Hyperparameter tuning in python

HYPERPARAMETER TUNING IN PYTHON

Alex Scriven
Data Scientist

Introduction

Why study this course?

- New, complex algorithms with many hyperparameters
- Tuning can take a lot of time
- Develops deeper understanding beyond the default settings

You may be surprised what you find under the hood!

The dataset

The dataset relates to credit card defaults.

It contains variables related to the financial history of some consumers in Taiwan. It has 30,000 users and 24 attributes.

Our modeling target is whether they defaulted on their loan

It has already been preprocessed and at times we will take smaller samples to demonstrate a concept

Extra information about the dataset can be found here:

```
https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients
```

Parameters Overview

What is a parameter?

- Components of the model learned during the modeling process
- You **do not** set these manually (you can't in fact!)
- The algorithm will discover these for you

Parameters in Logistic Regression

A simple logistic regression model:

```
log_reg_clf = LogisticRegression()
log_reg_clf.fit(X_train, y_train)

print(log_reg_clf.coef_)
```

```
array([[-2.88651273e-06, -8.23168511e-03,  7.50857018e-04,
         3.94375060e-04,  3.79423562e-04,  4.34612046e-04,
         4.37561467e-04,  4.12107102e-04, -6.41089138e-06,
        -4.39364494e-06,  cont... ]])
```

Parameters in Logistic Regression

Tidy up the coefficients:

```
# Get the original variable names
original_variables = list(X_train.columns)

# Zip together the names and coefficients
zipped_together = list(zip(original_variables, log_reg_clf.coef_[0]))
coefs = [list(x) for x in zipped_together]

# Put into a DataFrame with column labels
coefs = pd.DataFrame(coefs, columns=["Variable", "Coefficient"])
```

Parameters in Logistic Regression

Now sort and print the top three coefficients

```
coefs.sort_values(by=["Coefficient"], axis=0, inplace=True, ascending=False)
print(coefs.head(3))
```

Variable	Coefficient
PAY_0	0.000751
PAY_5	0.000438
PAY_4	0.000435

Where to find Parameters

To find parameters we need:

1. To know a bit about the algorithm
2. Consult the Scikit Learn documentation

Parameters will be found under the 'Attributes' section, *not* the 'parameters' section!

Parameters in Random Forest

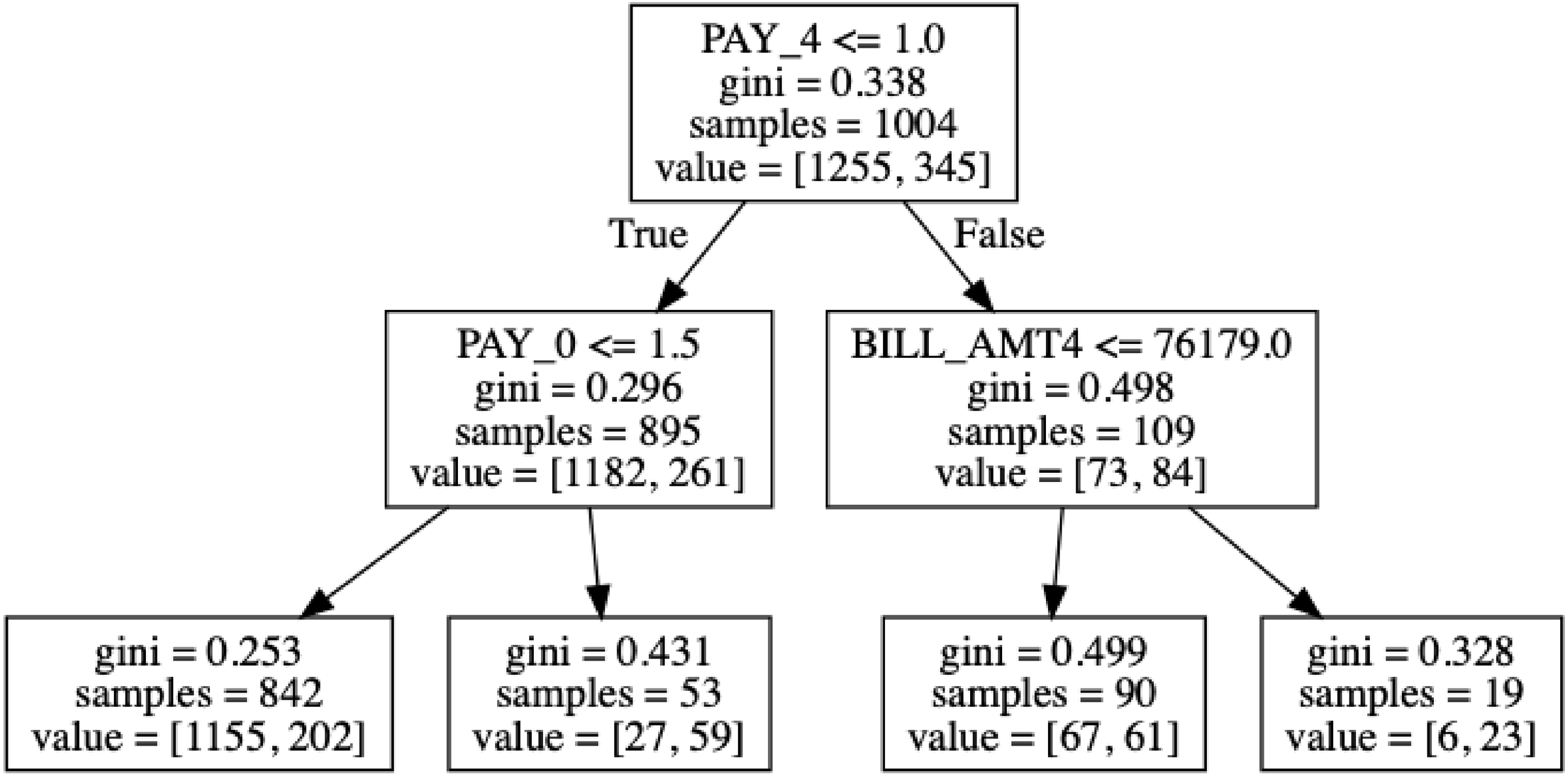
What about tree based algorithms?

