

DigitalHouse >
Coding School

DATA SCIENCE

UNIDAD 3
MÓDULO 5

Pipelines

Octubre 2017

PIPELINES

1

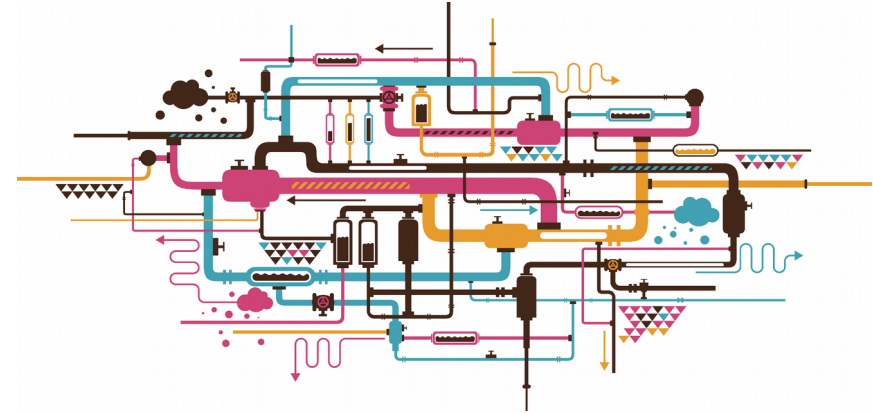
Crear pipelines para limpiar y manipular datos

2

Crear un transformador custom

3

Usar pipelines en combinación con clasificación, FeatureUnion y GridSearch



PIPELINES

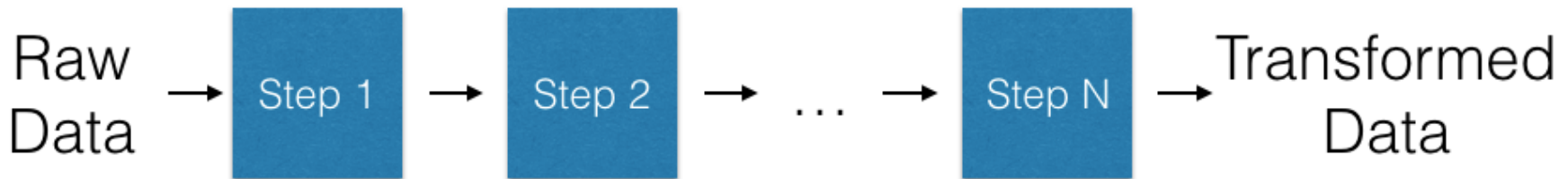


Un pipeline es una serie de pasos automatizados para transformar nuestros datos con el objetivo de asegurar su validez y consistencia.

Cada paso se “alimenta” del paso previo.

Al ser re-utilizables, los pipelines permiten ejecutar exactamente las mismas transformaciones sobre distintos datasets, asegurando consistencia en la operación

Al agrupar operaciones, también proveen un mayor nivel de abstracción



PIPELINES en SCIKIT-LEARN



Clase Pipeline del módulo sklearn (class sklearn.pipeline.Pipeline(steps)).

Steps es una lista de tuplas (key, value):

- Key: Nombre dado al paso
- Value: el transformador usado

Todos los pasos menos el último deben ser transformadores:

- Implementar el método *Transform*
- Implementar el método *Fit*

El último paso solo necesita implementar fit (puede ser transformador o clasificador)

<http://scikit-learn.org/stable/modules/pipeline.html>

DEMO I

Evergreen Stumbleupon Kaggle Competition

Kaggle(www.kaggle.com): red social para data scientists

StumbleUpon (www.StumbleUpon.com): “curador” de contenidos

2 Tipos de Contenidos:

- Efímeros: Pierden relevancia con el paso del tiempo. Ej: Noticias, recetas, etc.
- Perennes (Evergreen): Mantienen relevancia y pueden ser recomendados por más tiempo.

Competencia:

- Realizar esta distinción (clasificación) sin participación humana.



