

Introduction to Pipelines

 [_ \(https://github.com/learn-co-curriculum/dsc-pipelines-v2-1\)](https://github.com/learn-co-curriculum/dsc-pipelines-v2-1)  [_ \(https://github.com/learn-co-curriculum/dsc-pipelines-v2-1/issues/new/choose\)](https://github.com/learn-co-curriculum/dsc-pipelines-v2-1/issues/new/choose)

Introduction

You've learned a substantial number of different supervised learning algorithms. Now, it's time to learn about a handy tool used to integrate these algorithms into a single manageable pipeline.

Objectives

You will be able to:

- Explain how pipelines can be used to combine various parts of a machine learning workflow

Why Use Pipelines?

Pipelines are extremely useful tools to write clean and manageable code for machine learning. Recall how we start preparing our dataset: we want to clean our data, transform it, potentially use feature selection, and then run a machine learning algorithm. Using pipelines, you can do all these steps in one go!

Pipeline functionality can be found in scikit-learn's `Pipeline` module. Pipelines can be coded in a very simple way:

```
from sklearn.pipeline import Pipeline


# Create the pipeline
pipe = Pipeline([('mms', MinMaxScaler()),
                  ('tree', DecisionTreeClassifier(random_state=123))])
```

This pipeline will ensure that first we'll apply a Min-Max scaler on our data before fitting a decision tree. However, the `Pipeline()` function above is only defining the sequence of actions to perform. In order to actually fit the model, you need to call the `.fit()` method like so:

```
# Fit to the training data
pipe.fit(X_train, y_train)
```

Then, to score the model on test data, you can call the `.score()` method like so:

```
# Calculate the score on test data
pipe.score(X_test, y_test)
```

A really good blog post on the basic ideas of pipelines can be found [here](https://www.kdnuggets.com/2017/12/managing-machine-learning-workflows-scikit-learn-pipelines-part-1.html)  (<https://www.kdnuggets.com/2017/12/managing-machine-learning-workflows-scikit-learn-pipelines-part-1.html>).

Integrating Grid Search in Pipelines

Note that the above pipeline simply creates one pipeline for a training set, and evaluates on a test set. Is it possible to create a pipeline that performs grid search? And cross-validation? Yes, it is!

First, you define the pipeline in the same way as above. Next, you create a parameter grid. When this is all done, you use the function `GridSearchCV()`, which you've seen before, and specify the pipeline as the estimator and the parameter grid. You also have to define how many folds you'll use in your cross-validation.


```
# Create the pipeline
pipe = Pipeline([('mms', MinMaxScaler()),
                  ('tree', DecisionTreeClassifier(random_state=123))])

# Create the grid parameter
grid = [{'tree__max_depth': [None, 2, 6, 10],
        'tree__min_samples_split': [5, 10]}]

# Create the grid, with "pipe" as the estimator
gridsearch = GridSearchCV(estimator=pipe,
                           param_grid=grid,
                           scoring='accuracy',
                           cv=5)

# Fit using grid search
gridsearch.fit(X_train, y_train)

# Calculate the test score
gridsearch.score(X_test, y_test)
```

An article with a detailed workflow can be found [here](https://www.kdnuggets.com/2018/01/managing-machine-learning-workflows-scikit-learn-pipelines-part-2.html)  (<https://www.kdnuggets.com/2018/01/managing-machine-learning-workflows-scikit-learn-pipelines-part-2.html>).

Summary

Great, this wasn't too difficult! The proof of all this is in the pudding. In the next lab, you'll use this workflow to build pipelines applying classification algorithms you have learned so far in this module.

How do you feel about this lesson?



Have specific feedback?

[Tell us here! \(https://github.com/learn-co-curriculum/dsc-pipelines-v2-1/issues/new/choose\)](https://github.com/learn-co-curriculum/dsc-pipelines-v2-1/issues/new/choose)