MACHINE LEARNING MASTERY

Ensemble Learning Algorithms

FOR MACHINE LEARNING

7-Day Mini-Course

Jason Brownlee

Disclaimer

Ensemble Learning Algorithms With Python Crash Course

Edition: v1.11

Find the latest version of this guide online at: http://MachineLearningMastery.com

Contents

Before We Get Started...

Ensemble learning refers to machine learning models that combine the predictions from two or more models. Ensembles are an advanced approach to machine learning that are often used when the capability and skill of the predictions are more important than using a simple and understandable model. As such, they are often used by top and winning participants in machine learning competitions like the One Million Dollar Netflix Prize and Kaggle Competitions. Modern machine learning libraries like scikit-learn Python provide a suite of advanced ensemble learning methods that are easy to configure and use correctly without data leakage, a common concern when using ensemble algorithms. In this crash course, you will discover how you can get started and confidently bring ensemble learning algorithms to your predictive modeling project with Python in seven days. Let's get started.

Who Is This Crash-Course For?

Before we get started, let's make sure you are in the right place. This course is for developers that may know some applied machine learning. Maybe you know how to work through a predictive modeling problem end-to-end, or at least most of the main steps, with popular tools. The lessons in this course do assume a few things about you, such as: **You need to know**:

- You know your way around basic Python for programming.
- You may know some basic NumPy for array manipulation.
- You may know some basic scikit-learn for modeling.

You do NOT need to know:

- You do not need to be a math wiz!
- You do not need to be a machine learning expert!

This crash course will take you from a developer who knows a little machine learning to a developer who can effectively and competently apply ensemble learning algorithms on a predictive modeling project. Note that this crash course assumes you have a working Python 3 SciPy environment with at least NumPy installed. If you need help with your environment, you can follow the step-by-step tutorial here:

- How to Setup a Python Environment for Machine Learning

Crash-Course Overview

This crash course is broken down into seven lessons. You could complete one lesson per day (recommended) or complete all of the lessons in one day (hardcore). It really depends on the time you have available and your level of enthusiasm. Below is a list of the seven lessons that will get you started and productive with probability for machine learning in Python:

- **Lesson 01**: What Is Ensemble Learning?
- **Lesson 02**: Bagging Ensembles.
- **Lesson 03**: Random Forest Ensemble.
- **Lesson 04**: AdaBoost Ensemble.
- **Lesson 05**: Gradient Boosting Ensemble.
- **Lesson 06**: Voting Ensemble.
- **Lesson 07**: Stacking Ensemble.

Each lesson could take you 60 seconds or up to 30 minutes. Take your time and complete the lessons at your own pace. Ask questions and even share your results online. The lessons expect you to go off and find out how to do things. I will give you hints, but part of the point of each lesson is to force you to learn where to go to look for help on and about the best-of-breed tools in Python. (Hint: I have all of the answers directly on this blog; use the search box.) Share your results online, I'll cheer you on!

Hang in there, don't give up!

Lesson 01: What Is Ensemble Learning?

In this lesson, you will discover what ensemble learning is and why it is important. Applied machine learning often involves fitting and evaluating models on a dataset. Given that we cannot know which model will perform best on the dataset beforehand, this may involve a lot of trial and error until we find a model that performs well or best for our project. An alternate approach is to prepare multiple different models, then combine their predictions. This is called an ensemble machine learning model, or simply an ensemble, and the process of finding a well-performing ensemble model is referred to as *ensemble learning*. Although there is nearly an unlimited number of ways that this can be achieved, there are perhaps three classes of ensemble learning techniques that are most commonly discussed and used in practice. Their popularity is due in large part to their ease of implementation and success on a wide range of predictive modeling problems. They are:

- **Bagging**, e.g. bagged decision trees and random forest.
- **Boosting**, e.g. AdaBoost and gradient boosting
- **Stacking**, e.g. voting and using a meta-model.

There are two main reasons to use an ensemble over a single model, and they are related; they are:

- **Reliability**: Ensembles can reduce the variance of the predictions.
- **Skill**: Ensembles can achieve better performance than a single model.