Random forest has no coefficients, but node decisions (what feature and what value to split on).

```
# A simple random forest estimator
rf_clf = RandomForestClassifier(max_depth=2)
rf_clf.fit(X_train, y_train)

# Pull out one tree from the forest
chosen_tree = rf_clf.estimators_[7]
```

For simplicity we will show the final product (an image) of the decision tree. Feel free to explore the package used for this (graphviz & pydotplus) yourself.


PAY_4 <= 1.0
gini = 0.338
samples = 1004
value = [1255, 345]
True
False
PAY_0 <= 1.5
gini = 0.296
samples = 895
value = [1182, 261]
BILL_AMT4 <= 76179.0
gini = 0.498
samples = 109
value = [73, 84]
gini = 0.253
samples = 842
value = [1155, 202]
gini = 0.431
samples = 53
value = [27, 59]
gini = 0.499
samples = 90
value = [67, 61]
gini = 0.328
samples = 19
value = [6, 23]

Extracting Node Decisions

We can pull out details of the left, second-from-top node:

```
# Get the column it split on
split_column = chosen_tree.tree_.feature[1]
split_column_name = X_train.columns[split_column]

# Get the level it split on
split_value = chosen_tree.tree_.threshold[1]

print("This node split on feature {}, at a value of {}"
        .format(split_column_name, split_value))
```

> "This node split on feature PAY_0, at a value of 1.5"

Let's practice!

HYPERPARAMETER TUNING IN PYTHON

Hyperparameters Overview

HYPERPARAMETER TUNING IN PYTHON

Alex Scriven
Data Scientist

What is a hyperparameter

Hyperparameters:

- Something **you** set before the modelling process (like knobs on an old radio)
  - You also 'tune' your hyperparameters!
- The algorithm does not learn these

Hyperparameters in Random Forest

Create a simple random forest estimator and print it out:

```
rf_clf = RandomForestClassifier()
print(rf_clf)

RandomForestClassifier(n_estimators='warn', criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,bootstrap=True,
            class_weight=None, warm_start=False)
```

More info: http://scikit-learn.org

A single hyperparameter

Take the `n_estimators` parameter.

Data Type & Default Value:

> n_estimators : integer, optional (default=10)

Definition:

> The number of trees in the forest.

