**Combine Model Predictions Into Ensemble Predictions**

The three most popular methods for combining the predictions from different models are:

* **Bagging**. Building multiple models (typically of the same type) from different subsamples of the training dataset.
* **Boosting**. Building multiple models (typically of the same type) each of which learns to fix the prediction errors of a prior model in the chain.
* **Voting**. Building multiple models (typically of differing types) and simple statistics (like calculating the mean) are used to combine predictions.

This post will not explain each of these methods.

It assumes you are generally familiar with machine learning algorithms and ensemble methods and that you are looking for information on how to create ensembles in Python.

Each ensemble algorithm is demonstrated using 10 fold cross validation, a standard technique used to estimate the performance of any machine learning algorithm on unseen data.

## Bagging Algorithms

Bootstrap Aggregation or bagging involves taking multiple samples from your training dataset (with replacement) and training a model for each sample.

The final output prediction is averaged across the predictions of all of the sub-models.

The three bagging models covered in this section are as follows:

1. Bagged Decision Trees
2. Random Forest
3. Extra Trees

### 1. Bagged Decision Trees

Bagging performs best with algorithms that have high variance. A popular example are decision trees, often constructed without pruning.

In the example below see an example of using the BaggingClassifier with the Classification and Regression Trees algorithm ([DecisionTreeClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html)). A total of 100 trees are created.

# Bagged Decision Trees for Classification

import pandas

from sklearn import model\_selection

from sklearn.ensemble import BaggingClassifier

from sklearn.tree import DecisionTreeClassifier

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe = pandas.read\_csv(url, names=names)

array = dataframe.values

X = array[:,0:8]

Y = array[:,8]

seed = 7

kfold = model\_selection.KFold(n\_splits=10, random\_state=None)

cart = DecisionTreeClassifier()

num\_trees = 100

model = BaggingClassifier(base\_estimator=cart, n\_estimators=num\_trees, random\_state=None)

results = model\_selection.cross\_val\_score(model, X, Y, cv=kfold)

print(results.mean())

output: 0.7642686261107313

### 2. Random Forest

Random forest is an extension of bagged decision trees.

Samples of the training dataset are taken with replacement, but the trees are constructed in a way that reduces the correlation between individual classifiers. Specifically, rather than greedily choosing the best split point in the construction of the tree, only a random subset of features are considered for each split.

You can construct a Random Forest model for classification using of.

The example below provides an example of Random Forest for classification with 100 trees and split points chosen from a random selection of 3 features.

# Random Forest Classification

import pandas

from sklearn import model\_selection

from sklearn.ensemble import RandomForestClassifier

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe = pandas.read\_csv(url, names=names)

array = dataframe.values

X = array[:,0:8]

Y = array[:,8]

seed = 7

num\_trees = 100

max\_features = 3

kfold = model\_selection.KFold(n\_splits=10, random\_state=None)

model = RandomForestClassifier(n\_estimators=num\_trees, max\_features=max\_features)

results = model\_selection.cross\_val\_score(model, X, Y, cv=kfold)

print(results.mean())

output: 0.7799384825700615

### 3. Extra Trees

Extra Trees are another modification of bagging where random trees are constructed from samples of the training dataset.

You can construct an Extra Trees model for classification using the [ExtraTreesClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html) class.

The example below provides a demonstration of extra trees with the number of trees set to 100 and splits chosen from 7 random features.

# Extra Trees Classification

import pandas

from sklearn import model\_selection

from sklearn.ensemble import ExtraTreesClassifier

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe = pandas.read\_csv(url, names=names)

array = dataframe.values

X = array[:,0:8]

Y = array[:,8]

seed = 7

num\_trees = 100

max\_features = 7

kfold = model\_selection.KFold(n\_splits=10, random\_state=None)

model = ExtraTreesClassifier(n\_estimators=num\_trees, max\_features=max\_features)

results = model\_selection.cross\_val\_score(model, X, Y, cv=kfold)

print(results.mean())

Output: 0.7564422419685577

## Boosting Algorithms

Boosting ensemble algorithms creates a sequence of models that attempt to correct the mistakes of the models before them in the sequence.