These are both important concerns on a machine learning project and sometimes we may prefer one or both properties from a model.

Your Task

For this lesson, you must list three applications of ensemble learning. These may be famous examples, like a machine learning competition, or examples you have come across in tutorials, books, or research papers.

Next

In the next lesson, you will discover how to develop and evaluate a bagging ensemble.

Lesson 02: Bagging Ensembles

In this lesson, you will discover the bootstrap aggregation, or bagging, ensemble. Bagging works by creating samples of the training dataset and fitting a decision tree on each sample. The differences in the training datasets result in differences in the fit decision trees, and in turn, differences in predictions made by those trees. The predictions made by the ensemble members are then combined using simple statistics, such as voting or averaging. Key to the method is the manner in which each sample of the dataset is prepared to train ensemble members. Examples (rows) are drawn from the dataset at random, although with replacement. Replacement means that if a row is selected, it is returned to the training dataset for potential re-selection in the same training dataset. This is called a bootstrap sample, giving the technique its name. Bagging is available in scikit-learn via the `BaggingClassifier` and `BaggingRegressor` classes, which use a decision tree as the base-model by default and you can specify the number of trees to create via the `n_estimators` argument. The complete example of evaluating a bagging ensemble for classification is listed below.

```
# example of evaluating a bagging ensemble for classification
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import BaggingClassifier
# create the synthetic classification dataset
X, y = make_classification(random_state=1)
# configure the ensemble model
model = BaggingClassifier(n_estimators=50)
# configure the resampling method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate the ensemble on the dataset using the resampling method
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report ensemble performance
print('Mean Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

Listing 1: Example of a bagging ensemble for classification.

Your Task

For this lesson, you must run the example and review the results of the evaluated model. For bonus points, evaluate the effect of using more decision trees in the ensemble or even change the base learner that is used.

Next

In the next lesson, you will discover how to develop and evaluate a random forest ensemble.

Lesson 03: Random Forest Ensemble

In this lesson, you will discover the random forest ensemble. Random forest is an extension of the bagging ensemble. Like bagging, the random forest ensemble fits a decision tree on different bootstrap samples of the training dataset. Unlike bagging, random forest will also sample the features (columns) of each dataset. Specifically, split points are chosen in the data while constructing each decision tree. Rather than considering all features when choosing a split point, random forest limits the features to a random subset of features, such as 3 if there were 10 features.

The random forest ensemble is available in scikit-learn via the `RandomForestClassifier` and `RandomForestRegressor` classes. You can specify the number of trees to create via the `n_estimators` argument and the number of randomly selected features to consider at each split point via the `max_features` argument, which is set to the square root of the number of features in your dataset by default. The complete example of evaluating a random forest ensemble for classification is listed below.

```
# example of evaluating a random forest ensemble for classification
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import RandomForestClassifier
# create the synthetic classification dataset
X, y = make_classification(random_state=1)
# configure the ensemble model
model = RandomForestClassifier(n_estimators=50)
# configure the resampling method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate the ensemble on the dataset using the resampling method
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report ensemble performance
print('Mean Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

Listing 2: Example of a random forest ensemble for classification.

Your Task

For this lesson, you must run the example and review the results of the evaluated model. For bonus points, evaluate the effect of using more decision trees in the ensemble or tuning the number of features to consider at each split point.

Next

In the next lesson, you will discover how to develop and evaluate an AdaBoost ensemble.

Lesson 04: AdaBoost Ensemble

In this lesson, you will discover the adaptive boosting or AdaBoost ensemble. Boosting involves adding models sequentially to the ensemble where new models attempt to correct the errors made by prior models already added to the ensemble. As such, the more ensemble members that are added, the fewer errors the ensemble is expected to make, at least to a limit supported by the data and before overfitting the training dataset. The idea of boosting was first developed as a theoretical idea, and the AdaBoost algorithm was the first successful approach to realizing a boosting-based ensemble algorithm.