Setting hyperparameters

Set some hyperparameters at estimator creation:

```
rf_clf = RandomForestClassifier(n_estimators=100, criterion='entropy')
```

```
print(rf_clf)
RandomForestClassifier(n_estimators=100, criterion='entropy',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,bootstrap=True,
            class_weight=None, warm_start=False)
```

Hyperparameters in Logistic Regression

Find the hyperparameters of a Logistic Regression:

```
log_reg_clf = LogisticRegression()

print(log_reg_clf)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

There are less hyperparameters to tune with this algorithm!

Hyperparameter Importance

Some hyperparameters are more important than others.

Some will **not** help model performance:

For the random forest classifier:

- `n_jobs`
- `random_state`
- `verbose`

Not all hyperparameters make sense to 'train'

Random Forest: Important Hyperparameters

Some important hyperparameters:

- `n_estimators` (high value)
- `max_features` (try different values)
- `max_depth` & `min_sample_leaf` (important for overfitting)
- (maybe) `criterion`

Remember: this is only a guide

How to find hyperparameters that matter?

Some resources for learning this:

- Academic papers
- Blogs and tutorials from trusted sources (Like DataCamp!)
- The Scikit Learn module documentation
- Experience

Let's practice!

HYPERPARAMETER TUNING IN PYTHON

Hyperparameter Values

HYPERPARAMETER TUNING IN PYTHON

Alex Scriven
Data Scientist

Hyperparameter Values

Some hyperparameters are more important than others to begin tuning.

But which *values* to try for hyperparameters?

- Specific to each algorithm & hyperparameter
- Some best practice guidelines & tips do exist

Let's look at some top tips!

Conflicting Hyperparameter Choices

Be aware of conflicting hyperparameter choices.

- `LogisticRegression()` conflicting parameter options of `solver` & `penalty` that conflict.

> `The 'newton-cg', 'sag' and 'lbfgs' solvers support only l2 penalties.`

Some aren't explicit but will just 'ignore' (from `ElasticNet` with the `normalize` hyperparameter):

> `This parameter is ignored when fit_intercept is set to False`

Make sure to consult the Scikit Learn documentation!

Silly Hyperparameter Values

Be aware of setting 'silly' values for different algorithms:

- Random forest with low number of trees
  - Would you consider it a 'forest' with only 2 trees?
- 1 Neighbor in KNN algorithm
  - Averaging the 'votes' of one person doesn't sound very robust!
- Increasing a hyperparameter by a very small amount

Spending time documenting sensible values for hyperparameters is a valuable activity.

Automating Hyperparameter Choice

In the previous exercise, we built models as:

```
knn_5 =  KNeighborsClassifier(n_neighbors=5)
knn_10 = KNeighborsClassifier(n_neighbors=10)
knn_20  = KNeighborsClassifier(n_neighbors=20)
```

This is quite inefficient. Can we do better?

Automating Hyperparameter Tuning

Try a for loop to iterate through options:

```
neighbors_list = [3,5,10,20,50,75]

for test_number in neighbors_list:
    model = KNeighborsClassifier(n_neighbors=test_number)
    predictions = model.fit(X_train, y_train).predict(X_test)

    accuracy = accuracy_score(y_test, predictions)
    accuracy_list.append(accuracy)
```

Automating Hyperparameter Tuning

We can store the results in a DataFrame to view:

```
results_df = pd.DataFrame({'neighbors':neighbors_list, 'accuracy':accuracy_list})
print(results_df)
```

Neighbors	3	5	10	20	50	75
Accuracy	0.71	0.7125	0.765	0.7825	0.7825	0.7825

Learning Curves

Let's create a learning curve graph

We'll test many more values this time

```
neighbors_list = list(range(5,500, 5))

