SWE404/DMT413
BIG DATA ANALYTICS

Lecture 9: Classification and Regression Algorithms II

Lecturer: Dr. Yang Lu

Email: luyang@xmu.edu.my

Office: A1-432

Office hour: 2pm-4pm Mon & Thur

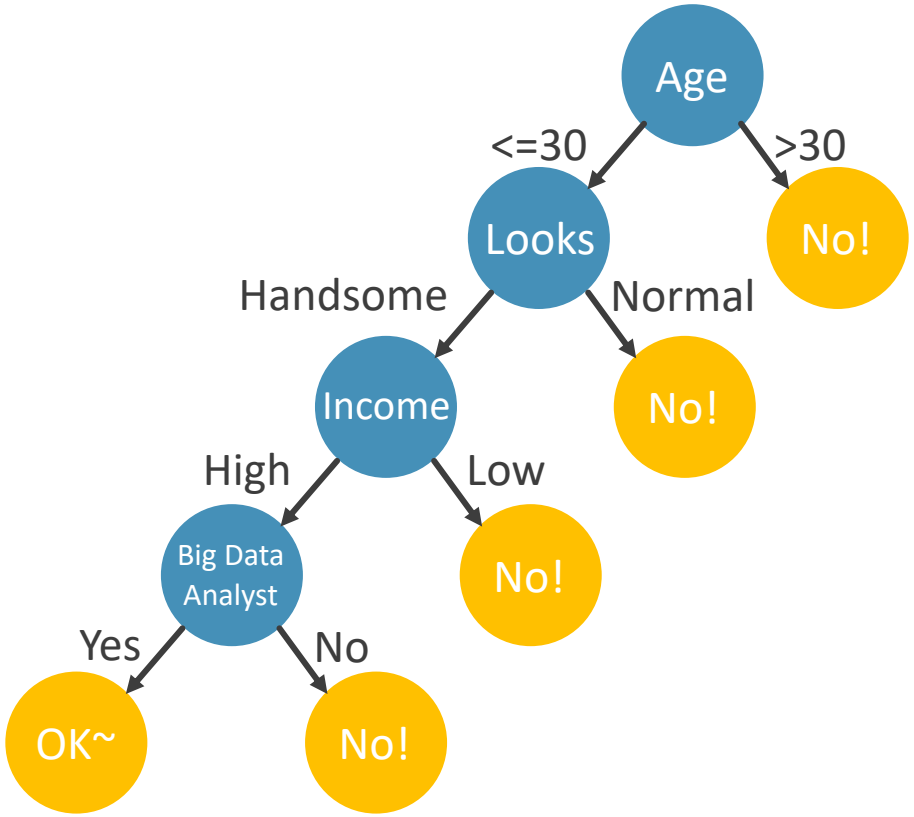
Outlines

- Decision Tree
- Ensemble Methods
  - Random Forest
  - Gradient Boosting Decision Trees
  - XGBoost

DECISION TREE

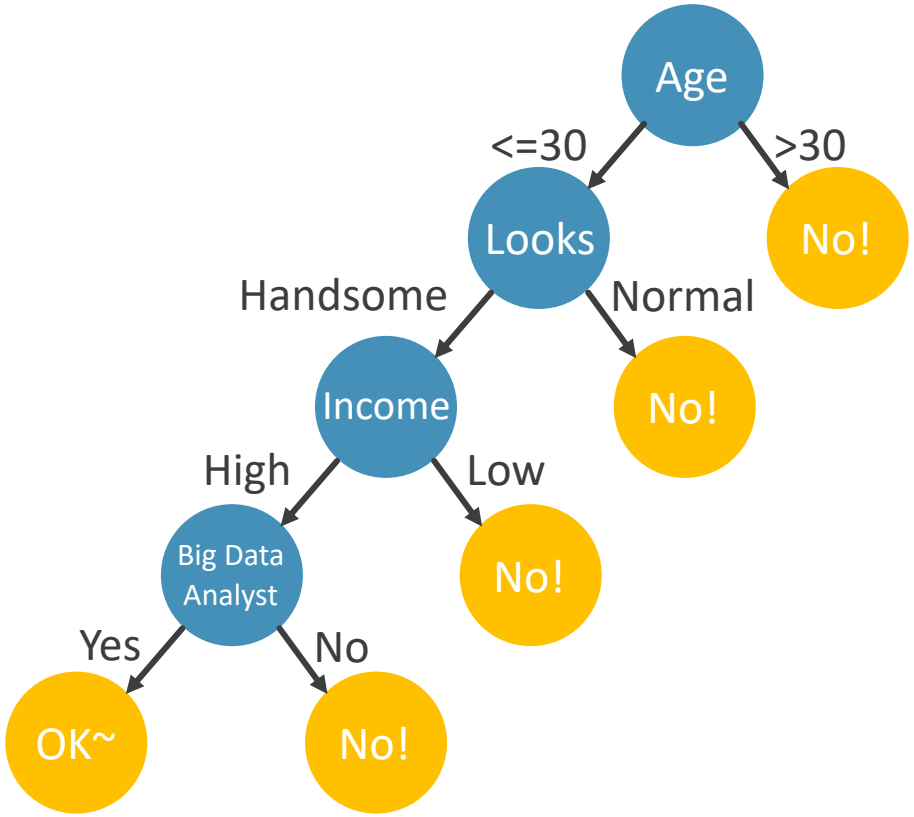
Daily Life Example of Decision Tree

- Your friend wants to introduce you a boyfriend, because you have been single for a few years…



