

# Scalable Decision Trees and Ensembles

Tahsin Khan

COM6012 Scalable Machine Learning



# Contents

- What is a decision tree
  - Advantages
- Review of decision trees
- How do decision trees work in Spark?
  - Building trees and impurity measures in Spark
  - Split candidates
  - Decision tree classes in `spark.ml`
  - PLANET algorithm
- Review of ensemble methods
- Ensemble methods in PySpark
- Acknowledgement
- References

# What is a decision tree

- ❑ Decision tree is a machine learning algorithm that tries to build predictive models using the most informative features
- ❑ An informative feature is a descriptive feature whose values split the instances in the dataset into homogeneous sets with respect to the target value
- ❑ More information can be found in the MLAI module and resources provided at the end
- ❑ APIs used in `spark.ml` are very similar to the ones used in `scikit-learn`

# Advantages of decision trees

- ❑ Easy to interpret
- ❑ Can handle categorical features well
- ❑ Extends to the multiclass classification setting
- ❑ Do not require feature scaling
- ❑ Able to capture non-linearities and feature interactions

# Contents

- What is a decision tree
  - Advantages
- **Review of decision trees**
- How do decision trees work in Spark?
  - Building trees and impurity measures in Spark
  - Split candidates
  - Decision tree classes in `spark.ml`
  - PLANET algorithm
- Review of ensemble methods
- Ensemble methods in PySpark
- Acknowledgement
- References

# Nodes in a decision tree

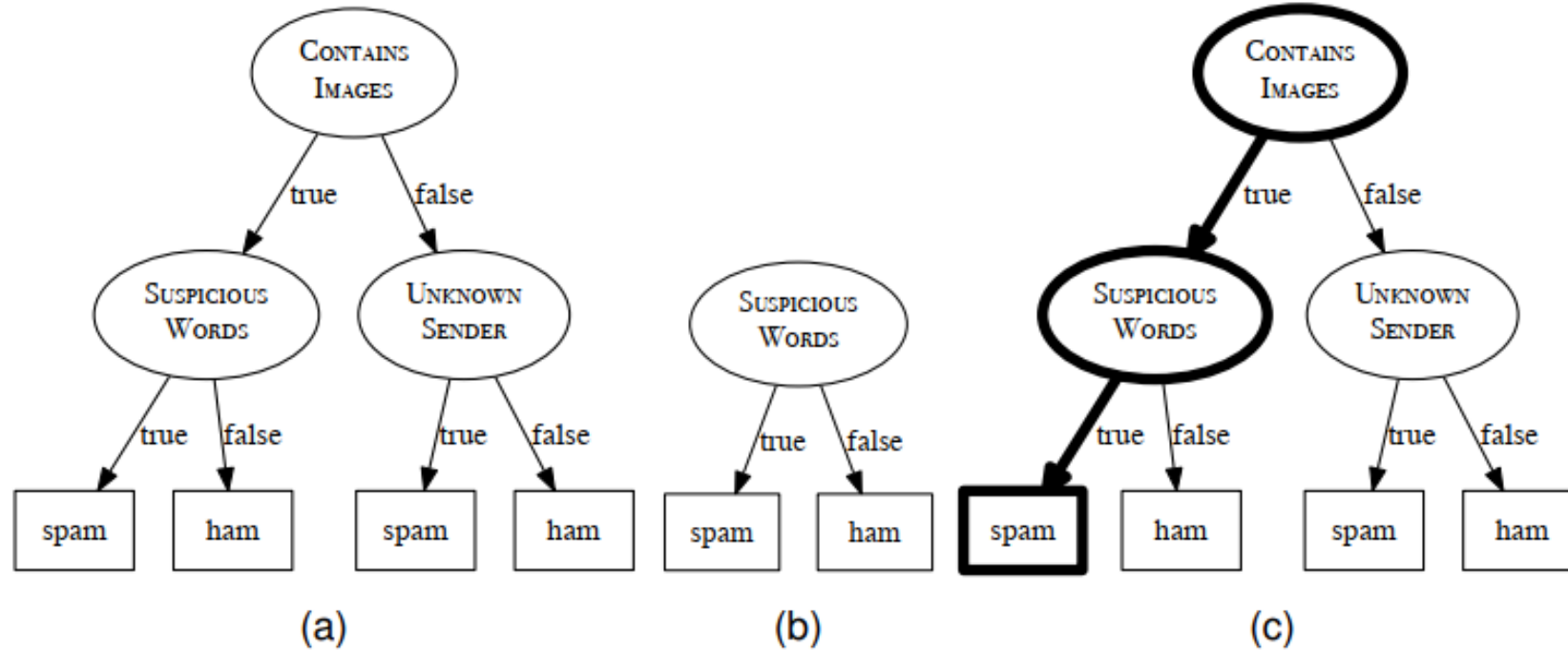
- ❑ A decision tree consists of:
  - A root node (or starting node)
  - Interior nodes
  - Leaf nodes (or terminating nodes)
- ❑ Each of the non-leaf nodes (root and interior) in the tree specifies a test to be carried out on one of the query's descriptive features
- ❑ Each of the leaf nodes specifies a predicted classification or predicted regression value for the query

# Binary classification: spam prediction

An email spam prediction dataset.

ID	SUSPICIOUS WORDS	UNKNOWN SENDER	CONTAINS IMAGES	CLASS
376	true	false	true	spam
489	true	true	false	spam
541	true	true	false	spam
693	false	true	true	ham
782	false	false	false	ham
976	false	false	false	ham

# Two decision trees and a query instance



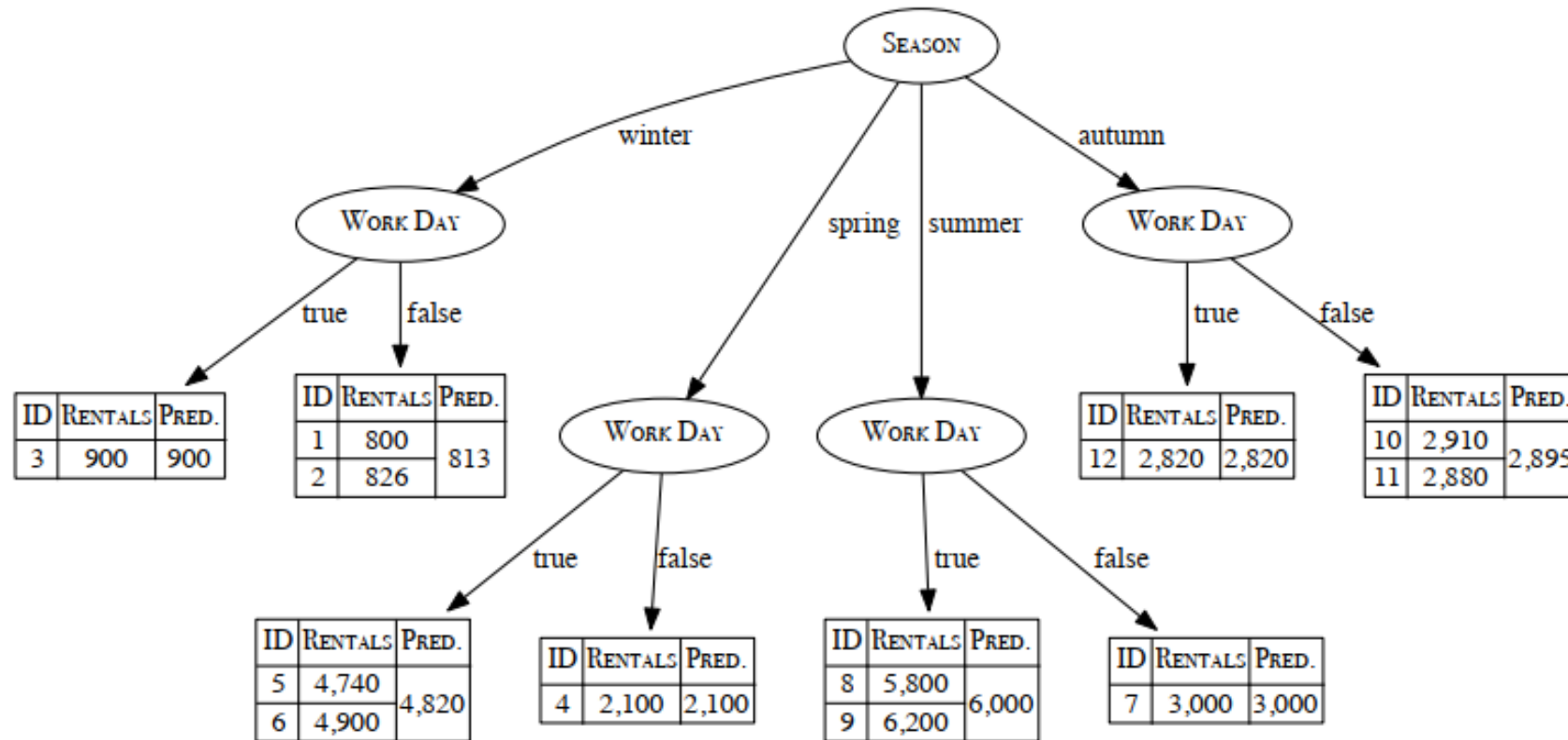
(a) and (b) show two decision trees that are consistent with the instances in the spam dataset. (c) shows the path taken through the tree shown in (a) to make a prediction for the query instance: SUSPICIOUS WORDS = 'true', UNKNOWN SENDER = 'true', CONTAINS IMAGES = 'true'.



# Regression: bike rentals per day

ID	SEASON	WORK DAY	RENTALS
1	winter	false	800
2	winter	false	826
3	winter	true	900
4	spring	false	2 100
5	spring	true	4 740
6	spring	true	4 900
7	summer	false	3 000
8	summer	true	5 800
9	summer	true	6 200
10	autumn	false	2 910
11	autumn	false	2 880
12	autumn	true	2 820

# Regression tree



The final decision tree induced from the dataset. This tree lists the instances that ended up at each leaf node and the prediction (PRED.) made by each leaf node.

