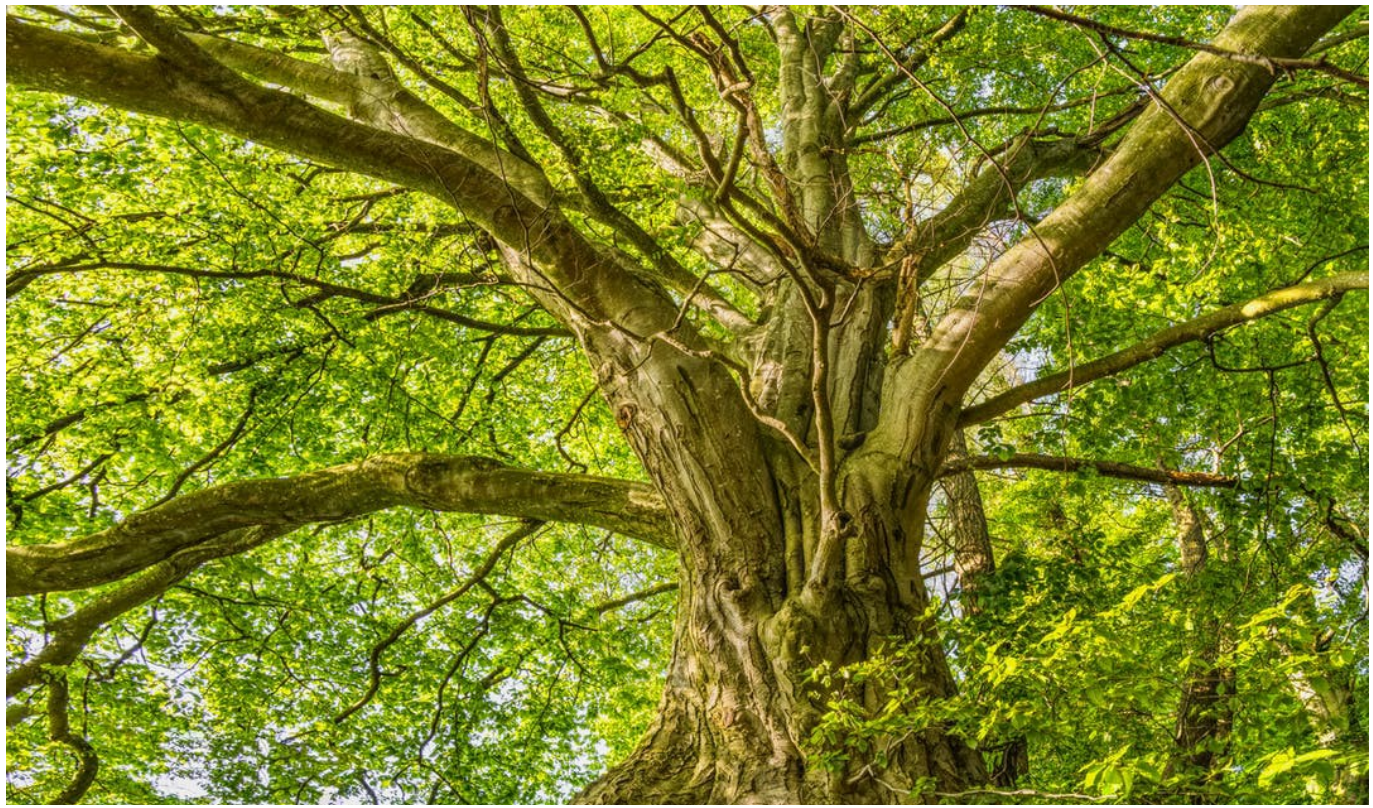
Implementing a Random Forest Classification Model in Python

Jake Huneycutt Follow

May 18, 2018 · 5 min read



Random forests algorithms are used for classification and regression. The random forest is an ensemble learning method, composed of multiple decision trees. By averaging out the impact of several decision trees, random forests tend to improve prediction.

There are many different models available to make predictions on classification data. Logistic regression is one of the most common for

binomial data. Other methodologies include support vector machines ("SVMs"), naive Bayes, and k-nearest neighbors. Random forests tend to shine in scenarios where a model has a large number of features that individually have weak predicative power but much stronger power collectively[1].

In this article, I'll give you a quick guide on how to implement a random forest model in Python for classification problems.

Loading Data

In order to understand how to implement a random forest model in Python, we'll do a very simple example with the Pima Indians diabetes data set. You can find it from numerous sources, or you can copy the code below.

```
import pandas as pd


# list for column headers
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi',
'age', 'class']


# open file with pd.read_csv
df =
pd.read_csv("https://raw.githubusercontent.com/jbrownlee/Datasets/mas
```

```
ter/pima-indians-diabetes.data.csv", names=names)
print(df.shape)


# print head of data set
print(df.head())
```

While the Pima Indian diabetes data set isn't necessarily an example of data best suited for random forest over all other models, the real goal here is just to walk through the process of implementing the model with a simple data set.

Creating a Random Forest Model

Let's create our model. We are trying to predict whether a patient has diabetes. This coincides with the 'class' column, which will be our independent variable. We'll use all the other columns as features for our model.

```
X = df.drop('class', axis=1)
y = df['class']
```

We'll use train-test-split to split the data into training data and testing data.

```
from sklearn.model_selection import train_test_split


# implementing train-test-split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=66)
```

Now, we can create the random forest model.

```
from sklearn import model_selection


# random forest model creation
rfc = RandomForestClassifier()
rfc.fit(X_train,y_train)


# predictions
rfc_predict = rfc.predict(X_test)
```

Let's next evaluate how the model performed.

Evaluating Performance

We'll import cross_val_score, classification_report, and confusion_matrix.