Steps of Building a Decision Tree

1. Select a feature.
2. Determine a value to split the feature.
3. Check if all the samples in each branch after split belongs to the same class.
    a. Yes, we are done.
    b. No, go back to step 1.



Entropy

- Entropy measures the *uncertainty* or *purity* of a system:

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

- c is the number of events and p_i is the probability that the ith event happens.
- For example when $c = 2$
  - The probability of tomorrow's weather is $P(rainy) = 0.5$, $P(sunny) = 0.5$. Then $Entropy(S) = 0.5 + 0.5 = 1$.
  - The probability of tomorrow's weather is $P(snowy) = 0.0001$, $P(sunny) = 0.9999$. Then $Entropy(S) = 0.00133 + 0.00014 = 0.00147$.
- Entropy is related to the the amount of information:
  - You don't want to know if the sun will rise tomorrow. (small amount of information)
  - You want to know if Lakers can beat Rockets tomorrow. (large amount of information)

Entropy

- Here, c is the total number of classes or attributes and p_i is number of examples belonging to the ith class.
- For this data, decision is the label with two classes.

$$Entropy(\text{Decision}) = -p(\text{Yes})\log_2 p(\text{Yes}) - p(\text{No})\log_2 p(\text{No})$$
$$= -\frac{9}{14}\log_2 p\left(\frac{9}{14}\right) - \frac{5}{14}\log_2 p\left(\frac{5}{14}\right) \approx 0.940$$

Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Information Gain

- Using entropy, the information gain can be calculated for selection of each feature A:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

- For feature A, we calculate the entropy of each of its value v.
  - For example, for feature Wind, we calculate $Entropy(\text{Decision}|\text{Strong})$ and $Entropy(\text{Decision}|\text{Weak})$.

Information Gain

Day	Wind	Decision
1	Weak	No
2	Strong	No
3	Weak	Yes
4	Weak	Yes
5	Weak	Yes
6	Strong	No
7	Strong	Yes
8	Weak	No
9	Weak	Yes
10	Weak	Yes
11	Strong	Yes
12	Strong	Yes
13	Weak	Yes
14	Strong	No

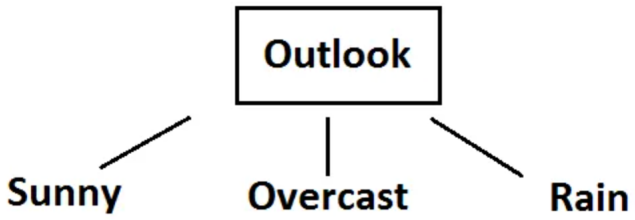
$$
\begin{aligned}
&Entropy(\text{Decision}|\text{Strong}) \\
&\quad = -p(\text{Yes}|\text{Strong})\log_2 p(\text{Yes}|\text{Strong}) - p(\text{No}|\text{Strong})\log_2 p(\text{No}|\text{Strong}) \\
&\quad = -\frac{3}{6}\log_2 p\left(\frac{3}{6}\right) - \frac{3}{6}\log_2 p\left(\frac{3}{6}\right) = 1 \\
&Entropy(\text{Decision}|\text{Weak}) \\
&\quad = -p(\text{Yes}|\text{Weak})\log_2 p(\text{Yes}|\text{Weak}) - p(\text{No}|\text{Weak})\log_2 p(\text{No}|\text{Weak}) \\
&\quad = -\frac{6}{8}\log_2 p\left(\frac{6}{8}\right) - \frac{2}{8}\log_2 p\left(\frac{2}{8}\right) \approx 0.811 \\
&Gain(\text{Decision}, \text{Wind}) \\
&\quad = Entropy(\text{Decision}) - \left(\frac{6}{14}Entropy(\text{Decision}|\text{Strong}) + \frac{8}{14}Entropy(\text{Decision}|\text{Weak})\right) \\
&\quad = 0.940 - \left(\frac{6}{14}\times 1 + \frac{8}{14}\times 0.811\right) \approx 0.048
\end{aligned}
$$

Information Gain

- Similarly, we can get:

$$Gain(\text{Decision}, \text{Wind}) = 0.048$$
$$Gain(\text{Decision}, \text{Outlook}) = 0.246$$
$$Gain(\text{Decision}, \text{Temp.}) = 0.029$$
$$Gain(\text{Decision}, \text{Humidity}) = 0.151$$