# Contents

- What is a decision tree
  - Advantages
- Review of decision trees
- **How do decision trees work in Spark?**
  - Building trees and impurity measures in Spark
  - Split candidates
  - Decision tree classes in `spark.ml`
  - PLANET algorithm
- Review of ensemble methods
- Ensemble methods in PySpark
- Acknowledgement
- References

# Binary decision trees

- ❑ Apache Spark only builds binary decision trees
  - Each node can only have two branches
- ❑ Binary partitions at each node are done recursively
- ❑ Each partition is chosen greedily by selecting the best split from a set of possible splits

# Best split

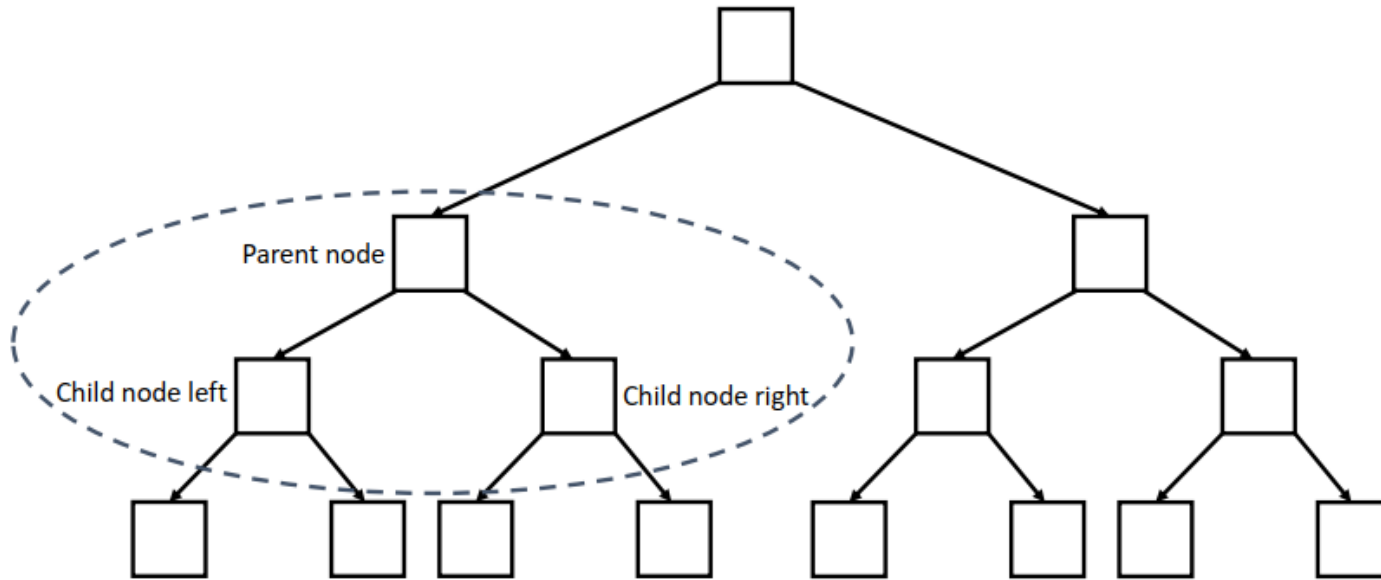
- ❑ The best split maximizes the information gain at a tree node
- ❑ The split chosen at each tree node is chosen from the set,

$$\arg \max_s IG(D, s)$$

Here,  $IG(D, s)$  is the information gain when split  $s$  is applied to dataset  $D$

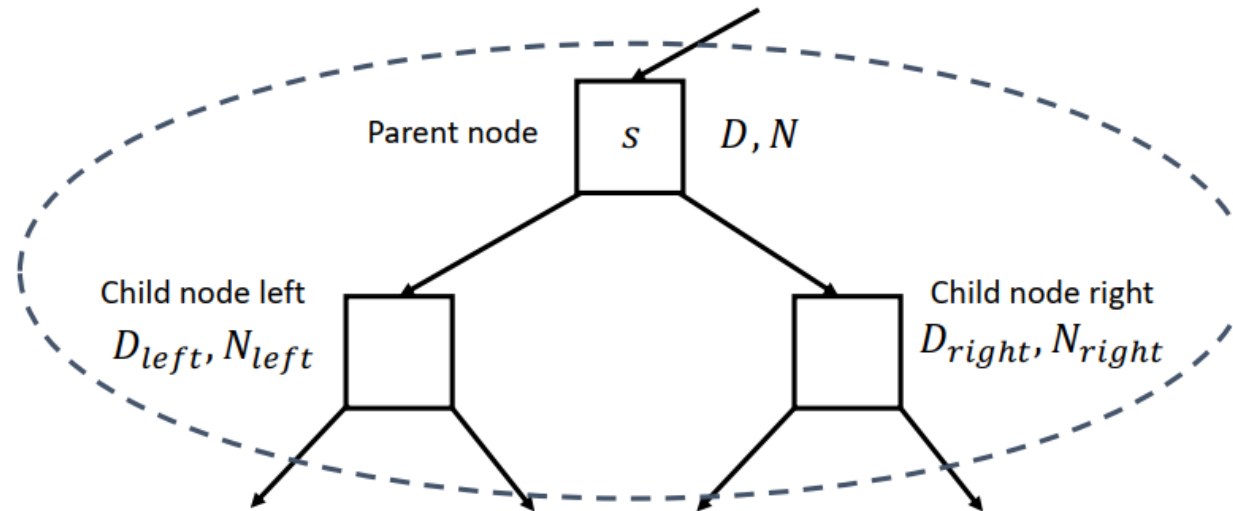
# Information gain

- The information gain is the difference between the parent node impurity and the weighted sum of the two child node impurities



# Information gain

- Assuming, that a split  $s$  partitions the dataset  $D$  of size  $N$  into two datasets  $D_{\text{left}}$  and  $D_{\text{right}}$  of size  $N_{\text{left}}$  and  $N_{\text{right}}$



The information gain is computed as,

$$IG(D, s) = \text{Impurity}(D) - \left( \frac{N_{\text{left}}}{N} \text{Impurity}(D_{\text{left}}) + \frac{N_{\text{right}}}{N} \text{Impurity}(D_{\text{right}}) \right)$$

# Node impurity

- ❑ Recall, node impurity is a measure of how homogeneous are the labels at a node
- ❑ The table below shows the impurity measures currently implemented in Spark

Impurity	Task	Formula	Description
Gini impurity	Classification	$\sum_{i=1}^C f_i(1 - f_i)$	$f_i$ is the frequency of label $i$ at a node and $C$ is the number of unique labels.
Entropy	Classification	$\sum_{i=1}^C -f_i \log(f_i)$	$f_i$ is the frequency of label $i$ at a node and $C$ is the number of unique labels.
Variance	Regression	$\frac{1}{N} \sum_{i=1}^N (y_i - \mu)^2$	$y_i$ is label for an instance, $N$ is the number of instances and $\mu$ is the mean given by $\frac{1}{N} \sum_{i=1}^N y_i$ .

\*frequency refers to probability



# Contents

- What is a decision tree
  - Advantages
- Review of decision trees
- **How do decision trees work in Spark?**
  - Building trees and impurity measures in Spark
  - **Split candidates**
  - Decision tree classes in `spark.ml`
  - PLANET algorithm
- Review of ensemble methods
- Ensemble methods in PySpark
- Acknowledgement
- References

# How are the split candidates chosen?

- We saw that Spark only implements binary decision trees

- This means splits are always of the kind:

**if** feature  $i \leq \text{threshold}$  **then**

    Move to the left/right branch

**else**

    Move to the right/left branch

**end if**

- Computation of `threshold` values depend on whether the features are continuous or categorical

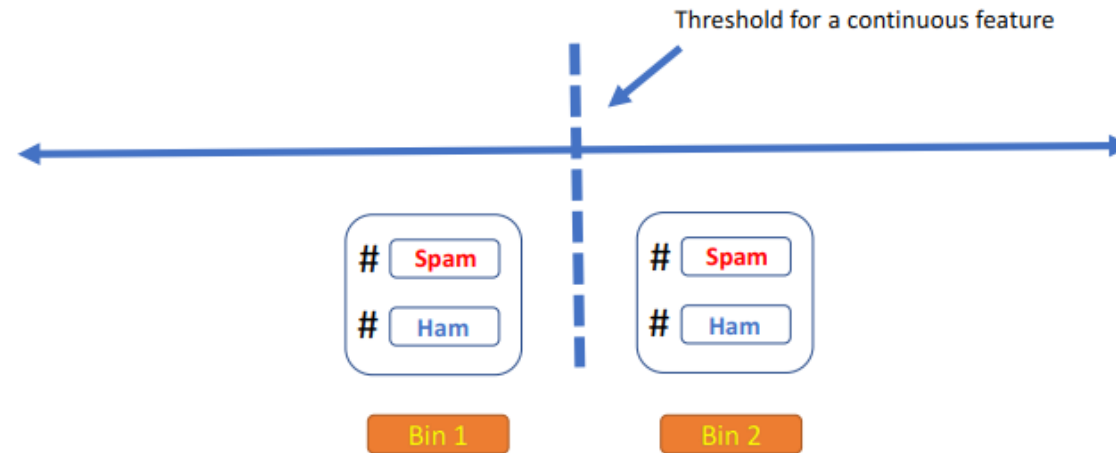
# Handling continuous features

- ❑ When working with small datasets, we can either:
  - Directly use the values of the feature as the thresholds
  - Sort the values of the feature and compute the corresponding thresholds from the sorted values
  
- ❑ However, these alternatives are not possible for use for large distributed datasets
  - It is not feasible to use the values of the feature as the thresholds
  - Sorting is an option, but it is expensive