<https://www.kaggle.com/c/stumbleupon/overview>

- 7395 ejemplos
- 27 campos
- Nos vamos a concentrar en el título y cuerpo de las páginas, dentro del campo boilerplate

A	B	C	D	E	F	G
"url"	"urlid"	"boilerplate"	"alchemy_category_u"	"alchemy_category_v_score"	"avglinksiz"	"commonlin1"
"http://www.bom/ne2010-1predic holo air-bre batter by-201		<p> IBM Sees Holographic Calls Air Breathing Batteries ibm sees holographic calls, air-breathing batteries, A sign stands outside the International Business Machines Corp IBM Almaden Research Center campus in San Jose California Photographer Tony Avelar Bloomberg Buildings stand at the International Business Machines Corp IBM Almaden Research Center campus in the Santa Teresa Hills of San Jose California Photographer Tony Avelar Bloomberg By 2015 your mobile phone will project a 3 D image of anyone who calls and your laptop will be powered by kinetic energy At least that s what International Business Machines Corp sees in its crystal ball The predictions are part of an annual tradition for the Armonk New York based company which surveys its 3 000 researchers to find five ideas expected to take root in the next five years IBM the world s largest provider of computer services looks to Silicon Valley for input gleaning many ideas from its Almaden research center in San Jose California Holographic conversations projected from mobile phones lead this year s list The predictions also include air breathing batteries computer programs that can tell when and where traffic jams will take place environmental information generated by sensors in cars and phones and cities powered by the heat thrown off by computer servers These are all stretch goals and that s good said Paul Saffo managing director of foresight at the investment advisory firm Discern in San Francisco In an era when pessimism is the new black a little dose of technological optimism is not a bad thing For IBM it s not just idle speculation The company is one of the few big corporations investing in long range research projects and it counts on innovation to fuel growth Saffo said Not all of its predictions pan out though IBM was overly optimistic about the spread of speech technology for instance When the ideas do lead to products they can have broad implications for society as well as IBM s bottom line he said Research Spending They have continued to do research when all the other grand research organizations are gone said Saffo who is also a consulting associate professor at Stanford University IBM invested 5 8 billion in research and </p>				

AA
"label"
"0"

PIPELINES

Algunos atributos



Otra manera de crear un pipeline es utilizando el comando `make_pipeline`.

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

pipe1 = make_pipeline(StandardScaler(), LogisticRegression())

pipe2 = Pipeline(steps=[('standardscaler', StandardScaler()),
                        ('logistic_regr', LogisticRegression())
                      ])
```

Ambos pipelines creados son idénticos.

pipe1

```
Pipeline(steps=[('standardscaler', StandardScaler(copy=True, with_mean=True, with_std=True,
intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
verbose=0, warm_start=False))])
```

pipe2

```
Pipeline(steps=[('standardscaler', StandardScaler(copy=True, with_mean=True, with_std=True,
intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
verbose=0, warm_start=False))])
```

Para `make_pipeline` no hace falta dar un nombre a cada paso, lo toma automáticamente del transformador

Los estimadores del pipeline se guardan como una lista en el atributo *steps*:

```
>>> pipe.steps[0]
('reduce_dim', PCA(copy=True, iterated_power='auto', n_components=None,
random_state=None, svd_solver='auto', tol=0.0, whiten=False))
```

... y como un dict en *named_steps*:

```
>>> pipe.named_steps['reduce_dim']
PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
svd_solver='auto', tol=0.0, whiten=False)
```

Los parameters de un estimador se pueden acceder usando *<estimador>__<parámetro>*:

```
>>> pipe.set_params(clf__C=10)
```

```
Pipeline(steps=[('reduce_dim', PCA(copy=True, iterated_power='auto',
n_components=None, random_state=None, svd_solver='auto', tol=0.0, whiten=False)),
('clf', SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape=None, degree=3, gamma='auto', kernel='rbf', max_iter=-1,
probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))])
```

Podemos crear un pipeline con dos pasos:

```
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.decomposition import PCA
estimators = [('reduce_dim', PCA()), ('clf', SVC())]
pipe = Pipeline(estimators)
pipe
```

Y luego cambiar los modelos que ejecutan esos pasos:

```
from sklearn.linear_model import LogisticRegression
param_grid = dict(reduce_dim=[None, PCA(5), PCA(10)],
                  clf=[SVC(), LogisticRegression()],
                  clf__C=[0.1, 10, 100])
grid_search = GridSearchCV(pipe, param_grid=param_grid)
```

PREPROCESAMIENTO



Esta práctica nos permitirá conocer el módulo de pre-procesamiento.

El mismo viene repleto de clases muy útiles para dicha operación.

La finalidad es re-utilizar al máximo las funciones existentes

<http://scikit-learn.org/stable/modules/preprocessing.html>

Práctica:

1. Juntarse de a pares
2. Seleccionar una función por grupo
3. Leer la documentación de la misma
4. Explicar el funcionamiento al resto de la clase.

Data Manipulators

- Binarizer
- KernelCenterer
- MaxAbsScaler
- MinMaxScaler
- Normalizer
- OneHotEncoder
- PolynomialFeatures
- RobustScaler
- StandardScaler

Data Imputation

- Imputer

Function Transformer

- FunctionTransformer

Label Manipulators

- LabelBinarizer
- LabelEncoder
- MultiLabelBinarizer

Más allá de la riqueza del módulo de preprocesamiento, podemos encontrar casos donde no sean suficiente y nos sea conveniente crear transformadores custom. Tenemos dos maneras de hacerlo:

1. Extender la BaseClass en Scikit-Learn.

En este ejemplo creamos un transformador muy simple que devuelve la entrada multiplicada por un factor X:

```
from sklearn.base import BaseEstimator, TransformerMixin
import numpy as np

class FeatureMultiplier(BaseEstimator, TransformerMixin):
    def __init__(self, factor):
        self.factor = factor

    def transform(self, X, *_):
        return X * self.factor

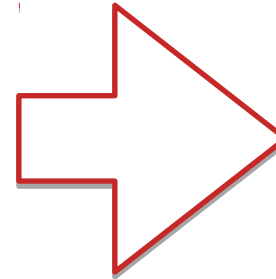
    def fit(self, *_):
        return self

fm = FeatureMultiplier(2)

test = np.diag((1,2,3,4))
print test

fm.transform(test)
```

```
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
```



```
[[2, 0, 0, 0],
 [0, 4, 0, 0],
 [0, 0, 6, 0],
 [0, 0, 0, 8]]
```

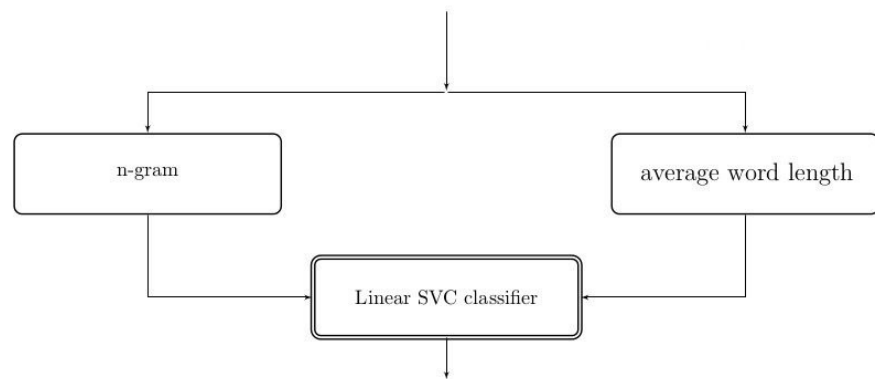
FEATURE UNION



Hay casos en los que nos va a interesar juntar features transformadas aplicando distintos métodos, y luego correr pasos siguientes.

Para ello, podemos usar FeatureUnion, que combina varios transformadores, en un nuevo transformador armado con la combinación de los outputs de cada transformador incluido. Cada transformador es aplicado de manera independiente y en paralelo, y los vectores de salida son combinados.

Ejemplo: un Pipeline con una Unión entre una matriz de palabras con CountVectorizer y el tamaño promedio de palabra (CustomTransformer), y un modelo que clasifique en base a `ambac`



```
from sklearn.pipeline import Pipeline, FeatureUnion
pipeline = Pipeline([('feats', FeatureUnion([
    ('ngram', ngram_count_pipeline),
    ('ave', AverageWordLengthExtractor())
])),
    ('clf', LinearSVC())
])
```

HERRAMIENTAS GRÁFICAS



Hemos podido observar la potencia de los Pipelines, más en combinación con GridSearch y FeatureUnion, además de las funciones de pre-procesamiento existentes y la posibilidad de extenderlos.

Sin embargo, en un ambiente complejo será conveniente utilizar una herramienta gráfica.

Dicha herramienta deberá soportar:

- Manejo de Ambientes (Dev, Test, Prod, Sandbox)
- Governance
- Seguridad
- Dependencias
- Alarmas
- Conectividad
- Priorización de procesamiento
- Estadísticas
- Errores
- Excepciones
- Reporting
- Etc.

www.digitalhouse.com



DAG: core_cx

Tree View | Graph View | Task Duration | Landing Times | Gantt | Code

Run: 2015-02-01 00:00:00 Layout: Left->Right Go

CONCLUSIONES

Pipelines

Los conceptos vistos hoy nos permiten:

- Generar soluciones más robustas
- Re-utilizar código
- Optimizar nuestra elección de modelos y parámetros
- Armar workflows de pro-cesamiento complejos
- Crear transformadores custom y adosarlos en los workflows