AdaBoost works by fitting decision trees on versions of the training dataset weighted so that the tree pays more attention to examples (rows) that the prior members got wrong, and less attention to those that the prior models got correct. Rather than full decision trees, AdaBoost uses very simple trees that make a single decision on one input variable before making a prediction. These short trees are referred to as decision stumps. AdaBoost is available in scikit-learn via the `AdaBoostClassifier` and `AdaBoostRegressor` classes, which use a decision tree (decision stump) as the base-model by default and you can specify the number of trees to create via the `n_estimators` argument. The complete example of evaluating an AdaBoost ensemble for classification is listed below.

```
# example of evaluating an AdaBoost ensemble for classification
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import AdaBoostClassifier
# create the synthetic classification dataset
X, y = make_classification(random_state=1)
# configure the ensemble model
model = AdaBoostClassifier(n_estimators=50)
# configure the resampling method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate the ensemble on the dataset using the resampling method
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report ensemble performance
print('Mean Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

Listing 3: Example of an AdaBoost ensemble for classification.

Your Task

For this lesson, you must run the example and review the results of the evaluated model. For bonus points, evaluate the effect of using more decision trees in the ensemble or even change the base learner that is used (note, it must support weighted training data).

Next

In the next lesson, you will discover how to develop and evaluate a gradient boosting ensemble.

Lesson 05: Gradient Boosting Ensemble

In this lesson, you will discover the gradient boosting ensemble. Gradient boosting is a framework for boosting ensemble algorithms and an extension to AdaBoost. It re-frames boosting as an additive model under a statistical framework and allows for the use of arbitrary loss functions to make it more flexible and loss penalties (shrinkage) to reduce overfitting. Gradient boosting also introduces ideas of bagging to the ensemble members, such as sampling of the training dataset rows and columns, referred to as stochastic gradient boosting. It is a very successful ensemble technique for structured or tabular data, although it can be slow to fit a model given that models are added sequentially. More efficient implementations have been developed, such as the popular extreme gradient boosting (XGBoost) and light gradient boosting machines (LightGBM).

Gradient boosting is available in scikit-learn via the `GradientBoostingClassifier` and `GradientBoostingRegressor` classes, which use a decision tree as the base-model by default. You can specify the number of trees to create via the `n_estimators` argument and the learning rate that controls the contribution from each tree via the `learning_rate` argument that defaults to 0.1. The complete example of evaluating a gradient boosting ensemble for classification is listed below.

```
# example of evaluating a gradient boosting ensemble for classification
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import GradientBoostingClassifier
# create the synthetic classification dataset
X, y = make_classification(random_state=1)
# configure the ensemble model
model = GradientBoostingClassifier(n_estimators=50)
# configure the resampling method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate the ensemble on the dataset using the resampling method
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report ensemble performance
print('Mean Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

Listing 4: Example of a gradient boosting ensemble for classification.

Your Task

For this lesson, you must run the example and review the results of the evaluated model. For bonus points, evaluate the effect of using more decision trees in the ensemble or try different learning rate values.

Next

In the next lesson, you will discover how to develop and evaluate a voting ensemble.

Lesson 06: Voting Ensemble

In this lesson, you will discover the voting ensemble. Voting ensembles use simple statistics to combine the predictions from multiple models. Typically, this involves fitting multiple different model types on the same training dataset, then calculating the average prediction in the case of regression or the class label with the most votes for classification, called hard voting. Voting can also be used when predicting the probability of class labels on classification problems by summing predicted probabilities and selecting the label with the largest summed probability. This is called soft voting and is preferred when the base-models used in the ensemble natively support predicting class probabilities as it can result in better performance.

