HOW TO CREATE A MACHINE LEARNING PIPELINE

In this article, I'll show you how to create a machine learning pipeline. In this example, we'll use the scikit-learn machine learning framework (see our scikit-learn guide or browse the topics in the right-hand menu). However, the concept of a pipeline exists for most machine learning frameworks.

To follow along, the data is available here, and the code here.

(This article is part of our scikit-learn Guide. Use the right-hand menu to navigate.)

What is a machine learning pipeline?

Generally, a machine learning pipeline describes or models your ML process: writing code, releasing it to production, performing data extractions, creating training models, and tuning the algorithm. An ML pipeline should be a continuous process as a team works on their ML platform.

Machine learning programs involve a series of steps to get the data ready before feeding it into the ML model. Those steps can include:

- Reading the data and converting it to a Pandas dataframe
- Dropping or adding some columns
- Running some calculations over the columns
- Normalizing the data

You can use the Pipeline object to do this one step after another. Let's look at an example.

ML pipeline example using sample data

We have looked at this data from Trip Advisor before. As you can see, the data is a combination of text and numbers. In a machine learning model, all the inputs must be numbers (with some exceptions.) So, we will use a pipeline to do this as Step 1: converting data to numbers.

We'll also use the pipeline to perform Step 2: normalizing the data. That means for each data point x we calculate the new value z = x - (average) / (standard deviation). This makes all large numbers small, which is useful because ML models work best when the inputs are normalized.

	User country	Nr. reviews	Nr. hotel reviews	Helpful votes	Score	Period of stay	Traveler type	Pool	Gym	Tennis court	Spa	Casino	Free internet	Hotel name	H s
0	USA	11	4	13	5	Dec-Feb	Friends	NO	YES	NO	NO	YES	YES	Circus Circus Hotel & Casino Las Vegas	
1	USA	119	21	75	3	Dec-Feb	Business	NO	YES	NO	NO	YES	YES	Circus Circus Hotel & Casino Las Vegas	

We start by importing the data using Pandas. Then, make an array of the non-numeric columns that we will convert to numbers.

```
import pandas as pd
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('/home/ubuntu/Documents/TripAdvisor.csv',sep=',',header=0)

cols =
```

We do this by using the Pandas **factorize()** function. This takes text and converts that to categorical values, meaning just turn them into them into unique numbers.

The pipeline method works like this:

You stack up functions in the order that you want to run them. These are called **transformers.** Below we create a custom transformer then use one built into scikit-learn.

The format is:

Pipeline constructor with tuples of **('a descriptive name', a function).** You can pass arguments to the first function's **init()** method where it says **some args.** Each method must implement the **fit()** and **transform()** functions. Except the last function only implements **fit().**

```
pipeline = Pipeline()

data = pipeline.fit_transform(dataframe)
```

Here is the first function that we call. It uses the **BaseEstimator** and **TransforMixin** objects, which saves us writing some code. For example, it creates a **fit_transform()** method for us and creates getters and setters that we can use to pass in other parameters.

We pass in the columns we want to convert to numbers in the **init()** constructor. The **X** object that is created for us to contained dataset that we pass to the transformer: **pipeline.fit_transform(dataframe).**

Then we loop over each column in the cols array and change those using **factorize().**

Finally we return the Pandas dataframe as a NumPy array using **X.values,** since machine learning models work with NumPy arrays, not dataframes (in most cases)

```
from sklearn.base import BaseEstimator, TransformerMixin

class ToNumbers(BaseEstimator, TransformerMixin):
    def __init__(self, cols):
        self.cols = cols
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        for c in cols:
            encoded, categories = X.factorize()
            X = encoded
        return X.values
```

The second step calls the **StandardScaler()** to normalize the values in the array. We don't have to pass it any arguments since it knows to use the data from the previous step.

```
pipeline = Pipeline()

data = pipeline.fit_transform(df)
```

Here is the converted data as a NumPy array.

```
data

array(,
       ,
       ,
       ...,
       ,
       ,
       ])
```