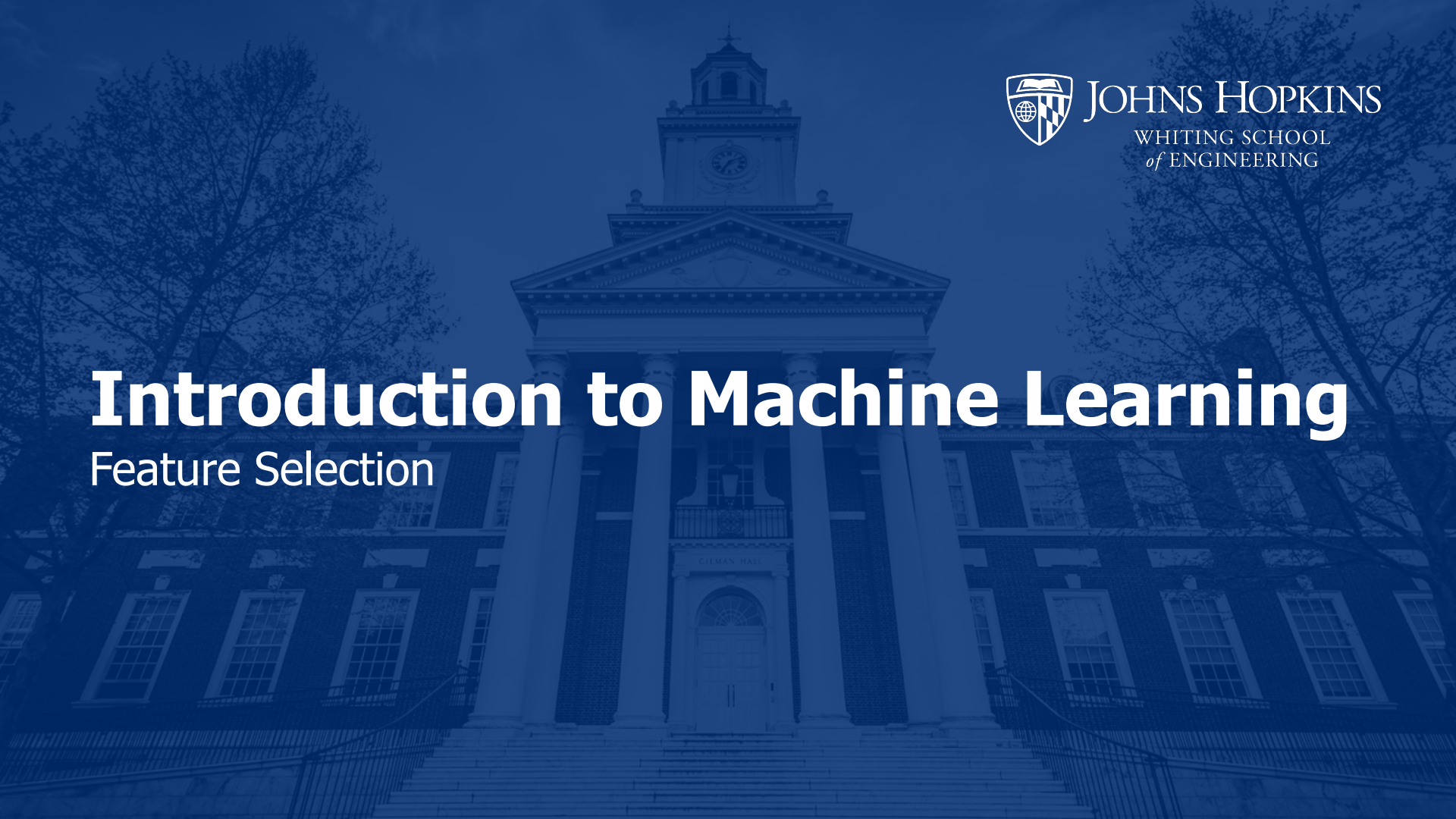
Rule Generation

Algorithm 2 Rule Generation

```
for each frequent k-itemset f_k, k ≥ 2 do
   H_1 ← {i | i ∈ f_k} // The 1-item consequents of the rule.
   call ap-genrules(f_k, H_1)
end for
```

```
ap-genrules(f_k, H_m)
   k ← |f_k| // Size of frequent itemset
   m ← |H_m| // Size of rule consequent
   if k > m + 1 then
      H_{m+1} ← apriori-gen(H_m) // Generate candidate itemsets
      for each h_{m+1} ∈ H_{m+1} do
         conf ← σ(f_k)/σ(f_k − h_{m+1})
         if conf ≥ minconf then
            output the rule (f_k − h_{m+1}) ⇒ h_{m+1}
         else
            delete h_{m+1} from H_{m+1}
         end if
      end for
      call ap-genrules(f_k, H_{m+1})
   end if
```



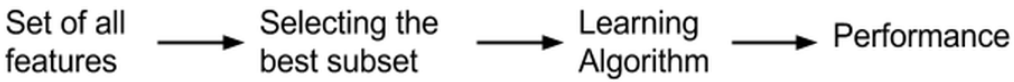


Introduction to Machine Learning

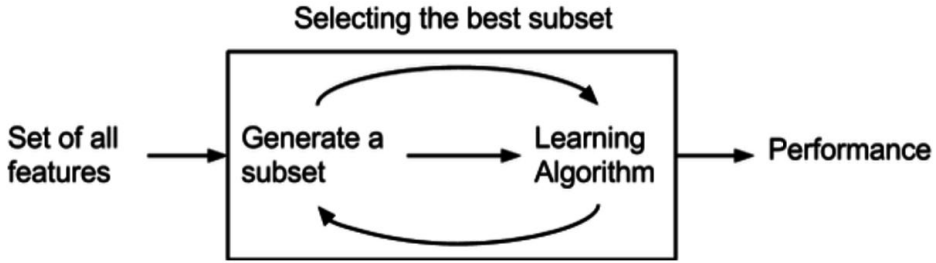
Feature Selection

Models of Feature Selection

- Filter Model



- Wrapper Model



Wrapper Methods

- Stepwise Forward Selection
- Given a feature set

$$\mathcal{F} = \langle F_1, \ldots, F_k \rangle$$

  and a data set $\mathcal{D}$, begin with an empty feature set $\mathcal{F}_0 = \langle \cdot \rangle$
- The process involves selecting one feature at a time, evaluating a model trained with the new feature, retaining the best single feature, and repeating the process.

Stepwise Forward Selection

Algorithm 10.1 Stepwise Forward Selection

```
1:  function SFS(F, D_train, D_valid, Learn())
2:      F_0 ← ⟨⟩
3:      basePerf ← −∞
4:      repeat
5:          bestPerf ← −∞
6:          for all F ∈ F do
7:              F_0 ← F_0 + F
8:              h ← Learn(F_0, D_train)
9:              currPerf ← Perf(h, D_valid)
10:             if currPerf > bestPerf then
11:                 bestPerf ← currPerf
12:                 bestF ← F
13:             end if
14:             F_0 ← F_0 − F
15:         end for
16:         if bestPerf > basePerf then
17:             basePerf ← bestPerf
18:             F ← F − bestF
19:             F_0 ← F_0 + bestF
20:         else
21:             exit
22:         end if
23:     until F ← ⟨⟩
24:     return F_0
25: end function
```

Wrapper Methods

- Stepwise Backward Elimination
- Given a feature set

$$\mathcal{F} = \langle F_1, \ldots, F_k \rangle$$

  and a data set $\mathcal{D}$, begin with an empty feature set $\mathcal{F}_0 = \langle F_1, \ldots, F_k \rangle$
- The process involves selecting one feature at a time, evaluating a model trained with the feature removed, removing the worst feature, and repeating the process.

Stepwise Backward Elimination

Algorithm 10.2 Stepwise Backward Elimination

```
1:  function SBE(F, D_train, D_valid, Learn())
2:      F_0 ← F
3:      basePerf ← −∞
4:      repeat
5:          bestPerf ← −∞
6:          for all F ∈ F do
7:              F_0 ← F_0 − F
8:              h ← Learn(F_0, D_train)
9:              currPerf ← Perf(h, D_valid)
10:             if currPerf > bestPerf then
11:                 bestPerf ← currPerf
12:                 bestF ← F
13:             end if
14:             F_0 ← F_0 + F
15:         end for
16:         if bestPerf > basePerf then
17:             basePerf ← bestPerf
18:             F ← F − bestF
19:             F_0 ← F_0 − bestF
20:         else
21:             exit
22:         end if
23:     until |F| = 1
24:     return F_0
25: end function
```

Local Search Methods

- Consider searching over the full space of possible features, rather than adding or eliminating features one at a time in a Greedy fashion.
- Referred to as "local search" since it makes local modifications to a complete solution.
- A powerful search method that may have been encountered in
  - Artificial Intelligence
  - Stochastic Search/Optimization
- We address three methods:
  - Hill Climbing
  - Simulated Annealing
  - Genetic Algorithms

Hill Climbing

- Simple Hill Climbing
  - Establish a starting state for search (e.g., a random binary vector of size equal to the number of features, where ”1” indicates the feature is selected and “0” indicates the feature is not selected).
  - Evaluate performance by training a model with the selected features.
  - Enter search loop: Incrementally flip bit in the vector, keeping track of performance, retaining the best bit flip.
- Stochastic Hill Climbing (aka first-choice HC or random mutation HC)
  - Establish a starting state for search.
  - Evaluate performance by training a model with the selected features.
  - Enter search loop: Randomly select and flip bit in the vector. If better, take it. If not, discard and try again.

Simulated Annealing

```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     T, a "temperature" controlling prob. of downward steps

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← VALUE[next] – VALUE[current]
        if ΔE > 0 then current ← next
        else current ← next only with probability e^(Δ E/T)
```

Genetic Algorithm

- Suppose

$$\mathcal{F} = \langle b_0 b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5 \quad b_6 \quad b_7 \quad b_8 \quad b_9 \rangle$$

$$\mathcal{F}_0 = \langle 100101011010 \rangle$$

$$\mathcal{P}(t) = \{\mathcal{F}_0, \ldots, \mathcal{F}_{n-1}\}$$

```
1:  function GeneticAlgorithm
2:      t ← 0
3:      initialize(P(t))
4:      f(t) ← evaluateFitness(P(t))
5:      while not(terminatep()) do
6:          t ← t + 1
7:          C(t) ← select(P(t − 1))
8:          C′(t) ← recombine(C(t))
9:          C″(t) ← mutate(C′(t))
10:         f(t) ← evaluateFitness(C″(t))
11:         P(t) ← replace(P(t − 1), C″(t), f(t))
12:     end while
13:     return P(t)
14: end function
```

Selection and Replacement

- Assume “steady state” replacement.
- This means only a small subset of the population is replaced in each generation, rather than the entire population.
- Several methods for selection of parents.
- We apply “fitness proportionate selection”

$$P(X_i) = \frac{f(X_i)}{\sum_{j=0}^{n-1} f(X_j)}$$

- Fitness is based on performance of the model trained with the subset of features.

Crossover

- A method for mating two (or more) parents selected from the population.
- We assume two parents and generate two offspring.
- Parents

[1 1 1 1 1 1 1 1 1 1]

[0 0 0 0 0 0 0 0 0 0]

- Offspring

[1 1 1 1 0 0 0 0 0 0]

[0 0 0 0 1 1 1 1 1 1]

Mutation

- Introduce random variation.
- “March” down the individuals and randomly swap a subset of bits.
- Offspring

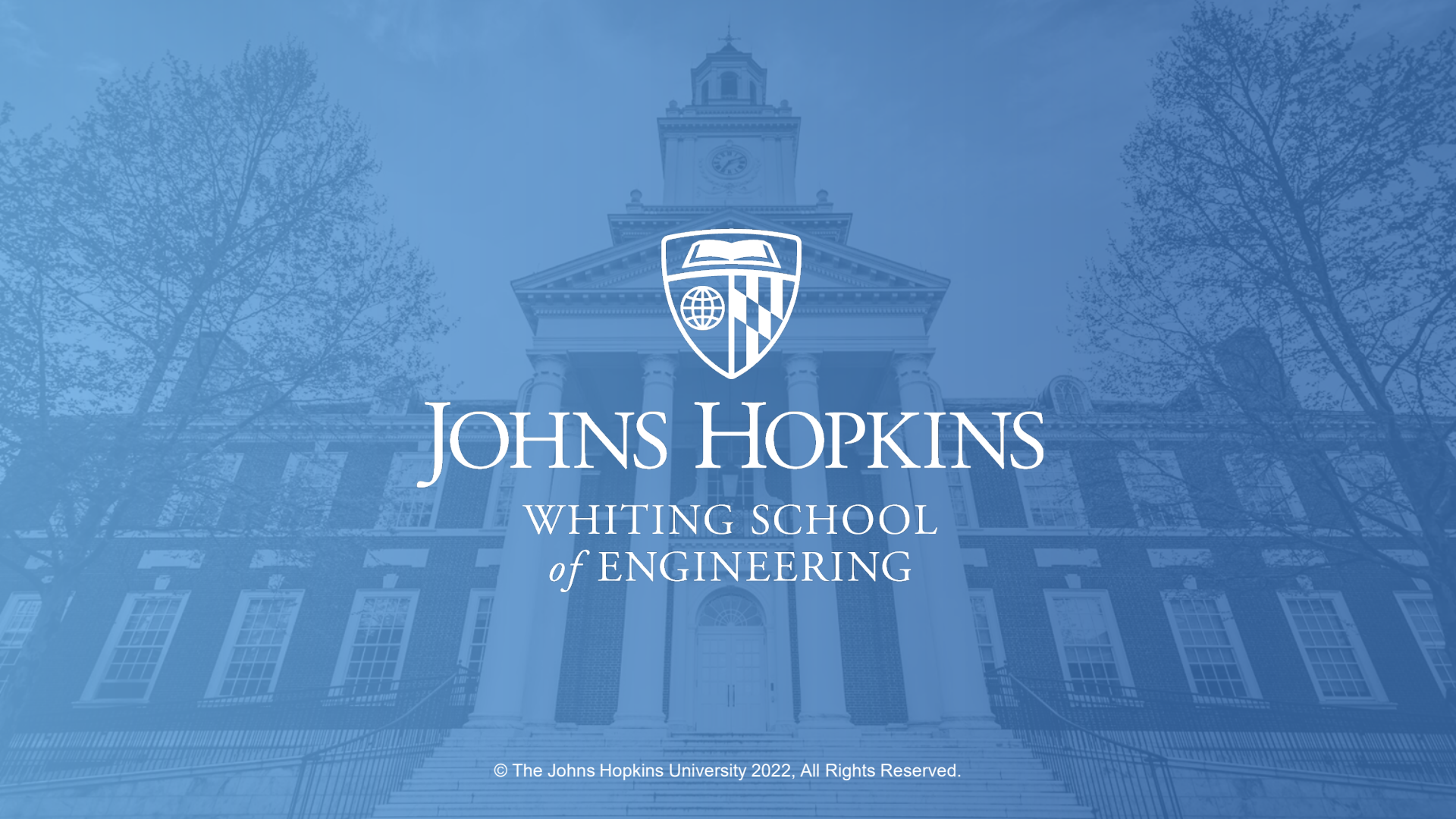
[1 1 1 1 0 0 0 0 0 0]

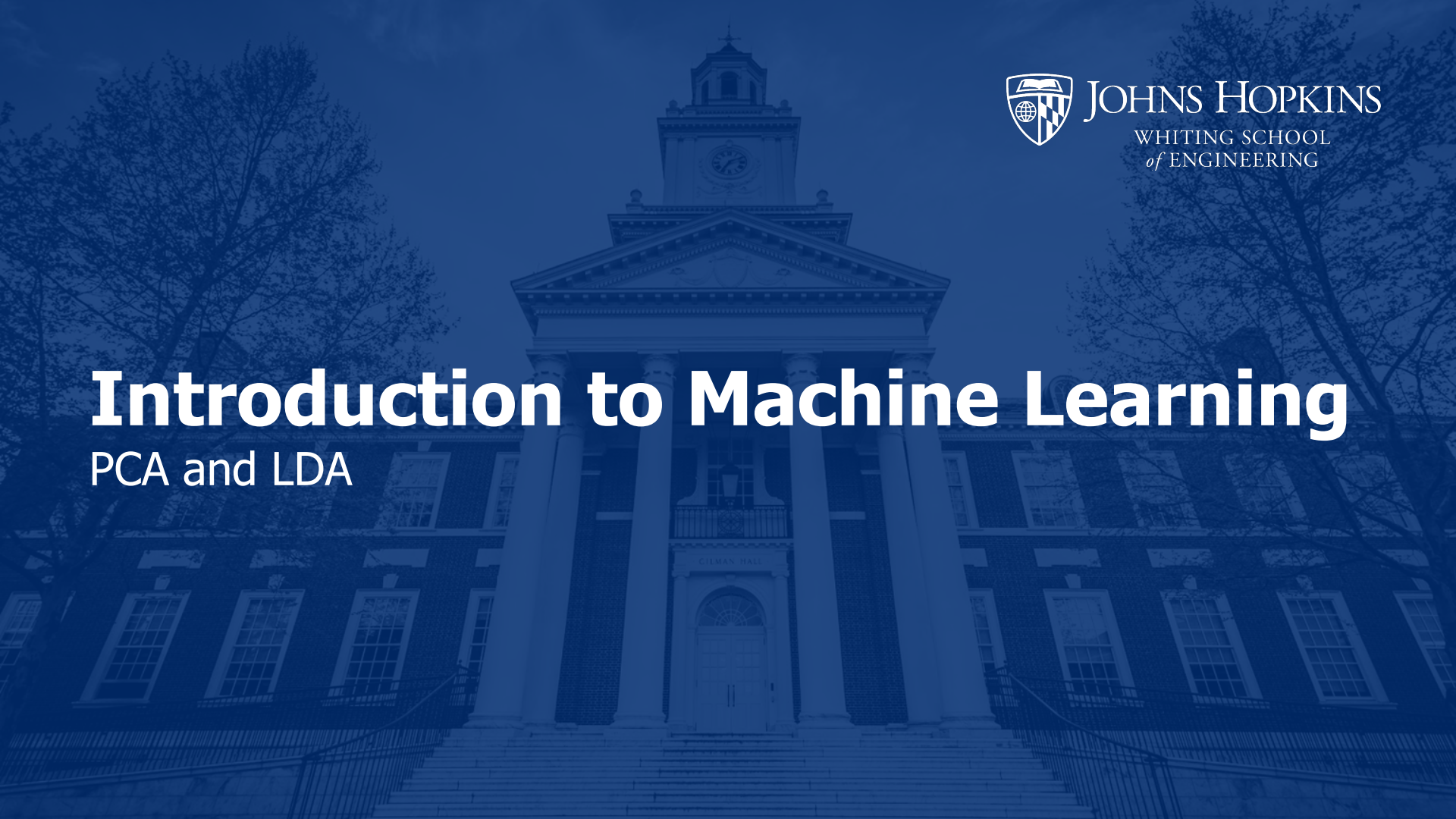
[0 0 0 0 1 1 1 1 1 1]

