Ensemble Learning

Ensemble Learning

- Based on “wisdom of the crowd”
  - Ask 1000 people a question
  - Average (aggregate) their answers
  - The aggregate of the crowd will be better than the answer of an expert
- Ensemble learning aggregates predictions from a group of predictors
- The aggregate prediction will be better than the prediction of the best individual model

Example

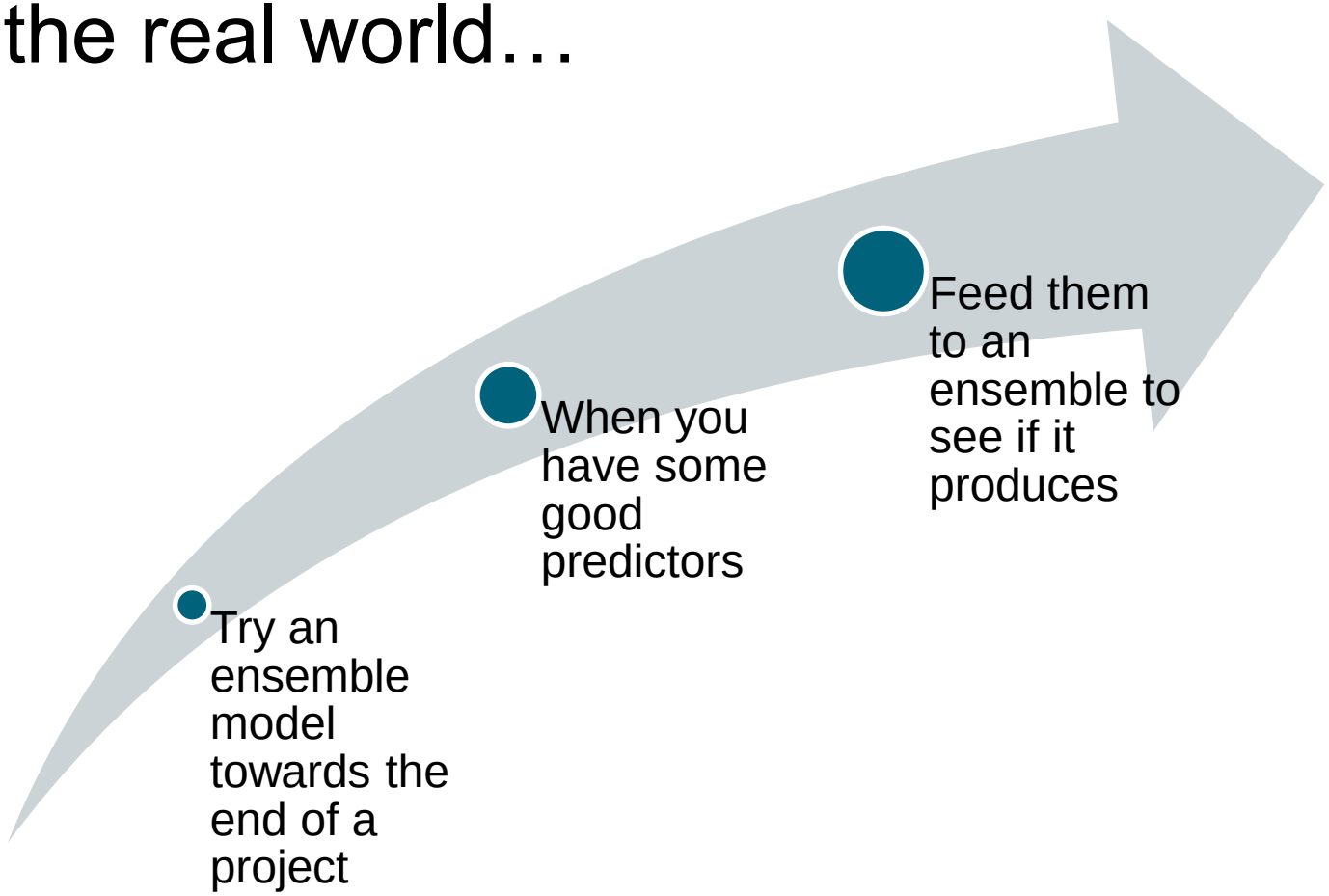
Train a group of decision tree classifiers each on different random subsets of the training data

Obtain the predictions of all the individual trees and predict the class that gets the most votes

