Ensemble

Hossam Ahmed Salah

MSP *Helwan*

Agenda

Bootstrapping

Bagging bootstrap aggregating

Boosting

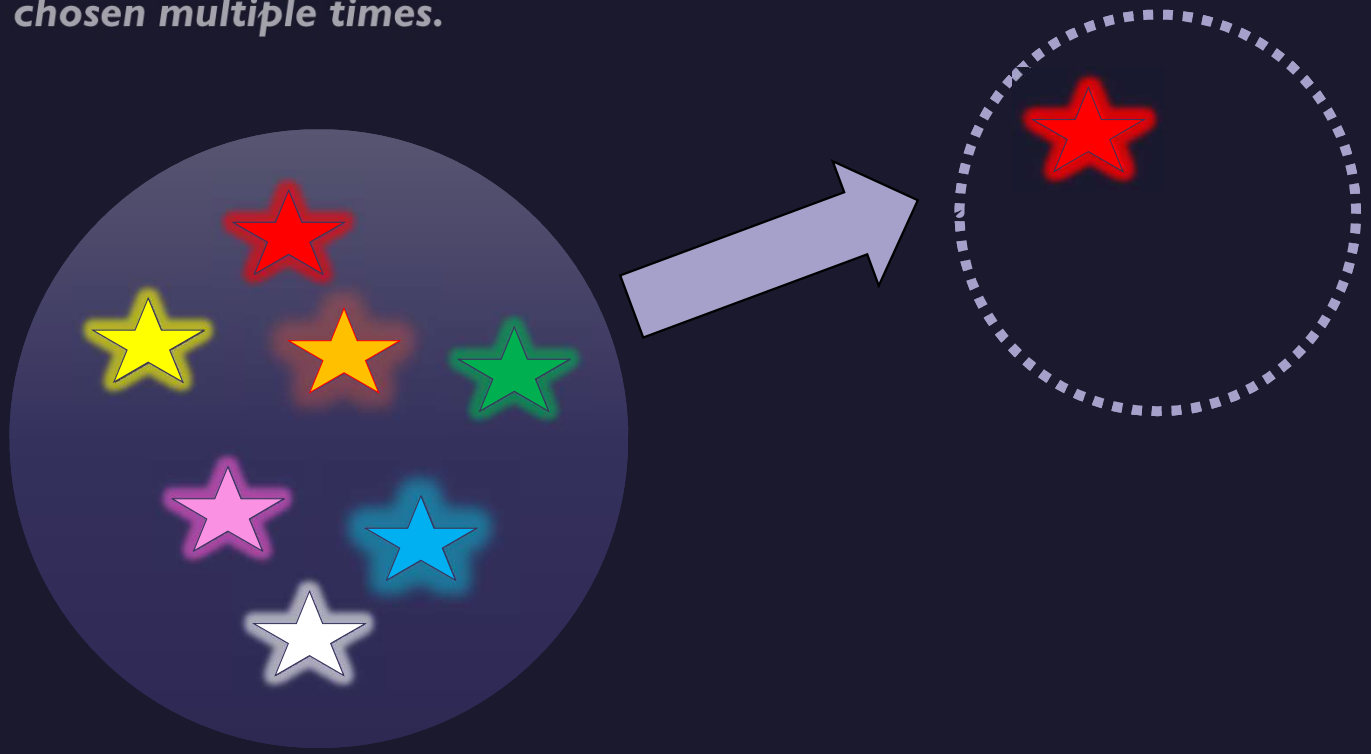
Random forest

Extra trees

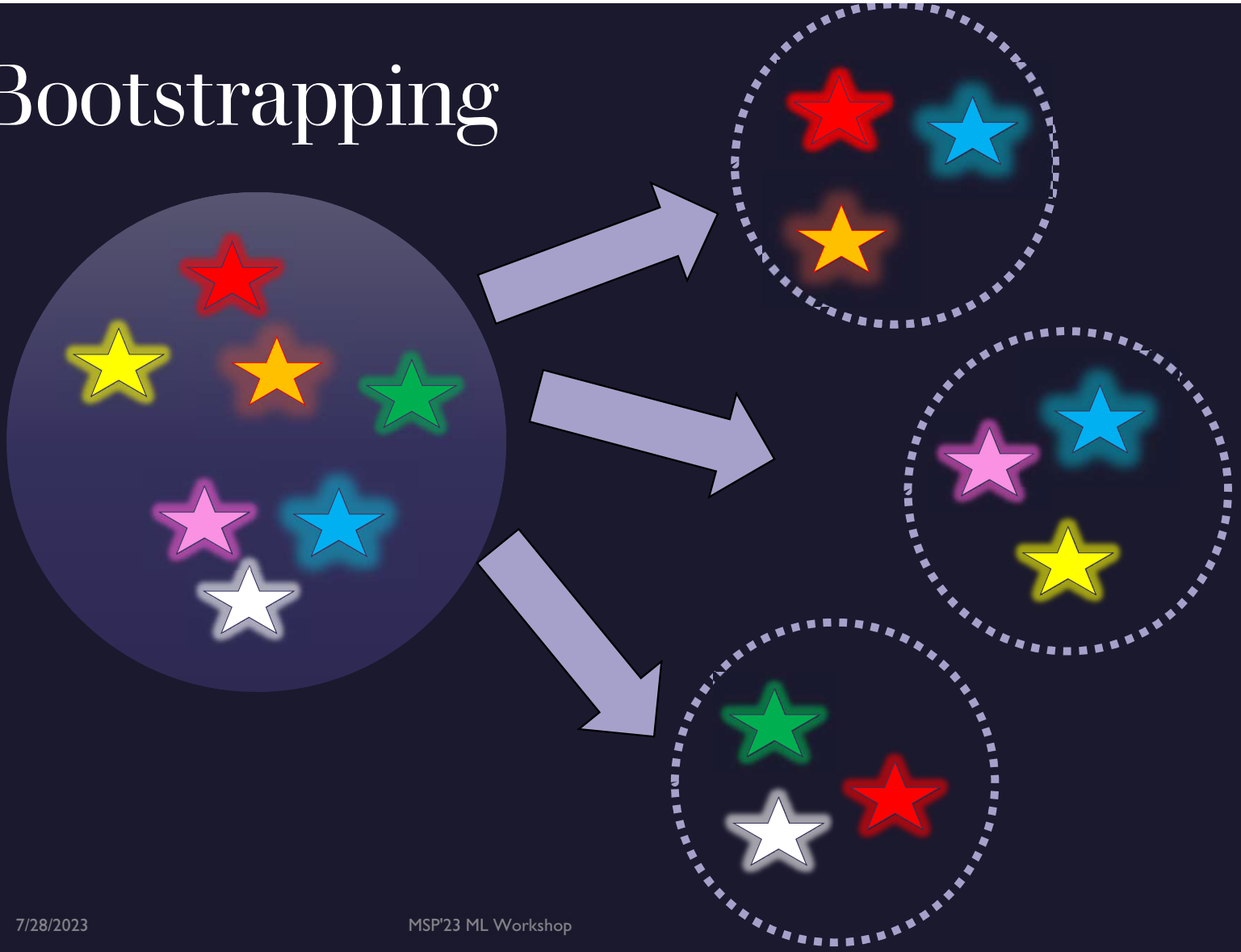
AdaBoost

Bootstrapping

- *Bootstrapping involves* ***random sampling*** *with* ***replacement****, allowing observations to be chosen multiple times.*



Bootstrapping



Out of Bag samples

Out of Bag samples

- The OOB sample is a **set of observations that are not used** to build a particular sample.
- The OOB sample can be used as a **test set** for each model in the ensembled model (forest).
- The OOB sample provides an unbiased **estimate of the model's performance** without the need for a separate validation set.
- The OOB sample can save time and computational resources, as well as reduce the risk of overfitting the model to a particular validation set.
- The OOB sample can be used to **identify the most important features** in the random forest model based on their contribution to the accuracy of the model on the OOB sample.

Example

- [0.1, 0.2, 0.3, 0.4, 0.5, 0.6] dataset with 6 observations
- The first step is to choose the **size of the sample**. Here, we will use 4.
- we must randomly choose the first observation from the dataset. Let's choose 0.2
- sample = [0.2]
- This observation is **returned to the dataset,** and we repeat this step 3 more times
- sample = [0.2, 0.1, 0.2, 0.6]
- Those observations not chosen for the sample may be used as out of sample observations.
- oob = [0.3, 0.4, 0.5]

Example

- [0.1, 0.2, 0.3, 0.4, 0.5, 0.6] dataset with 6 observations
- sample = [0.2, 0.1, 0.2, 0.6]
- In the case of evaluating a machine learning model, the model is fit on the drawn sample and evaluated on the out-of-bag sample.