- Mutated

[1 1 0 1 0 0 0 0 1 0]

[0 0 0 1 1 1 0 1 1 0]





Introduction to Machine Learning

PCA and LDA

Eigendecomposition

- Need math – in particular linear algebra.
- Given a matrix $\mathbf{A}_{m\times n} = \mathbf{U}_{m\times n}\mathbf{\Sigma}_{m\times n}\mathbf{V}^{\top}_{n\times n}$
- Orthogonality means $\mathbf{U}^{\top}\mathbf{U} = \mathbf{I}$ and $\mathbf{V}^{\top}\mathbf{V} = \mathbf{I}$

$$\mathbf{\Sigma} = \begin{bmatrix} \sqrt{\lambda_{1,1}} & 0 & \cdots & 0 \\ 0 & \sqrt{\lambda_{2,2}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sqrt{\lambda_{\min\{m,n\},\min\{m,n\}}} \end{bmatrix}$$

- Here, λ_i is the i^{th} eigenvalue of $\mathbf{A}^{\top}\mathbf{A}$
- Columns of $\mathbf{U}$ are ("left") eigenvectors of $\mathbf{A}\mathbf{A}^{\top}$
- columns of $\mathbf{V}$ are ("right") eigenvectors of $\mathbf{A}^{\top}\mathbf{A}$

Singular Value Decomposition

- First calculate $\mathbf{A}' = \mathbf{A}\mathbf{A}^\top$ and $\mathbf{A}'' = \mathbf{A}^\top\mathbf{A}$
- Find the eigenvalues and eigenvectors of $\mathbf{A}'$ Do the same for $\mathbf{A}''$
- Arrange the eigenvectors as columns for $\mathbf{U}$ and $\mathbf{V}$ respectively.
- Construct $\mathbf{\Sigma}$ as a diagonal matrix from $\sqrt{\lambda_{11}}$ to $\sqrt{\lambda_{kk}}$
  Only use the non-zero eigenvalues.
- This is the singular value decomposition of $\mathbf{A}$.

SVD Example

- Given the following matrix:

$$\mathbf{A} = \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix}$$

- Calculate $\mathbf{A}'$ and $\mathbf{A}''$

$$\mathbf{A}' = \mathbf{A}\mathbf{A}^\top = \begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix}$$

$$\mathbf{A}'' = \mathbf{A}^\top\mathbf{A} = \begin{bmatrix} 10 & 0 & 2 \\ 0 & 10 & 4 \\ 2 & 4 & 2 \end{bmatrix}$$

- Find eigenvalues and eigenvectors by solving

$$\begin{bmatrix} 11 & 1 \\ 1 & 11 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 10 & 0 & 2 \\ 0 & 10 & 4 \\ 2 & 4 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Finding the λ's

- Considering $\mathbf{A}'$ rewrite the equations:

$$11x_1 + x_2 = \lambda x_1$$
$$x_1 + 11x_2 = \lambda x_2$$

- Rearrange:

$$(11 - \lambda)x_1 + x_2 = 0$$
$$x_1 + (11 - \lambda)x_2 = 0$$

- Set determinant of coefficient matrix to zero and solve:

$$\begin{vmatrix} (11-\lambda) & 1 \\ 1 & (11-\lambda) \end{vmatrix} = 0$$

- We get $\lambda = 10$ and $\lambda = 12$ With $x_1 = 1$ we get $\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

Gram-Schmidt Orthonormalization

- Use projection

$$\text{proj}_{\mathbf{u}}(\mathbf{v}) = \frac{\langle \mathbf{v}, \mathbf{u} \rangle}{\langle \mathbf{u}, \mathbf{u} \rangle}$$

  where $\langle \mathbf{v}, \mathbf{u} \rangle$ is the "inner product" of vectors $\mathbf{v}$ and $\mathbf{u}$.

- For some matrix $\mathbf{V}$, convert column vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ to orthogonal $\mathbf{u}_1, \ldots, \mathbf{u}_k$ sequentially as $\mathbf{u}_j = \mathbf{v}_j - \sum_{i=1}^{j-1} \text{proj}_{\mathbf{u}_i}(\mathbf{v}_i)$
- Normalize to turn into orthonormal vectors

$$\mathbf{u}'_j = \frac{\mathbf{u}_j}{\|\mathbf{u}_j\|}$$

Back to the Example

Apply Gram-Schmidt to our example:

$$\mathbf{U} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\mathbf{V}^\top = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{30}} & \frac{2}{\sqrt{30}} & -\frac{5}{\sqrt{30}} \end{bmatrix}$$

Final Decomposition

- Eigenvalues are then

$$\mathbf{\Sigma} = \begin{bmatrix} \sqrt{12} & 0 & 0 \\ 0 & \sqrt{10} & 0 \end{bmatrix}$$

- Finally, we get

$$\mathbf{A}_{m\times n} = \mathbf{U}_{m\times m}\mathbf{\Sigma}_{m\times n}\mathbf{V}^{\top}_{n\times n}$$

$$= \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \sqrt{12} & 0 & 0 \\ 0 & \sqrt{10} & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{30}} & \frac{2}{\sqrt{30}} & -\frac{5}{\sqrt{30}} \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 1 & 1 \\ -1 & 3 & 1 \end{bmatrix}$$

Dimensionality Reduction

- Recall we have
  - Feature selection: Choosing a subset of features
  - Feature extraction: Transform input features to a new feature space
- The Fourier Transform
  - Data is generally provided in the “time” domain.
  - Data is converted to the “frequency” domain.
  - Example algorithms: Discrete Fourier Transform and Fast Fourier Transform.

Principal Component Analysis

- Find an orthogonal basis set that explains the variance in the data.
- Let $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ be a set of examples in d-dimensional space (i.e., $\mathbf{x}_i = \langle F_1, \ldots, F_d \rangle$)
- Calculate the sample mean: $\mathbf{m} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$
- Convert to zero-mean: $\mathbf{x}'_i = \mathbf{x}_i - \mathbf{m}$
- Calculate the variance of the features: $\mathbf{v} = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{m})^2 = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}'_i)^2$

Principal Component Analysis

- Consider the *covariance* between pairs of features F_a and F_b

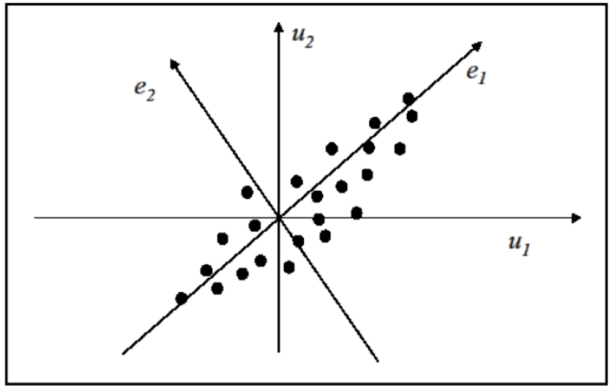
$$\mathbf{v}_{ab} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_{ai} - \mathbf{m})^{\top}(\mathbf{x}_{bi} - \mathbf{m}) = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}'^{\top}_{ai}\mathbf{x}'_{bi}$$

- Apply SVD to the covariance matrix and solve for the eigenvalue matrix.

$$\mathbf{\Lambda} = \mathbf{U}^{\top}\mathrm{cov}(\mathbf{X})\mathbf{U}$$

- The off-diagonal entries of $\mathbf{\Lambda}$ are all zero.
- The diagonal entries of $\mathbf{\Lambda}$ are the square roots of the eigenvalues of $\mathrm{cov}(\mathbf{X})$ which are the variances of the transformed data along the principal axes (components).
- The row vectors of $\mathbf{U}$ are the eigevectors of $\mathrm{cov}(\mathbf{X})$ and are called the "principal components" of $\mathrm{cov}(\mathbf{X})$
- Choose *k* < *d*, select the *k* eigenvectors with largest eigenvalues, and project data onto the the eigenvectors,

$$\mathbf{p} = \mathbf{u}_i^{\top}\mathbf{x}'$$



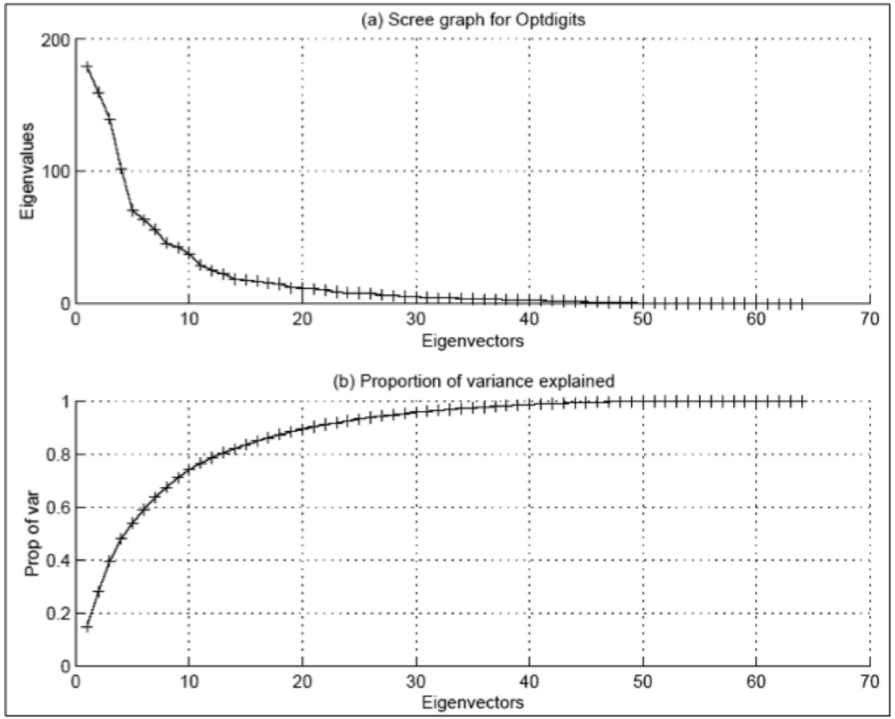
Determining k

- Sort the eigenvalues λ_i^2
  (recalling the singular values are the square root of the eigenvalues).
- Calculate the ”proportion of variance”:

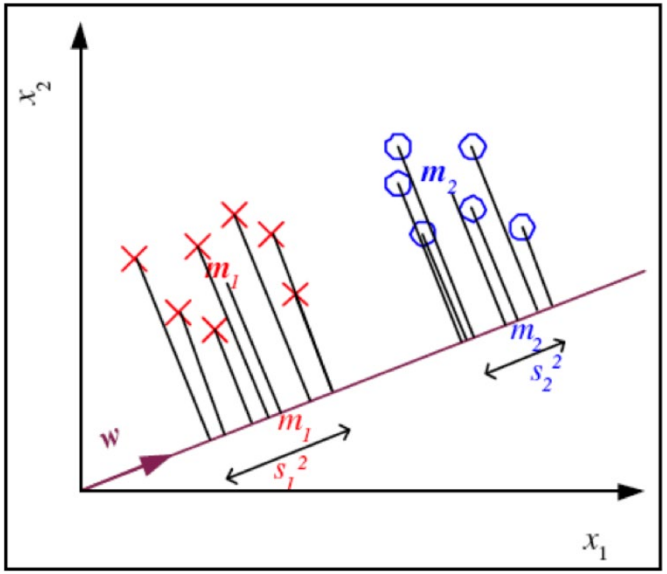
$$\text{PoV} = \frac{\sum_{i=1}^{k} \lambda_i^2}{\sum_{j=1}^{d} \lambda_j^2}$$

- Select *k* such that *PoV* exceeds some threshold (i.e., 90%).

Scree Graph


(a) Scree graph for Optdigits
Eigenvalues
200
100
0
0
10
20
30
40
50
60
70
Eigenvectors
(b) Proportion of variance explained
Prop of var
1
0.8
0.6
0.4
0.2
0
0
10
20
30
40
50
60
70
Eigenvectors

Fisher's Linear Discriminant Analysis



Finding the Linear Discriminant

- Let $\mathcal{F} = \langle F_1, \ldots, F_d \rangle$ be a feature vector.
- Let $\mathbf{m}_1$ be the mean vector of examples from class C_1 and $\mathbf{m}_2$ be the mean vector of examples from class C_2 (assuming two classes).
- Let $\mathbf{m}_1^p$ be the mean vector after projection similarly for $\mathbf{m}_2^p$
- Given a sample of data $\mathcal{D} = \left\{ \mathbf{x}^t, c^t \right\}$

$$\mathbf{m}_1^p = \frac{\sum_t \mathbf{w}^\top \mathbf{x}^t \mathbf{c}^t}{\sum_t \mathbf{c}^t} = \mathbf{w}^\top \mathbf{m}_1 \Rightarrow s_1^2 = \sum_t \left(\mathbf{w}^\top \mathbf{x} - \mathbf{m}_1^p \right)^2 \mathbf{c}^t$$

$$\mathbf{m}_2^p = \frac{\sum_t \mathbf{w}^\top \mathbf{x}^t \left(1 - \mathbf{c}^t\right)}{\sum_t \left(1 - \mathbf{c}^t\right)} = \mathbf{w}^\top \mathbf{m}_2 \Rightarrow s_2^2 = \sum_t \left(\mathbf{w}^\top \mathbf{x} - \mathbf{m}_1^p \right)^2 \left(1 - \mathbf{c}^t\right)$$

Following Projection

- We want the two classes to be well separated.

$$|\mathbf{m}_1^p - \mathbf{m}_2^p| \text{ large}$$

- We also want the two classes to have minimal scatter.

$$s_1^2 + s_2^2 \text{ small}$$

- Find the weight vector $\mathbf{w}$ such that

$$J(\mathbf{w}) = \frac{(\mathbf{m}_1^p - \mathbf{m}_2^p)^2}{s_1^2 + s_2^2}$$

Between-Class Scatter

Rewrite the numerator in terms of a projection onto $\mathbf{w}$.

$$\begin{aligned}\left(\mathbf{m}_1^p - \mathbf{m}_2^p\right)^2 &= (\mathbf{w}^\top \mathbf{m}_1 - \mathbf{w}^\top \mathbf{m}_2)^2 \\ &= \mathbf{w}^\top (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^\top \mathbf{w} \\ &= \mathbf{w}^\top \mathbf{S}_B \mathbf{w}\end{aligned}$$

where

$$\mathbf{S}_B = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^\top$$

Within Scatter

Rewrite the denominator in terms of a projection onto $\mathbf{w}$.

$$\begin{aligned} s_1^2 &= \sum_t (\mathbf{w}^\top \mathbf{x}^t - \mathbf{m}_1^p)^2 \boldsymbol{c}^t \\ &= \sum_t \mathbf{w}^\top (\mathbf{x}^t - \mathbf{m}_1)(\mathbf{x}^t - \mathbf{m}_1)^\top \mathbf{w} \boldsymbol{c}^t \\ &= \mathbf{w}^\top \mathbf{S}_1 \mathbf{w} \end{aligned}$$

where

$$\mathbf{s}_1 = \sum_t \boldsymbol{c}^t \left(\mathbf{x}^t - \mathbf{m}_1\right)\left(\mathbf{x}^t - \mathbf{m}_1\right)^\top$$

Total class scatter is $\mathbf{S}_W = \mathbf{S}_1 + \mathbf{S}_2$ and denominator becomes $s_1^2 + s_2^2 = \mathbf{w}^\top \mathbf{S}_W \mathbf{w}$