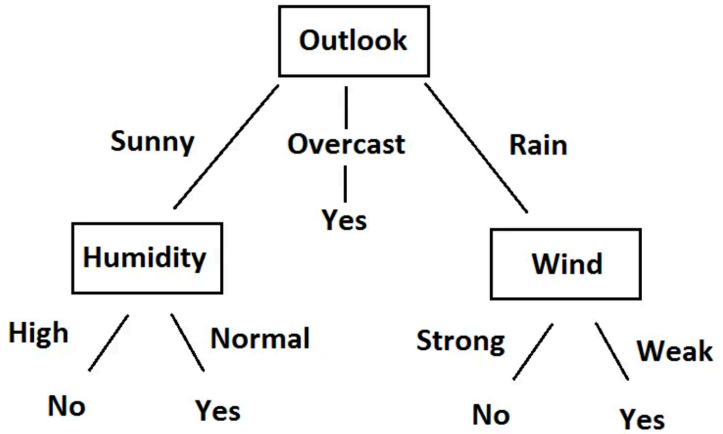
- As seen, outlook feature on decision produces the highest information gain. That's why, outlook decision will appear in the root node of the tree.



Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Information Gain

- For each branch, we continue to select the feature with highest information gain, until each node contains only one class.
- The final decision tree is:



Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Alternative Purity

- Entropy is only one of the purity criteria.
- Alternatives are:
  - Gini index:

$$\sum_{i=1}^{c} p_i(1 - p_i)$$

  - Misclassification error:

$$1 - \max_i p_i$$

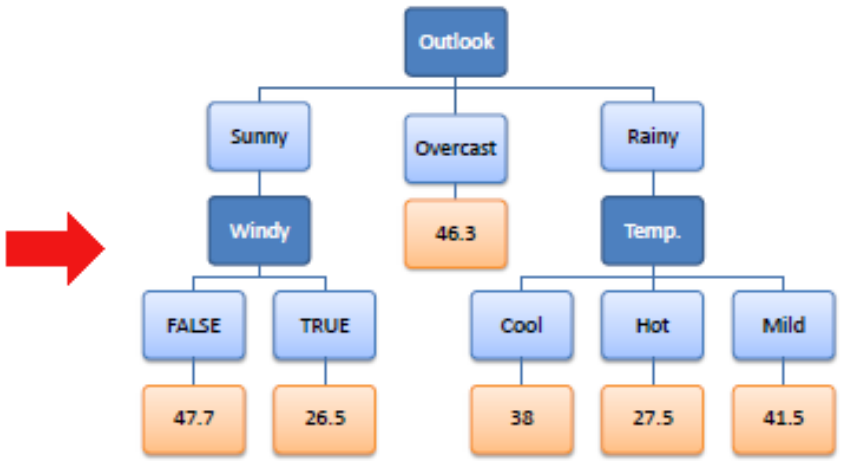
Decision Tree Pruning

- One of the questions that arises in a decision tree algorithm is the optimal size of the final tree.
- A tree that can correctly classify all training data risks overfitting the training data and poorly generalizing to new samples.
- Pruning should reduce the size of a decision tree without reducing predictive accuracy as measured by a cross-validation set.
- There are many techniques for tree pruning that differ in the measurement that is used to optimize performance.

Decision Tree Regression

- Decision tree can be easily extend to do regression.
- We can simply replace the entropy with **standard deviation** to measure the uncertainty.

Predictors				Target
Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	25
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	52
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	35
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	52
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30



Imgae source: https://saedsayad.com/decision_tree_reg.htm

Deal with Continuous Feature

- Previou example is about categorical feature, while each categoty is a branch in the tree.
- When a feature is continuous, we should find a cutting point.
- A straightforward way is to try all the cutting points with different results and select one according to the purity criterion.
- For example, we have samples on age feature: (20, 29, 40, 45), we can take the cutting points by the midpoints (24.5, 34.5, 45).

Advantages and Disadvantages

- Advantages:
  - Easy to understand and interpret.
  - Require less effort for data pre-processing.
- Disadvantages:
  - Too simple to use its own to predict.
  - Can’t handle complex data.

MLlib API

For regression: DecisionTreeRegressor

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*) [source]

- It supports both binary and multiclass labels, as well as both continuous and categorical features.
- Commonly used hyperparameter:
  - **maxDepth:** Maximum depth of the tree.
  - **maxBins:** Max number of bins for discretizing continuous features.
  - **minInstancesPerNode:** Minimum number of instances each child must have after split. If a split causes the left or right child to have fewer than minInstancesPerNode, the split will be discarded as invalid.
  - **minInfoGain:** Minimum information gain for a split to be considered at a tree node.
  - **impurity:** Criterion used for information gain calculation. Supported options: entropy, gini.

Source: https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#pyspark.ml.classification.DecisionTreeClassifier

ENSEMBLE METHODS

Ensemble Methods

- The performance of a binary classifier is poorest if its accuracy is 0.5.
  - 0.5 accuracy is just random guess.
  - Accuracy 0.01 is equivalent to accuracy 0.99, by simply flip the prediction over (0->1, 1->0).
- A classifier with accuracy slightly higher than 0.5 is called a *weak* classifier.
  - A pruned decision tree is usually a weak classifier.
- Assumption: Given many weak classifiers, we can combine them into a strong classifier.
  - Is that possible?