This is an example of random forest since it uses an ensemble of decision trees

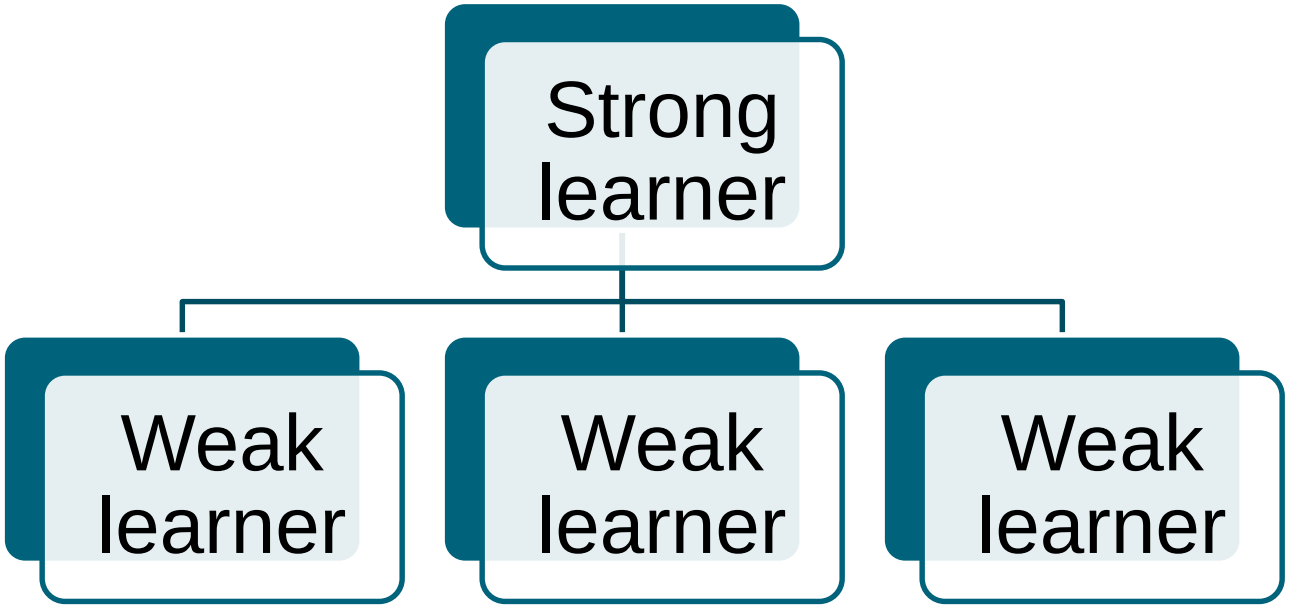
The Bigger Picture

- In the real world…



Something to Think About…

- Even if each classifier is a weak learner, the ensemble might become a strong one (meaning that it achieves higher accuracy)



How Can an Aggregate Become a Strong Learner?

- Each base predictor contributes to an estimate of the true probability of the set
- The more predictors you have the closer you will get to the true probability

This is called the law of large numbers

Implementation

- Scikit-learn has an ensemble module where all the ensemble classes are implemented
- Here is the link to the user guide (chapter 1.11) describing ensemble methods
  - 1.11 Ensemble Methods link

The End

Voting Classifiers

Voting classifier

- VotingClassifier() implements a voting classifier for unfitted estimators
- Paramaters for voting classifier include
  - Estimators: the list of estimators you want to include
  - Voting: hard or soft
    - Hard uses a straight majority vote based on base model predictions
    - Soft voting predicts the class based on class probability

Voting Classifier (cont.)

- Voting: hard or soft (cont.)
  - In order to use soft voting, the included base estimators have to be able to compute class probabilities
- Weights: allow you to specify a weight for each classifier that is then multiplied by the class probability
- n_jobs - the number of jobs to run in parallel

Voting Classifier Attributes

- Attributes
  - estimators_: gives you a list of the fitted base estimators defined in the estimators parameter
  - named_estimators: lets you access fitted sub-estimators by name
  - classes_: shows you the class labels
- There is a VotingRegressor class available as well

Two Ways to Improve Results

- There are two ways to improve results in ensemble methods
  - Choosing different classifiers
  - Choosing the same classifier on random subsets of data
- We've already covered the first option in voting classifiers
- Let's look at the second option, using random subsets of data now