# Handling continuous features

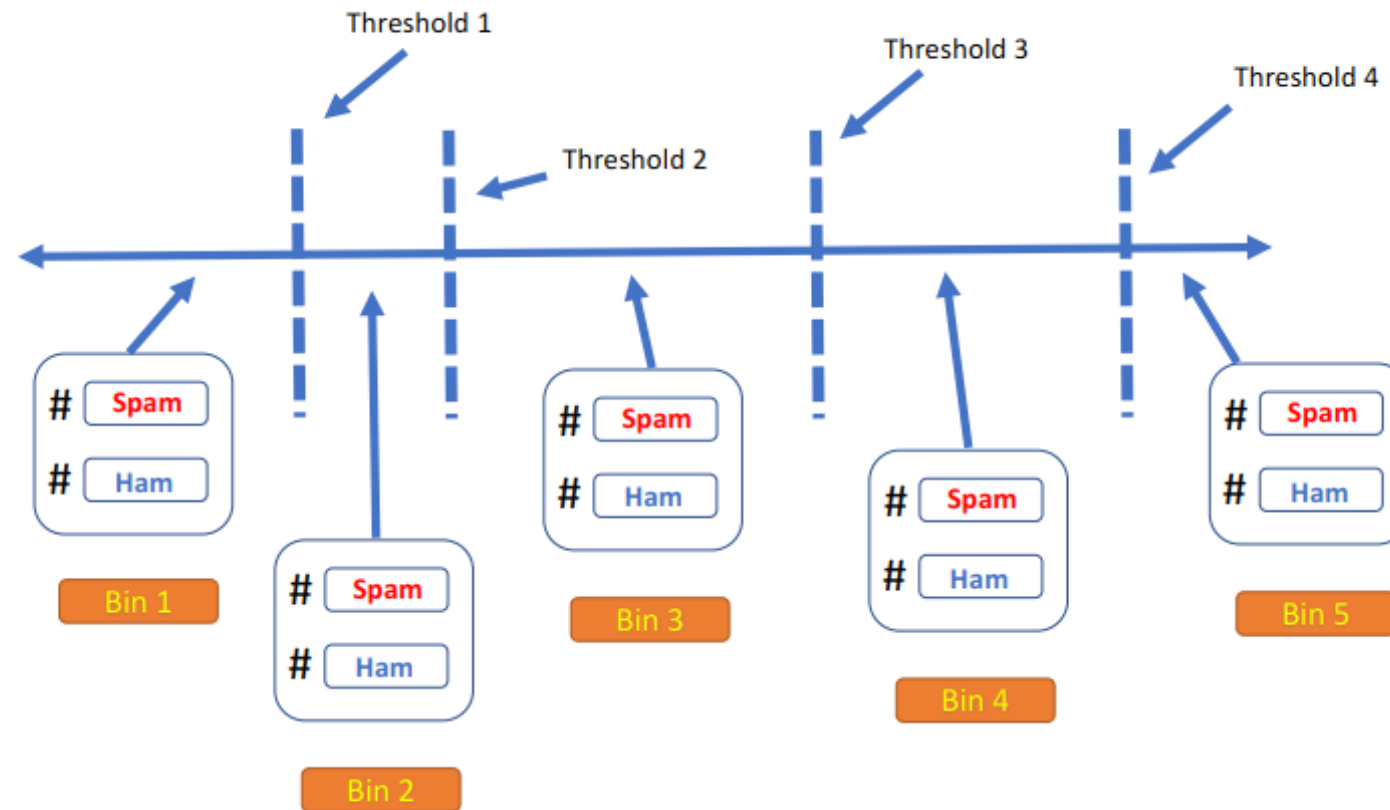
- ❑ Spark's implementation computes an approximate set of split candidates by performing a quantile calculation over a sample fraction of the data
- ❑ The ordered splits create “bins” and the maximum number of such bins can be specified using the `maxBins` parameter

# Bins for continuous features



Bins in continuous features

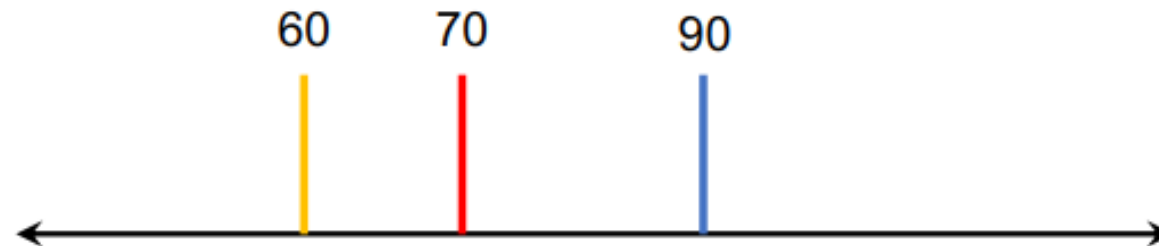
# Bins for continuous features



Bins in continuous features

# Why bins? Naïve approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



# Why bins? Naïve approach

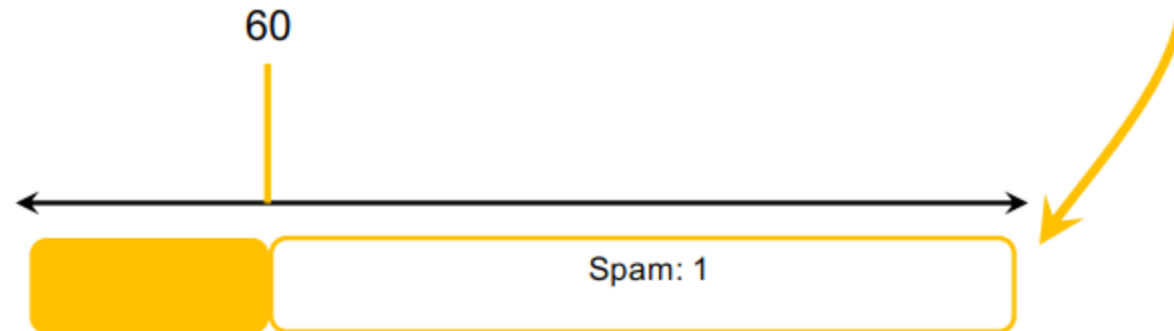
ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham





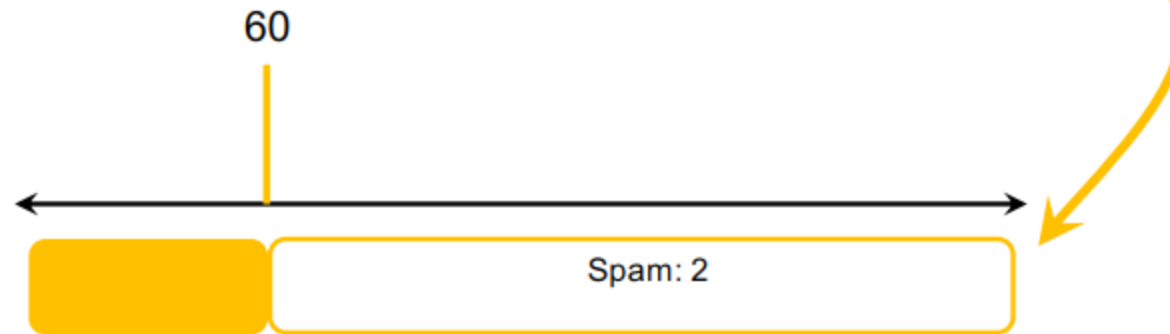
# Why bins? Naïve approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



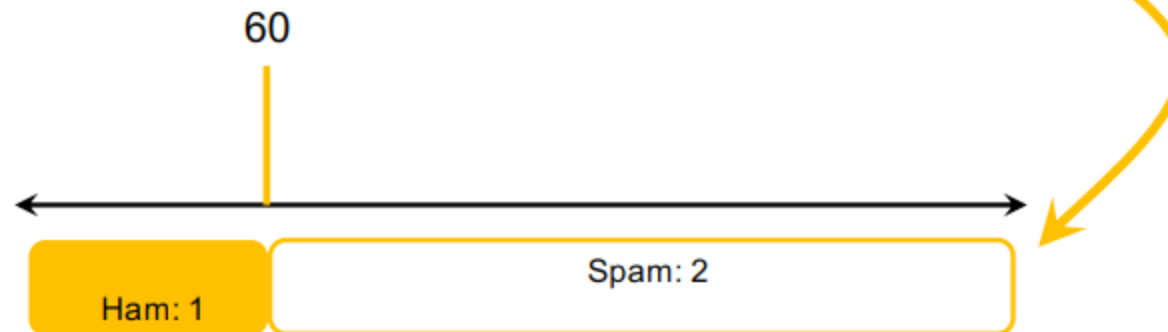
# Why bins? Naïve approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



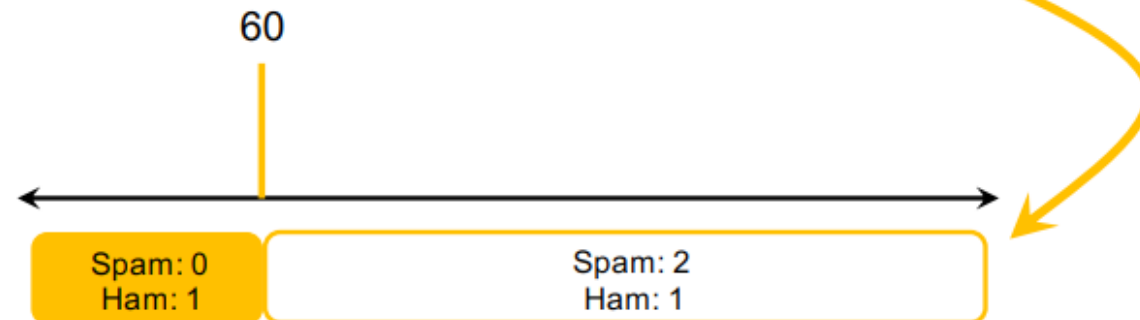
# Why bins? Naïve approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



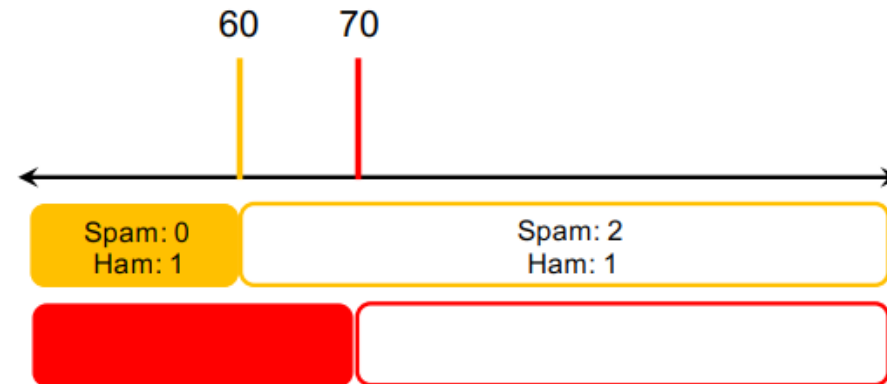
# Why bins? Naïve approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