```
train = [0.2, 0.1, 0.2, 0.6] #Sample

test = [0.3, 0.4, 0.5] #OOB

model = fit(train) #Model fitted on the sample

statistic = evaluate(model, test) #Evaluated on the OOB
```

Example

```
statistic = evaluate(model, test) #Evaluated on the OOB
```

- That concludes one **repeat of the procedure**. It can be repeated 30 or more times to give a sample of calculated statistics.

```
statistics = [...]
```

- This sample of statistics can then be **summarized by calculating a mean**, standard deviation, or other summary values to give a final usable estimate of the statistic

```
estimate = mean([...])
```

Implementing Bootstrapping sample

- resample() scikit-learn function
- It takes as arguments the data array, whether or not to sample with replacement, the size of the sample, and the seed for the pseudorandom number generator used prior to the sampling.
- we can create a bootstrap that creates a sample with replacement with 4 observations and uses a value of 1 for the pseudorandom number generator.

```
boot = resample(data, replace=True, n_samples=4, random_state=1)
```

- we can gather the out-of-bag observations using a simple Python list comprehension.
- *# out of bag observations*
- oob = [x for x in data if x not in boot]

```
# scikit-learn bootstrap
from sklearn.utils import resample
# data sample
data = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
# prepare bootstrap sample
boot = resample(data, replace=True, n_samples=4, random_state=1)
print('Bootstrap Sample: %s' % boot)
# out of bag observations
oob = [x for x in data if x not in boot]
print('OOB Sample: %s' % oob)
```

Bootstrap Sample: [0.6, 0.4, 0.5, 0.1]
OOB Sample: [0.2, 0.3]