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
```

We'll also run cross-validation to get a better overview of the results.

```
rfc_cv_score = cross_val_score(rfc, X, y, cv=10, scoring='roc_auc')
```

Now, we'll print out the results.

```
print("=== Confusion Matrix ===")
print(confusion_matrix(y_test, rfc_predict))
print('\n')


print("=== Classification Report ===")
print(classification_report(y_test, rfc_predict))
print('\n')


print("=== All AUC Scores ===")
```

```
print(rfc_cv_score)
print('\n')


print("=== Mean AUC Score ===")
print("Mean AUC Score - Random Forest: ", rfc_cv_score.mean())
```

You should get a printout that looks something like this:

```
=== Confusion Matrix ===
[[153  23]
 [ 39  39]]


=== Classification Report ===
             precision    recall  f1-score   support

          0       0.80      0.87      0.83       176
          1       0.63      0.50      0.56        78

avg / total       0.75      0.76      0.75       254



=== All AUC Scores ===
[0.77703704 0.74407407 0.77814815 0.67962963 0.74481481 0.83777778
 0.83148148 0.88851852 0.77461538 0.87      ]


=== Mean AUC Score ===
Mean AUC Score - Random Forest:  0.7926096866096867
```

The confusion matrix is useful for giving you false positives and false negatives. The classification report tells you the accuracy of your model. The ROC curve plots out the true positive rate versus the false positive rate at various thresholds. The roc_auc scoring used in the cross-validation model shows the area under the ROC curve.

We'll evaluate our model's score based on the roc_auc score, which is .792. The next thing we should do is tune our hyperparameters to see if we can improve the performance of the model.

Tuning Hyperparameters

I'd recommend William Koehrsen's article, "Hyperparameter Tuning the Random Forest in Python" for a more detailed description of the process. We'll do a cliff-notes version.

We'll use RandomizedSearchCV from sklearn to optimize our hyperparamaters. Koehrsen uses a full grid of hyperparameters in his article, but I found that this could take a very substantial time to run in practice. I decided to focus on 3 hyperparameters: n_estimators, max_features, and max_depth.

```
from sklearn.model_selection import RandomizedSearchCV


# number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000,
num = 10)]


# number of features at every split
max_features = ['auto', 'sqrt']

# max depth
max_depth = [int(x) for x in np.linspace(100, 500, num = 11)]
max_depth.append(None)


# create random grid
random_grid = {
 'n_estimators': n_estimators,
 'max_features': max_features,
 'max_depth': max_depth
 }


# Random search of parameters
rfc_random = RandomizedSearchCV(estimator = rfc, param_distributions
= random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42,
n_jobs = -1)


# Fit the model
rfc_random.fit(X_train, y_train)


# print results
print(rfc_random.best_params_)
```

This should take a few minutes to run even with the slimmed down version. The print-out at the bottom shows us the best hyperparameters for our model.

```
[CV]  n_estimators=600, max_features=auto, max_depth=420, total=   1.4s
[CV] n_estimators=1400, max_features=auto, max_depth=380 .............
[CV]  n_estimators=1400, max_features=auto, max_depth=380, total=   3.1s
[CV] n_estimators=1400, max_features=auto, max_depth=380 .............
[CV]  n_estimators=1400, max_features=auto, max_depth=380, total=   3.2s
[CV]  n_estimators=1400, max_features=auto, max_depth=380, total=   2.4s
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  6.9min finished
{'n_estimators': 600, 'max_features': 'sqrt', 'max_depth': 300}
```

My results were: ‘n_estimators’ = 600; ‘max_features’ = ‘sqrt’; ‘max_depth’: 300. Now we can plug these back into the model to see if it improved our performance.

```
rfc = RandomForestClassifier(n_estimators=600, max_depth=300,
max_features='sqrt')
rfc.fit(X_train,y_train)
rfc_predict = rfc.predict(X_test)
rfc_cv_score = cross_val_score(rfc, X, y, cv=10, scoring='roc_auc')


print("=== Confusion Matrix ===")
print(confusion_matrix(y_test, rfc_predict))
print('\n')
```

```
print("=== Classification Report ===")
print(classification_report(y_test, rfc_predict))
print('\n')


print("=== All AUC Scores ===")


print(rfc_cv_score)
print('\n')


print("=== Mean AUC Score ===")
print("Mean AUC Score - Random Forest: ", rfc_cv_score.mean())
```

We can see from the output that there was a slight improvement in the results. Our roc_auc score improved from .793 to .829. The downside is that our number of false positives increased slightly (but false negatives declined).

```
=== Confusion Matrix ===
[[150  26]
 [ 34  44]]


=== Classification Report ===
             precision    recall  f1-score   support

          0       0.82      0.85      0.83       176
          1       0.63      0.56      0.59        78

avg / total       0.76      0.76      0.76       254
```

```
=== All AUC Scores ===
[0.79259259 0.82888889 0.83259259 0.73592593 0.81333333 0.85777778
 0.86851852 0.91074074 0.79884615 0.85384615]


=== Mean AUC Score ===
Mean AUC Score - Random Forest:  0.8293062678062677
```

And that's the quick and dirty version of how to implement a random forest model in Python. For more reading, check out Jake VanderPlas's excellent overview of decision trees and random forests in Python.

Sources and Additional Reading

1. *Dan Benyamin. "A Gentle Introduction to Random Forests, Ensembles, and Performance Metrics in a Commercial System."* Link.
2. *Niklas Dongas. "The Random Forest Algorithm."* Link.
3. *Jason Brownlee. "How to Implement Random Forest from Scratch in Python."* Link.

Machine Learning | Python | Random Forest | Data Science | Statistics