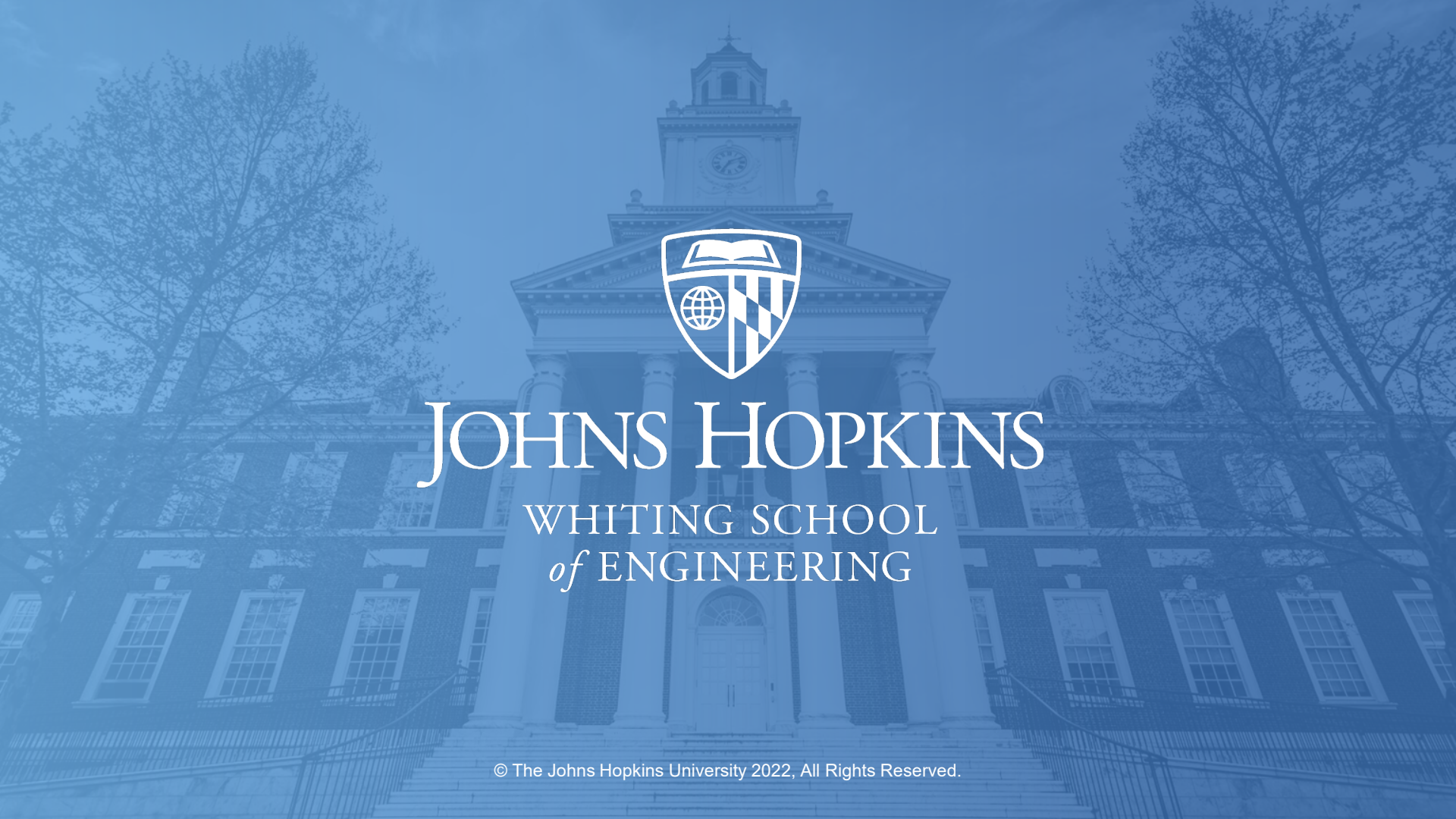
Final Objective Function

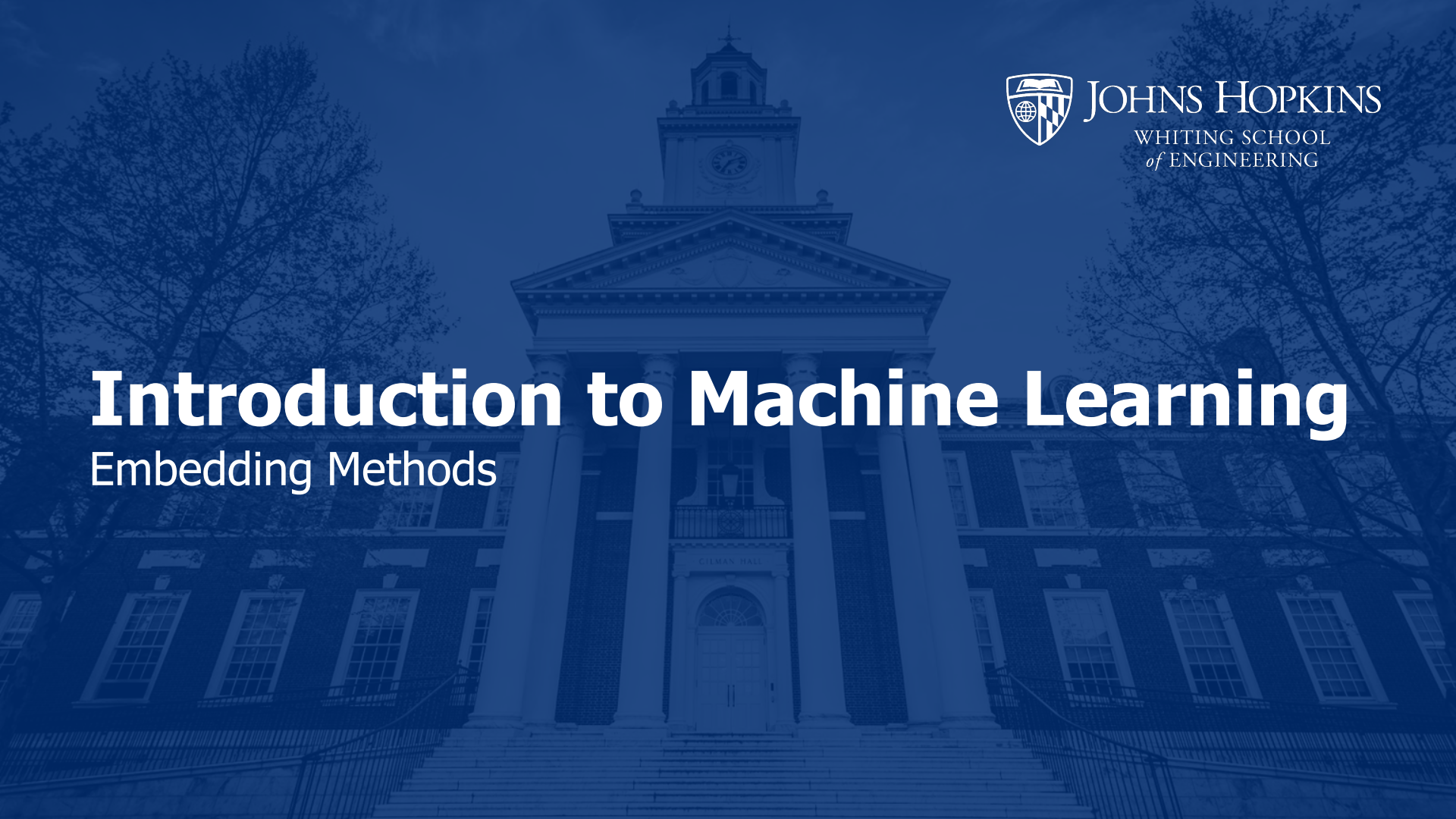
- The final objective function to optimize then becomes

$$J(\mathbf{w}) = \frac{\mathbf{w}^\top \mathbf{S}_B \mathbf{w}}{\mathbf{w}^\top \mathbf{S}_W \mathbf{w}}$$

- Taking the derivative, setting to zero, and solving yields

$$\mathbf{w} = \mathbf{S}_W^{-1} (\mathbf{m}_1 - \mathbf{m}_2)$$





Introduction to Machine Learning

Embedding Methods

Multidimensional Scaling

- Let $\mathcal{D}$ be a data set, where $n = |\mathcal{D}|$ Denote the corresponding data matrix $\mathbf{X}$.
- Assume a similarity matrix has been constructed, consisting of all of the pairwise distances between each pair of points in $\mathcal{D}$.

$$M = \begin{bmatrix} \delta_{11} & \delta_{12} & \cdots & \delta_{1n} \\ \delta_{21} & \delta_{22} & \cdots & \delta_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \delta_{n1} & \delta_{n2} & \cdots & \delta_{nn} \end{bmatrix}$$

- Any distance function can be used.
- Our goal is to embed the data in two dimensions such that relative distances are preserved.
- Also assume all of the data is ”centered” at the origin and shift data as necessary.

Some Preprocessing

- Assume data is real-valued (technically not necessary).
- Assume we use Euclidean distance (technically not necessary).

$$\delta_{ij}^2 = \left\|\mathbf{x}^i - \mathbf{x}^j\right\|_2^2 = \sum_{k=1}^{d} \left(x_k^i\right)^2 - 2\sum_{k=1}^{d} x_k^i x_k^j + \sum_{k=1}^{d} \left(x_k^j\right)^2$$

- Then we can calculate the following:

$$\sum_i \delta_{ij}^2 = \sum_{t=1}^{n}\sum_{k=1}^{d} \left(x_k^t\right)^2 + n\sum_{k=1}^{d} \left(x_k^j\right)^2$$

$$\sum_j \delta_{ij}^2 = \sum_{t=1}^{n}\sum_{k=1}^{d} \left(x_k^t\right)^2 + n\sum_{k=1}^{d} \left(x_k^i\right)^2$$

$$\sum_i \sum_j \delta_{ij}^2 = 2n\sum_{t=1}^{n}\sum_{k=1}^{d} \left(x_k^t\right)^2$$

Some Preprocessing

- Some useful notation:

$$\delta_{\cdot j}^2 = \frac{1}{n}\sum_i \delta_{ij}^2$$

$$\delta_{i\cdot}^2 = \frac{1}{n}\sum_j \delta_{ij}^2$$

$$\delta_{\cdot\cdot}^2 = \frac{1}{n^2}\sum_i\sum_j \delta_{ij}^2$$

- Now let $\beta_{ij} = \sum_{k=1}^{d} x_k^i x_k^j$ Then $\beta_{ij} = \frac{1}{2}\left(\delta_{i\cdot}^2 + \delta_{\cdot j}^2 + \delta_{\cdot\cdot}^2 + \delta_{ij}^2\right)$

The Spectral Approach

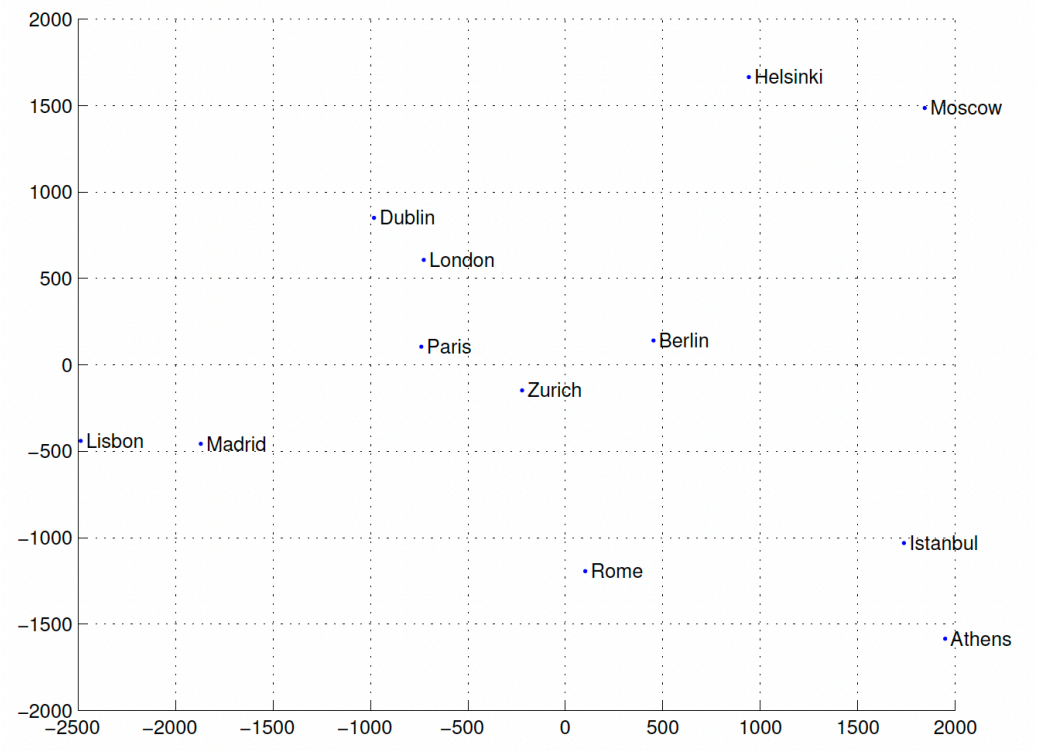
- Begin by constructing $\mathbf{B}_{n \times n}$ where each cell is β_{ij}
- We will approximate the original data $\mathbf{X} \approx \mathbf{C}\mathbf{D}^{1/2}$
  - $\mathbf{C}$ is a matrix whose columns are the eigenvectors of $\mathbf{B}$.

    $\mathbf{D}^{1/2}$ is a diagonal matrix containing the square roots of the eigenvalues of $\mathbf{B}$.
- Our goal is to reduce the dimensions to $k < d < n$, where d is the number of dimensions in the original space. Often, for visualization purposes, we set $k = 2$.
- Project the data to the new space:

$$z_j^t = c_j^t \sqrt{\lambda_j}; \quad j = 1, \ldots, d'; \quad t = 1, \ldots, n$$

where $\mathbf{c}_j^t$ is the j^{th} eigenvector, and λ_j is the j^{th} eigenvalue.

An Example


2000
1500
1000
500
0
−500
−1000
−1500
−2000
−2500
−2000
−1500
−1000
−500
0
500
1000
1500
2000
Helsinki
Moscow
Dublin
London
Paris
Berlin
Zurich
Lisbon
Madrid
Istanbul
Rome
Athens

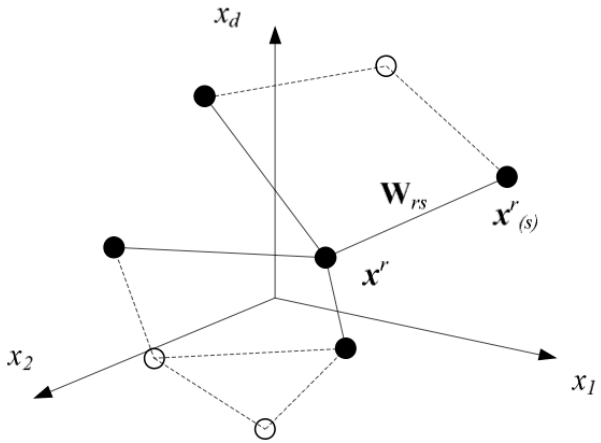
Local Linear Embedding

- Attempts to capture nonlinear relationships in the data.
- Approximates the nonlinear relationships by building locally linear models.
- Consider a point and its immediate neighbors

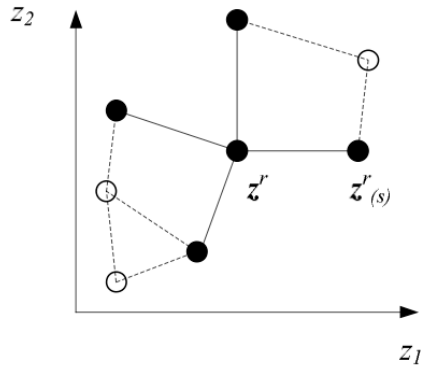
$$\mathbf{x}^t \rightsquigarrow \mathbf{x}^{\mathcal{N}(\mathbf{x}^t)}$$

- So find the k nearest neighbors of $\mathbf{x}^t$
- Then find a set of "reconstruction weights" $\mathbf{W}$ to generate a local linear fit of the $k + 1$ points.
- Repeat this for each of the n points in $\mathcal{D}$.

An Example



x space



z space

Finding the Weight Matrices

- Define an error function

$$E(\mathbf{W} \mid \mathcal{D}) = \sum_{\mathbf{x} \in \mathcal{D}} \left\| \mathbf{x} - \sum_{\mathbf{y} \in \mathcal{D}} \mathbf{w}_{\mathbf{xy}} \mathbf{y} \right\|^2$$

  employing least squares regression.

- This assumes $\mathbf{W}_{\mathbf{xx}} = 0$ and $\sum_{\mathbf{y} \in \mathcal{N}(\mathbf{x})} \mathbf{W}_{\mathbf{xy}} = 1$

- Define a new error function

$$E(z \mid \mathbf{W}) = \sum_{\mathbf{x} \in \mathcal{D}} \left\| \mathbf{z}^{\mathbf{x}} - \sum_{\mathbf{y} \in \mathcal{N}(\mathbf{x})} \mathbf{w}_{\mathbf{xy}} \mathbf{z}^{\mathbf{y}} \right\|^2$$

  and minimize.