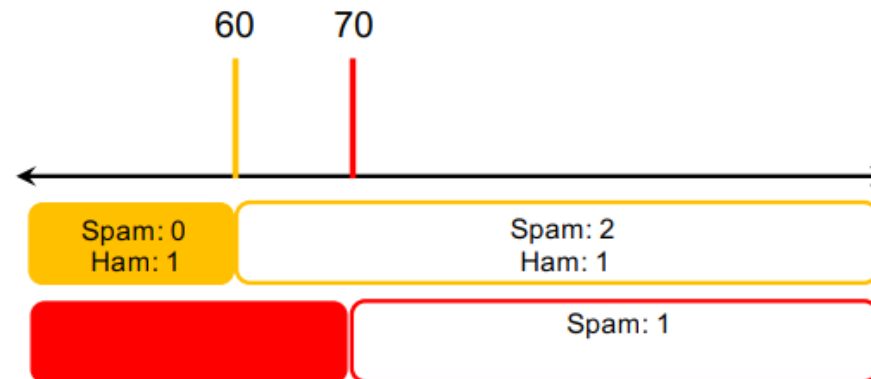
# Why bins? Naïve approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



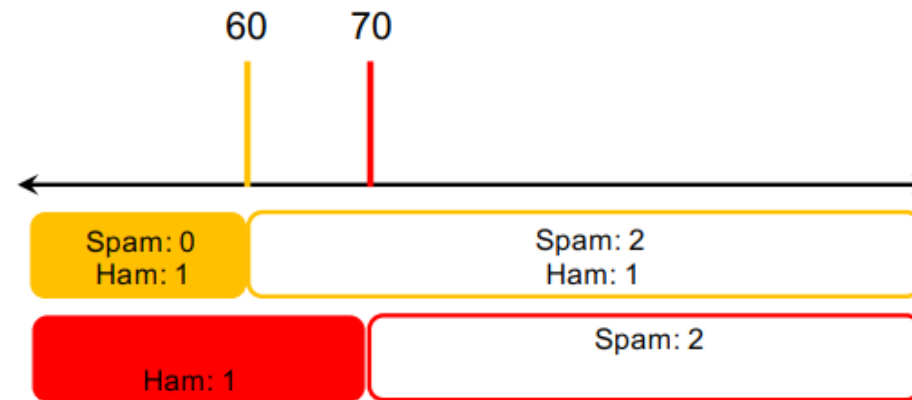
# Why bins? Naïve approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



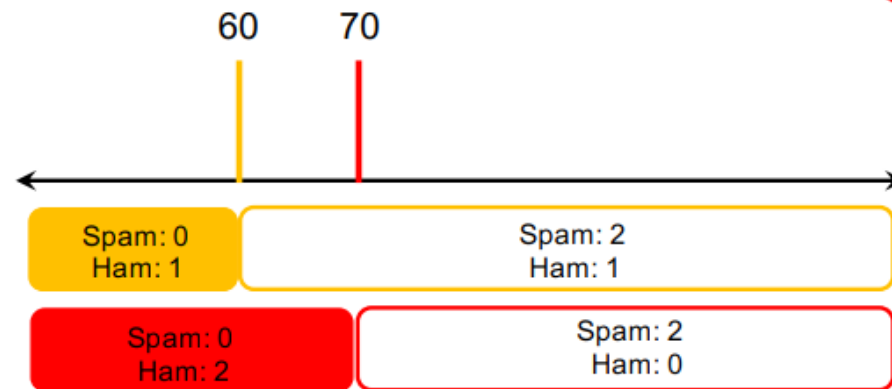
# Why bins? Naïve approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



# Why bins? Naïve approach

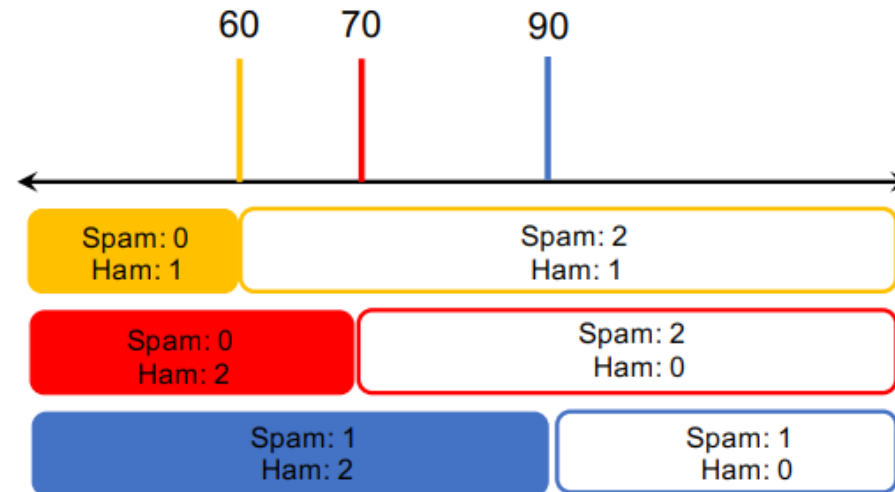
ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham





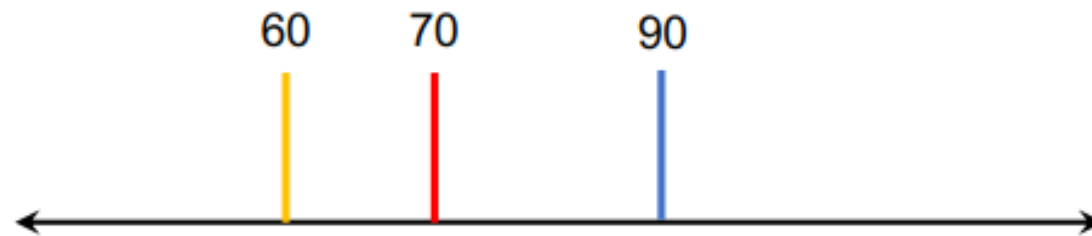
# Why bins? Naïve approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



