Managing Machine Learning Workflows with Scikit-learn Pipelines Part 1: A Gentle Introduction

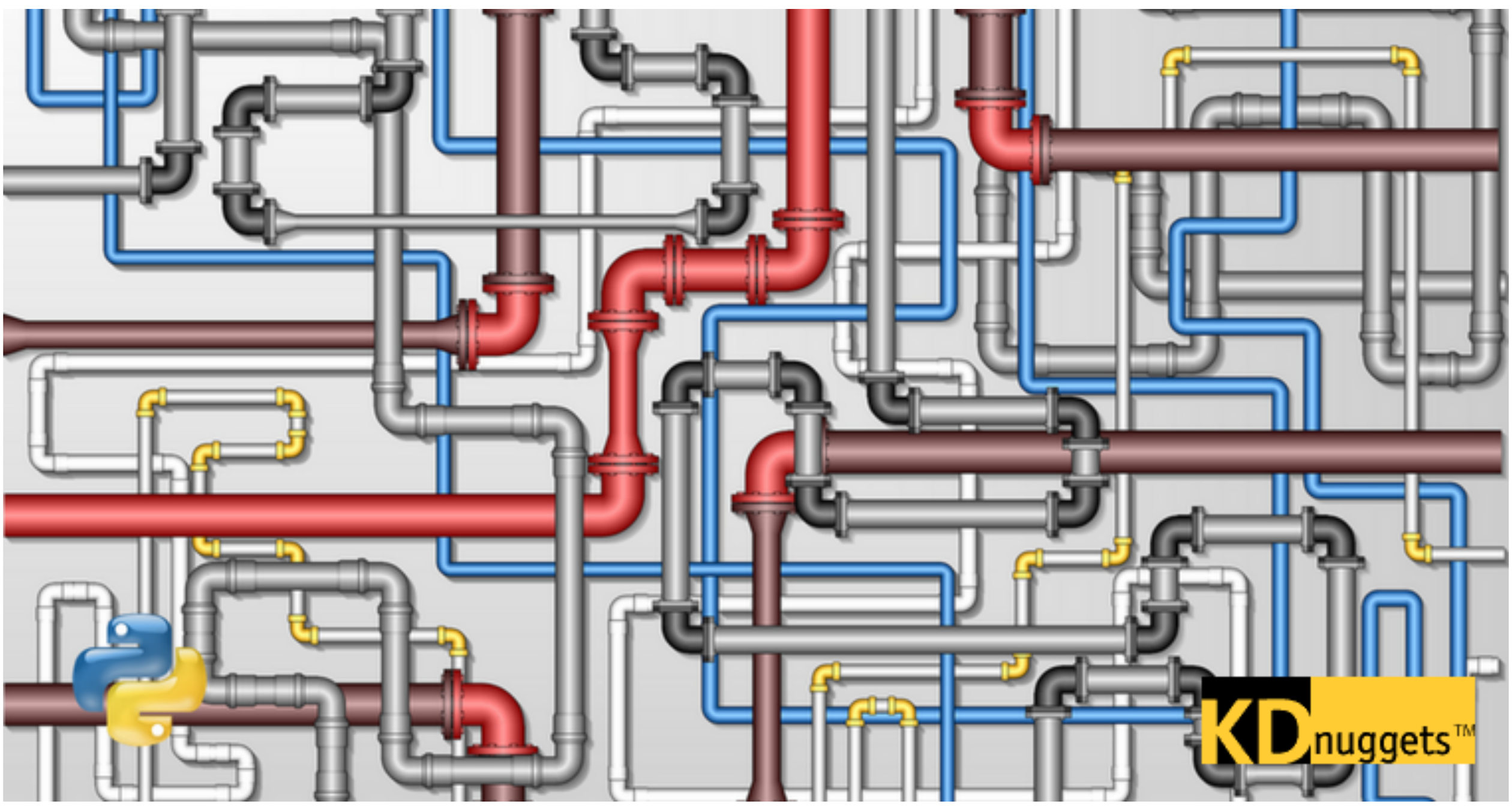
Previous post
Next post

Tags: Data Preprocessing, Pipeline, Python, scikit-learn, Workflow

Scikit-learn's Pipeline class is designed as a manageable way to apply a series of data transformations followed by the application of an estimator.

By Matthew Mayo, KDnuggets.

comments



Are you familiar with Scikit-learn Pipelines?

They are an extremely simple yet very useful tool for managing machine learning workflows.

A typical machine learning task generally involves data preparation to varying degrees. We won't get into the wide array of activities which make up data

preparation here, but there are many. Such tasks are known for taking up a large proportion of time spent on any given machine learning task.

After a dataset is cleaned up from a potential initial state of massive disarray, however, there are still several less-intensive yet no less-important transformative data preprocessing steps such as feature extraction, feature scaling, and dimensionality reduction, to name just a few.

Maybe your preprocessing requires only one of these tansformations, such as some form of scaling. But maybe you need to string a number of transformations together, and ultimately finish off with an estimator of some sort. This is where Scikit-learn Pipelines can be helpful.