Alternate Space

- Similar to MDS, we want nearby points in the original space to be nearby in the projected space.
- Define an alternative matrix:

$$\mathbf{M}_{\mathbf{xy}} = \delta_{\mathbf{xy}} - \mathbf{W}_{\mathbf{xy}} - \mathbf{W}_{\mathbf{yx}} + \sum_{i} \mathbf{W}_{i\mathbf{y}} \mathbf{W}_{i\mathbf{x}}$$

- While $\mathbf{W}$ is not likely to be symmetric, this new matrix $\mathbf{M}$ is guaranteed to be symmetric. It is also positive semi-definite and sparse.
- Rewrite the objective function (error) as

$$(z \mid \mathbf{W}) = \sum_{\mathbf{x},\mathbf{y}} \mathbf{M}_{\mathbf{xy}} (\mathbf{z}^{\mathbf{x}})^{\top} \mathbf{z}^{\mathbf{y}}$$

- Find the $m + 1$ eigenvectors with the smallest eigenvalues and drop the eigenvector with the smallest eigenvalue.

Projection

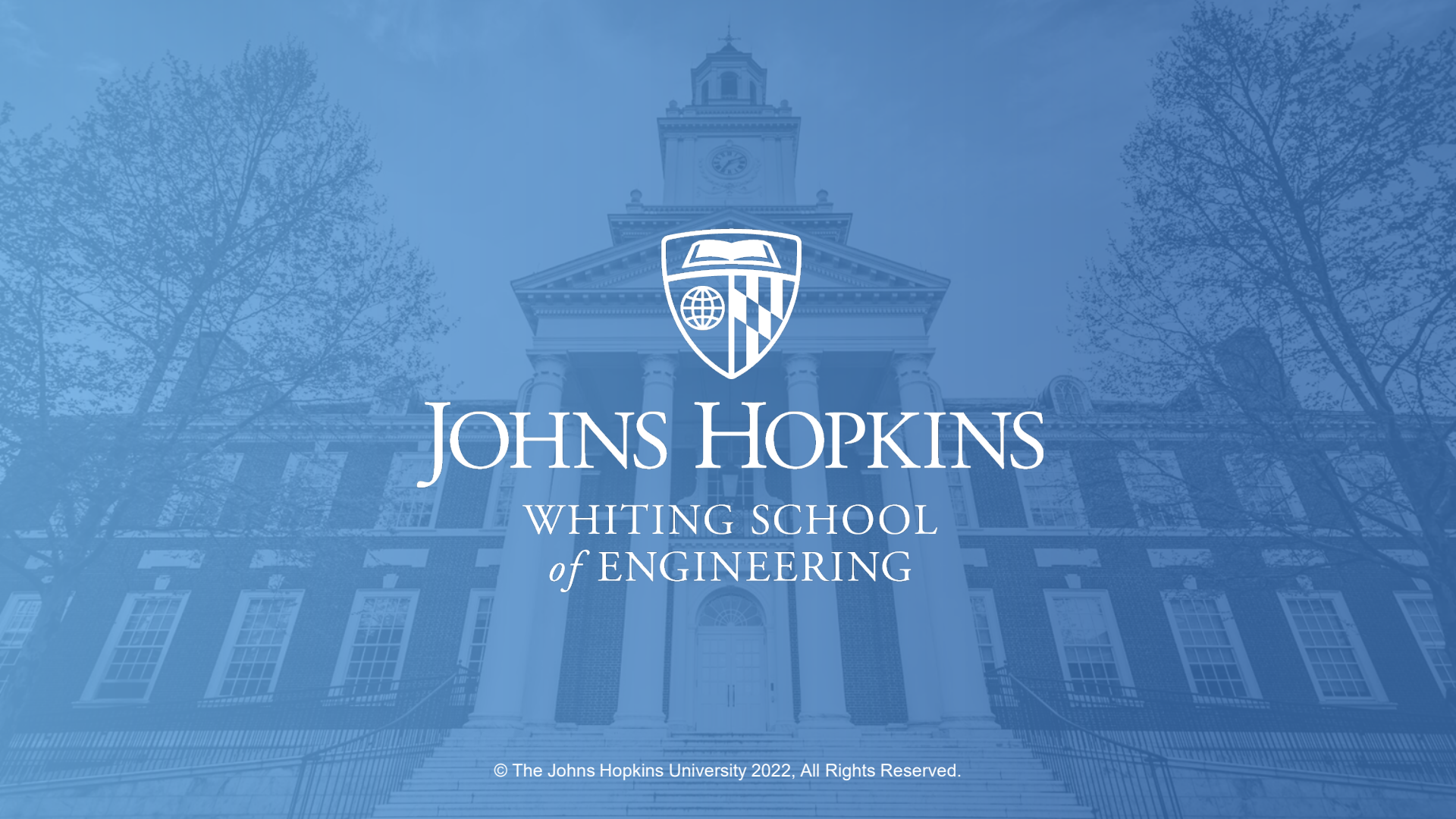
- Option 1
  - For new point $\mathbf{x}'$ find k nearest neighbors and its local linear model.
  - Find projected point $\mathbf{z}'$ in new space using

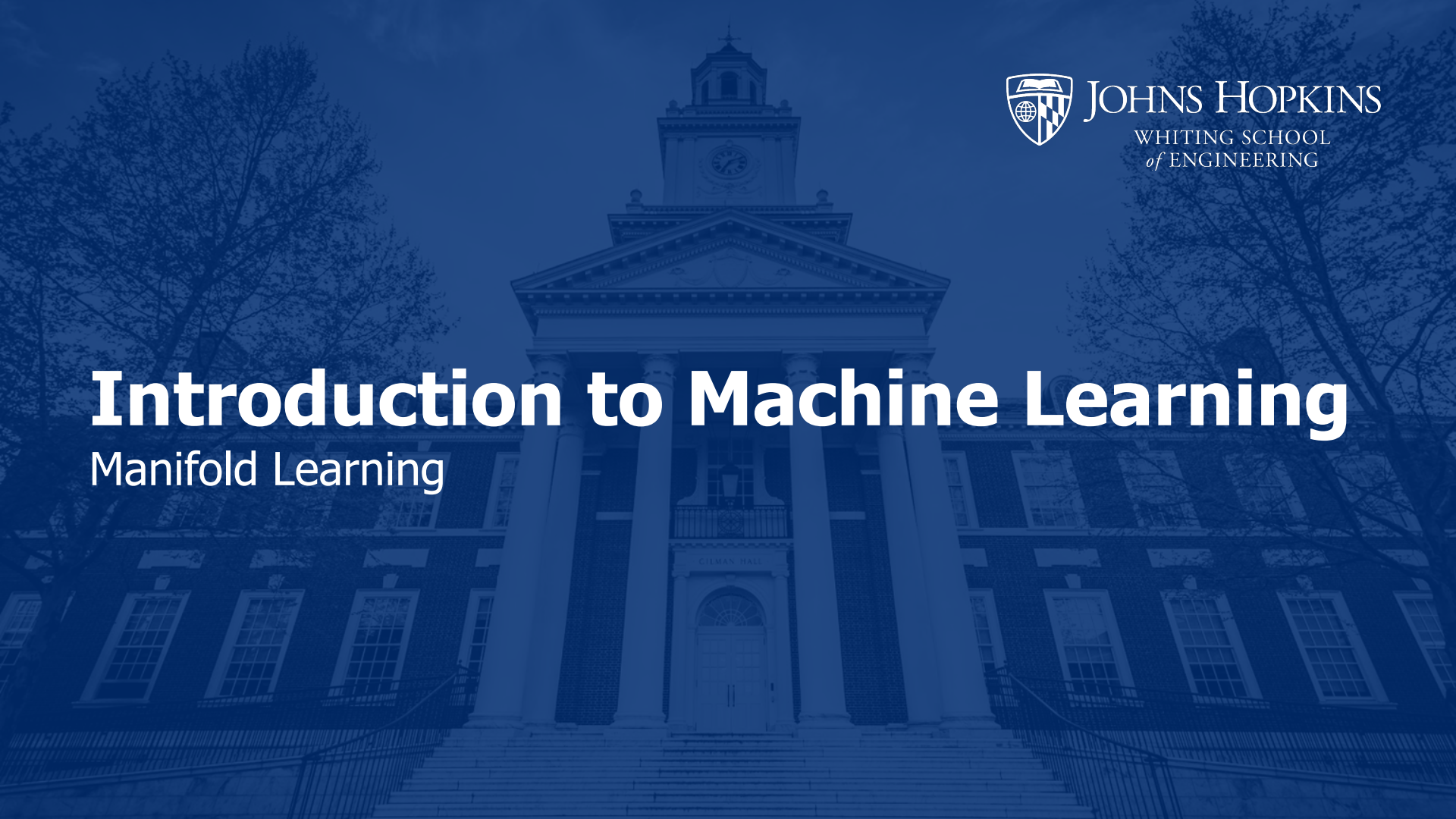
$$\mathbf{z}' = \sum_{\mathbf{y} \in \mathcal{N}(\mathbf{x}')} \mathbf{w}_{\mathbf{y}} \mathbf{z}^{\mathbf{y}}$$

- Option 2
  - Use mapping with original data $f : \mathbf{X} \rightarrow \mathbf{Z}$
  - Create training set $\mathcal{D}^{\ell} = \left\{ \mathbf{x}^t, \mathbf{z}^t \right\}_{t=1}^{n}$
  - Employ a supervised learning method to learn the mapping.

Neighborhood Effects

- The definition of the neighborhood can have a drastic effect on the LLE projection.
- So far, we have assumed k-nearest neighbors for the neighborhood.
- Alternative is to apply a Parzen window approach.
- Tradeoff
  - Small windows may lead to disconnected regions or lack sufficient points to create a locally linear model.
  - Large windows could over-smooth
- Could vary the window size or k based on domain knowledge, but the actual practice for doing this is unknown.





Introduction to Machine Learning

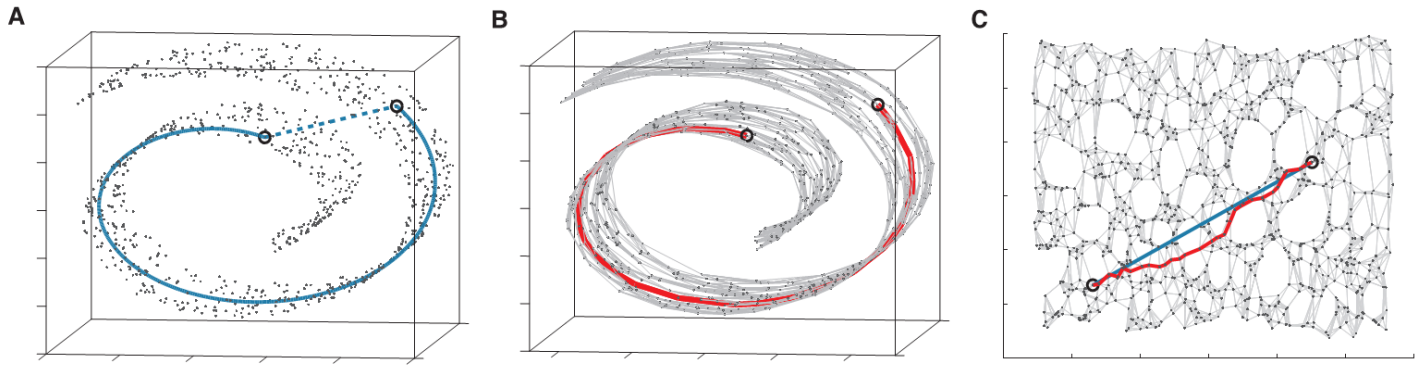
Manifold Learning

Limitations

- PCA is limited to linear subspaces
- Kernel-PCA relaxes linearity assumption but has added complication of determining the right kernel
- LDA is supervised
- MDS imposes a "metric" assumption and is linear in the metric space
- LLE imposes a locally linear assumption to approximate the nonlinearity of the space
- Goal: To find an embedding method that more "naturally" follows the low-dimensional manifold of the data
- Note that LLE and MDS, as well as the ones being presented in this video, are also limited by the fact that there is no general mapping function; the reduction applies only to the data analyzed with no way to map new points to the reduced space

Geodesic Distance

- What if we assume a metric space that follows the local shape of the manifold?
- **Def:** *Geodesic distance* is the length of the shortest curve between two points along the surface of a manifold



Swiss Roll (Tenenbaum *et al.* 2000)

Isometric Mapping (ISOMAP)

- A method to reduce dimensionality by focusing on a manifold defined by geodesic distance
- Proceeds in three steps
  1. Construct neighborhood graph
  2. Estimate pairwise geodesic distance between all points
  3. Apply multidimensional scaling (MDS) to the resulting similarity matrix

Neighborhood Graph

- Define a graph $\mathcal{G}$ over the data points $\mathcal{D}$
- Each vertex in the graph corresponds to a data point
- An edge is created between two points $\mathbf{x}_i$ and $\mathbf{x}_j$ if $\|x_i - x_j\| < \epsilon$
- For the norm calculation, it is customary to use the L_2 norm (i.e., Euclidean distance), but any norm can be used
- An alternative to using the distance threshold ε (which is similar to a Parzen window) is to use the k nearest neighbors, leading to the k-ISOMAP procedure

Estimating Geodesic Distance

- Geodesic distance is approximated by ”walking” the neighborhood graph
- Compute the pairwise shortest path distances between all pairs of points/vertices in the graph
- An algorithm such as the Floyd-Warshall all-pairs shortest path algorithm can be used
- The resulting pairwise distances correspond to the approximations of the geodesic distance

Applying MDS

- Let $\mathbf{S}$ be a matrix of squared geodesic distances between all pairs of points/vertices $\mathbf{x}_i$ and $\mathbf{x}_j$

$$s_{ij} = \delta(\mathbf{x}_i, \mathbf{x}_j)^2 = \|x_i - x_j\|$$

- Let $\mathbf{H}$ be a "centering" matrix, defined such that

$$h_{ij} = \delta(\mathbf{x}_i, \mathbf{x}_j)^2 - \frac{1}{n}$$

  where $n = |\mathcal{D}|$

- Compute embedding matrix $\tau(\mathcal{D}) = -\mathbf{HSH}^\top/2$
- Find eigenbasis for $\tau(\mathcal{D})$ and sort eigenvectors in decreasing order of eigenvalues
- For reduced number of d eigenvectors, let v_p^i be the ith component of the pth eigenvector and λ_p be the pth eigenvalue; then the pth component of the d-dimensional coordinate vector is calculated as $x_i' = \sqrt{\lambda_p} v_p^i$

A Statistical Approach

- Given the current limitations of lacking a mapping function, the primary use for the proposed methods is data visualization; including new data requires regenerating the manifold
- A statistical approach depends upon a t-distributed interpretation of the data to be embedded in the underlying manifold
- The result is also a nonlinear mapping
- The resulting method is referred to as "t-Stochastic Neighbor Embedding" (t-SNE)

Phases of t-SNE

- There are two main phases of the t-SNE algorithm.
- Phase 1: Construct a probability distribution over pairs of the data points such that similar data points have a higher probability than dissimilar points
- Phase 2: Construct a probability distribution over the same points in the reduced space by minimizing the Kullback-Leibler divergence between the two distributions
- Note that any similarity metric can be used, but it is customary to use the L_2 norm
- Assuming a uniform distribution in geodesic space (like ISOMAP) results in a variant called Uniform Manifold Approximation and Projection (UMAP)

Computing Initial Distribution

- For each pair of points $\mathbf{x}_i$ and $\mathbf{x}_j$ set $P(\mathbf{x}_i \mid \mathbf{x}_i) = 0$ and then compute

$$P(\mathbf{x}_j \mid \mathbf{x}_i) = \frac{\exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|\mathbf{x}_i - \mathbf{x}_k\|^2 / 2\sigma_i^2\right)}$$

- Next define $P(\mathbf{x}_i, \mathbf{x}_j) = [P(\mathbf{x}_j \mid \mathbf{x}_i) + P(\mathbf{x}_i \mid \mathbf{x}_j)]/2|\mathcal{D}|$
- Notice a Gaussian kernel is used, where the bandwidth σ is set based on the density of the data

Computing Low Dimensional Distribution

- Typically, a 2- or 3-dimensional map is learned since the purpose is visualization
- Low dimensional points are denoted $\mathbf{y}_1, \ldots, \mathbf{y}_{|\mathcal{D}|}$