The End

Bagging and Pasting

Bagging and Pasting

- Random sampling with replacement is called bagging, short for bootstrap aggregation
- Random sampling without replacement is called pasting
- Bagging allows an individual predictor to see a certain instance several times

Implementation

- For classification problems, scikit-learn has the BaggingClassifier() estimator
- For regression problems, scikit-learn has the BaggingRegressor() estimator
- Both of these estimators has a bootstrap parameter
  - Setting bootstrap to true implements bagging
  - Setting bootstrap to false implements pasting

Aggregation

- The aggregation calculated for bagging in a classification problem is the mode
- The aggregation calculated for bagging in a regression problem is the average
- The result of limiting individual predictors to train on training subsets increases bias (the number of errors) they experience
- But individual predictors will also have less variance on the random subset of data

Aggregation Results

- Aggregating predictions form multiple base estimators reduces both bias and variance
- Long story short:

An ensemble model will have similar bias but lower variance than a single model trained on the original data set

N-jobs

- One benefit of ensemble models is that base predictors can be trained in parallel by using different CPU cores at the same time
- This allows ensemble methods to scale very well

Other Parameters for the Bagging Estimators

- base-estimator
- n_estimators
- max_samples
- max_features
- bootstrap
- bootstrap_features
- oob_score

Voting for Bagging Methods

- Soft voting is the default voting method if the base estimator supports class probabilities

Out of Bag Evaluation

- If you are using bagging, some instances will be seen multiple times because of ”with replacement” sampling
- Some training instances, because of this, will not be seen
- The training instances not seen by a predictor during bagging can be used to evaluate the performance of that predictor
- This is called out-of-bag (oob) evaluation

Oob_score Parameter

- The oob-score parameter in a bagging classifier evaluates predictors on out-of-bag instances
- The value for oob_score is Boolean
- If it is set to true, it tells the classifier to produce oob_scores for each predictor
- It then averages these scores for the entire ensemble as well
- The result is available in the oob_score_variable when training is done

Sampling Features

- To have a good bagging result, it helps if the individual base estimators have some diversity
- We've talked about random sampling to help us with this
- We can also sample the features
- Two parameters control feature sampling:
  - max_features
  - Boostrap_features

Feature Parameters

Max_features

- Defines how many features are used to train each estimator

Bootstrap_features

- Defines whether features are drawn with replacement

Terminology

- When max_features is set to true but bootstrap is set to false this is called random subspaces
  - In this case, your diversity only comes by sampling the features, not the instances
- When max_features and bootstrap are set to true this is called random patches
  - In this case, both training instances and features are sampled

The End

Random Forests

Random Forest

- A convenient scikit-learn class for an ensemble of decision trees is called a random forest
- You implement it by calling sklearn.ensemble.RandomForest
- Random forests typically use bagging
- The default max_samples in a random forest is the size of the training set

Scikit-learn Random Forest Classes

- For regression problems use RandomForestRegressor
- For classification problems use RandomForestClassifier
- Trees are grown by searching for the best feature among a random subset of features
- This diversity results in higher bias and lower variance than if the model was trained on the entire input set

Feature Importances

- One of the outputs of random forest is a measure of the relative importance of each feature
- It looks at the nodes of each tree that use that feature and sees how that feature reduces impurity on average
- This average is scaled so all the feature importances add up to one