Ensemble Methods

- The answer is YES!
- We can let them do majority voting. It is called *ensemble*.
- Each classifier to construct the ensemble is called *individual* or *base* classifier.
- The necessary conditon is that, they should make different mistakes. The weak classifiers should be *diverse*.
  - Ensemble of many classifier with the same error is useless.

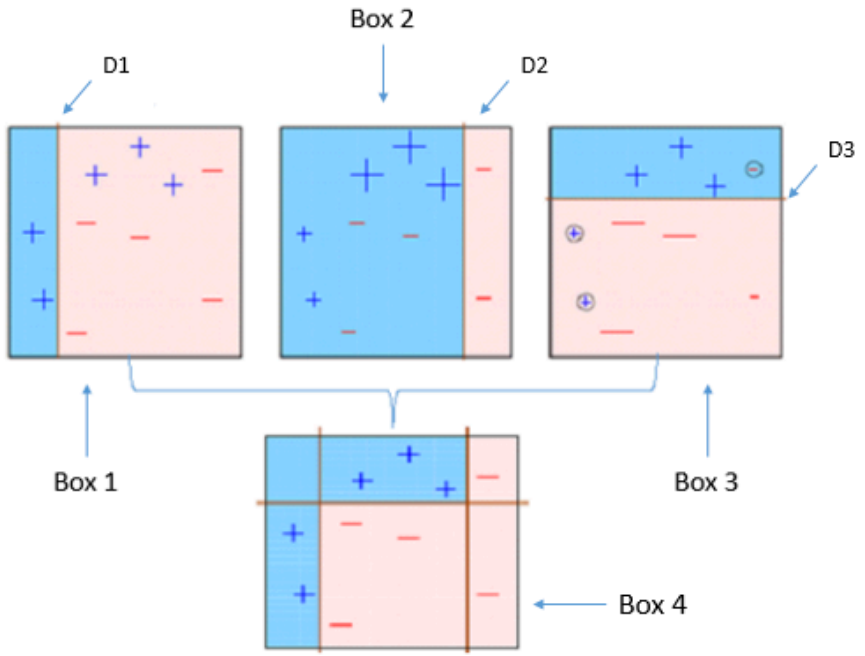


Image source: https://towardsdatascience.com/understanding-adaboost-2f94f22d5bfe

Bagging

- Bagging, is short for *bootstrap aggregating*, is the aggregation of multiple versions of a predicted model.
- Each model is trained individually, and the prediction is combined by averaging.
- As shown in the previous example, ensemble only works when the individual classifiers are diverse.
  - How to make them diverse?

Bootstrapping

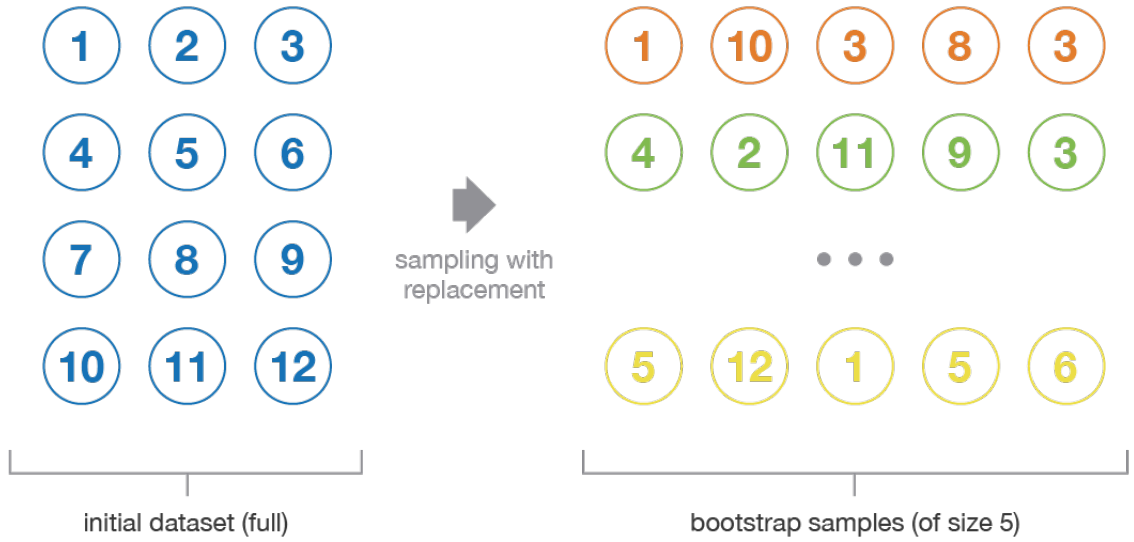
- Bootstrapping is the process of randomly sampling the data points **with replacement**.
- In such way, how many samples will be sampled if there are totally n samples?