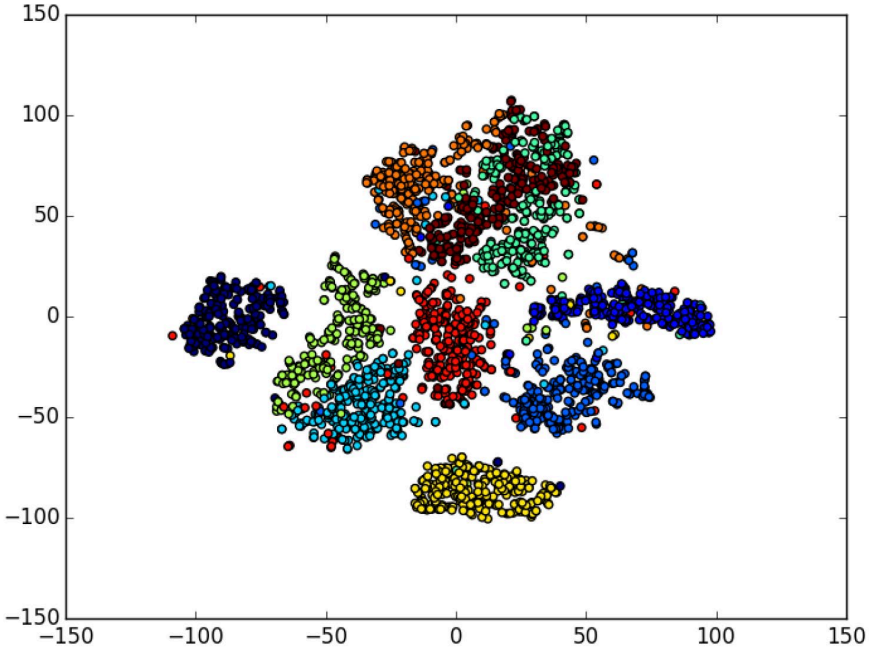
  and the reduced distribution is defined such that

$$Q(\mathbf{y}_i, \mathbf{y}_j) = \frac{\left(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2\right)^{-1}}{\sum_k \sum_{\ell \neq k} \left(1 + \|\mathbf{y}_k - \mathbf{y}_\ell\|^2\right)^{-1}}$$

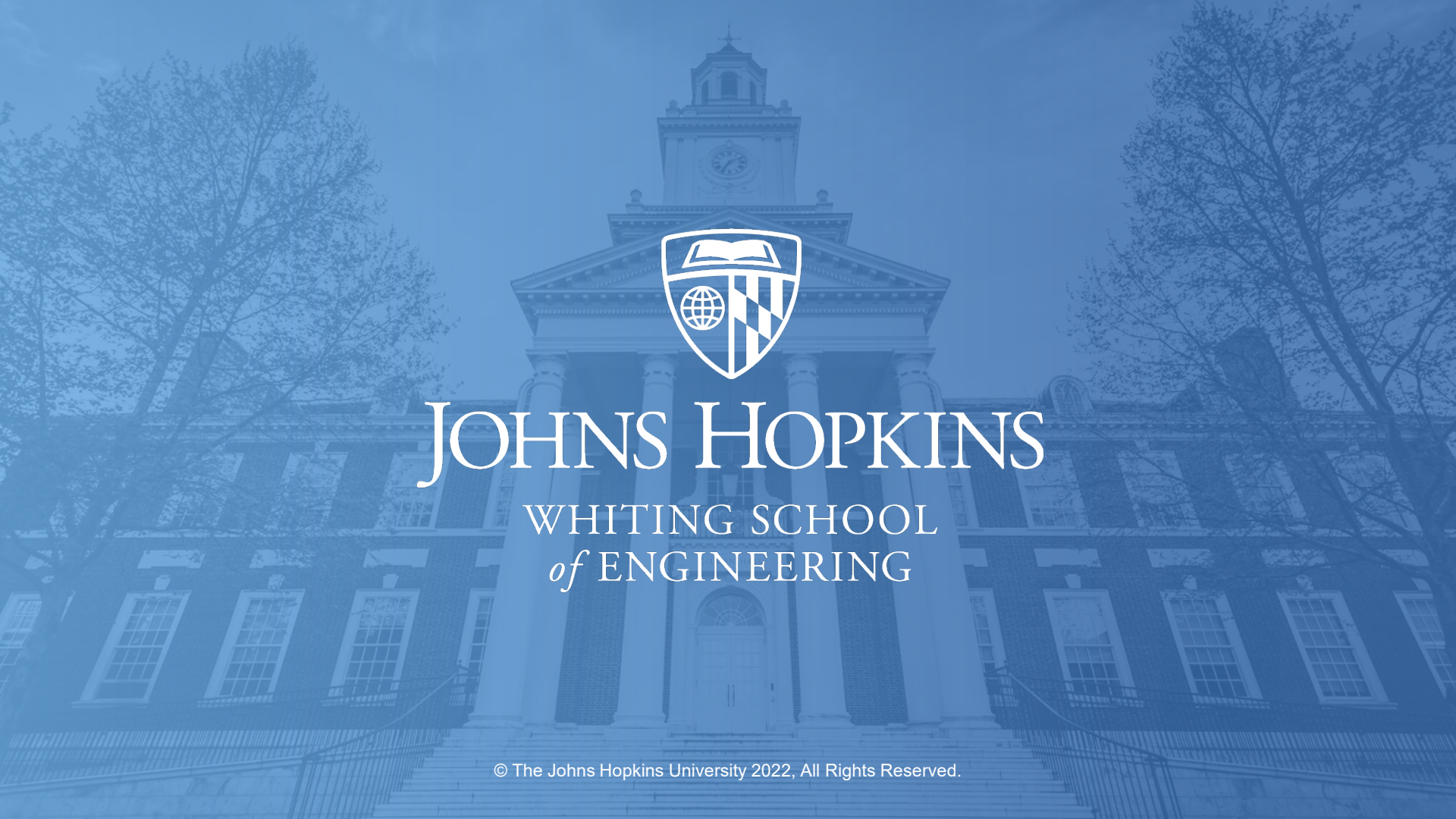
- The points $\mathbf{y}_1, \ldots, \mathbf{y}_{|\mathcal{D}|}$ are found by minimizing KL-divergence using gradient descent

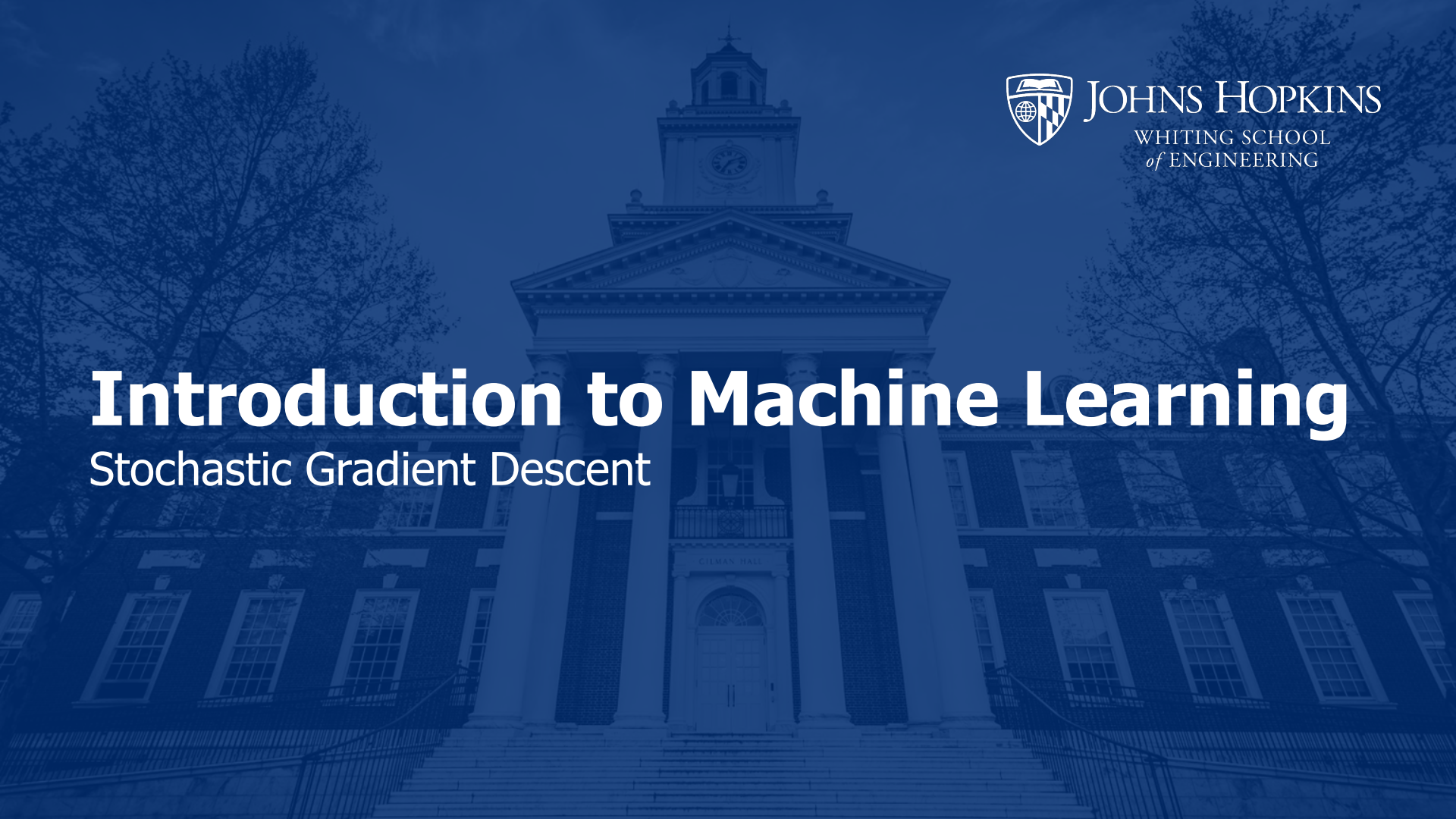
$$D_{KL}(P\|Q) = \sum_{i \neq j} P(\mathbf{x}_i, \mathbf{x}_j) \log \frac{P(\mathbf{x}_i, \mathbf{x}_j)}{Q(\mathbf{y}_i, \mathbf{y}_j)}$$

Example Mapping – MNIST



Tosun, 2016





Introduction to Machine Learning

Stochastic Gradient Descent

Optimization

- To this point, we have been approaching machine learning as an optimization problem.
- Specifically, we seek to find model parameters $\boldsymbol{\Theta}$ such that some loss function $\mathcal{L}(f(\mathcal{D}, \boldsymbol{\Theta}))$ is minimized over data set $\mathcal{D}$.
- If we know the functional form of the model to be learned, we can apply analytical methods for optimization:

$$\boldsymbol{\Theta} = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}^*}{\operatorname{argmin}} \mathcal{L}(f(\mathcal{D}, \boldsymbol{\theta}))$$

$$\nabla_{\boldsymbol{\Theta}} \mathcal{L}(f(\mathcal{D}, \boldsymbol{\Theta})) = 0$$

Solve for Θ.

Gradient Descent

- Generally, we do not know the complete functional form of the loss function due to its dependence on the model's functional form $f(\mathcal{D}, \boldsymbol{\Theta})$
- Furthermore, the resulting function is likely to be highly nonlinear, resulting in
  - Several critical points (minima and saddle points)
  - Some critical points worse than others (local minima)
- Finding the global optimum is therefore intractable.
- Therefore, we approximate the solution by computing a local gradient and following the gradient to the nearest local optimum.

One-Step Gradient Descent

- Select an initial point in the search space: $\boldsymbol{\theta}^0$
- Repeat

  $t \leftarrow t + 1$

  - Calculate the gradient of the loss function at that point: $\nabla_{\boldsymbol{\theta}} \mathcal{L}(f(\mathcal{D}, \boldsymbol{\theta}))$
  - Update the parameters: $\theta^t \leftarrow \theta^{t-1} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(f\left(\mathcal{D}, \boldsymbol{\theta}^{t-1}\right)\right.$
- Until some convergence property is met.

- Here $\eta \in (0, 1)$ is a learning rate, corresponding to the "step size" of the update.

Types of Gradient Descent

- Generally, three types of gradient descent are employed:
  - Batch gradient descent
  - Incremental (stochastic) gradient descent
  - Mini-batch (stochastic) gradient descent
- Technically, the incremental approach is a type of mini-batch where the batch size is set equal to one.
- Key to mini-batch gradient descent is randomization; this is the stochastic part.

Batch Gradient Descent

- Assume we have a training set, $\mathcal{D} = \{\mathbf{x}^i, y^i\}_{i=1}^{|\mathcal{D}|}$
- Initialize parameters $\boldsymbol{\theta}^0$
- Repeat

  $t \leftarrow t + 1$

  $\Delta\boldsymbol{\theta} \leftarrow 0$

  For $i = 1, |\mathcal{D}|$ do

  $\Delta\boldsymbol{\theta} \leftarrow \Delta\boldsymbol{\theta} - \eta_t \nabla_\theta \mathcal{L}\left(f\left(x^i, \boldsymbol{\theta}^{t-1}\right)\right)$

  End For

  $\boldsymbol{\theta}^t \leftarrow \boldsymbol{\theta}^{t-1} + \frac{1}{|\mathcal{D}|}\Delta\boldsymbol{\theta}$
- Until some convergence property is met

Incremental Gradient Descent

- Assume we have a training set, $\mathcal{D} = \{\mathbf{x}^i, y^i\}_{i=1}^{|\mathcal{D}|}$
- Initialize parameters $\boldsymbol{\theta}^0$
- Repeat
  - Generate a random permutation of $\mathcal{D}$

    For $i = 1, |\mathcal{D}|$ do

    $$t \leftarrow t + 1$$
    $$\Delta\boldsymbol{\theta} \leftarrow -\eta_t \nabla_\theta \mathcal{L}\left(f\left(x^i, \boldsymbol{\theta}^{t-1}\right)\right)$$
    $$\boldsymbol{\theta}^t \leftarrow \boldsymbol{\theta}^{t-1} + \Delta\boldsymbol{\theta}$$

    End For
- Until some convergence property is met

Mini-batch Gradient Descent

- Assume we have a training set, $\mathcal{D} = \left\{\mathbf{x}^i, y^i\right\}_{i=1}^{|\mathcal{D}|}$
- Initialize parameters $\boldsymbol{\theta}^0$
- Repeat

  $t \leftarrow t + 1$

  Let $\mathcal{B}^t \sim \mathbb{U}(\mathcal{D})$ be a randomly generated mini-batch of some size $\left|\mathcal{B}^t\right| < |\mathcal{D}|$

  $\Delta\boldsymbol{\theta} \leftarrow 0$

  For $i = 1, \left|\mathcal{B}^t\right|$ do

  $\Delta\boldsymbol{\theta} \leftarrow \Delta\boldsymbol{\theta} - \eta_t \nabla_\theta \mathcal{L}\left(f\left(x^i, \boldsymbol{\theta}^{t-1}\right)\right)$

  End For

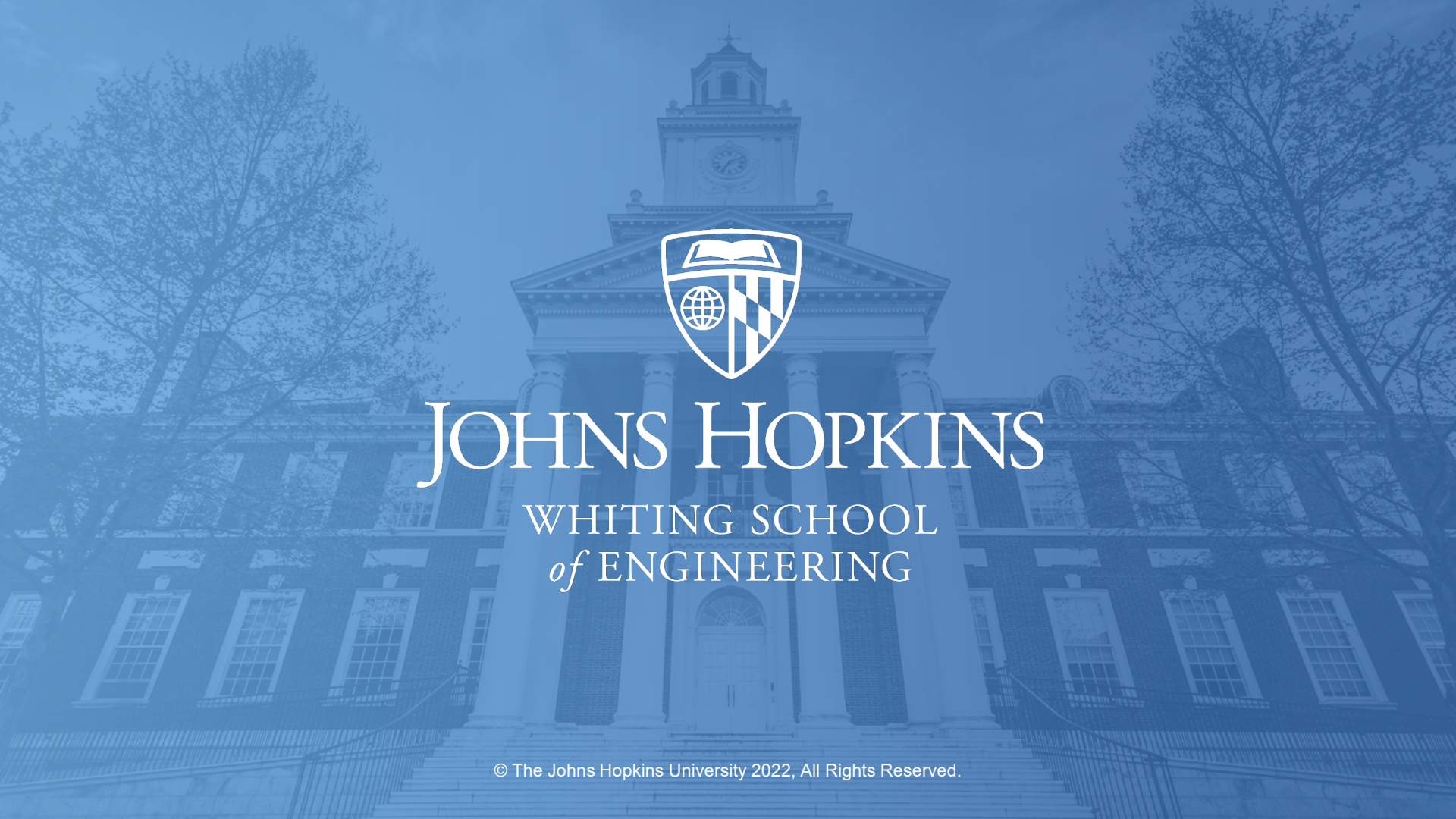
  $\boldsymbol{\theta}^t \leftarrow \boldsymbol{\theta}^{t-1} + \frac{1}{|\mathcal{B}^t|}\delta\boldsymbol{\theta}$
- Until some convergence property is met