Scikit-learn's Pipeline class is designed as a manageable way to apply a series of data transformations followed by the application of an estimator. In fact, that's really all it is:

> Pipeline of transforms with a final estimator.

That's it. Ultimately, this simple tool is useful for:

- Convenience in creating a coherent and easy-to-understand workflow
- Enforcing workflow implementation and the desired order of step applications
- Reproducibility
- Value in persistence of entire pipeline objects (goes to reproducibility and convenience)

So let's have a quick look at Pipelines. Specifically, here is what we will do.

Build 3 pipelines, each with a different estimator (classification algorithm), using default hyperparameters:

- Logisitic Regression
- Support Vector Machine
- Decision Tree

To demonstrate pipeline **transforms**, will perform:

- feature scaling
- dimensionality reduction, using PCA to project data onto 2 dimensional space

We will then end with fitting to our final **estimators**.

Afterward, and almost completely unrelated, in order to make this a little more like a full-fledged workflow (it still isn't, but closer), we will:

- Followup with scoring test data
- Compare pipeline model accuracies
- Identify the "best" model, meaning that which has the highest accuracy on our test data
- Persist (save to file) the entire pipeline of the "best" model

Granted, given that we will use default hyperparameters, this likely won't result in the most accurate possible models, but it will provide a sense of how to use simple pipelines. We will come back to the question of more complex modeling, hyperparameter tuning, and model evaluation afterward.

Oh, and for additional simplicity, we are using the iris dataset. The code is well-commented, and should be easy to follow.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.externals import joblib
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn import tree

# Load and split the data
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=42)

# Construct some pipelines
pipe_lr = Pipeline([('scl', StandardScaler()),
                    ('pca', PCA(n_components=2)),
                    ('clf', LogisticRegression(random_state=42))])

pipe_svm = Pipeline([('scl', StandardScaler()),
```

```
                        ('pca', PCA(n_components=2)),
                        ('clf', svm.SVC(random_state=42))])

pipe_dt = Pipeline([('scl', StandardScaler()),
                        ('pca', PCA(n_components=2)),
                        ('clf', tree.DecisionTreeClassifier(random_state=42))])

# List of pipelines for ease of iteration
pipelines = [pipe_lr, pipe_svm, pipe_dt]

# Dictionary of pipelines and classifier types for ease of reference
pipe_dict = {0: 'Logistic Regression', 1: 'Support Vector Machine', 2: 'Decision Tree'}

# Fit the pipelines
for pipe in pipelines:
        pipe.fit(X_train, y_train)

# Compare accuracies
for idx, val in enumerate(pipelines):
        print('%s pipeline test accuracy: %.3f' % (pipe_dict[idx], val.score(X_test, y_test)))

# Identify the most accurate model on test data
best_acc = 0.0
best_clf = 0
best_pipe = ''
for idx, val in enumerate(pipelines):
        if val.score(X_test, y_test) > best_acc:
                best_acc = val.score(X_test, y_test)
                best_pipe = val
                best_clf = idx
print('Classifier with best accuracy: %s' % pipe_dict[best_clf])

# Save pipeline to file
joblib.dump(best_pipe, 'best_pipeline.pkl', compress=1)
print('Saved %s pipeline to file' % pipe_dict[best_clf])
```



```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.externals import joblib
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn import tree

# Load and split the data
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=42)

# Construct some pipelines
pipe_lr = Pipeline([('scl', StandardScaler()),
                        ('pca', PCA(n_components=2)),
                        ('clf', LogisticRegression(random_state=42))])

pipe_svm = Pipeline([('scl', StandardScaler()),
```

```python
                        ('pca', PCA(n_components=2)),
                        ('clf', svm.SVC(random_state=42))])

pipe_dt = Pipeline([('scl', StandardScaler()),
                        ('pca', PCA(n_components=2)),
                        ('clf', tree.DecisionTreeClassifier(random_state=42))])

# List of pipelines for ease of iteration
pipelines = [pipe_lr, pipe_svm, pipe_dt]

# Dictionary of pipelines and classifier types for ease of reference
pipe_dict = {0: 'Logistic Regression', 1: 'Support Vector Machine', 2: 'Decision Tree'}

# Fit the pipelines
for pipe in pipelines:
        pipe.fit(X_train, y_train)

# Compare accuracies
for idx, val in enumerate(pipelines):
        print('%s pipeline test accuracy: %.3f' % (pipe_dict[idx], val.score(X_test, y_test)))

# Identify the most accurate model on test data
best_acc = 0.0
best_clf = 0
best_pipe = ''
for idx, val in enumerate(pipelines):
        if val.score(X_test, y_test) > best_acc:
                best_acc = val.score(X_test, y_test)
                best_pipe = val
                best_clf = idx
print('Classifier with best accuracy: %s' % pipe_dict[best_clf])

# Save pipeline to file
joblib.dump(best_pipe, 'best_pipeline.pkl', compress=1)
print('Saved %s pipeline to file' % pipe_dict[best_clf])
```


Let's run our script and see what happens.