Once created, the models make predictions which may be weighted by their demonstrated accuracy and the results are combined to create a final output prediction.

The two most common boosting ensemble machine learning algorithms are:

1. AdaBoost
2. Stochastic Gradient Boosting

1. AdaBoost

AdaBoost was perhaps the first successful boosting ensemble algorithm. It generally works by weighting instances in the dataset by how easy or difficult they are to classify, allowing the algorithm to pay or or less attention to them in the construction of subsequent models.

You can construct an AdaBoost model for classification using the [AdaBoostClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html) class.

The example below demonstrates the construction of 30 decision trees in sequence using the AdaBoost algorithm.

# AdaBoost Classification

import pandas

from sklearn import model\_selection

from sklearn.ensemble import AdaBoostClassifier

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe = pandas.read\_csv(url, names=names)

array = dataframe.values

X = array[:,0:8]

Y = array[:,8]

seed = 7

num\_trees = 30

kfold = model\_selection.KFold(n\_splits=10, random\_state=None)

model = AdaBoostClassifier(n\_estimators=num\_trees, random\_state=None)

results = model\_selection.cross\_val\_score(model, X, Y, cv=kfold)

print(results.mean())

output: 0.760457963089542

### 2. Stochastic Gradient Boosting

Stochastic Gradient Boosting (also called Gradient Boosting Machines) are one of the most sophisticated ensemble techniques. It is also a technique that is proving to be perhaps of the the best techniques available for improving performance via ensembles.

You can construct a Gradient Boosting model for classification using the [GradientBoostingClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html) class.

The example below demonstrates Stochastic Gradient Boosting for classification with 100 trees.

# Stochastic Gradient Boosting Classification

import pandas

from sklearn import model\_selection

from sklearn.ensemble import GradientBoostingClassifier

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe = pandas.read\_csv(url, names=names)

array = dataframe.values

X = array[:,0:8]

Y = array[:,8]

seed = 7

num\_trees = 100

kfold = model\_selection.KFold(n\_splits=10, random\_state=None)

model = GradientBoostingClassifier(n\_estimators=num\_trees, random\_state=None)

results = model\_selection.cross\_val\_score(model, X, Y, cv=kfold)

print(results.mean())

output: 0.7681989063568012

## Voting Ensemble

Voting is one of the simplest ways of combining the predictions from multiple machine learning algorithms.

It works by first creating two or more standalone models from your training dataset. A Voting Classifier can then be used to wrap your models and average the predictions of the sub-models when asked to make predictions for new data.

The predictions of the sub-models can be weighted, but specifying the weights for classifiers manually or even heuristically is difficult. More advanced methods can learn how to best weight the predictions from submodels, but this is called [stacking](https://machinelearningmastery.com/stacking-ensemble-for-deep-learning-neural-networks/) (stacked generalization) and is currently not provided in scikit-learn.

You can create a voting ensemble model for classification using the [VotingClassifier](http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html) class.

The code below provides an example of combining the predictions of logistic regression, classification and regression trees and support vector machines together for a classification problem.

# Voting Ensemble for Classification

import pandas

from sklearn import model\_selection

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.ensemble import VotingClassifier

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"

names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataframe = pandas.read\_csv(url, names=names)

array = dataframe.values

X = array[:,0:8]

Y = array[:,8]

seed = 7

kfold = model\_selection.KFold(n\_splits=10, random\_state=seed)

# create the sub models

estimators = []

model1 = LogisticRegression()

estimators.append(('logistic', model1))

model2 = DecisionTreeClassifier()

estimators.append(('cart', model2))

model3 = SVC()

estimators.append(('svm', model3))

# create the ensemble model

ensemble = VotingClassifier(estimators)

results = model\_selection.cross\_val\_score(ensemble, X, Y, cv=kfold)

print(results.mean())

Output: 0.7669002050580997