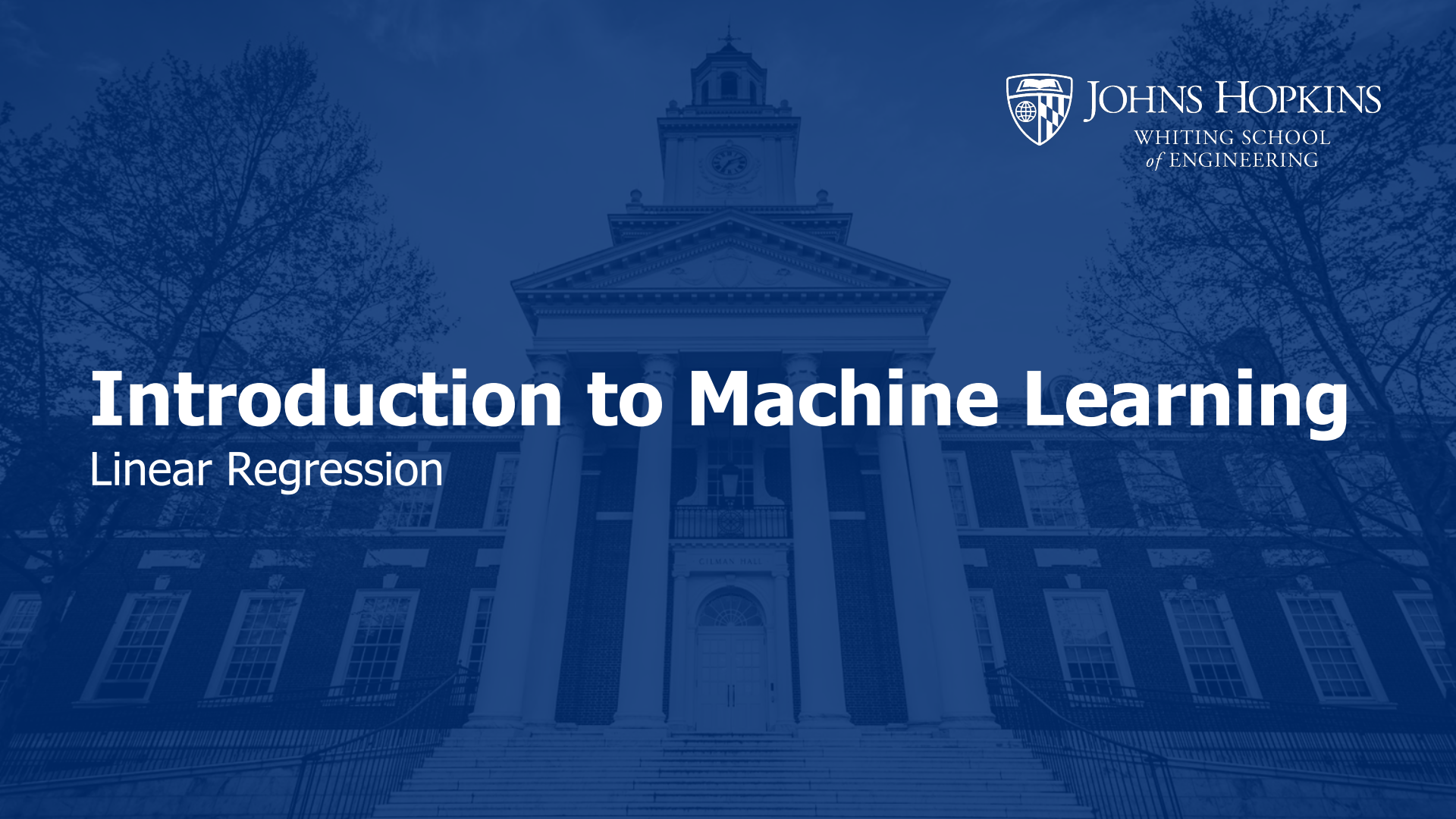
The Learning Rate

- Much of the behavior of gradient descent is driven by the selected learning rate.
- It is critical that the learning rate be tuned.
- In appropriate learning rates can have negative results:
  - Too small – slow convergence
  - Too large – unstable convergence due to oscillation (over-shooting)
- Ideally:
  - Want the learning rate to be large when we have long, shallow gradients
  - Want the learning rate to be small when we have short, steep gradients
- Gradient magnitudes are determined directly, but the length is difficult to determine.
- One approach to handling is to set the initial learning rate large and gradually reduce over time (referred to as "annealing").

Convergence

- Gradient descent, in all of its forms, is an iterative search process.
- As such, we need to have a way to determine when to stop the search.
- There are a number of common approaches that can be considered:
  - Pre-set a maximum number of iterations and stop when that number is reached (difficult to manage over-fitting or under-fitting)
  - Keep track of a recent history of updates; when the magnitudes of the updates appear to stagnate, stop (can lead to getting trapped on a ridge or plateau, or can lead to overfitting)
  - Track performance against a separate, held-out data set; stop when performance starts to degrade on this data set (manages over/under fitting but requires some level of smoothing to manage noise)





Introduction to Machine Learning

Linear Regression

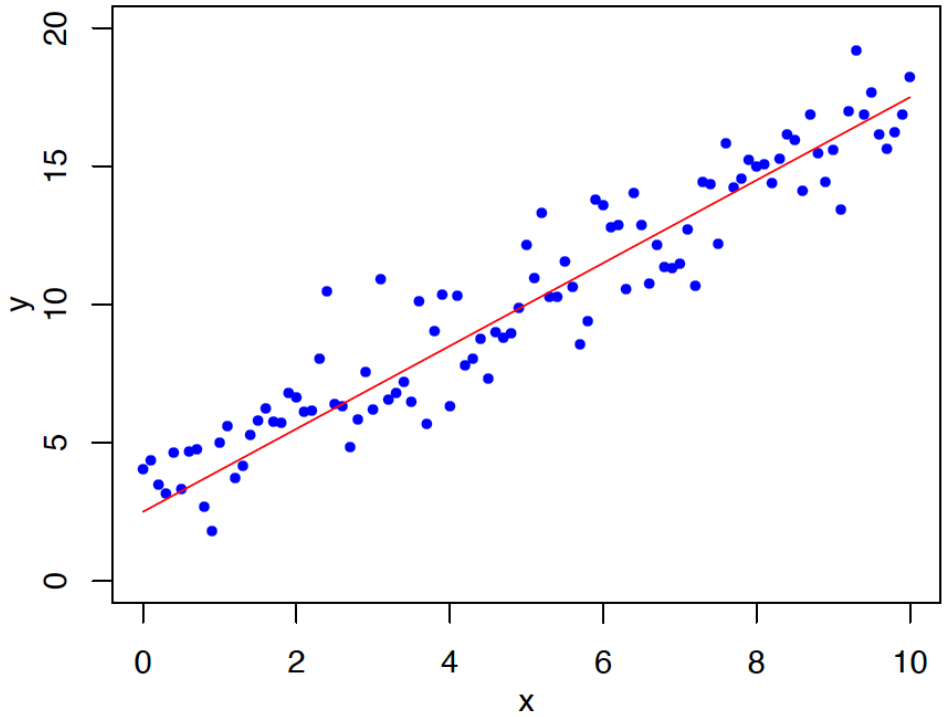
Linear Models

- So far, we have considered learning methods that are not linear:
  - Nonparametric learning such as nearest neighbor
  - Decision trees
  - Decision rules
- Here we take a step back and consider only linear models.

$$f(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i$$

- The objective is to learn the weights $\mathbf{w} = [w_0, w_1, \ldots, w_d]^\top$
- We provide this as a foundation for more powerful models to follow.
- We begin by considering the regression problem.

Linear Regression



Linear Regression

- The task is to learn the weights $\mathbf{w}$ in a linear model.
- The typical loss function is mean squared error:

$$\mathcal{L}(f(\mathcal{D}, \mathbf{w})) = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left(y^i - f\left(\mathbf{x}^i, \mathbf{w}\right)\right)^2$$

- Since the model is linear, this becomes

$$\mathcal{L}(f(\mathcal{D}, \mathbf{w})) = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \left(y^i - w_0 - \sum_{j=1}^{d} w_j x_j^i\right)^2$$

- This can be solved analytically: $\mathbf{w} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{Y}$

  where $\mathbf{X}$ is the data matrix (augmented by a vector of 1's for the intercept) and $\mathbf{Y}$ is the response vector.

Assumptions

- The error (residual) terms are independent.
- The error (residual) terms all have the same variance across the entire regression function (homoscedastic).
- The errors all have a conditional mean of zero.
- The covariates are linearly independent.
- Sometimes it is assumed that the errors are normally distributed

Linear Regression by Gradient Descent

- Solving the linear regression problem involves inverting the matrix $\mathbf{X}^\top\mathbf{X}$
- For a large data set, this can be very expensive.
- Given the assumptions just presented, the loss function is convex, but note that some of the assumptions (e.g., equal variance and Gaussian residuals) can be relaxed.
- Therefore, applying gradient descent will find the globally optimal set of weights, under the added assumption that the data is truly linear.
- Any of the three methods (iterative, batch, or mini-batch) can be used; however, the full batch approach is typically applied for stability.

Batch Gradient Descent

- Assume we have a training set, $\mathcal{D} = \{\mathbf{x}_i, r_i\}_{i=1}^{|\mathcal{D}|}$
- Initialize parameters $\mathbf{w}^0$
- Repeat

  $t \leftarrow t + 1$

  $\Delta\mathbf{w} \leftarrow 0$

  For $i = 1, |\mathcal{D}|$ do

  $$\Delta\mathbf{w} \leftarrow \Delta\mathbf{w} - \eta\left(r_i - \left(\mathbf{w}^t\right)^\top \mathbf{x}_i\right)\mathbf{x}_i$$

  End for

  $$\mathbf{w}^t \leftarrow \mathbf{w}^{t-1} + \frac{1}{|\mathcal{D}|}\Delta\mathbf{w}$$

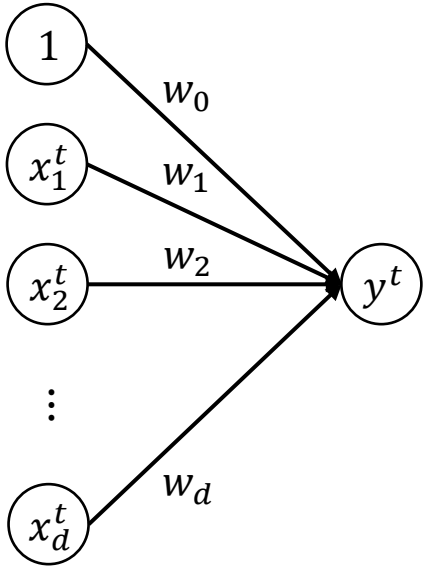
- Until some convergence property is met