accuracy_list = []
for test_number in neighbors_list:
    model = KNeighborsClassifier(n_neighbors=test_number)
    predictions = model.fit(X_train, y_train).predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    accuracy_list.append(accuracy)
results_df = pd.DataFrame({'neighbors':neighbors_list, 'accuracy':accuracy_list})
```

Learning Curves

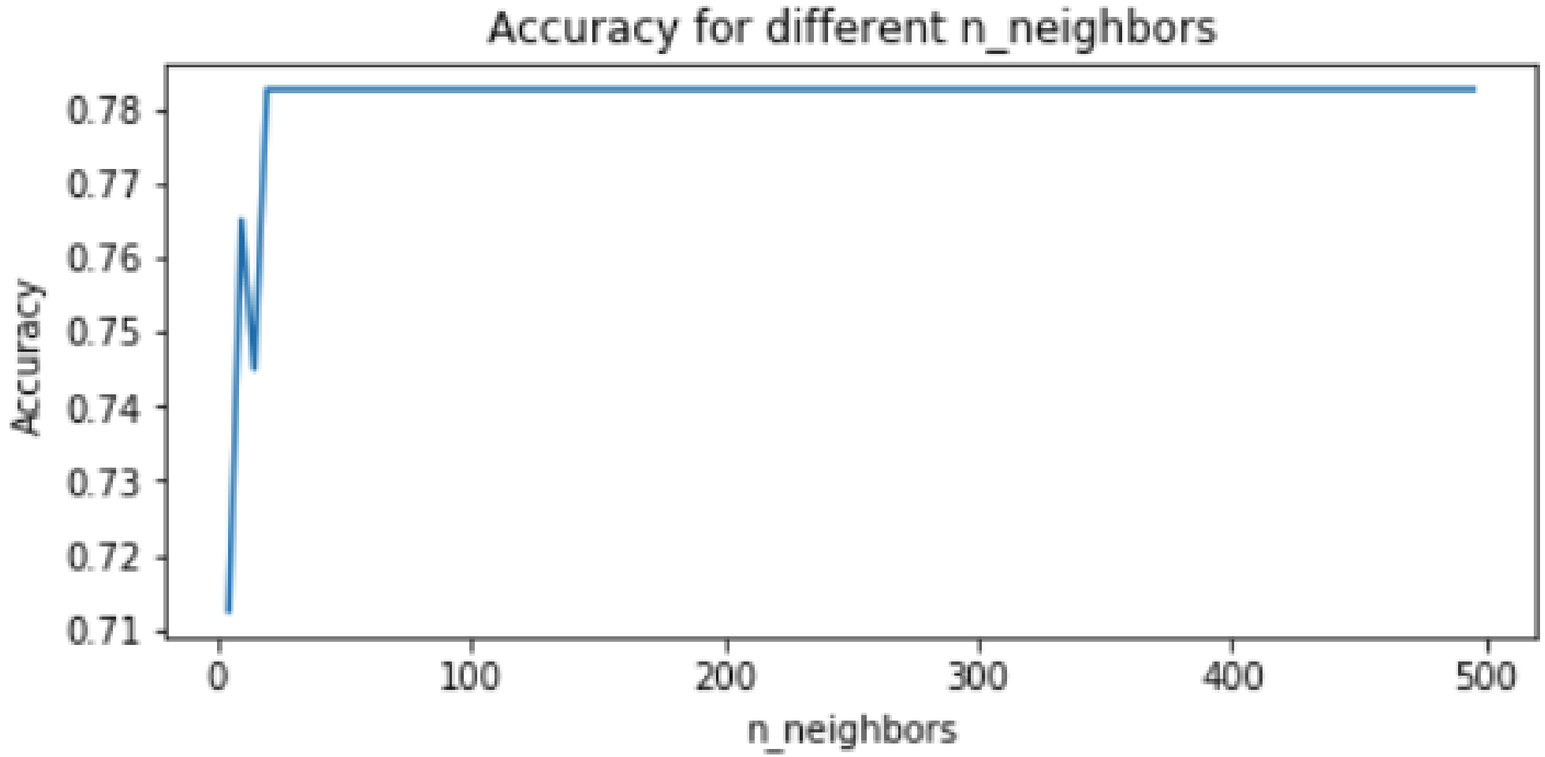
We can plot the larger DataFrame:

```
plt.plot(results_df['neighbors'],
    results_df['accuracy'])

# Add the labels and title
plt.gca().set(xlabel='n_neighbors', ylabel='Accuracy',
    title='Accuracy for different n_neighbors')
plt.show()
```

Learning Curves

Our graph:



A handy trick for generating values

Python's `range` function does not work for decimal steps.

A handy trick uses NumPy's `np.linspace(start, end, num)`

- Create a number of values (`num`) evenly spread within an interval (`start` , `end`) that you specify.

```
print(np.linspace(1,2,5))
```

```
[1. 1.25 1.5 1.75 2. ]
```

Let's practice!

HYPERPARAMETER TUNING IN PYTHON