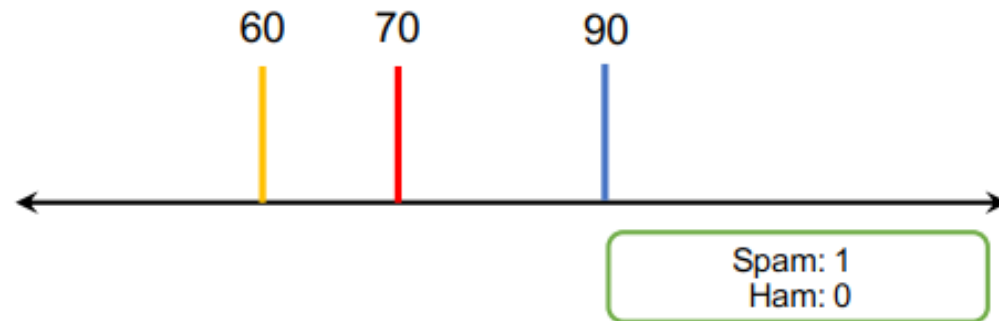
# Why bins? Binary search

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



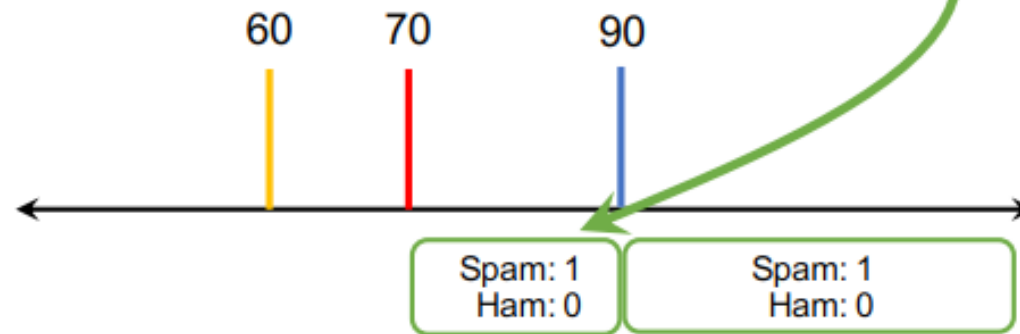
# Why bins? Binary search

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham

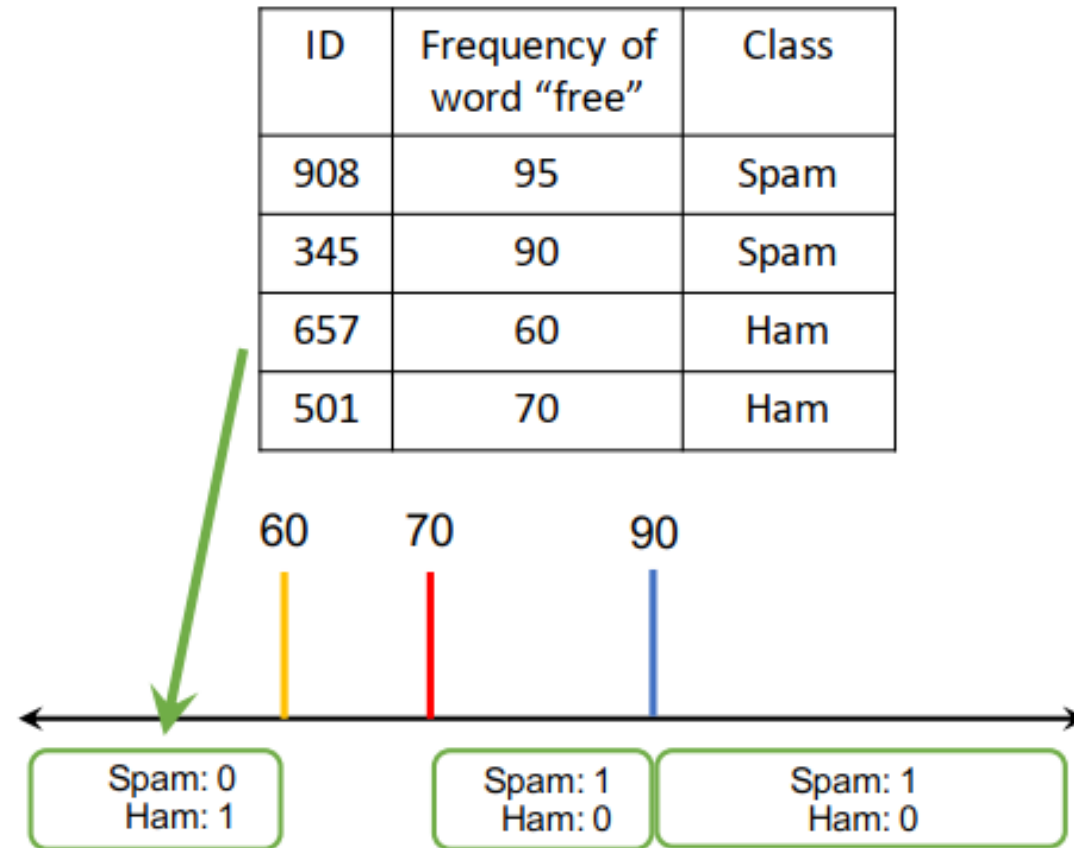


# Why bins? Binary search

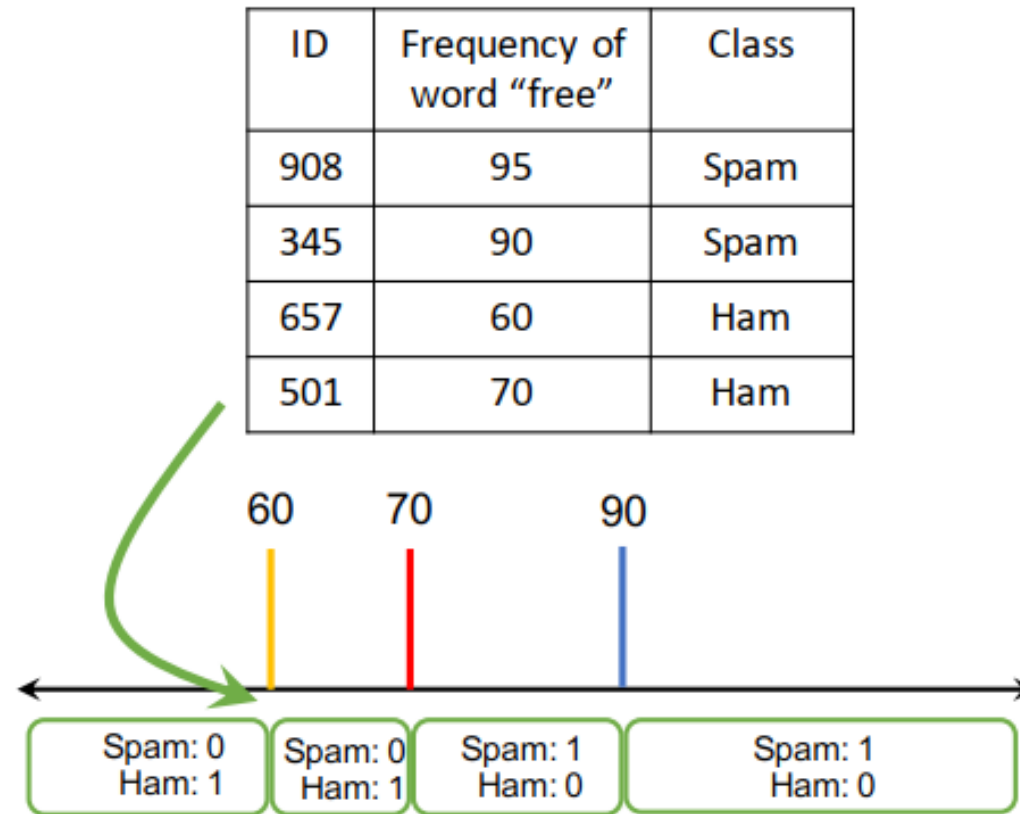
ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



# Why bins? Binary search

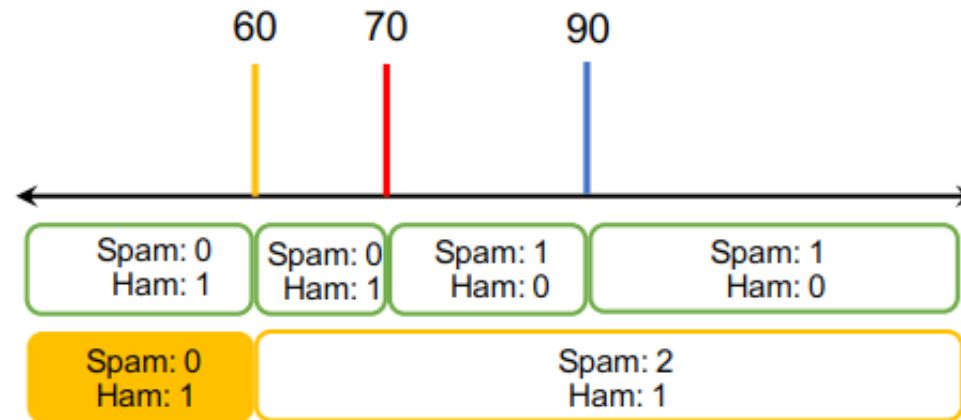


# Why bins? Binary search



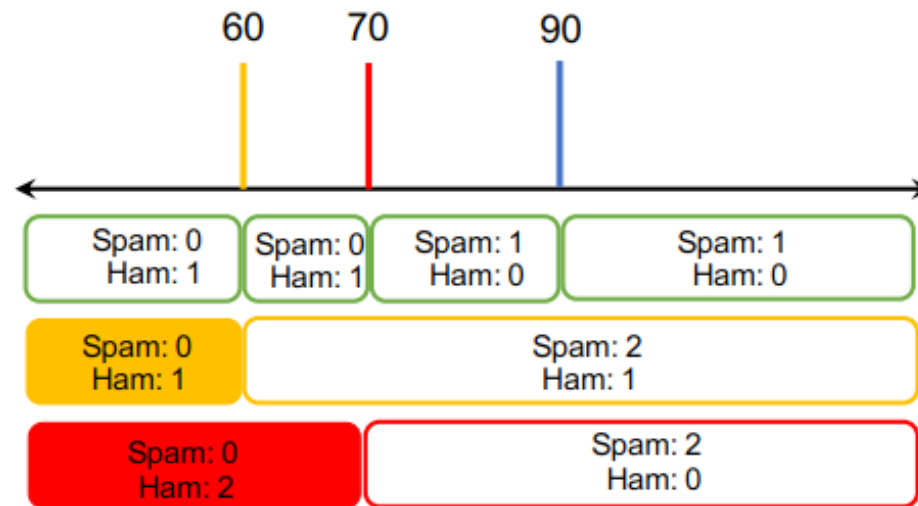
# Why bins? Binary search

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



# Why bins? Binary search

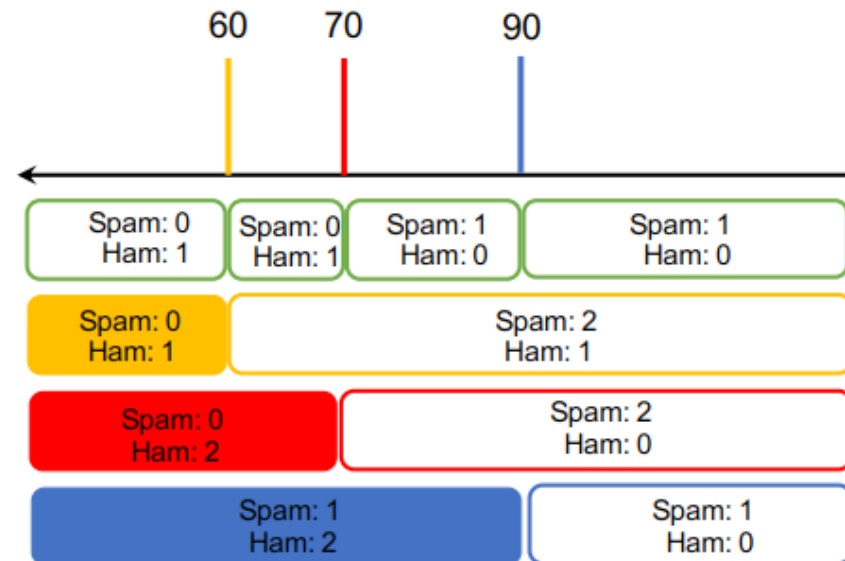
ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham





# Why bins? Binary search

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



# What is the difference?

- ❑ Assume there are  $m$  thresholds
- ❑ In the naïve approach, for each observation, we need to perform  $m$  operations
- ❑ When using binning, it is possible to use binary search for each observation to fill out the bins
- ❑ Binary search takes  $\log(m)$  operations

# Categorical features

- Say we have a categorical feature that can take  $M$  unordered values
- We need to make a binary partition based on this categorical feature
- It can be shown there are  $2^{M-1} - 1$  possible partitions of the  $M$  values into two groups

# Categorical features

- ❑ As an example, consider the feature `weather` that takes values `[spring, summer, autumn, winter]`
- ❑ We would then have  $2^{M-1} - 1 = 2^{4-1} - 1 = 7$  possible partitions of two groups

Option	Group 1	Group 2
1	spring	summer, autumn, winter
2	summer	spring, autumn, winter
3	autumn	summer, spring, winter
4	winter	summer, autumn, spring
5	spring, summer	autumn, winter
6	spring, autumn	summer, winter
7	spring, winter	summer, autumn

- ❑ For large  $M$ , computation becomes prohibitive
- ❑ There are heuristics to optimally reduce the number of partitions

# Categorical features: binary classification

- ❑ For binary classification, the number of partitions can be reduced to  $M - 1$  by order the categorical feature values by proportion falling in the outcome class 1
- ❑ For instance, lets consider the `weather` example and consider that the proportions contributed by the categorical values for outcome class 1 are as follows