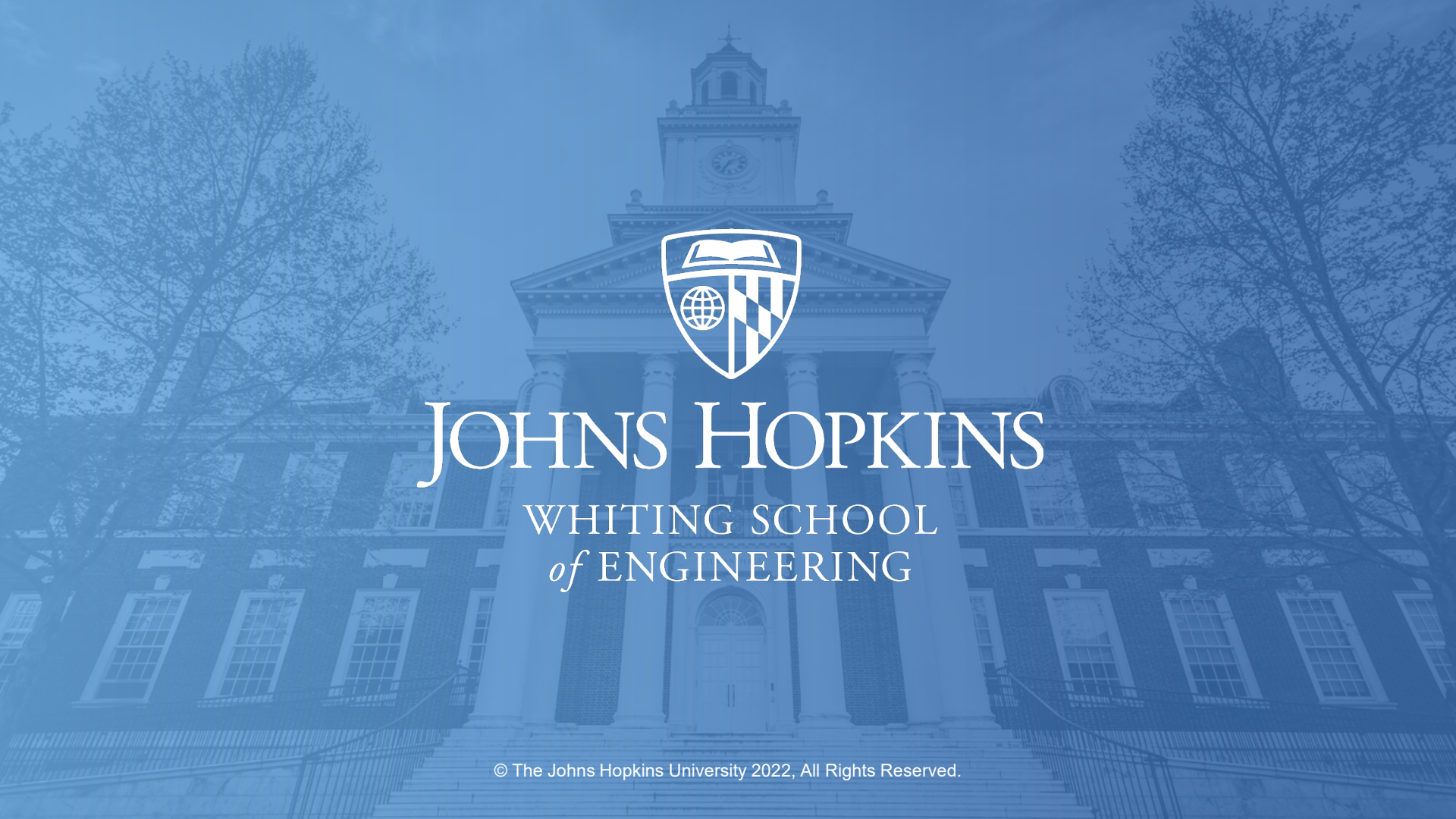
Coefficient of Determination

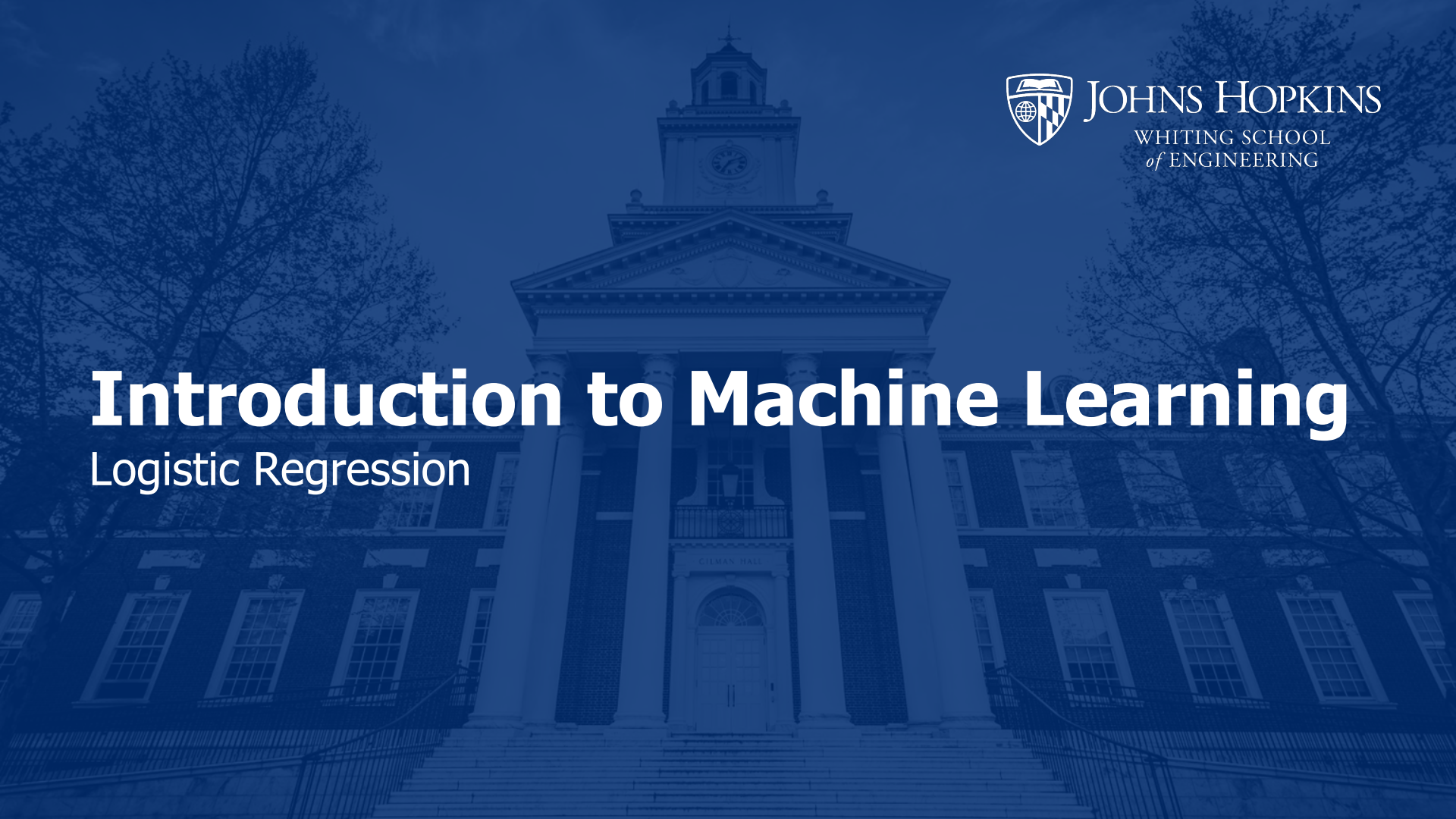
- In addition to considering mean squared error, it is common to evaluate the results of linear regression using the *coefficient of determinations* i.e., R^2
- Note that R^2 is only valid for *linear* models.
- Reflects the proportion of the variation in the dependent (response) variable predictable by the covariates (features).
- Define $e^i = y^i - f^i$ as the "residual" of a prediction.
  - Residual sum of squares: $SS_{\text{res}} = \sum_i \left(y^i - f^i\right)^2 = \sum_i \left(e^i\right)^2$
  - Total sum of squares: $SS_{tot} = \sum_i \left(y^i - \bar{y}\right)^2$
  - Coefficient of determination: $R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$

A Neural Network View

Looking ahead, linear regression can be represented in a network form:







Introduction to Machine Learning

Logistic Regression

Discriminant Function

- Maximize posterior probability: $P(c \mid \mathbf{x})$
- Consider the two-class classification problem, where $c \in \{0, 1\}$
- We seek to find $P(c = 1 \mid \mathbf{x})$ and $P(c = 0 \mid \mathbf{x})$
- We use the logit distribution

$$\text{logit}(P) = \log \frac{P}{1 - P}$$

- Suppose we consider the natural log. Then we derive the logistic function:

$$\text{logit}^{-1}(P) = \frac{1}{1 + \exp(-P)} = \frac{\exp(P)}{1 + \exp(P)}$$

Estimating the Posteriors

- From the logistic function, we get

$$P(c=1 \mid \mathbf{x}) = \frac{\exp\left(w_0 + \sum_{i=1}^{d} w_i x_i\right)}{1 + \exp\left(w_0 + \sum_{i=1}^{d} w_i x_i\right)}$$

$$P(c=0 \mid \mathbf{x}) = \frac{1}{1 + \exp\left(w_0 + \sum_{i=1}^{d} w_i x_i\right)}$$

- Then we choose $c = 1$ if

$$1 < \frac{P(c=1|\mathbf{x})}{P(c=0|\mathbf{x})} < \frac{\exp\left(w_0 + \sum_{i=1}^{d} w_i x_i\right)}{\frac{1 + \exp\left(w_0 + \sum_{i=1}^{d} w_i x_i\right)}{1}}$$

Estimating the Posteriors

- Note that this is the same as saying

$$1 < \exp\left(w_0 + \sum_{i=1}^{d} w_i x_i\right)$$

- Taking the log of both sides yields

$$0 < w_0 + \sum_{i=1}^{d} w_i x_i$$

- Notice that this is the linear discriminant function.

Learning the Weights

- The goal is to find weights such that

$$\mathbf{w} = \underset{\mathbf{w}}{\operatorname{argmax}} \prod_k P\left(C^k \mid \mathbf{x}^i, \mathbf{w}\right)$$

- It is customary to work in log space since probabilities can be quite small, yielding

$$\mathbf{w} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_k \ln P\left(C^k \mid \mathbf{x}^i, \mathbf{w}\right)$$

- Find conditional log likelihood as

$$\begin{aligned}
\ell(\mathbf{w}) &= \sum_k \left(C^k \ln P\left(C^k = 1 \mid \mathbf{x}^i, \mathbf{w}\right) + \left(1 - C^k\right) \ln P\left(C^k = 0 \mid \mathbf{x}^i, \mathbf{w}\right)\right. \\
&= \sum_k \left(C^k \ln \frac{P\left(C^k = 1 \mid \mathbf{x}^i, \mathbf{w}\right)}{P\left(C^k = 0 \mid \mathbf{x}^i, \mathbf{w}\right)} + \ln P\left(C^k = 0 \mid \mathbf{x}^i, \mathbf{w}\right)\right) \\
&= \sum_k \left[C^k \left(w_0 + \sum_{j=1}^d w_j x_j^i\right) - \ln\left(1 + \exp \sum_{j=1}^d w_j x_j^i\right)\right].
\end{aligned}$$

Gradient Descent

- The classic approach to solve is via gradient descent.
- We seek $\mathbf{w}$ such that for all w_j

$$\frac{\partial \ell(\mathbf{w})}{\partial w_j} = 0$$

- Note that

$$\frac{\partial \ell(\mathbf{w})}{\partial w_j} = \sum_k x_j^i \left(C^k - \hat{P}\left(C^k = 1 \mid \mathbf{w} \right) \right)$$

- Weight update rule:

$$w_j \leftarrow w_j + \eta \sum_k x_j^i \left(C^k - \hat{P}\left(C^k = 1 \mid \mathbf{w} \right) \right)$$

Regularization

- Avoiding learning noise, which would lead to overfitting.
- Penalize large weights.

$$\mathbf{w} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_k \ln P\left(C^k \mid \mathbf{x}^i, \mathbf{w}\right) - \frac{\lambda}{2}\|\mathbf{w}\|^2$$

- New gradient

$$\frac{\partial \ell(\mathbf{w})}{\partial\left(w_j\right)} = \sum_k x_j^i \left(C^k - \hat{P}\left(C^k = 1 \mid \mathbf{w}\right)\right) - \lambda w_j$$

- New update

$$w_j \leftarrow w_j + \eta \sum_k x_j^i \left(C^k - \hat{P}\left(C^k = 1 \mid \mathbf{w}\right)\right) - \eta\lambda w_j$$

Two-Class Logistic Regression

For $j = 0, \ldots, d$
  $w_j \leftarrow$rand(-0.01,0.01)
Repeat
  For $j = 0, \ldots, d$
    $\Delta w_j \leftarrow 0$
  For $t = 1, \ldots, N$
    $o \leftarrow 0$
    For $j = 0, \ldots, d$
      $o \leftarrow o + w_j x_j^t$
    $y \leftarrow \text{sigmoid}(o)$
    $\Delta w_j \leftarrow \Delta w_j + (r^t - y) x_j^t$
  For $j = 0, \ldots, d$
    $w_j \leftarrow w_j + \eta \Delta w_j$
Until convergence

Multi-Class Classification

- Extend the log likelihood function.
- For $i = 1, \ldots, K-1$

$$P(C = c_i \mid \mathbf{x}) = \frac{\exp\left(w_0 + \sum_{j=1}^{d} w_j x_j\right)}{1 + \sum_{c=1}^{K-1} \exp\left(w_{c0} + \sum_{j=1}^{d} w_j x_j\right)}$$

- For $i = K$

$$P(C = c_i \mid \mathbf{x}) = \frac{1}{1 + \sum_{c=1}^{K} \exp\left(w_{c0} + \sum_{j=1}^{d} w_j x_j\right)}$$

- Update rule

$$w_{ji} \leftarrow w_{ji} + \eta \sum_{k} x_j^i \delta\left(C = c_k\right) - \hat{P}\left(C = c_k \mid \mathbf{x}^i, \mathbf{w}\right) \Bigg) - \eta \lambda w_{ji}$$

Classification with Softmax

- In logistic regression (and similar models), probabilistic classification is often done via the softmax function (based on the logistic, sigmoid function):

$$y_i = P\left(C_i \mid \mathbf{x}\right) = \frac{\exp\left[\mathbf{w}_i^\top \mathbf{x} + w_{i0}\right]}{\sum_{j=1}^{K} \exp\left[\mathbf{w}_j^\top \mathbf{x} + w_{j0}\right]}$$

- This assumes we treat *all* of the classes uniformly.
- We assume a multinomial class distribution, $\mathbf{r}^t \mid \mathbf{x}^t \sim \text{Mult}_k\left(\mathbf{y}^t\right)$ where $y_i^t \equiv P\left(C_i \mid \mathbf{x}^t\right)$

Cross Entropy Loss

- Using softmax, we want to maximize the sample likelihood of the weights given the data is

$$L\left(\{\mathbf{w}_i, w_{i0}\} \mid X\right) = \prod_t \prod_i \left(y_i^t\right)^{r_i^t}$$

- To learn the parameters $\mathbf{w}_i^\top$ for each class C_i

  we minimize the cross-entropy loss, defined as the negative log of the likelihood:

$$\ell\left(\{\mathbf{w}_i, w_{i0}\} \mid X\right) = -\sum_t \sum_i r_i^t \log y_i^t$$

Gradient Descent

- We use gradient descent to learn the weights.
- We can show that for $y_i = \exp(a_i) / \sum_j \exp(a_j)$

$$\frac{\partial y_i}{\partial a_j} = y_i(\delta_{ij} - y_j)$$

where

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

Update Rule

- Based on this, we can derive the update rule as follows:

$$\begin{aligned}
\Delta \mathbf{w}_j &= \eta \sum_t \sum_i \frac{r_i^t}{y_i^t} y_i^t \left(\delta_{ij} - y_j^t\right) \mathbf{x}^t \\
&= \eta \sum_t \sum_i r_i^t \left(\delta_{ij} - y_j^t\right) \mathbf{x}^t \\
&= \eta \sum_t \left[\sum_i r_i^t \delta_{ij} - y_j^t \sum_i r_i^t \right] \mathbf{x}^t \\
&= \eta \sum_t \left(r_j^t - y_j^t\right) \mathbf{x}^t.
\end{aligned}$$

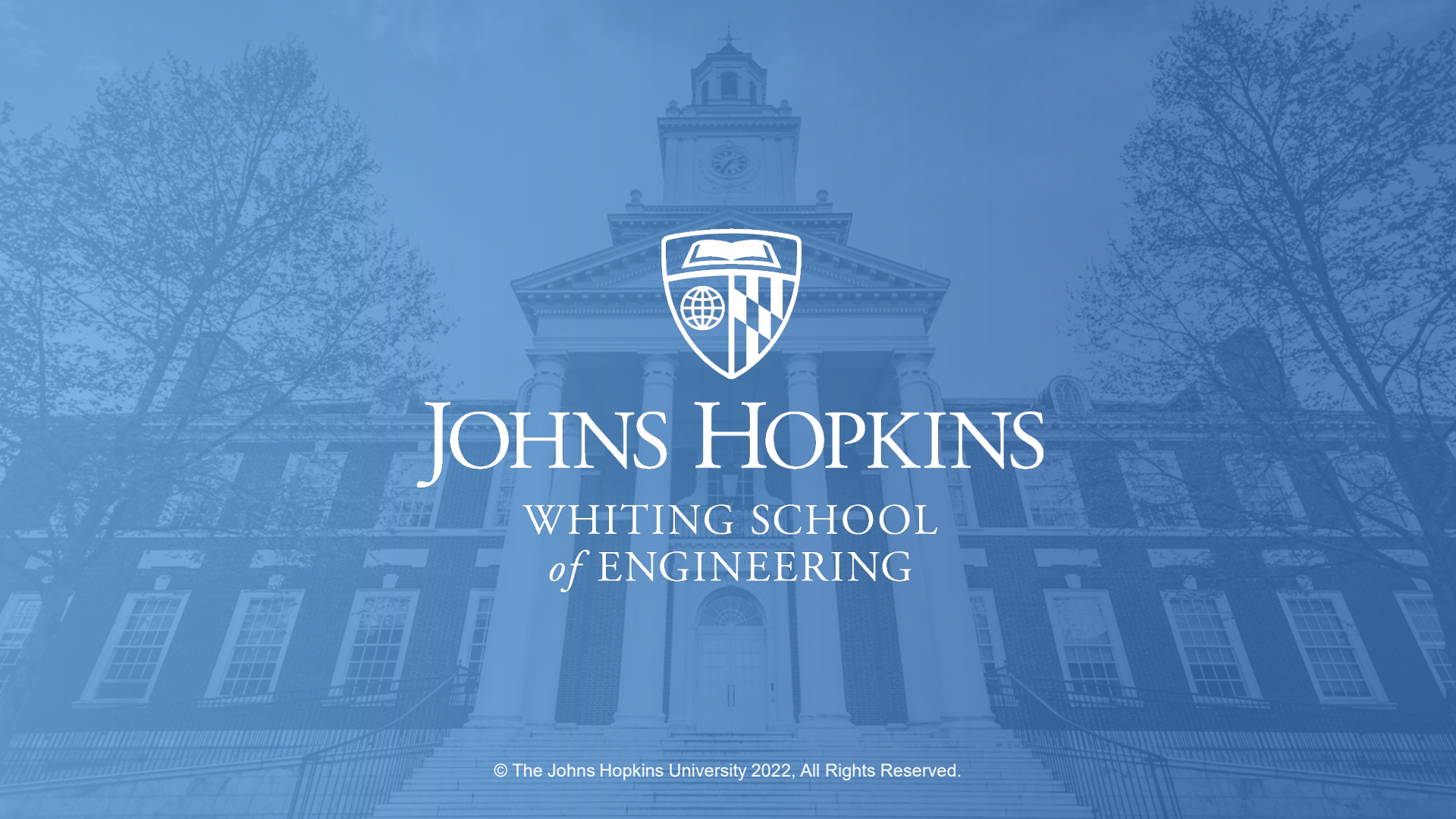
- For the bias term,

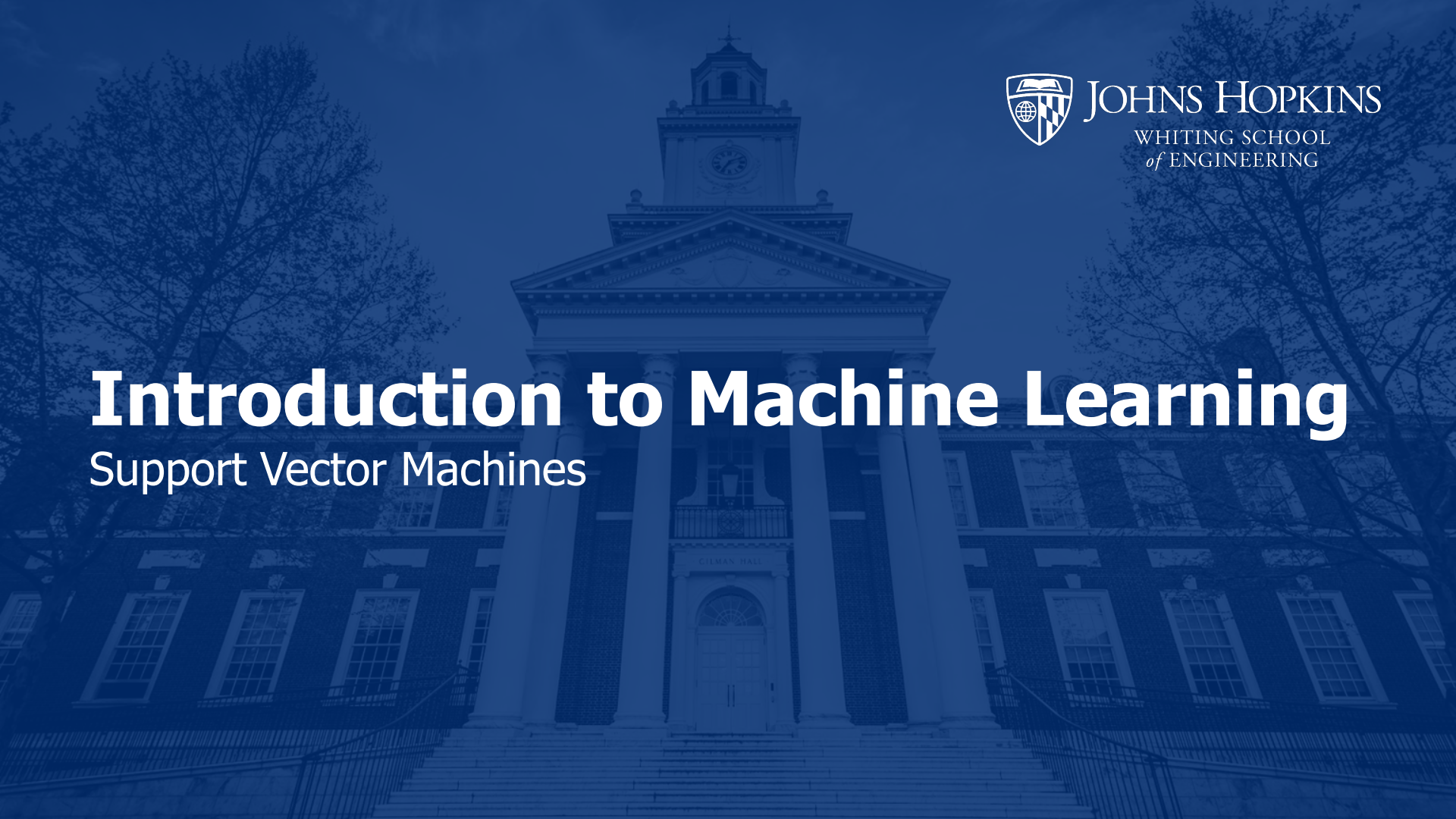
$$\Delta w_{j0} = \eta \sum_t \left(r_j^t - y_j^t\right)$$

since, by definition, $x_0^t = 1$

Multi-Class Pseudocode

For $i = 1, \ldots, K$
  For $j = 0, \ldots, d$
    $w_{ij} \leftarrow \text{rand}(-0.01, 0.01)$
Repeat
  For $i = 1, \ldots, K$
    For $j = 0, \ldots, d$
      $\Delta w_{ij} \leftarrow 0$
  For $t = 1, \ldots, N$
    For $i = 1, \ldots, K$
      $o_i \leftarrow 0$
      For $j = 0, \ldots, d$
        $o_i \leftarrow o_i + w_{ij} x_j^t$
    For $i = 1, \ldots, K$
      $y_i \leftarrow \exp(o_i) / \sum_k \exp(o_k)$
    For $i = 1, \ldots, K$
      For $j = 0, \ldots, d$
        $\Delta w_{ij} \leftarrow \Delta w_{ij} + (r_i^t - y_i) x_j^t$
  For $i = 1, \ldots, K$
    For $j = 0, \ldots, d$
      $w_{ij} \leftarrow w_{ij} + \eta \Delta w_{ij}$
Until convergence