The End

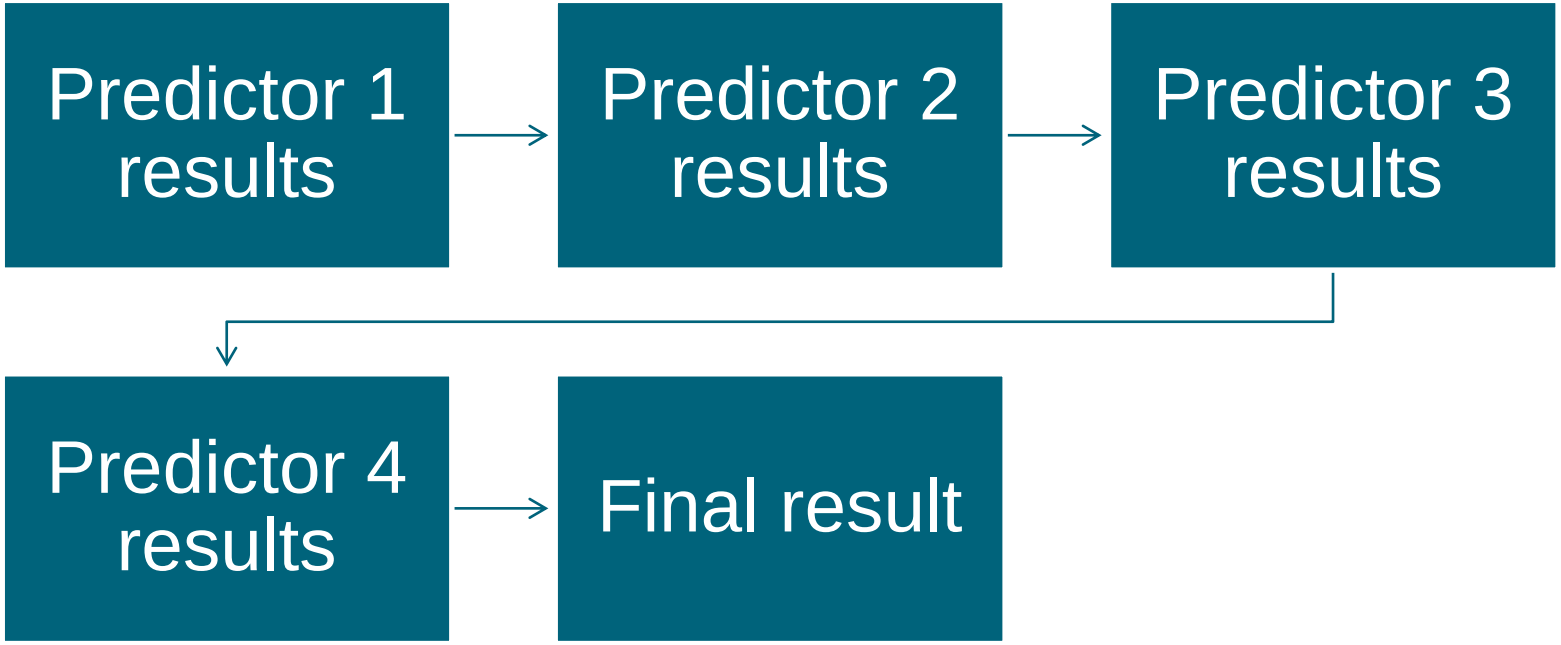
Boosting

Boosting

- Boosting is an ensemble method that combine multiple weak learners into a strong learner
- Instead of building entire trees as the base estimator, boosting uses stumps, which have one branching node and two leaf nodes off of it (max_depth = 1)
- These stumps are by definition weak learners

Boosting

- With boosting you train predictors sequentially
- New predictors correct older ones