Categorical value	Proportion
spring	0.2
summer	0.3
autumn	0.05
winter	0.45

# Categorical features: binary classification

□ By ordering the categorical values, we get

- `[autumn, spring, summer, winter]`
- Number of partitions =  $M - 1 = 3$

Option	Group 1	Group 2
1	<code>autumn</code>	<code>spring, summer, winter</code>
2	<code>autumn, spring</code>	<code>summer, winter</code>
3	<code>autumn, spring, summer</code>	<code>winter</code>

□ Essentially, we are transforming an unordered feature into an ordered feature

# Categorical features: regression

- ❑ One can show this gives the optimal split, in terms of cross-entropy or Gini index, among all possible  $2^M - 1$  splits
- ❑ This result also holds for a regression problem when using the square error loss
- ❑ In this case, the categories are ordered by increasing mean of the outcome
- ❑ More details can be found in book *The Elements of Statistical Learning*, 2<sup>nd</sup> edition, section 9.2.4

# Categorical features: multi-class classification

- ❑ In Spark, for multiclass classification, all  $2^{M-1} - 1$  possible split are used whenever possible
- ❑ When  $2^{M-1} - 1$  is greater than the `maxBins` parameter, Spark uses a heuristic method similar to the method used for binary classification and regression
- ❑ The  $M$  categorical feature values are ordered by impurity and the resulting  $M - 1$  split candidates are considered



# Contents

- What is a decision tree
  - Advantages
- Review of decision trees
- **How do decision trees work in Spark?**
  - Building trees and impurity measures in Spark
  - Split candidates
  - **Decision tree classes in `spark.ml`**
  - PLANET algorithm
- Review of ensemble methods
- Ensemble methods in PySpark
- Acknowledgement
- References

# APIs for decision trees

- ❑ Class for Classification: `DecisionTreeClassifier`

- ❑ Class for Regression: `DecisionTreeRegressor`

# The DecisionTreeClassifier class

## □ In scikit-learn

```
class sklearn.tree. DecisionTreeClassifier (criterion='gini', splitter='best',  
max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,  
max_features=None, random_state=None, max_leaf_nodes=None,  
min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)
```

## □ In PySpark

```
class pyspark.ml.classification. DecisionTreeClassifier (featuresCol='features',  
labelCol='label', predictionCol='prediction', probabilityCol='probability',  
rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32, minInstancesPerNode=1,  
minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10,  
impurity='gini', seed=None)
```

# The `DecisionTreeRegressor` class

## ❑ In scikit-learn

```
class sklearn.tree.DecisionTreeRegressor(*, criterion='mse', splitter='best', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None,
random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None,
ccp_alpha=0.0)
```

[\[source\]](#)

## ❑ In PySpark

```
class pyspark.ml.regression.DecisionTreeRegressor(featuresCol='features',
labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1,
minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10,
impurity='variance', seed=None, varianceCol=None, weightCol=None, leafCol="",
minWeightFractionPerNode=0.0)
```

[\[source\]](#)

# Parameters to adjust

- ❑ `maxDepth`: Maximum depth of a tree
- ❑ `maxBins`: Maximum number of bins for discretizing continuous features. It must be  $\geq 2$  and  $\geq$  number of categories for any categorical feature
- ❑ `impurity`: Criterion used for information gain calculation.
  - Supported options:
    - `entropy` or `gini` for classification
    - `variance` for regression
- ❑ Several other parameters will be explored in the lab session

# Contents

- What is a decision tree
  - Advantages
- Review of decision trees
- **How do decision trees work in Spark?**
  - Building trees and impurity measures in Spark
  - Split candidates
  - Decision tree classes in `spark.ml`
  - **PLANET algorithm**
- Review of ensemble methods
- Ensemble methods in PySpark
- Acknowledgement
- References

# PLANET: horizontal partitioning

- PLANET is the standard algorithm to train a decision in a distributed dataset or horizontal partitioning

- Let  $X \in \mathbb{R}^{n \times p}$  be the input matrix

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix}$$

- Here,  $\mathbf{x}_k \in \mathbb{R}^{p \times 1}$ , meaning there are  $p$  features
- Horizontal partitioning refers to the fact that each worker will receive a subset of the  $n$  instances or row vectors

# Setup

- Assume there are  $B$  potential thresholds for each of the  $p$  features that were considered
- Let us define  $S$  as the set of cardinality  $p \times B$  that contains all the split candidates
- Assume there are  $k$  workers. Each worker  $j$  computes  $c$  sufficient statistics over a subset of the original data,  $\mathbf{X}_j$
- What are these sufficient statistics?
  - For classification: label counts (e.g.  $c = 4$  for binary classification, i.e., 2 per branch)
  - For regression: count, sum, and  $\text{sum}^2$  (so,  $c = 6$ , i.e., 3 per branch)
- Details can be found in the paper, **Yggdrasil: An Optimized System for Training Deep Decision Trees at Scale**



# Setup

- Each worker  $j$  then computes the sufficient statistics  $g_j(s)$  over  $\mathbf{X}_j$  for each  $s \in S$
- For a parent node  $l$  in the tree, the optimal split  $s^*$  is found by solving

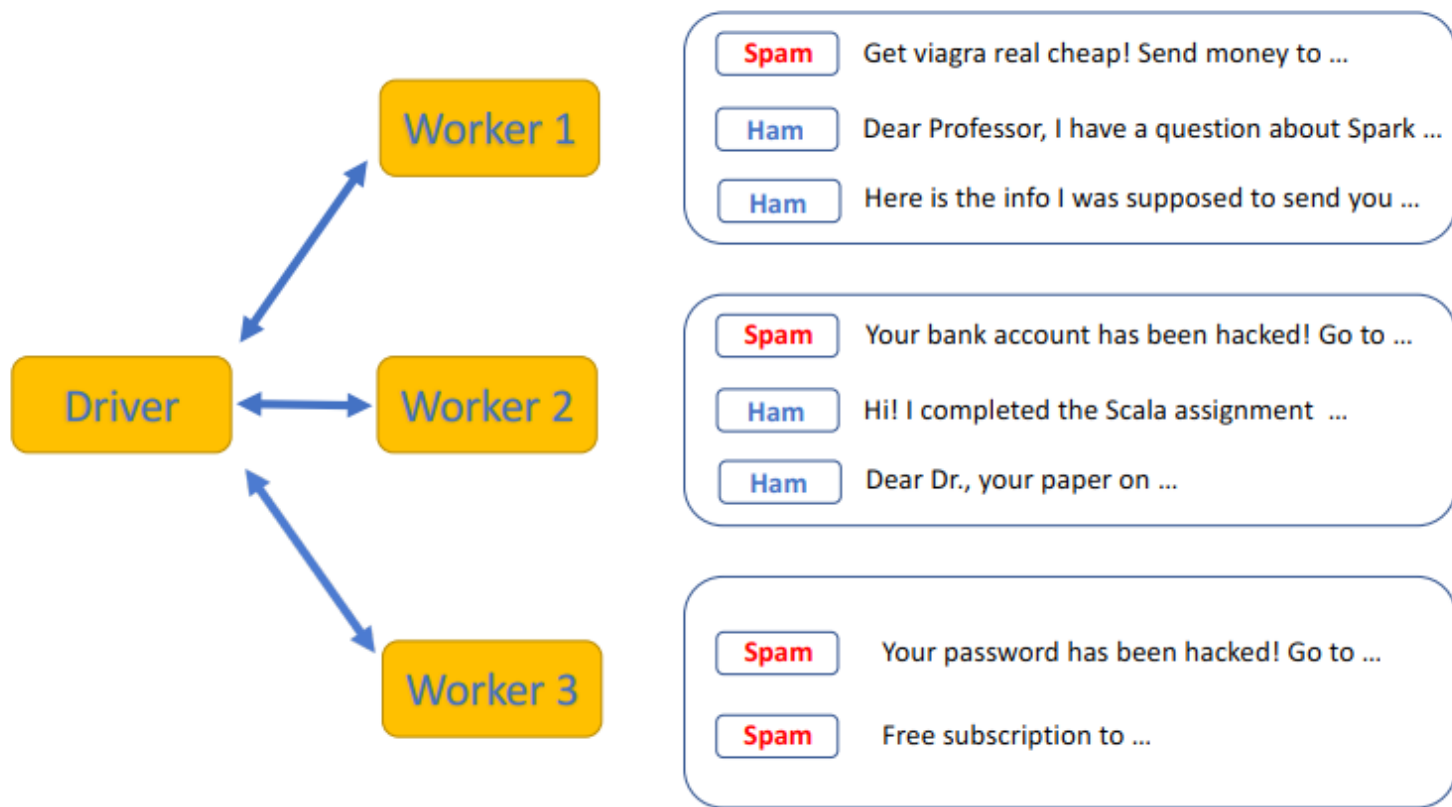
$$s^* = \arg \max_{s \in S} f\left(\sum_{j=1}^k g_j(s)\right)$$

- Where,  $f: \mathbb{R}^c \rightarrow \mathbb{R}$  computes the information gains