$$\lim_{n\to\infty} 1-\left(1-\frac{1}{n}\right)^n = 1-\frac{1}{e} \approx 0.632$$

- About two-thirds.

```
import random
n = 1000000
bs = random.choices(list(range(n)), k=n)
print(len(set(bs)) / n)
```

```
0.632216
```



Bagging

- General steps of Bagging:
  - The bootstrapped sample sets are first created.
  - Either a regression or classification algorithm is applied to each set.
  - Finally, make prediction by averaging
    - For regression, an average is taken over all the outputs predicted by the individual learners.
    - For classification either the most voted class is accepted (*hard-voting*), or the highest average of all the class probabilities is taken as the output (*soft-voting*).
- Since classifier diversity is so important for ensemble, is there any way to produce more diversity?

Random Forest

- In Random Forest, along with the division of data, the features are also divided, and not all features are used to grow the trees.
- This technique is known as *feature bagging*. Each tree has its own set of features allocated to it.
- Random Forest has the same steps as Bagging, except:
  - When training each individual classifier, use only d' of d features ($d' < d$).
  - Use decision tree as individual classifier.

Advantages and Disadvantages

- Advantages:
  - Default hyperparameters used to give a good prediction.
  - Solves the overfitting problem.
  - It can be used as a feature selection tool.
  - It handles high dimensional data well.
- Disadvantages:
  - It is computationally expensive.
  - It is difficult to interpret.

MLlib API

For regression: RandomForestRegressor

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*) ¶ [source]

- It supports both continuous and categorical features.
- Commonly used hyperparameter:
  - **maxDepth**, **maxBins**, **minInstancesPerNode**, **minInfoGain**, **impurity**: same as decision tree.
  - **numTrees:** Number of trees to train (>= 1).
  - **subsamplingRate:** Fraction of the training data used for learning each decision tree, in range (0, 1].

MLlib API

- **featureSubsetStrategy:** The number of features to consider for splits at each tree node (default = 'auto'). Supported options:
  - 'auto' (choose automatically for task: If numTrees == 1, set to 'all'. If numTrees > 1 (forest), set to 'sqrt' for classification and to 'onethird' for regression),
  - 'all' (use all features),
  - 'onethird' (use 1/3 of the features),
  - 'sqrt' (use sqrt(number of features)),
  - 'log2' (use log2(number of features)),
  - 'n' (when n is in the range (0, 1.0], use n * number of features. When n is in the range (1, number of features), use n features).

MLlib Example

```
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer

data = spark.read.format("libsvm").load("sample_libsvm_data.txt")
labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)
li_data = labelIndexer.transform(data)
featureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(data)
fi_data = featureIndexer.transform(li_data)
rf = RandomForestClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures", numTrees=10)
model = rf.fit(fi_data)
```

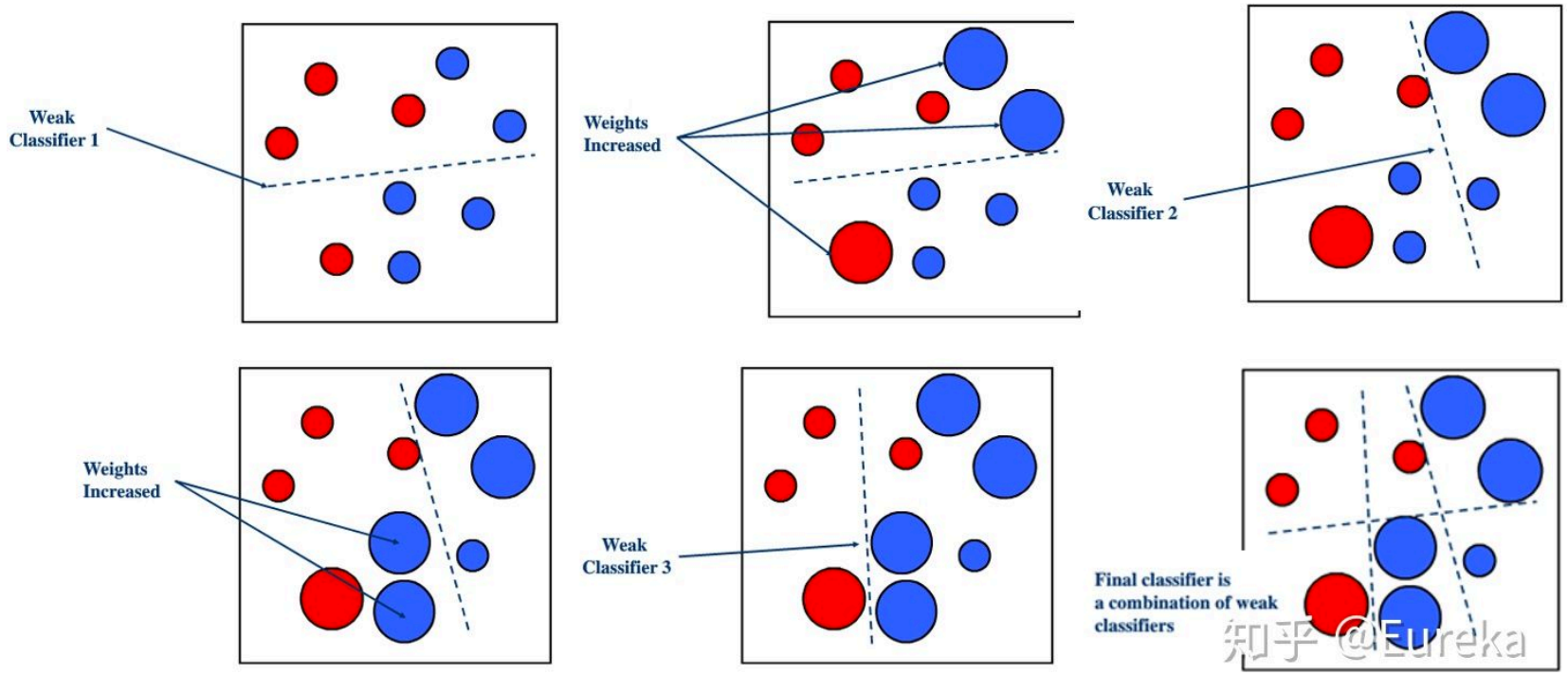
```
# Each feature's importance is the average of its importance across all trees in the ensemble.
# The importance vector is normalized to sum to 1.
model.featureImportances
```

```
SparseVector(692, {231: 0.0014, 237: 0.0005, 241: 0.0014, 295: 0.0036, 323: 0.0042, 330: 0.0038, 378: 0.0887, 379: 0.
0853, 380: 0.002, 384: 0.0805, 400: 0.0338, 407: 0.1223, 435: 0.0744, 461: 0.0961, 462: 0.1224, 490: 0.0377, 510: 0.0
705, 518: 0.0689, 547: 0.0062, 549: 0.0095, 553: 0.0195, 569: 0.0073, 578: 0.0519, 604: 0.0014, 660: 0.0066})
```