Bagging

Bootstrapping aggregating

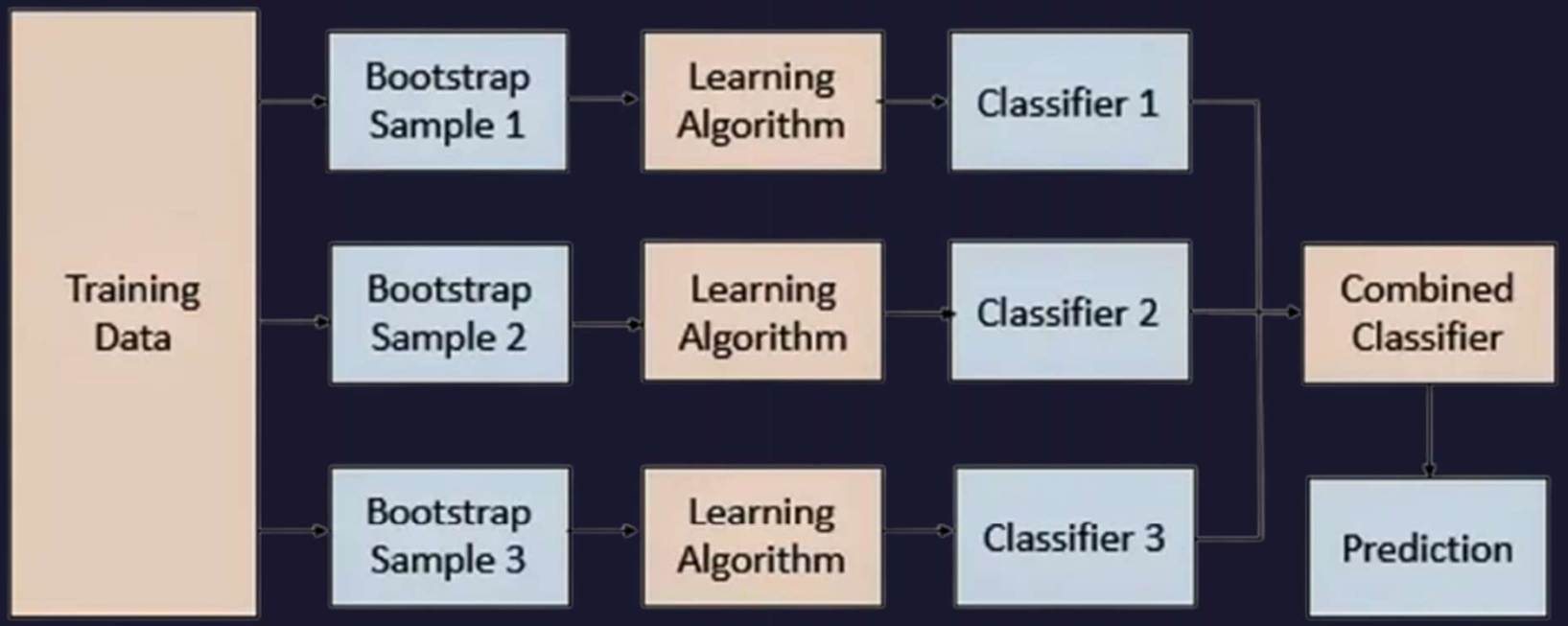
Bagging

Bootstrapping aggregating

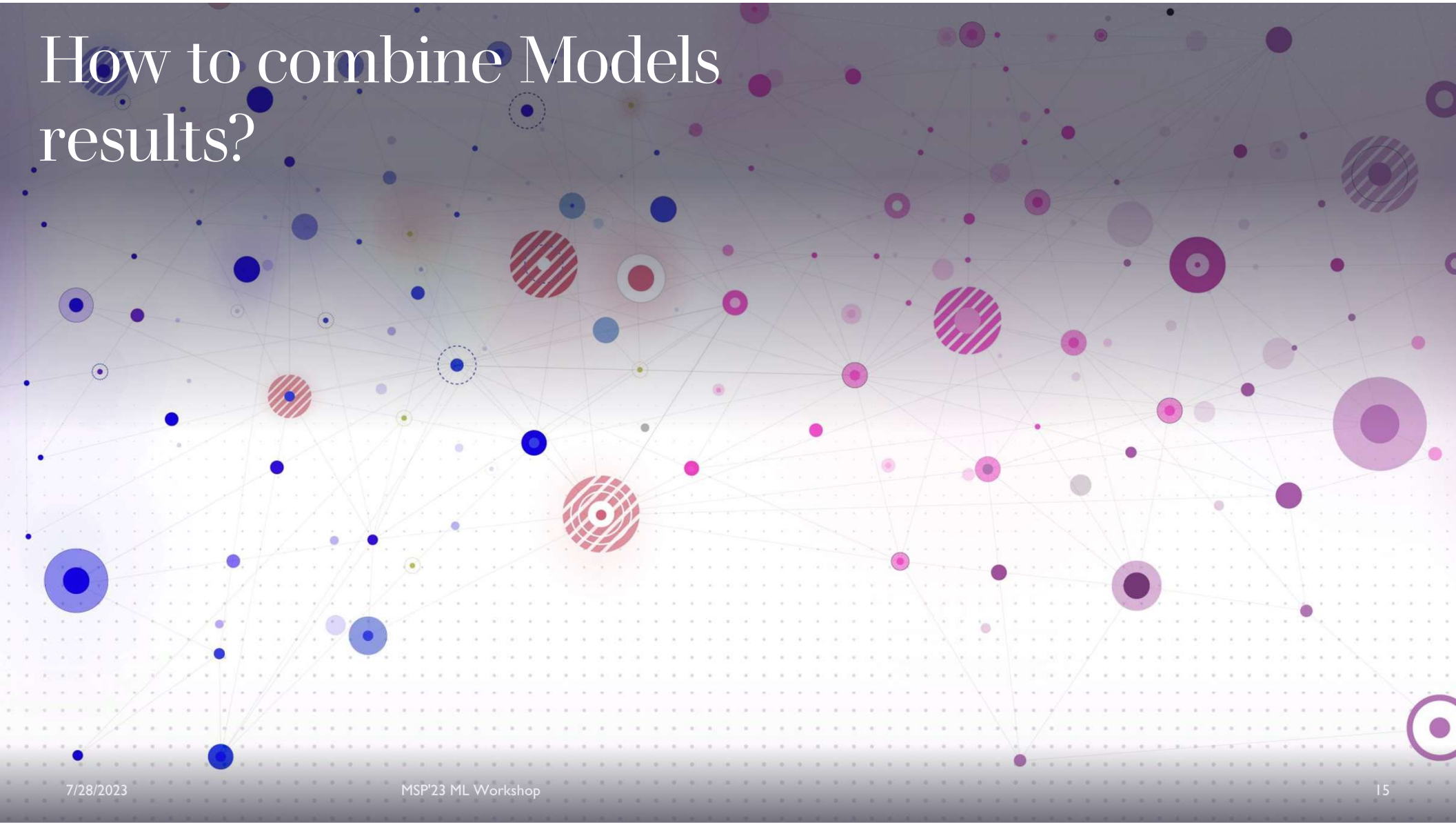
- *Bagging uses **this idea of asking multiple "models"** (which are like your friends) to help **make a better prediction or decision**. These models are **trained using different samples** taken from the original data set. **Each model learns something slightly different** from the data, just like each friend might have a different perspective or opinion.*
- *By combining the predictions of all these models, bagging helps to make a more accurate and reliable prediction. It's like taking the average or consensus of your friends' answers to make a better decision.*
- *So, bagging is a way to improve the accuracy of predictions by using multiple models trained on different samples of the original data.*

Bagging

Bootstrapping aggregating



How to combine Models results?