Voting ensembles are available in scikit-learn via the `VotingClassifier` and `VotingRegressor` classes. A list of base-models can be provided as an argument to the model and each model in the list must be a tuple with a name and the model, e.g. `('lr', LogisticRegression())`. The type of voting used for classification can be specified via the `voting` argument and set to either `'soft'` or `'hard'`. The complete example of evaluating a voting ensemble for classification is listed below.

```
# example of evaluating a voting ensemble for classification
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import VotingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
# create the synthetic classification dataset
X, y = make_classification(random_state=1)
# configure the models to use in the ensemble
models = [('lr', LogisticRegression()), ('nb', GaussianNB())]
# configure the ensemble model
model = VotingClassifier(models, voting='soft')
# configure the resampling method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate the ensemble on the dataset using the resampling method
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report ensemble performance
print('Mean Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

Listing 5: Example of a voting ensemble for classification.

Your Task

For this lesson, you must run the example and review the results of the evaluated model. For bonus points, evaluate the effect of trying different types of models in the ensemble or even change the type of voting from soft voting to hard voting.

Next

In the next lesson, you will discover how to develop and evaluate a stacking ensemble.

Lesson 07: Stacking Ensemble

In this lesson, you will discover the stacked generalization or stacking ensemble. Stacking involves combining the predictions of multiple different types of base-models, much like voting. The important difference from voting is that another machine learning model is used to learn how to best combine the predictions of the base-models. This is often a linear model, such as a linear regression for regression problems or logistic regression for classification, but can be any machine learning model you like. The meta-model is trained on the predictions made by base-models on out-of-sample data. This involves using k-fold cross-validation for each base-model and storing all of the out-of-fold predictions. The base-models are then trained on the entire training dataset, and the meta-model is trained on the out-of-fold predictions and learns which model to trust, the degree to trust them, and under which circumstances.

Although internally stacking uses k-fold cross-validation to train the meta-model, you can evaluate stacking models any way you like, such as via a train-test split or k-fold cross-validation. The evaluation of the model is separate from this internal resampling-for-training process. Stacking ensembles are available in scikit-learn via the `StackingClassifier` and `StackingRegressor` classes. A list of base-models can be provided as an argument to the model and each model in the list must be a tuple with a name and the model, e.g. `('lr', LogisticRegression())`. The meta-learner can be specified via the `final_estimator` argument and the resampling strategy can be specified via the `cv` argument and can be simply set to an integer indicating the number of cross-validation folds. The complete example of evaluating a stacking ensemble for classification is listed below.

```
# example of evaluating a stacking ensemble for classification
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import StackingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
# create the synthetic classification dataset
X, y = make_classification(random_state=1)
# configure the models to use in the ensemble
models = [('knn', KNeighborsClassifier()), ('tree', DecisionTreeClassifier())]
# configure the ensemble model
model = StackingClassifier(models, final_estimator=LogisticRegression(), cv=3)
# configure the resampling method
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate the ensemble on the dataset using the resampling method
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
```

```
# report ensemble performance
print('Mean Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

Listing 6: Example of a stacking ensemble for classification.

Your Task

For this lesson, you must run the example and review the results of the evaluated model. For bonus points, evaluate the effect of trying different types of models in the ensemble and different meta-models to combine the predictions. This was the final lesson in the mini-course.

Final Word Before You Go...

You made it. Well done! Take a moment and look back at how far you have come. You discovered:

- What ensemble learning is and why you would use it on a predictive modeling project.
- How to use a bootstrap aggregation, or bagging, ensemble.
- How to use a random forest ensemble as an extension to bagging.
- How to use an adaptive boosting or AdaBoost ensemble.
- How to use a gradient boosting ensemble.
- How to combine the predictions of models using a voting ensemble.
- How to learn how to combine the predictions of models using a stacking ensemble.

This is just the beginning of your journey with ensemble learning. Keep practicing and developing your skills. Take the next step and check out my book on **Ensemble Learning Algorithms With Python**.

How Did You Go With The Crash-Course?

Did you enjoy this crash-course?
Do you have any questions or sticking points?

Let me know, send me an email at: **jason@MachineLearningMastery.com**

Take the Next Step

Looking for more help with Ensemble Learning?

Grab my new book:
Ensemble Learning Algorithms With Python
https://machinelearningmastery.com/ensemble-learning-algorithms-with-python/