Ensemble Learning and Random Forests

Ensembles

- If we aggregate the predictions of a group of predictors, we will often get better predictions than with the best individual predictor.
- A group of predictors is called an ensemble, and aggregating individual predictors is called ensemble learning. An ensemble learning algorithm is called an ensemble method.
- For example, we can train a group of decision tree classifiers, each on a different subset of the training set, get their predictions, and get the class with the most votes. This ensemble is called a random forest.

Voting Classifiers

- We can aggregate the predictions of each classifier and get the majority vote. This classifier is a hard voting classifier.
- Even if each classifier is a weak learner, the ensemble can still be a strong learner.
    - There has to be a sufficient number of weak learners in the ensemble, and they must be sufficiently diverse to create a strong learner.
    - One way to get diverse learners is to train them using different algorithms.
- We can use Scikit-Learn's **VotingClassifier()** class, which takes in a list of name/predictor tuples.
    - Note that moving forward, every class or function mentioned originates from the Scikit-Learn API unless specified otherwise.
- The "estimators" attribute fits a clone of every predictor that was provided to it.
- Then, we can simply fit() and predict().
- Voting classifiers can also perform soft voting, where the algorithm predicts the highest class probability, averaged over all individual classifiers.
    - Soft voting often performs better than hard voting because it gives more weight to highly confident votes.
    - We simply set the voting parameter to "soft."

Bagging and Pasting

- Another way of getting diverse classifiers is to use the same training algorithm for every predictor but train them on different random subsets of the training set.

    - Bagging is sampling performed with replacement, also known as bootstrap aggregating.
    - Pasting is sampling performed without replacement.
- The aggregation function is typically the statistical mode for classification (i.e., the most frequent prediction, just like with a hard voting classifier) or the average for regression.
- We can use the **BaggingClassifier()** class (or **BaggingRegressor()** class for regression) with a DecisionTreeClassifier() as an input.
    - By default, this class performs bootstrapping. To perform pasting, we set the "bootstrap" parameter to false.
    - The bagging classifier also performs soft voting by default.
- An ensemble's predictions likely generalize much better than a single decision tree.
    - An ensemble has a similar amount of bias but a smaller variance.
    - Bagging introduces more bias than pasting, but the overall ensemble's variance is smaller.

Out-of-Bag Evaluation

- Not all training instances are sampled for any given predictor. The instances that are not sampled are called out-of-bag (OOB) instances.
- A bagging ensemble can be evaluated using OOB instances without a separate validation set.
- We can simply set the bagging classifier's "oob_score" attribute to true and check its evaluation score with the **oob_score_** attribute.
- The classifier also has an **oob_decision_function_** attribute to get the class probabilities for each instance.

Random Patches and Random Subspaces

- The bagging classifier can also sample features.
- We can use the "bootstrap_features" hyperparameter to train each predictor on a random subset of input features.
- Sampling both instances and features is called the random patches method.
- Keeping all training instances except the sampling features is called the random subspaces method.

Random Forests

- We can use **RandomForestClassifier()** to use an ensemble of decision trees.
    - There's also the **RandomForestRegressor()** class for regression tasks.
- The random forest algorithm searches for the best feature among a random subset of features.
    - It samples $\sqrt{n}$ features, where n is the total number of features.
    - This results in a higher bias but lower variance.
- The RandomForestClassifier() class has the same hyperparamaters as a DecisionTreeClassifier() and as the BaggingClassifer() class.

- We can find the most important features using the RandomForestClassifier's **feature_importances_** attribute.

Extra-Trees

- We can make trees even more random by using random thresholds for each feature.
  - We simply set the "splitter" attribute to "random" when creating a decision tree classifier.
  - These forests are called extremely randomized trees or extra-trees.
- Extra-Trees have a higher bias but lower variance than random forests.
- Extra-Tree classifiers are much faster than random forests because it simply uses a random threshold at each feature rather than finding the best possible threshold at each feature.
- We can use the **ExtraTreesClassifier()** class whose API is identical to the RandomForestClassifier() class.

Boosting

- Boosting refers to any ensemble method that can combine weak learners into a strong learner.
- Boosting methods train predictors sequentially, each trying to correct its predecessor.

AdaBoost

- An AdaBoost classifier pays more attention to the training instances that the predecessor underfit.
  - The algorithm first trains a base classifier, and then, it increases the relative weight of misclassified training instances.
  - The algorithm repeats this process a set number of times.
- We can use the **AdaBoostClassifier()** class using a base classifier, like a decision tree, as an input.
- An important drawback of AdaBoost is that it does not scale as well as bragging or pasting.
- If the AdaBoost ensemble overfits, we can try to reduce the number of estimators or regularizing the base estimator more.

Gradient Boosting

- This method tries to fit the new predictor to the residual errors of the predecessor rather than the relative weight of misclassified training instances.
- We can use the **GradientBoostingRegressor()** class or the **GradientBoostingClassifier()** class depending on the task.
  - These classes have a "learning_rate" hyperparameter that controls the contribution of each decision tree.
  - The the learning rate value is lower, then the algorithm needs more trees in the ensemble, but it will likely generalize better. This technique is called shrinkage.

    - If the learning rate value is too low, the model will likely underfit, and if it is set too high, it will likely overfit.
- To find the optimal number of trees, we can perform cross-validation using GridSearchCV() or RandomizedSearchCV().
    - We can also set the gradient booster class's "n_iter_no_change" attribute to an integer value, which will stop adding more trees during training if the last few trees did not help.
    - Then, we can simply access the optimal number of trees using the class's **n_estimators_** attribute.

Histogram-Based Gradient Boosting

- HGB bins the input features, replacing them with integers.
    - The number of bins is controlled by the "max_bins" hyperparameter, which defaults to 255 and cannot go any higher.
    - Working with integers allows the algorithm to use faster and more memory-efficient data structures.
    - This implementation has a complexity of O(b x m) where b is the number of bins and m is the number of training instances.
- We can use the **HistGradientBoostingRegressor()** class or the **HistGradientBoostingClassifier()** class.
    - Early stopping is automatically activated if the number of instances is greater than 10,000. This is controlled by setting the "early_stopping" hyperparameter to true or false.
- HGB classes support categorical features and missing values, which means that we can use an encoder to encode text as integers.

Stacking

- Stacked generalization trains a model to perform aggregation.
    - Each predictor predicts a different value, and the final predictor (the blender or meta learner) takes these predictions as inputs and makes the final prediction.
- The blending training set will contain one input feature per predictor.
- Once the algorithm trains the blender, it trains the base predictors one last time on the full original training set.
- We can have several layers, or stacks, of blenders, but this increases both training time and system complexity.
- We can use the **StackingClassifier()** class, which has a similar API as the VotingClassifier() class.
    - The StackingClassifier() class has a "final_estimator" attribute, which controls which estimator, such as a random forest, the algorithm uses as the blender.