1. Max voting

- *The max voting method is used for **classification problems**.*
- *Multiple models are used to make predictions, and each **model's prediction** is considered as a '**vote**'.*
- *The **final prediction** is determined based on **the majority of votes** from the models.*
- *It can be compared to asking colleagues to rate a movie and taking the mode of their ratings as the final rating.*

1. Max voting

```
from sklearn.ensemble import VotingClassifier
model1 = LogisticRegression(random_state=1)
model2 = tree.DecisionTreeClassifier(random_state=1)
model = VotingClassifier(estimators=[('lr', model1), ('dt', model2)], voting='hard')
model.fit(x_train,y_train)
model.score(x_test,y_test)
```

2. Averaging

- Averaging is a technique where multiple predictions are made for each data point.
- The final prediction is obtained by taking the average of all the predictions.
- Averaging is commonly used in **regression** problems or for calculating probabilities in classification problems

2. Averaging

```
model1 = tree.DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()

model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)

pred1=model1.predict(x_test)
pred2=model2.predict(x_test)
pred3=model3.predict(x_test)

finalpred=(pred1+pred2+pred3)/3
```

3. Weighted Averaging

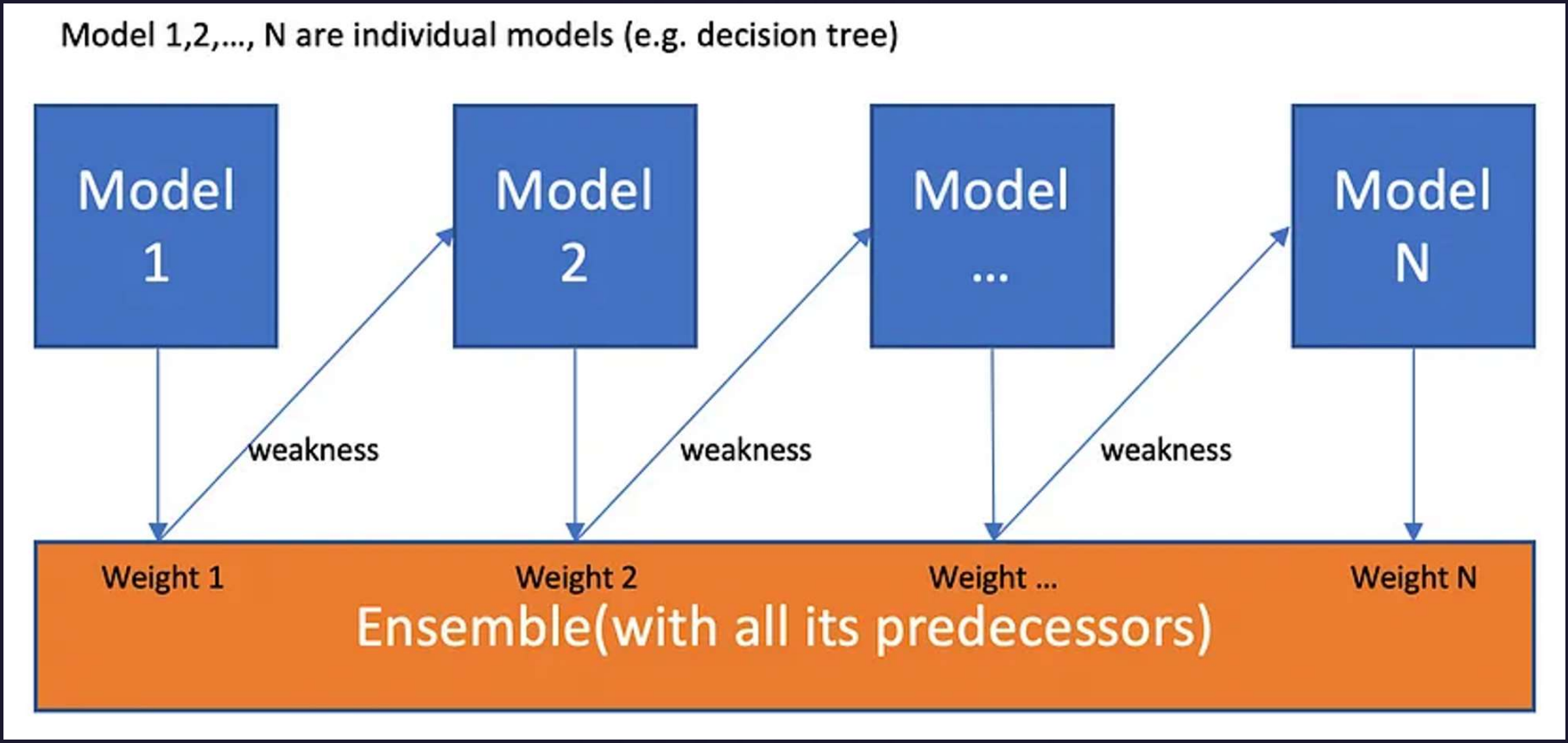
- Weighted averaging is an extension of the averaging method where **each model is assigned a weight.**
- The weights determine the importance of each model's prediction in the final result.
- Models with higher weights contribute more to the final prediction.