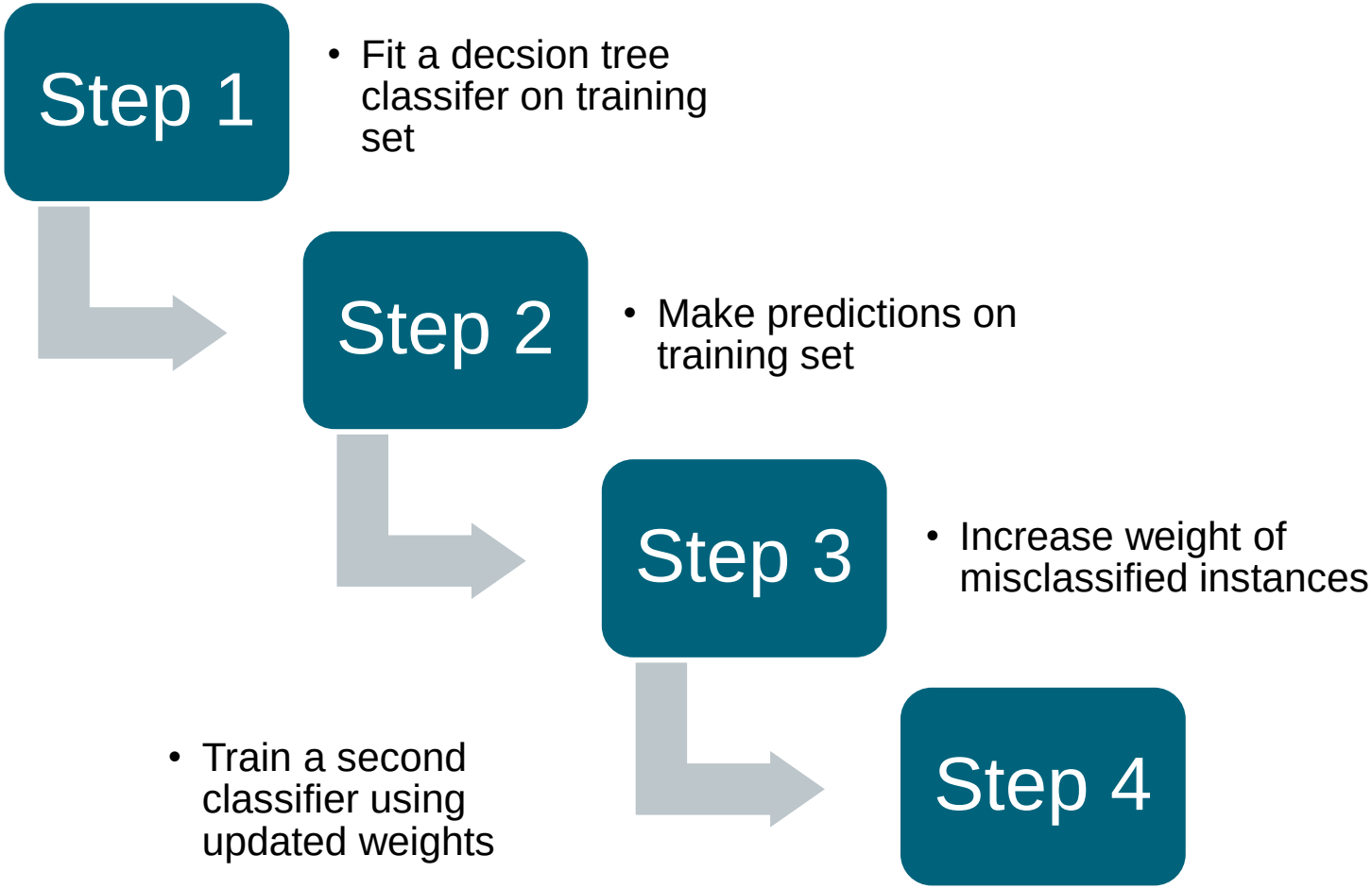
Boosting Algorithms

- There are two main boosting algorithms
  - Adaboost
    - Pays attention to underfitted training instances
  - Gradient boosting
    - Fits the new predictor to the residual errors of the previous predictor

Adaboost

- Adaboost improves previous predictors by looking at instances it got wrong
- Additional predictors added sequentially look for and find instances that are harder and harder to classify

Adaboost Process



Adaboost Base Classifiers

- As I stated before, the base trees trained by adaboost are called stumps
  - There max depth is one
- The second weight adaboost uses is a weight for each predictor
- Predictors with high accuracy get a higher weight

Prediction Time

- At prediction time:
  1. Adaboost computes the predictions of all the predictors
  2. Applies the weights to them
  3. Adds the weights votes to come up with a final aggregated prediction

Implementation

- From scikit-learn.ensemble import AdaboostClassifier
- From scikit-learn.ensemble import AdaboostRegressor

Regularization

- Ways to handle overfitting:
  - Reducing the number of estimators
  - Regularizing the base estimator

Gradient Boosting

- Instead of updating instance weights as it builds sequential predictions
- It builds a new predictor to the residual errors and targets of the initial one
- Gradient boosting is implemented in sklearn.ensemble
  - From sklearn.ensemble import GradientBoostingClassifier
  - From sklearn.ensemble import GradientBoostingRegressor

Gradient Boosting Parameters

- learning_rate scales the contribution of each tree
  - A lower value requires you to have more trees
- warm_start keeps the results of existing trees when the fit method is called
  - This allows incremental training
  - You can write a for loop to stop training when validation error does not improve for a certain number of iterations

XGBoost

- XGBoost is an open-source library that implements an optimized version of Gradient Boosting
- Here is a link for more information:
  - XFBoost Python Package Introduction link

The End

Stacking

Stacking

- In stacking, the aggregation of the predictions is handled by the model itself
- The portion of the model that trains the aggregation is called a blender
- The training set is broken into two subsets
- The first layer is trained on the first subset of data
- The first layer then makes predictions on the second subset of training data

Training the Blender

- A new training set is built using the predicted values as new input features in addition to the target features
- The blender is trained on this new dataset, so it learns to predict the target value based on the first layer’s predictions

The End