# Algorithm to compute $s^*$ in a distributed fashion

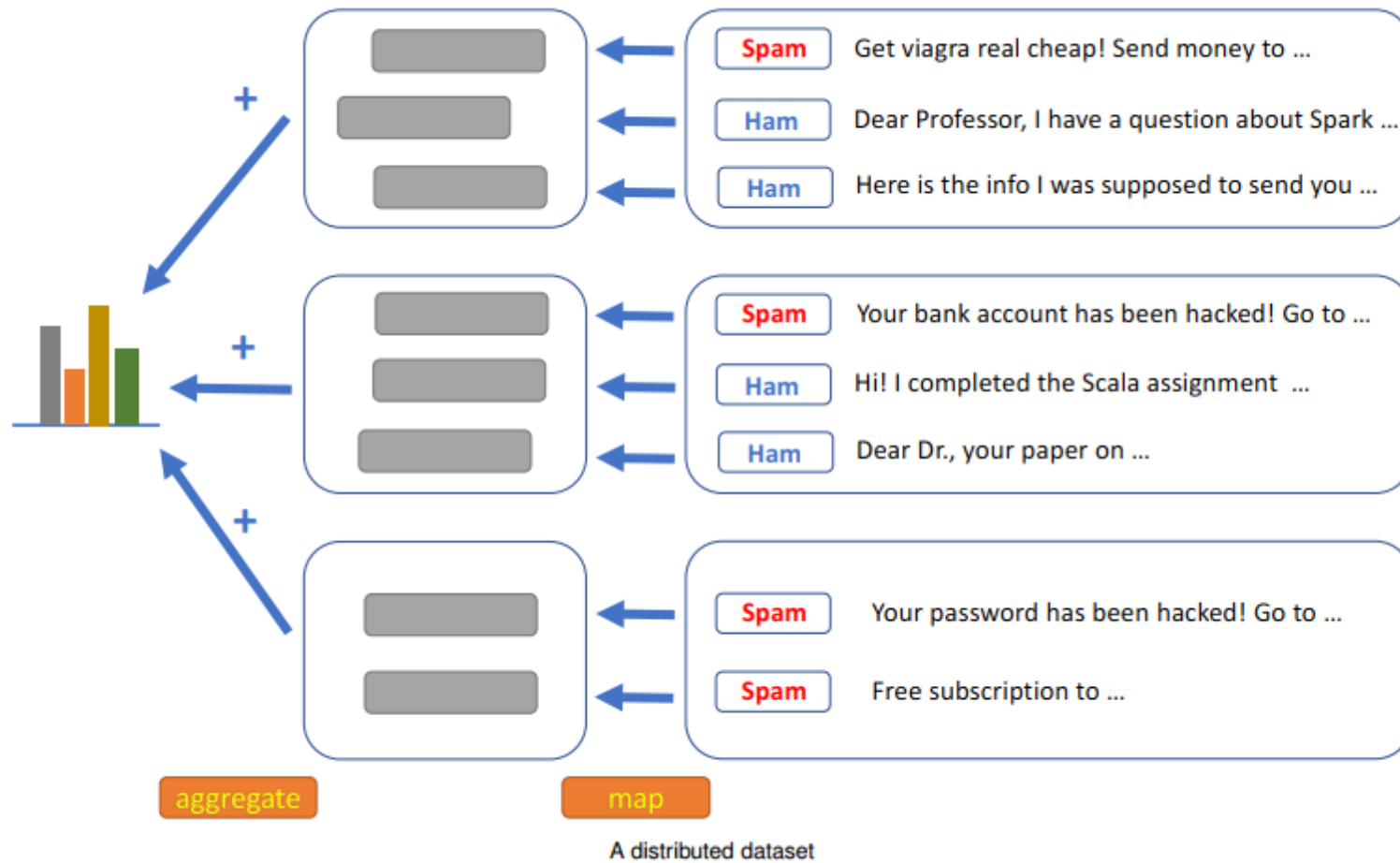
- We start by assuming the depth of the tree is fixed to  $D$
  
- At iteration  $t$ ,
  - the algorithm computes the optimal splits for all the nodes on the level  $t$  of the tree
  - there is a single round trip of communication between the master and the workers
  
- Each tree node  $i$  is split as follows
  1. The  $j$ -th worker locally computes sufficient statistics  $g_j(s)$  for all  $s \in S$
  2. Each worker communicates all statistics  $g_j(s)$  to the master ( $Bp$  in total)
  3. The master computes the best split  $s^*$
  4. The master broadcasts  $s^*$  to the workers, who update their local states to keep track of which instances are assigned to which child nodes

# The distributed dataset

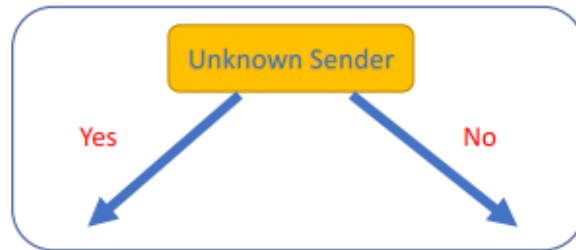


A distributed dataset

# The distributed dataset

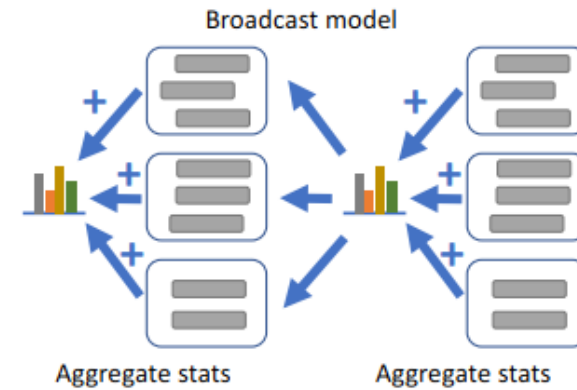
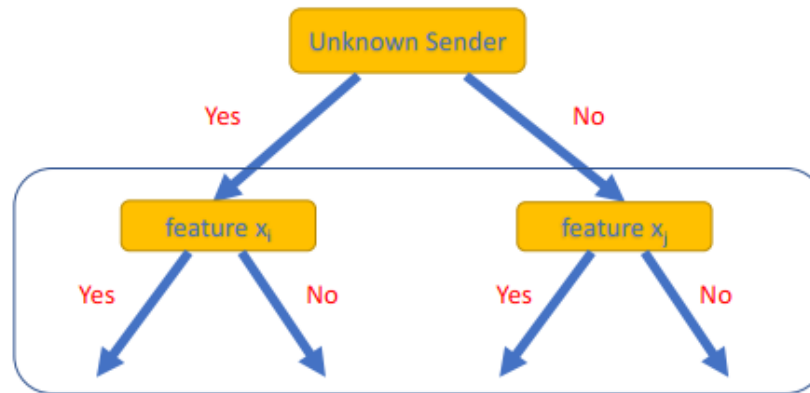


# Building a decision tree



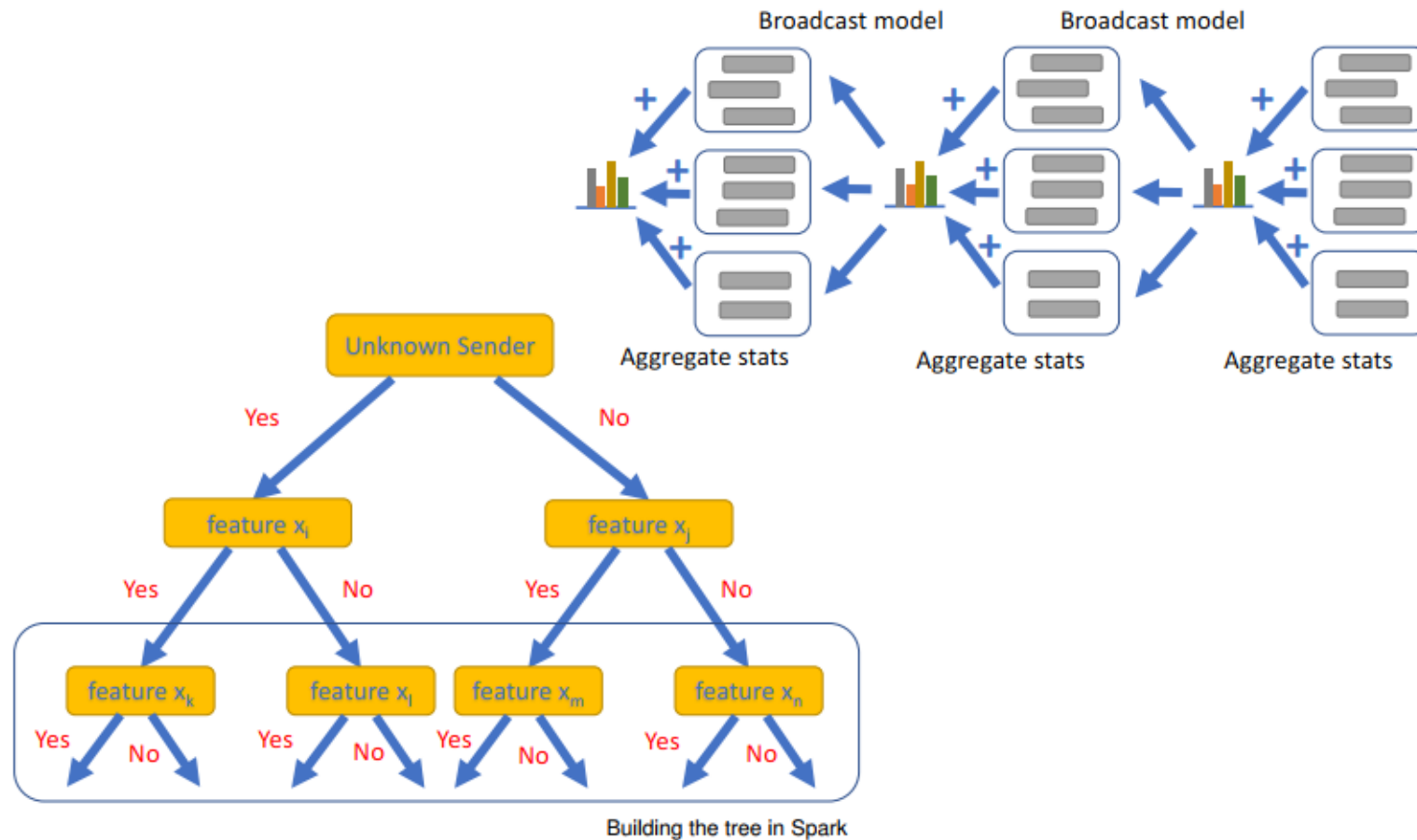
Building the tree in Spark

# Building a decision tree



Building the tree in Spark

# Building a decision tree



# Computational and communication complexity

- ❑ Computation is linear in  $n$ ,  $p$  and  $D$ , so it's easy to parallelize
- ❑ The issue is the communication overhead
- ❑ For each tree node, step 2 in the algorithm before, require communicating  $kBp$  tuples of size  $c$
- ❑ With  $2^D$  total nodes, the total communication is  $2^D kBpc$  floating point values
- ❑ Communication is then exponential in tree depth  $D$  and linear in thresholds  $B$



# Contents

- What is a decision tree
  - Advantages
- Review of decision trees
- How do decision trees work in Spark?
  - Building trees and impurity measures in Spark
  - Split candidates
  - Decision tree classes in `spark.ml`
  - PLANET algorithm
- **Review of ensemble methods**
- Ensemble methods in PySpark
- Acknowledgement
- References