3. Weighted Averaging

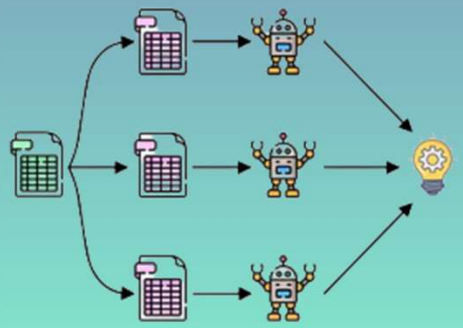
```
model1 = tree.DecisionTreeClassifier()
model2 = KNeighborsClassifier()
model3= LogisticRegression()

model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)

pred1=model1.predict(x_test)
pred2=model2.predict(x_test)
pred3=model3.predict(x_test)

finalpred=(pred1*0.3+pred2*0.3+pred3*0.4)
```

Boosting

Boosting

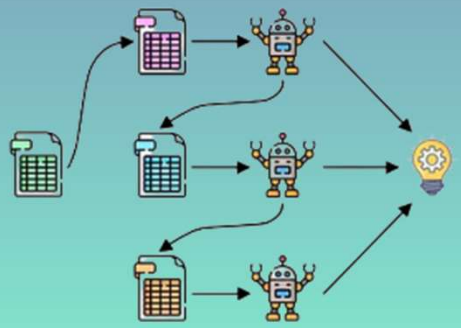

Model 1,2,…, N are individual models (e.g. decision tree)
Model 1
Model 2
Model …
Model N
weakness
weakness
weakness
Weight 1
Weight 2
Weight …
Weight N
Ensemble(with all its predecessors)

Bagging



Parallel

Boosting



Sequential

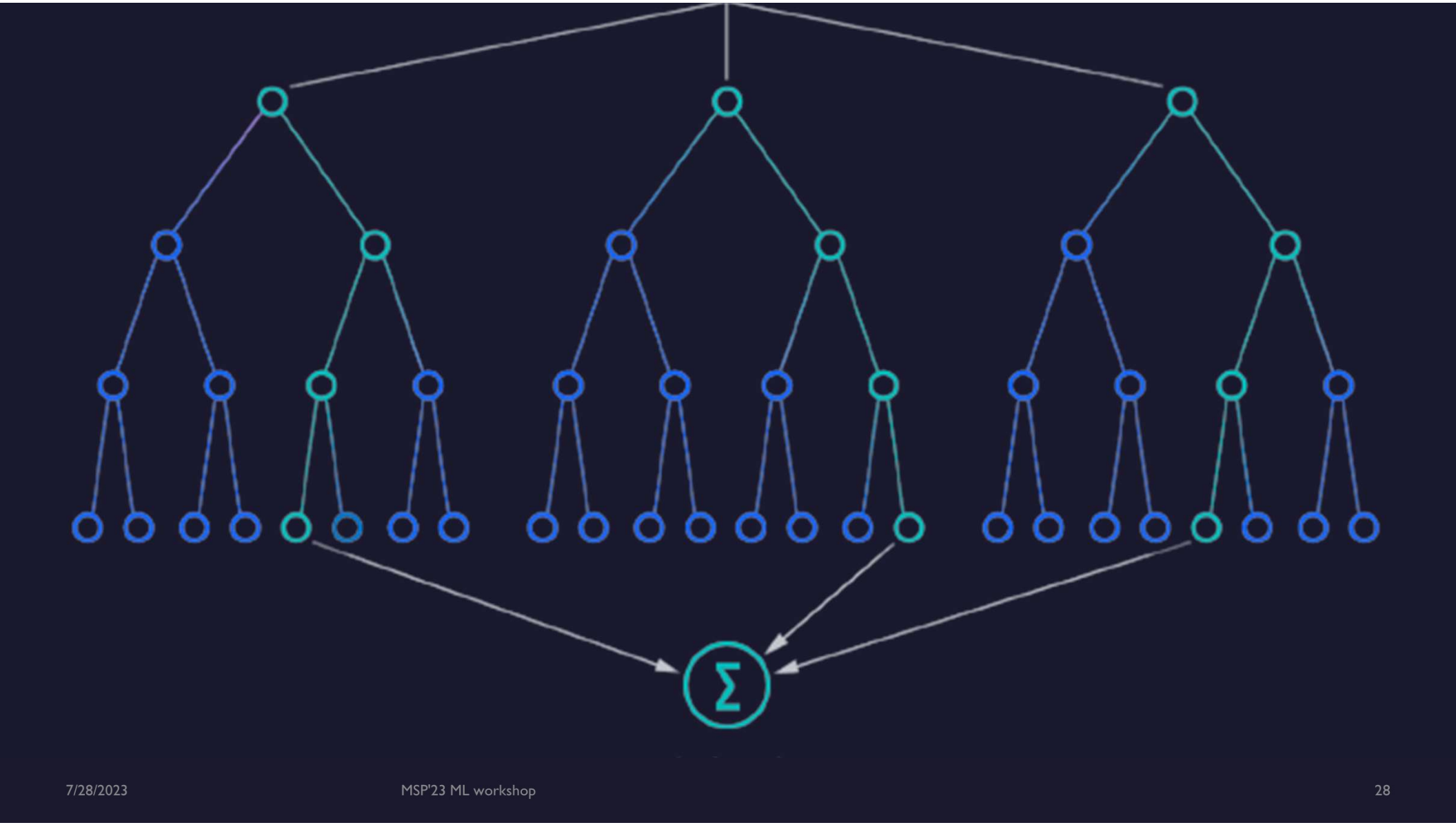
Boosting

- Boosting is an advanced version of Bagging.
- Models are **built sequentially**, one after another.
- Each model focuses on **improving** the classification of data points that were **incorrectly classified** by the previous model.
- The iterative process of boosting leads to increased overall accuracy of the model.
- A subset is created from the original dataset for each iteration of boosting.
- **Initially**, all data points are given **equal weights**.
- A base model is created using this subset.
- The base model is used to make predictions on the entire dataset.
- Errors are calculated using the actual values and predicted values.
- The observations which are **incorrectly predicted**, are **given higher weights**.
- Another model is created, and predictions are made on the dataset.(This model tries to correct the errors from the previous model)