Boosting

- In Bagging, each individual classifier has the same weight in the ensemble, and is created in parallel.
- In contrast, boosting assigns different weights to each individual classifier, and is created in sequential.
  - In each stage, introduce an individual classifier to compensate the shortcomings of existing individual classifiers.

AdaBoost



AdaBoost increases the weight of the samples that is misclassified in previous rounds.

Image source: https://zhuanlan.zhihu.com/p/39920405

Generalization of AdaBoost as Gradient Boosting

- The statistical framework cast boosting as a numerical optimization problem.
  - The objective is to minimize the loss of the model by adding weak classifiers using a gradient descent like procedure.
  - Just like updating weight by $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla J(\boldsymbol{w})$, while now $\nabla J(\boldsymbol{w})$ is a new weak classifier.
- Gradient boosting involves three elements:
  - A loss function to be optimized.
  - A weak classifier to make predictions.
  - An additive model to add weak classifiers to minimize the loss function.

Gradient Boosting Decision Trees

- **Loss Function**
  - The loss function used depends on the type of problem being solved.
  - It must be differentiable.
- **Weak Classifier**
  - Decision trees are used as the weak classifier in gradient boosting. Therefore, it is usually called Gradient Boosting Decision Trees (GBDT).
- **Additive Model**
  - Trees are added one at a time, and existing trees in the model are not changed.
  - A gradient descent procedure is used to minimize the loss when adding trees.

Advantages and Disadvantages

- Advantages:
  - Often provides predictive accuracy that cannot be beat.
  - Lots of flexibility - can be used to solve almost all derivable objective function.
- Disadvantages:
  - More sensitive to overfitting if the data is noisy.
  - Training generally takes longer because of the fact that trees are built sequentially.

MLlib API

For regression: GBTRegressor

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, featureSubsetStrategy='all'*) [source]

- It supports both continuous and categorical features.
- Commonly used hyperparameter:
  - **maxDepth**, **maxBins**, **minInstancesPerNode**, **minInfoGain**, **impurity**, **subsamplingRate**: same as random forest.
  - **lossType:** Loss function which GBT tries to minimize. Supported options: logistic.
- GBDT automatically determines the number of trees in ensemble by checking its convergence. Thus, we don't have hyperparameter **numTrees**.

MLlib Example

```
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.feature import IndexToString, StringIndexer, VectorIndexer

data = spark.read.format("libsvm").load("sample_libsvm_data.txt")
labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)
li_data = labelIndexer.transform(data)
featureIndexer = VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(data)
fi_data = featureIndexer.transform(li_data)
rf = GBTClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures")
model = rf.fit(fi_data)
```

```
model.treeWeights
```

```
[1.0,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1,
 0.1]
```

XGBoost

- XGBoost stands for eXtreme Gradient Boosting.
  - It is the most powerful machine learning model before the age of deep learning.
  - It wins almost every kaggle competition.
  - It is still used in many IT companies now.
- XGBoost is based on GBDT but with more generalized configurations for real industry applications.
- If you are struggling in selecting models, try XGBoost first!