# Bagging and boosting

- ❑ Rather than creating a single model, they generate a set of models and then make predictions by aggregating the outputs of these models
- ❑ A prediction model that is composed of a set of models is called a model ensemble
- ❑ In order for this approach to work, the models that are in the ensemble must be different from each other
- ❑ There are two standard approaches to creating ensembles:
  - Bagging
  - Boosting

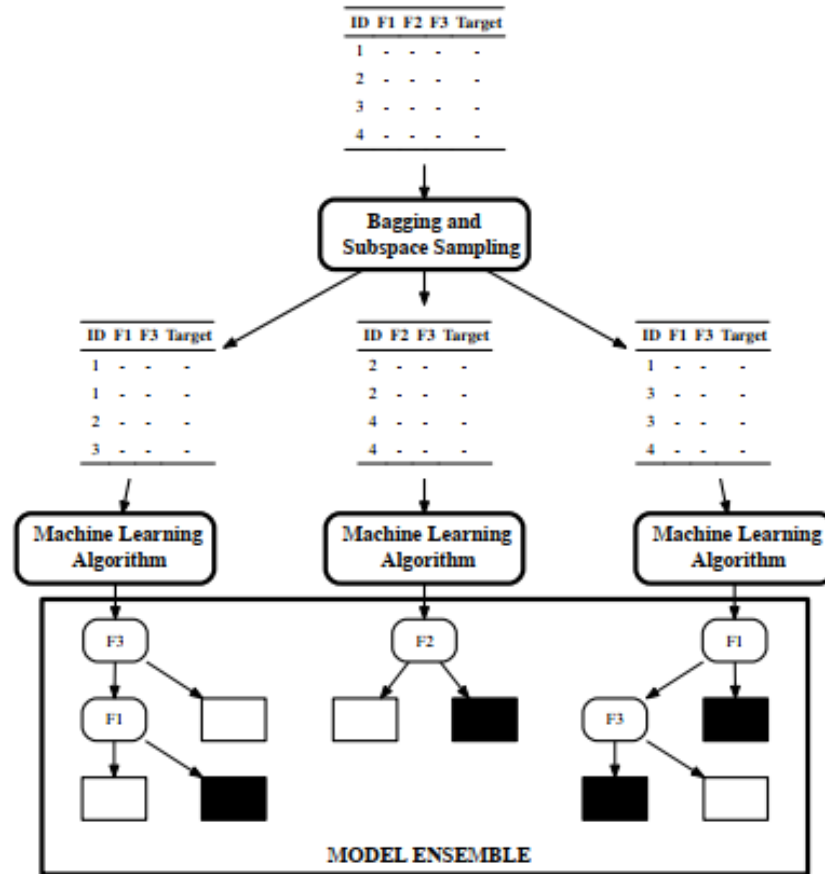
# Bagging: Definition

- ❑ In **bagging** (or **bootstrap aggregating**) each model in the ensemble is trained on a random sample of the dataset known as **bootstrap samples**
- ❑ Each random sample is the same size as the dataset and **sampling with replacement** is used
- ❑ Hence, every bootstrap sample will be missing some of the instance from the dataset. Consequently,
  - Each bootstrap sample will be different
  - Therefore, models trained on different bootstrap samples will also be different

# Bagging: Random forest

- When bagging is used with decision trees, each bootstrap sample only uses a randomly selected subset of the descriptive features in the dataset
  - This is known as **subspace sampling**
- The combination of bagging, subspace sampling and decision trees is known as a **random forest** model

# Example of using bagging



The process of creating a model ensemble using bagging and subspace sampling.

# Boosting: How does it work?

- ❑ Boosting works by iteratively creating models and adding them to the ensemble
- ❑ The iteration stops when a predefined number of models have been added
- ❑ When we use **boosting** each new model added to the ensemble is biased to pay more attention to instance that previous models misclassified
- ❑ This is done by incrementally adapting the dataset used to train the models. To do this, we use a **weighted dataset**

# Boosting: Weighted Dataset

- ❑ Each instance has an associated weight,  $w_i \geq 0$
- ❑ Initially set to  $\frac{1}{n}$  where  $n$  is the number of instances in the dataset
- ❑ After each model is added to the ensemble, it is tested on the **training data**
  - Weights of the instances that the model **accurately predicts** are **decreased**
  - Weights of the instances that the model **predicts incorrectly** are **increased**
- ❑ These weights are used as a distribution over which the dataset is sampled to create a **replicated training set**
  - Replication of an instance is proportional to its weight

# Algorithm

□ During each **training iteration**, the algorithm does the following:

1. Induces a model and calculates the total error,  $\epsilon$ , by summing the weights of the training instances for which the predictions made by the model were incorrect
2. Increases the weights for the instances that were misclassified by using the formula:

$$\mathbf{w}[i] \leftarrow \mathbf{w}[i] \times \left( \frac{1}{2 \times \epsilon} \right)$$

3. Decreases the weights for the instances correctly classified:

$$\mathbf{w}[i] \leftarrow \mathbf{w}[i] \times \left( \frac{1}{2 \times (1 - \epsilon)} \right)$$

4. Calculates a confidence factor,  $\alpha$ , for the model such that  $\alpha$  increases as  $\epsilon$  decreases

$$\alpha = \frac{1}{2} \times \log_e \left( \frac{1 - \epsilon}{\epsilon} \right)$$



# Predictions

- Once the set of models have been created, the ensemble makes **predictions** using a weighted aggregate of the predictions made by the individual models
- The weights used in this aggregation are simply the confidence factors associated with each model

# Contents

- What is a decision tree
  - Advantages
- Review of decision trees
- How do decision trees work in Spark?
  - Building trees and impurity measures in Spark
  - Split candidates
  - Decision tree classes in `spark.ml`
  - PLANET algorithm
- Review of ensemble methods
- Ensemble methods in PySpark
- Acknowledgement
- References

# Bagging and Boosting in PySpark

- ❑ Random forests can be implemented in PySpark
- ❑ The boosting model implemented in PySpark is the Gradient-Boosted Trees (GBTs)

# Random forests

- ❑ `spark.ml` supports random forests for binary and multiclass classification and for regression, using both continuous and categorical features
- ❑ `spark.ml` implements random forests using the existing decision tree implementation

# Random forest API

## ❑ RandomForestClassifier class in PySpark

```
class pyspark.ml.classification.RandomForestClassifier(featuresCol='features',  
labelCol='label', predictionCol='prediction', probabilityCol='probability',  
rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32, minInstancesPerNode=1,  
minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10,  
impurity='gini', numTrees=20, featureSubsetStrategy='auto', seed=None, subsamplingRate=1.0,  
leafCol="", minWeightFractionPerNode=0.0, weightCol=None, bootstrap=True) \[source\]
```

## ❑ RandomForestRegressor class in PySpark

```
class pyspark.ml.regression.RandomForestRegressor(featuresCol='features',  
labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1,  
minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10,  
impurity='variance', subsamplingRate=1.0, seed=None, numTrees=20,  
featureSubsetStrategy='auto', leafCol="", minWeightFractionPerNode=0.0, weightCol=None,  
bootstrap=True) \[source\]
```

# Important parameter to adjust

- ❑ numTrees: Number of trees in the forest
- ❑ maxDepth: Maximum depth of each tree in the forest
- ❑ Several other parameters will be explored in the lab session

# Gradient-boosted trees

- ❑ `spark.ml` supports GBTs for binary classification and for regression, using both continuous and categorical features
- ❑ `spark.ml` implements GBTs using the existing decision tree implementation
- ❑ GBTs do not yet support multiclass classification

# Gradient-boosted trees API

## □ GBTClassifier class in PySpark

```
class pyspark.ml.classification.GBTClassifier(featuresCol='features', labelCol='label',  
predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0,  
maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, lossType='logistic',  
maxIter=20, stepSize=0.1, seed=None, subsamplingRate=1.0, impurity='variance',  
featureSubsetStrategy='all', validationTol=0.01, validationIndicatorCol=None, leafCol="",  
minWeightFractionPerNode=0.0, weightCol=None)
```

[\[source\]](#)

## □ GBRegressor class in PySpark

```
class pyspark.ml.regression.GBRegressor(featuresCol='features', labelCol='label',  
predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0,  
maxMemoryInMB=256, cacheNodeIds=False, subsamplingRate=1.0, checkpointInterval=10,  
lossType='squared', maxIter=20, stepSize=0.1, seed=None, impurity='variance',  
featureSubsetStrategy='all', validationTol=0.01, validationIndicatorCol=None, leafCol="",  
minWeightFractionPerNode=0.0, weightCol=None)
```

[\[source\]](#)



# Parameters to adjust

- ❑ The parameters to adjust are similar to the ones used for random forests
- ❑ Some of these will be reviewed in the lab session

# Contents

- What is a decision tree
  - Advantages
- Review of decision trees
- How do decision trees work in Spark?
  - Building trees and impurity measures in Spark
  - Split candidates
  - Decision tree classes in `spark.ml`
  - PLANET algorithm
- Review of ensemble methods
- Ensemble methods in PySpark
- Acknowledgement
- References

# References

- Lecture slides were adapted from [Dr Mauricio Alvarez](#), who contributed to this module from 2017 to 2022

# Contents

- What is a decision tree
  - Advantages
- Review of decision trees
- How do decision trees work in Spark?
  - Building trees and impurity measures in Spark
  - Split candidates
  - Decision tree classes in `spark.ml`
  - PLANET algorithm
- Review of ensemble methods
- Ensemble methods in PySpark
- Acknowledgement
- References

# References

- ❑ Book: Fundamentals of Machine Learning for Predictive Analytics by Kelleher, Mac Namee and D'Arcy, 2015
- ❑ Website: [Spark API documentation](#)
- ❑ YouTube videos:
  - [Scalable Decision Trees in Spark MLlib by Manish Amde](#) (contributor to the MLlib implementation of Decision Trees in Spark)
  - [Decision Trees on Spark by Joseph Bradley](#) (Databricks)
- ❑ Paper: *PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce* by B. Panda et al. (Google)