Boosting

- Boosting slowly improves the fitted model by fitting small trees to the residuals in areas where it does not perform well.
- In boosting, the construction of each tree depends strongly on the trees that have already been grown, unlike in bagging.
- Boosting has three tuning parameters: the number of trees B, the shrinkage parameter λ, and the number of splits d in each tree.
- Cross-validation is used to select the **number of trees B to avoid overfitting.**
- The shrinkage parameter **λ** is a small positive number that **controls the rate** at which boosting **learns**.
- The **number of splits d** in each tree **controls the complexity** of the boosted ensemble.
- A smaller value of d, such as d=1, often works well, in which case each tree is a stump consisting of a single split, and the boosted stump ensemble is fitting an additive model.

Random forest
example on Bagging


Σ

Random forest

- Random forests reduce the risk of overfitting because the **averaging of uncorrelated decision trees** *lowers* the ***overall variance*** and prediction error.
- Random forests provide flexibility because they can handle both regression and classification tasks with high accuracy.
- **Feature bagging(**unique set of features to each tree**)** in random forests can be used to estimate missing values while maintaining accuracy.
- Random forests make it easy to determine feature importance using measures such as Gini importance, mean decrease in impurity (MDI), and permutation importance (MDA) or even using OOB.
- Permutation importance (MDA) is a method for evaluating feature importance that identifies the average decrease in accuracy by randomly permutating the feature values in OOB samples

Random forest

- In **feature bagging,** a random subset of features (also known as predictors or variables) is selected for each tree in the random forest. This means that **each tree is built using a different set of features,** and **no single feature is used in all the trees**. The number of features in each subset is typically smaller than the total number of features in the dataset.
- The **purpose of feature bagging** is to **reduce the correlation between trees** in the random forest, which can lead to overfitting and decreased accuracy. By randomly selecting a subset of features for each tree, the trees are forced to rely on different aspects of the data, which helps to reduce the variance of the model and improve its generalization performance.

Random forest Feature importance

- Though bagged trees are hard to interpret as a whole, we can still summarize the importance of each predictor variable.
- For bagged regression trees, we calculate the total decrease in RSS due to splits on each variable, averaged over all trees.
- For bagged classification trees, we calculate the total decrease in the Gini index due to splits on each variable, averaged over all trees.
- **A larger value** indicates a **more important predictor**, as it leads to greater reductions in impurity or error when used for splits.
- The variable importance provide an overall summary of the predictors' contributions to the bagged model, though not for any individual tree.