```
$ python3 pipelines.py
```

```
Logistic Regression pipeline test accuracy: 0.933
Support Vector Machine pipeline test accuracy: 0.900
Decision Tree pipeline test accuracy: 0.867
Classifier with best accuracy: Logistic Regression
Saved Logistic Regression pipeline to file
```

So there you have it; a simple implementation of Scikit-learn pipelines. In this particular case, our logistic regression-based pipeline with default parameters scored the highest accuracy.

As mentioned above, however, these results likely don't represent our best efforts. What if we did want to test out a series of different hyperparameters? Can we use grid search? Can we incorporate automated methods for tuning these hyperparameters? Can AutoML fit in to this picture somewhere? What about using cross-validation?

Over the next couple of posts we will take a look at these additional issues, and see how these simple pieces fit together to make pipelines much more powerful than they may first appear to be given our initial example.

Related:

- 7 Steps to Mastering Data Preparation with Python
- Machine Learning Workflows in Python from Scratch Part 1: Data Preparation
- Machine Learning Workflows in Python from Scratch Part 2: k-means Clustering

Alex de Sá • a year ago

Hi,

I have been working with AutoML methods in order to optimize Machine Learning (ML) pipelines. You can give a look in our approach (namely, RECIPE -- REsilient ClassifIcation Pipeline Evolution):
https://github.com/RecipeML...
or
https://recipeml.github.io/...
and
https://www.researchgate.ne...

The problem of finding the most suitable ML pipeline is very challenging because it really depends on the characteristics of the dataset. In addition, depending on the search space size, a simple random search is enough.

Nevertheless, we are using evolutionary techniques to do that. Given a good representation of the algorithms and hyper-parameters (individuals in the population), these techniques could perform very well. More specifically, we are using a Grammar-based Genetic Programming (GGP) to do that. In this case, all the prior ML knowledge is encompassed into a grammar. This is a bit different from what other methods (e.g., Auto-WEKA and Auto-sklearn, TPOT and HyperBand) do.

Regards,

see more

1 • Reply • Share ›

Ernest Murray • 8 months ago

What is the function of PCA in the pipeline?

• Reply • Share ›

Davis David ➦ Ernest Murray • 6 months ago

The function of PCA is to reduce dimensionality of the the dataset

• Reply • Share ›

Previous post
Next post

Top Stories Past 30 Days

Most Popular

1. 9 Must-have skills you need to become a Data Scientist, updated
2. The cold start problem: how to build your machine learning portfolio
3. Why You Shouldn't be a Data Science Generalist
4. How to go from Zero to Employment in Data Science
5. The Essence of Machine Learning
6. The Five Best Data Visualization Libraries
7. 10 More Must-See Free Courses for Machine Learning and Data Science

Most Shared

1. The Essence of Machine Learning
2. 10 More Must-See Free Courses for Machine Learning and Data Science
3. Introduction to Statistics for Data Science
4. Papers with Code: A Fantastic GitHub Resource for Machine Learning
5. A Guide to Decision Trees for Machine Learning and Data Science
6. Top Python Libraries in 2018 in Data Science, Deep Learning, Machine Learning
7. Top 10 Books on NLP and Text Analysis

Latest News

- Advance Your Career in Analytics
- 2019 CDO Virtual Event: Empowering Business Through Data
- The Data Science Gold Rush: Top Jobs in Data Science an...
- Data Science Project Flow for Startups
- Top tweets, Jan 16-22: How to build an API for a mach...
- How To Fine Tune Your Machine Learning Models To Improv...

Top Stories
Last Week

Most Popular

1. NEW How to go from Zero to Employment in Data Science

2. NEW Ontology and Data Science
3. 9 Must-have skills you need to become a Data Scientist, updated
4. NEW Data Scientists Dilemma: The Cold Start Problem - Ten Machine Learning Examples
5. NEW End To End Guide For Machine Learning Projects
6. NEW Math for Programmers
7. NEW The 6 Most Useful Machine Learning Projects of 2018

Most Shared

1. How to build an API for a machine learning model in 5 minutes using Flask

2. End To End Guide For Machine Learning Projects
3. How to solve 90% of NLP problems: a step-by-step guide
4. Data Scientists Dilemma: The Cold Start Problem Ten Machine Learning Examples
5. How to go from Zero to Employment in Data Science
6. Math for Programmers
7. The 6 Most Useful Machine Learning Projects of 2018

Top Stories
Last Week

Most Popular

1. NEW How to go from Zero to Employment in Data Science

2. NEW Ontology and Data Science
3. 9 Must-have skills you need to become a Data Scientist, updated
4. NEW Data Scientists Dilemma: The Cold Start Problem - Ten Machine Learning Examples
5. NEW End To End Guide For Machine Learning Projects
6. NEW Math for Programmers
7. NEW The 6 Most Useful Machine Learning Projects of 2018

Most Shared

1. How to build an API for a machine learning model in 5 minutes using Flask

2. **End To End Guide For Machine Learning Projects**
3. **How to solve 90% of NLP problems: a step-by-step guide**
4. **Data Scientists Dilemma: The Cold Start Problem Ten Machine Learning Examples**
5. **How to go from Zero to Employment in Data Science**
6. **Math for Programmers**
7. **The 6 Most Useful Machine Learning Projects of 2018**