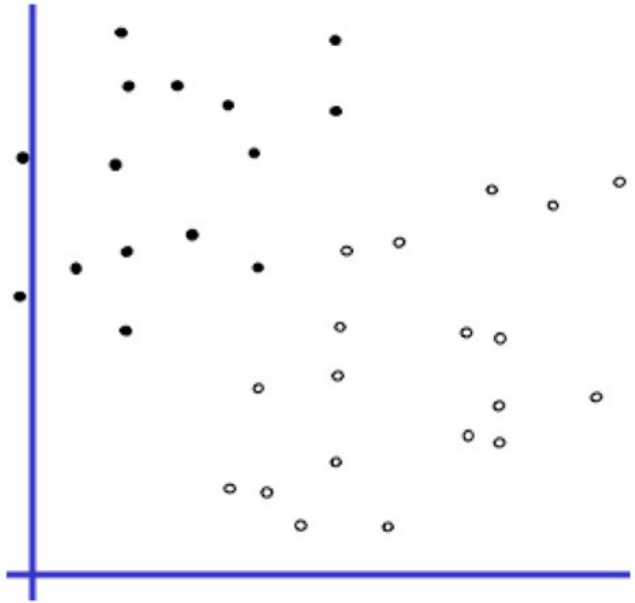




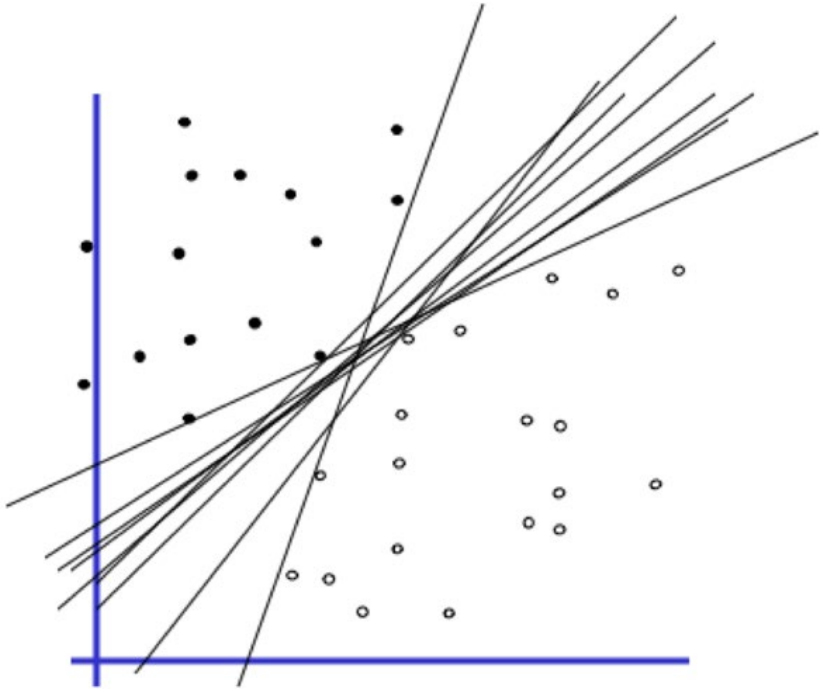
Introduction to Machine Learning

Support Vector Machines

A Data Set



A Data Set



Structural Risk Minimization

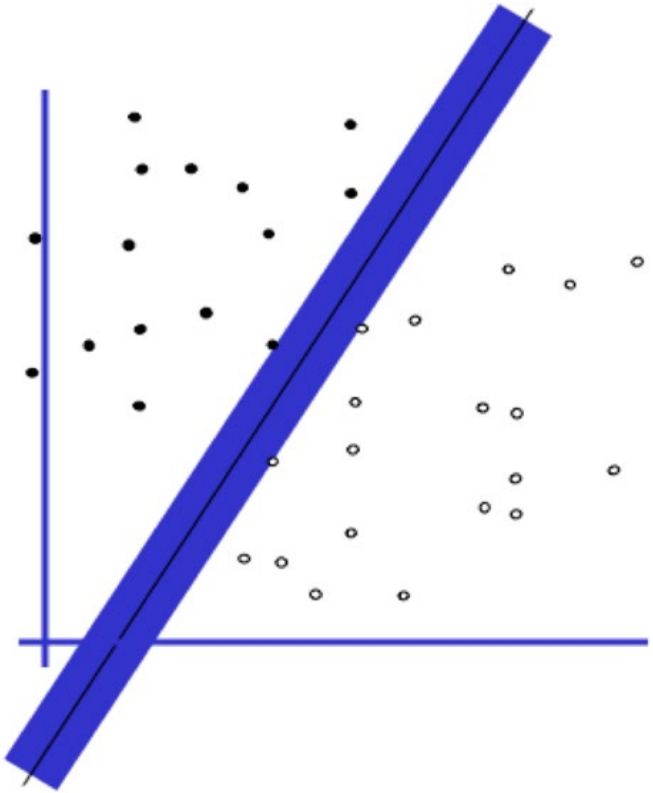
- **Theorem:** The class of optimal linear separators has VC dimension *h*, bound from above as

$$h \leq \min\left\{\left\lceil \frac{D^2}{\rho^2} \right\rceil, m_0\right\} + 1$$

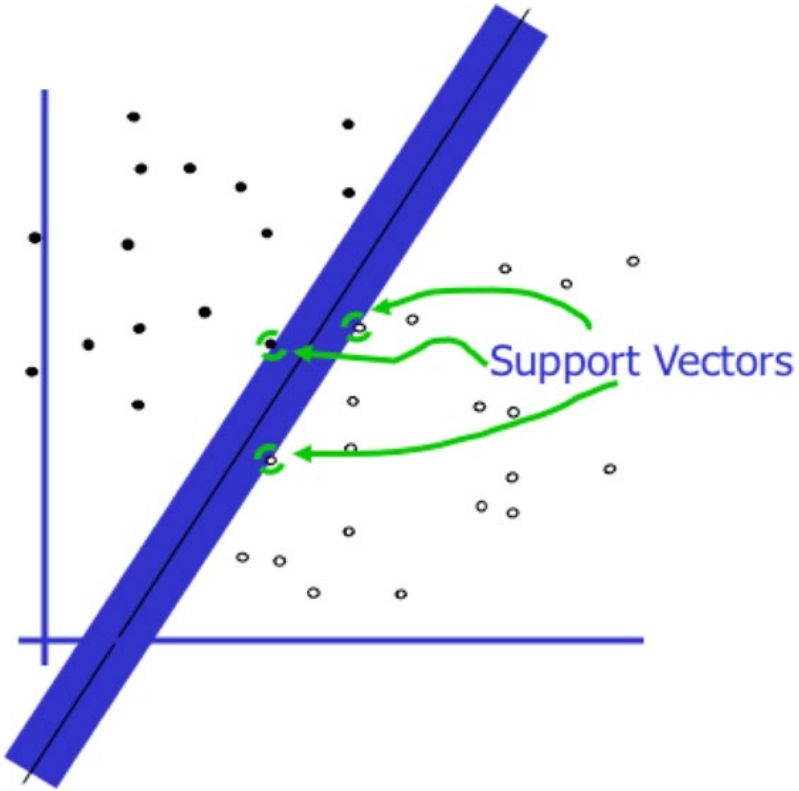
where ρ is the margin, D is the diameter of the smallest sphere that can enclose all of the training examples, and m_0 is the dimensionality of the input space.

- **Interpretation:** The margin corresponds to the width of the band (or hyper-rectangle) and is the widest the band can be without including one of the data points.

Margin



Support Vectors



Finding the Support Vectors

- Recall a linear discriminant can be represented as $y = \mathbf{w}^\top \mathbf{x} + b$
- Assuming two-class classification, we can use this as

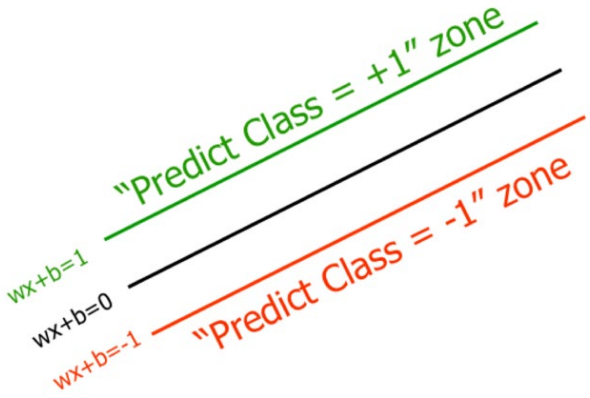
$$\text{class}(\mathbf{x}) = \begin{cases} +1 & \mathbf{w}^\top \mathbf{x} + b \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- Rewrite decision rule as

$$\text{class}(\mathbf{x}) = \begin{cases} +1 & \mathbf{w}^\top \mathbf{x} + b \geq +1 \\ -1 & \mathbf{w}^\top \mathbf{x} + b \leq -1 \end{cases}$$

- Margin width

$$\rho = \frac{2}{\sqrt{\mathbf{w}^\top \mathbf{W}}}$$



Maximizing the Margin

- Objective function

$$\max \rho = \max \left\{ 2/\sqrt{\mathbf{w}^\top \mathbf{w}} \right\}$$

- Alternatively

$$\min \left\{ \mathbf{w}^\top \mathbf{w} \right\}$$

- This is a "quadratic programming" problem with $k = 1, \ldots, n$ constraints.

$$\text{If} \quad y_k = +1 \ \text{ then} \quad \mathbf{w}^\top \mathbf{x} + b \geq +1$$

$$\text{If} \quad y_k = -1 \ \text{ then} \quad \mathbf{w}^\top \mathbf{x} + b \leq -1$$

A Limitation

- We are working with a simple linear discriminant.
- This means we are assuming the data is "linearly separable."
- What if it isn't?
- Two perspectives:
  - Separability breaks down because of noise.
  - Separability breaks down because classes really aren't linearly separable.

Handling Noise

There are three possible approaches to handling noise.

1. Minimize $\mathbf{w}^\top \mathbf{w}$ and error simultaneously: A multi-objective optimization problem.
2. Minimize $\mathbf{w}^\top \mathbf{w} + C\left(\#\,\text{error}_{\,\text{train}}\right)$: Second term is an integer.
3. Minimize $\mathbf{w}^\top \mathbf{w} + C\left(\text{dist}\left(\mathbf{x}_{\text{err}},\ \text{plane}\right)\right)$: A continuous objective function.

The third method has the nice property that the objectives don't compete directly and can be solved via quadratic programming.

Soft Margin

- Minimizing distance of erroneous points to correct side leads to the ”soft margin.”
- We can then solve the optimization problem by introducing “slack variables.”
- Denote the slack variable for example $\mathbf{x}_k$ as ξ_k
- New objective function:

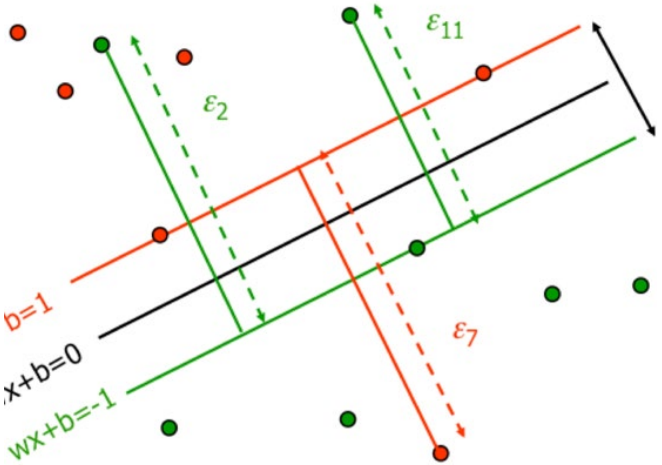
$$\text{minimize } \frac{1}{2}\mathbf{w}^\top\mathbf{w} + C\sum_{k=1}^{n}\xi_k$$

subject to

$$\mathbf{w}^\top\mathbf{x} + b \geq 1 - \xi_k \, (y_k = +1)$$

$$\mathbf{w}^\top\mathbf{x} + b \geq -1 + \xi_k \, (y_k = -1)$$

$$\forall k \xi_k \geq 0$$



Lagrangian Method

- Improve efficiency (solving by gradient descent)
- Identify support vectors more easily

$$\text{maximize} \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k=1}^{n} \sum_{\ell=1}^{n} \alpha_k \alpha_\ell Q_{k\ell}$$

where

$$Q_{k\ell} = y_k y_\ell \left(\mathbf{x}_k^\top \mathbf{x}_\ell \right)$$

- The corresponding constraints become

$$\forall k 0 \leq \alpha_k \leq C$$

$$\sum_{k=1}^{n} \alpha_k y_k = 0$$

Lagrange Multipliers

- We note that the support vectors correspond to the data points where $\alpha_k > 0$
- We can recover the weights we wish to learn.
- First, let $K = \underset{k}{\operatorname{argmax}}\, \alpha_k$
- Then

$$b = y_K \left(1 - \xi_k\right) - \mathbf{x}_K^\top \mathbf{w}$$

and

$$\mathbf{w} = \sum_{k=1}^{n} \alpha_k y_k \mathbf{x}_k$$

- Finally

$$\text{class}\left(\mathbf{x}'\right) = \text{sgn}\left(\mathbf{w}^\top \mathbf{x} - b\right)$$

Support Vector Regression (ϵ-SVR)

- Let $\mathcal{D} = \left\{(\mathbf{x}_i, y_i\right\}_{i=1}^{n} \subset \mathcal{X} \times \mathbb{R}$
- Find function of the form $f(x) = \mathbf{w}^\top \mathbf{x} + b$
- Minimize $\|\mathbf{w}\|^2 = \mathbf{w}^\top \mathbf{w}$
- New optimization problem.

$$\text{minimize } \frac{1}{2}\mathbf{w}^\top \mathbf{w}$$

subject to

$$y_i - \left(\mathbf{w}^\top \mathbf{x}_i + b\right) \leq \epsilon$$

$$\left(\mathbf{w}^\top \mathbf{x}_i + b\right) - y_i \leq \epsilon$$

ϵ-SVR with Soft Margin

Minimize

$$\frac{1}{2}\mathbf{w}^\top\mathbf{w} + C\sum_{k=1}^{n}(\xi_i + \xi_i^*)$$

subject to

$$y_i - \left(\mathbf{w}^\top\mathbf{x} + b\right) \leq \epsilon + \xi_i$$
$$\left(\mathbf{w}^\top\mathbf{x} + b\right) - y_i \leq \epsilon + \xi_i^*$$
$$\xi_i, \xi_i^* \geq 0$$

Slack cost

$$|\xi|_\epsilon = \begin{cases} 0 \text{ if } |\xi| \leq \epsilon \\ |\xi| - \epsilon \text{ otherwise} \end{cases}$$