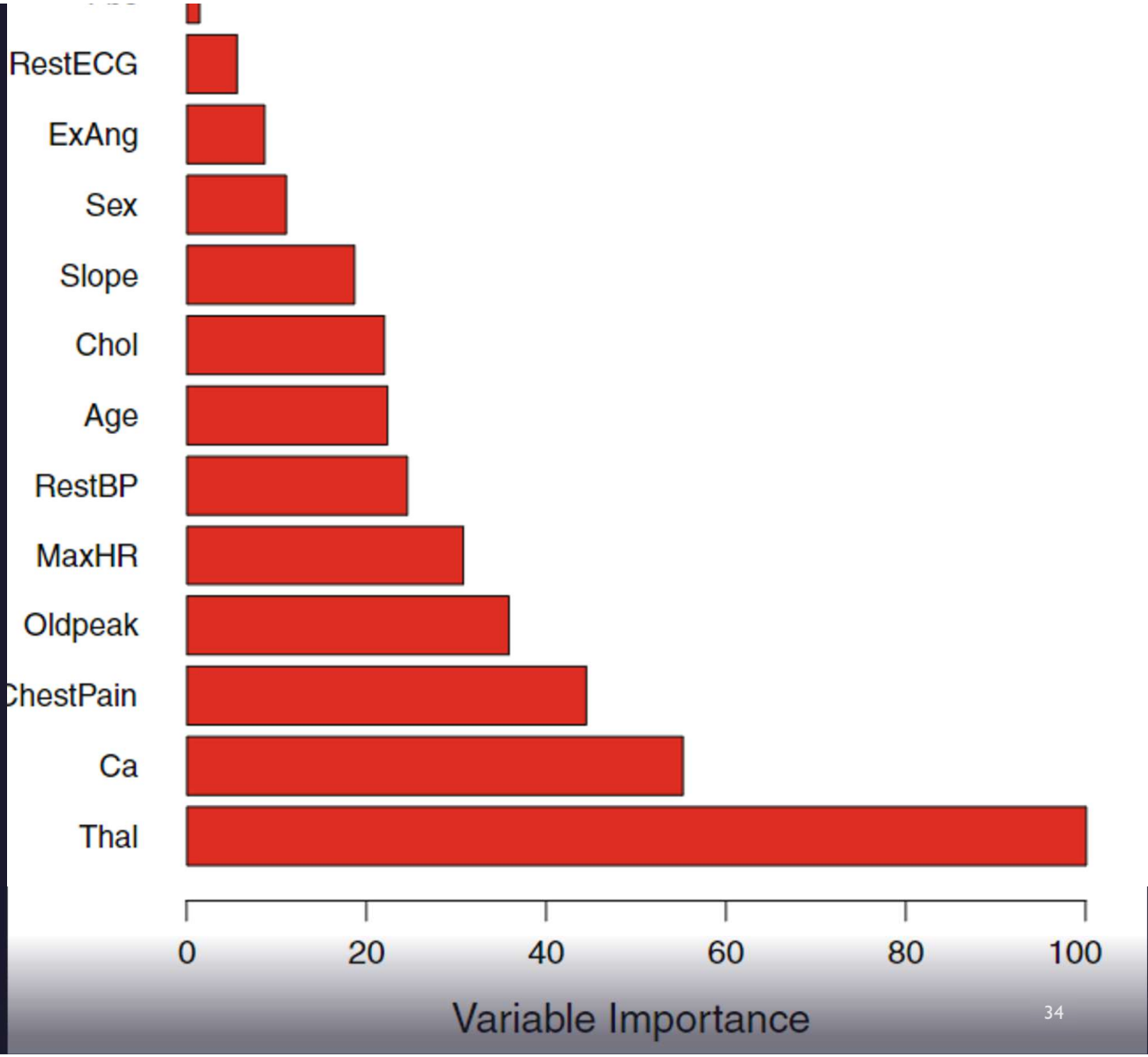
Random forest Feature importance

- Though bagged trees are hard to interpret as a whole, we can still summarize the importance of each predictor variable.
- For bagged regression trees, we calculate the total decrease in RSS due to splits on each variable, averaged over all trees.
- For bagged classification trees, we calculate the total decrease in the Gini index due to splits on each variable, averaged over all trees.
- **A larger value** indicates a **more important predictor**, as it leads to greater reductions in impurity or error when used for splits.
- The variable importance provide an overall summary of the predictors' contributions to the bagged model, though not for any individual tree.

Random forest Feature importance

```
from sklearn.ensemble import RandomForestClassifier

# Train random forest classifier
rfc = RandomForestClassifier(n_estimators=100, random_state=42)
rfc.fit(X, y)

# Get feature importances
importances = rfc.feature_importances_

# Print feature importances
for feature, importance in zip(X.columns, importances):
    print(f"{feature}: {importance}")
```

Feature importance



Extra-Trees

Extra tree (Extra randomness mean more bias less variance)

- Extra Trees is an ensemble learning method that uses decision trees to make predictions.
- Extra Trees is similar to Random Forests but can be faster due to the random selection of feature splits.
- Extra Trees creates many decision trees, each with a randomly sampled but without replacement dataset and a randomly selected subset of features.
- The most unique characteristic of Extra Trees is the random selection of a splitting value for a feature, which makes the trees diversified and uncorrelated.
- It is **computationally efficient** and can handle large datasets with reasonable resources(Why to use?)

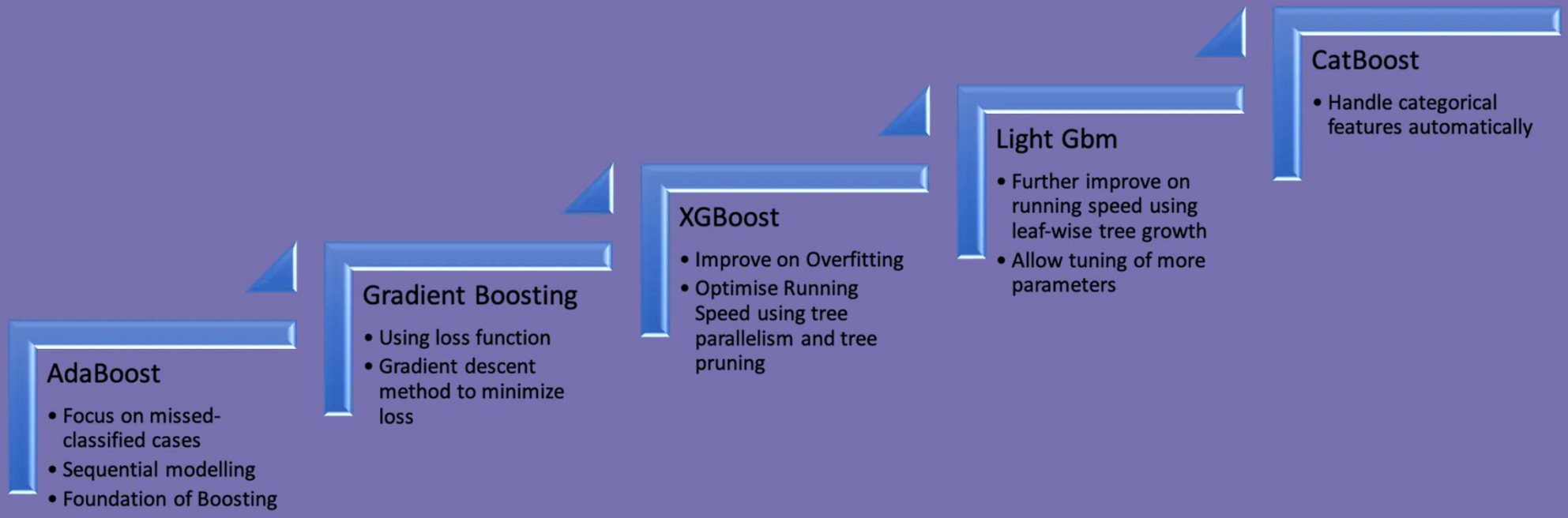
Extra tree (Extra randomness mean more bias less variance)