XGBoost

- Both XGBoost and GBDT follows the principle of gradient boosting. There are however, the difference in modeling details.
  - **Regularization:** XGBoost has regularization module to reduce overfitting while standard GBDT implementation doesn't have. In fact, XGBoost is also known as a '*regularized boosting*' technique.
  - **Parallel Processing:** XGBoost implements parallel processing and is blazingly faster as compared to GBDT.
  - **High Flexibility:** XGBoost allows users to define custom optimization objectives and evaluation criteria.
  - **Handling Missing Values:** XGBoost has an in-built routine to handle missing values.

XGBoost

- If you are interested in XGBoost, I highly suggest you to read the original paper.

Xgboost: A scalable tree boosting system
T Chen, C Guestrin - Proceedings of the 22nd acm sigkdd international …, 2016 - dl.acm.org
Tree boosting is a highly effective and widely used machine learning method. In this paper, we describe a scalable end-to-end tree boosting system called XGBoost, which is used widely by data scientists to achieve state-of-the-art results on many machine learning …
☆ 99 Cited by 5612 Related articles All 27 versions

- For a more comprehensive discussion of XGBoost, you can read this master thesis:

[PDF] **Tree boosting with xgboost-why does xgboost win" every" machine learning competition?**
D Nielsen - 2016 - ntnuopen.ntnu.no
Tree boosting has empirically proven to be a highly effective approach to predictive modeling. It has shown remarkable results for a vast array of problems. For many years, MART has been the tree boosting method of choice. More recently, a tree boosting method …
☆ 99 Cited by 71 Related articles All 2 versions »

XGBoost in Spark

- XGBoost is not officially supported by PySpark.
- You can use the scala version of Spark with package XGBoost4J-Spark.
- A good alternative of XGBoost is LightGBM, whose performance is as good as XGBoost.
  - Follow https://github.com/Azure/mmlspark/blob/master/docs/lightgbm.md.

MACHINE LEARNING RELATED ISSUES

Hyperparameter Tuning

- Almost every model has some hyperparameters.
  - C and λ in SVM.
  - Number of layers and neurons on each layer in neural networks.
  - Number of trees and size of feature subset in random forest.
- For different datasets, the best hyperparameters are different.
  - We need to select them by trial and error.
- This process is called *hyperparameter tuning* or *model selection*.

Parameter Tuning

- However, we can't select hyperparameters based on the performance on the test data.
  - It is like cheating if you know questions and answers of the final exam and then adjust your study plan.
- Generally, we have two strategies:
  - K-fold cross validation.
  - Fixed validation set.

K-Fold Cross Validation

- We cut the training data into K folds, where
  - K-1 folds are used for training.
  - 1 fold is used for validation.
  - Repeat K times with different combinations and select the parameter with the highest average performance.
- For example, 3-fold cross validation will separate the training data into 5 folds.
  - Train on fold [1, 2] and test on [3].
  - Train on fold [1, 3] and test on [2].
  - Train on fold [2, 3] and test on [1].
- After selecting the best hyperparameters, train the model again with all training data.

Fixed Validation Set

- If the dataset is large enough, we can fix the training/validation/test data.
  - 8-1-1 split is usually adopted.
- In this way, we can compare training/validation/test error, because the size of training data is same.
  - The data size used for training in cross validation is actually shrinked.

Grid Search

- If you have only one hyperparameter, simply try all the values you want.
  - Learning rate in logistic regression: [0.0001, 0.001, 0.01, 0.05, 0.1, 0.5].
- If you have multiple hyperparameters, we should try all the combinations of them. This is called grid search.
  - Learning rate in logistic regression: [0.0001, 0.001, 0.01, 0.05, 0.1, 0.5].
  - Regularization parameter: [0.001, 0.01, 0.1, 0, 1, 10, 100]
  - Totally $6 \times 7 = 42$ combinations should be tried.

General Strategy for Multiclass Classification

- If a classifier can only handle binary classification, e.g. logistic regression or SVM, we can also make them able to do multiclass classification.
- Two strategies:
  - **One-Vs-Rest (OVR)**
  - **One-Vs-One (OVO)**

One-Vs-Rest for Multiclass Classification

- Split the multiclass dataset into *c* binary classification problems.
- For example, given a multiclass classification problem with examples for each class '*red*,' '*blue*,' and '*green*'. This could be divided into three binary classification datasets as follows:
  - Binary classification problem 1: red vs [blue, green],
  - Binary classification problem 2: blue vs [red, green],
  - Binary classification problem 3: green vs [red, blue].
- The final prediction is made by selection of the class with highest probability.