- Extra Trees is an ensemble learning method that uses decision trees to make predictions.
- Extra Trees is similar to Random Forests but can be faster due to the random selection of feature splits.
- Extra Trees creates many decision trees, each with a randomly sampled but without replacement dataset and a randomly selected subset of features.
- The most unique characteristic of Extra Trees is the random selection of a splitting value for a feature, which makes the trees diversified and uncorrelated.
- It is **computationally efficient** and can handle large datasets with reasonable resources(Why to use?)
- It can be less interpretable than other algorithms due to the randomness of the split

```python
from sklearn.ensemble import ExtraTreesClassifier


# Create an Extra Trees classifier with 100 trees
clf = ExtraTreesClassifier(n_estimators=100, random_state=42)

# Train the classifier on the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Evaluate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

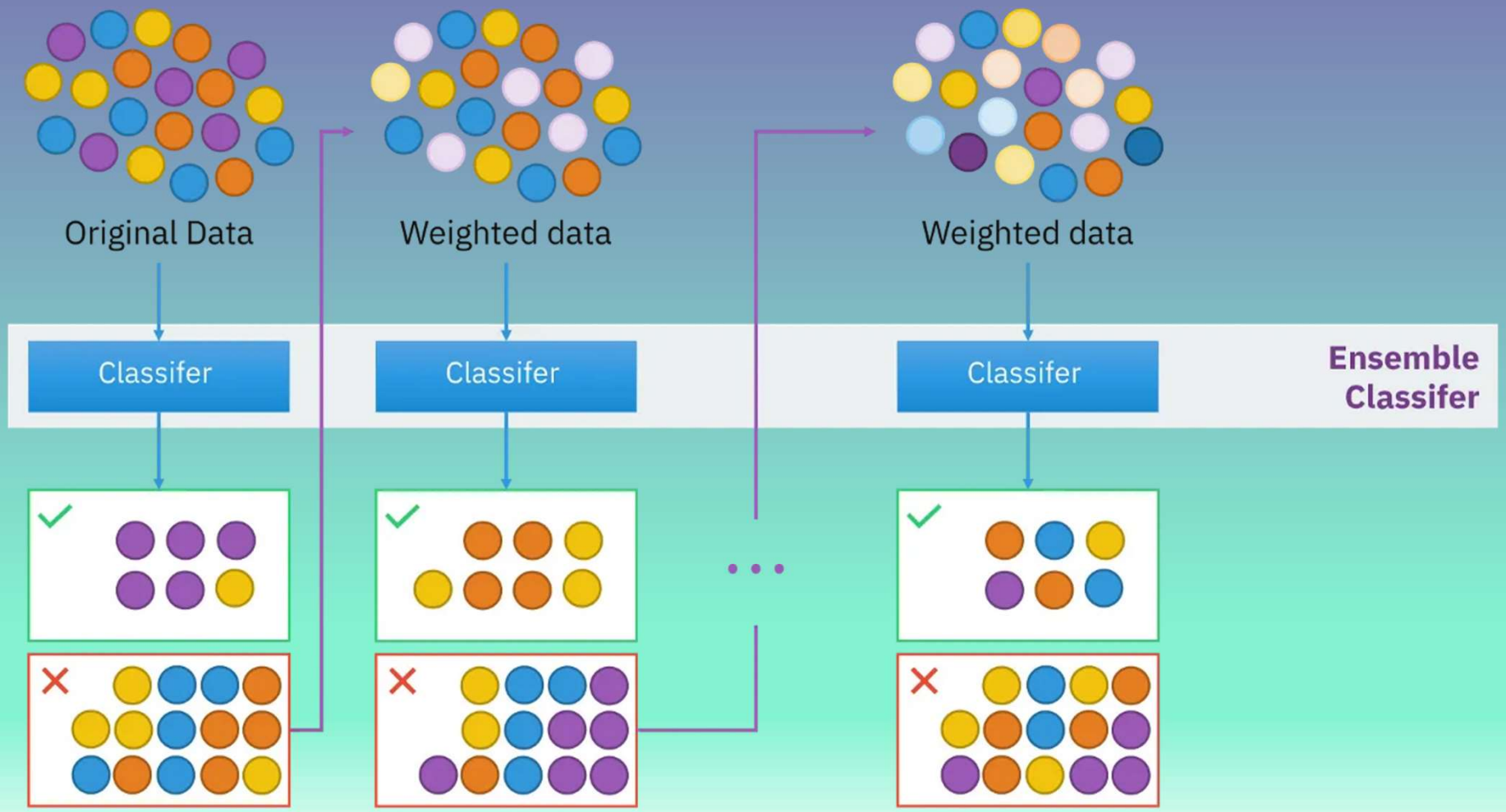
AdaBoost and Boosting algorithms

Boosting algorithms



AdaBoost

- Initialize the weights of each training example to be equal.
- Train a weak learner on the training data, and compute its error rate on the training data.
- Increase the weights of the training examples that the weak learner classified incorrectly, and decrease the weights of the training examples that it classified correctly.
- Repeat steps 2-3 for a set number of iterations or until the error rate becomes acceptably low.
- Combine the weak learners into a strong learner by assigning each one a weight based on its accuracy on the training data.
- To make a prediction on a new input, apply the weak learners to the input and combine their predictions using their assigned weights


Original Data
Weighted data
Weighted data
Classifer
Classifer
Classifer
Ensemble
Classifer

AdaBoost

```python
from sklearn.ensemble import AdaBoostClassifier

# Create an AdaBoost classifier with 100 weak learners
clf = AdaBoostClassifier(n_estimators=100, random_state=42)

# Train the classifier on the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Evaluate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

References

- https://machinelearningmastery.com/agentle-introduction-to-the-bootstrap-method/
- https://en.wikipedia.org/wiki/Ensemble_learning
- https://hossam-ahmed.notion.site/Articles-4d8f0c0a7af84bbb95816cc867e2163c?pvs=4
- https://hossam-ahmed.notion.site/8-Tree-based-Model c3a1186914ed40f7b4298dd5493f3fb3?pvs=4
- https://www.ibm.com/topics/random-forest
- https://orbi.uliege.be/bitstream/2268/9357/1/geurts-mlj-advance.pdf

Thank You