One-Vs-One for Multiclass Classification

- Split the multiclass dataset into $c(c-1)/2$ binary classification problems.
- For example, consider a multiclass classification problem with four classes: '*red*,' '*blue*,' and '*green*,' '*yellow*.' This could be divided into six binary classification datasets as follows:
  - Binary classification problem 1: red vs. blue
  - Binary classification problem 2: red vs. green
  - Binary classification problem 3: red vs. yellow
  - Binary classification problem 4: blue vs. green
  - Binary classification problem 5: blue vs. yellow
  - Binary classification problem 6: green vs. yellow
- The final prediction is made by majority voting.

Evaluation Metric

- Evaluating your machine learning algorithm is an essential part of any project.
- For different applications, we may adopt different evaluation metrics.
- Most of the times we use classification accuracy to measure the performance of our model, however it is not enough to truly judge our model.

Classification Accuracy

- Classification accuracy is what we usually mean, when we use the term accuracy.
- It is the ratio of number of correct predictions to the total number of input samples.

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

- However, sometimes accuracy cannot well reflect the performance.
  - Consider training a disease diagnose system that there are 98% samples of healthy people and 2% samples of patients in our training set.
  - Then our model can easily get **98% training accuracy** by simply predicting every training sample as healthy.

Confusion Matrix

- Confusion Matrix as the name suggests gives us a matrix as output and describes the complete performance of the model.
- Confusion Matrix forms the basis for the other types of metrics.

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

		Ground Truth	
Prediction		Positive (sick)	Negative (healthy)
	Positive (sick)	True Positive (TP)	False Positive (FP)
	Negative (healthy)	False Negative (FN)	True Negative (TN)

Precision and Recall

- *Precision* refers to how much of your positive predictions are correct. It doesn't care if you cover all of the positive samples.

$$Precision = \frac{tp}{tp + fp}.$$

  - For the previous example, if you only predict one sample as sick and it is correct, the precision will be 1.
- *Recall* (aka *sensitivity, TPR*) refers to how much of your positive samples are correctly classified. It doesn't care how many positive predictions you make.

$$Recall = \frac{tp}{tp + fn}.$$

  - For the previous example, if you predict all of the samples as sick, the recall will be 1.

F1 Score

- Therefore, a good classifier should take a balance between precision and recall.
- We use F1 score to measure this trade-off. It is the harmonic mean of precision and recall.

$$F1 = \frac{2 \times precision \times recall}{precision + recall}.$$

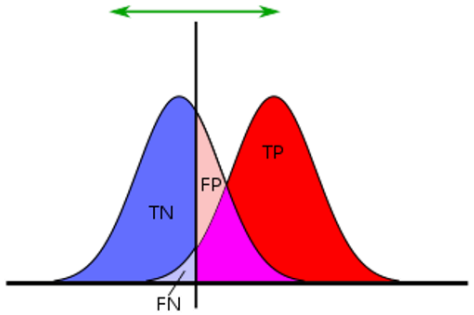
- F1 score will be high only if both precision and recall be high.
  - One of them is close to 0 will make F1 score close to 0.

Receiver Operating Characteristic

- Another trade-off can be made between TPR (recall) and FPR.

$$TPR = \frac{tp}{tp + fn}, \qquad FPR = \frac{fp}{fp + tn}.$$

- By moving the threshold for binary classification, the change of TPR and FPR can be drawn in a figure. It is called *Receiver Operating Characteristic (ROC)*.
- Usually, we calculate the Area Under Curve (AUC) of ROC to compare.



TP	FP
FN	TN

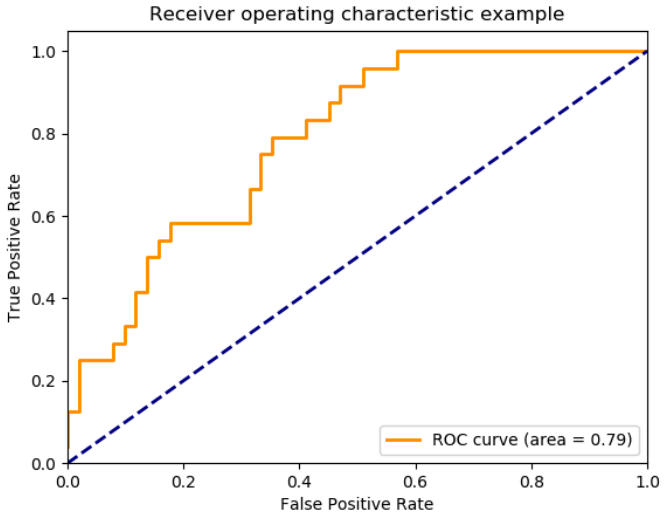


Image source: https://en.wikipedia.org/wiki/Receiver_operating_characteristic

Conclusion

After this lecture, you should know:

- How to build a decision tree.
- Why weak classifier can be combined to become strong classifier by ensemble.
- What is the difference between bagging and boosting.
- How to tune hyperparameters.
- What are the common used evaluation metrics.

Assignment 3

- Assignment 3 is released. The deadline is **18:00, 15th June**.

Thank you!

- Any question?
- Don’t hesitate to send email to me for asking questions and discussion. ☺