Member-only story

Customize your Sklearn Pipelines with BaseEstimator, TransformerMixin and FeatureUnion

How to create a custom transformer for an Sklearn Pipeline

Andrea D'Agostino · Follow

Published in Towards Data Science · 5 min read · Dec 14, 2021

Photo by Arseny Togulev on Unsplash

One of Scikit-Learn's most used and popular features are *pipelines*. Although their use is optional, they can be employed to make our code cleaner and easier to maintain. Pipelines accept *estimators* as input, which are classes inheriting from sklearn.base.BaseEstimator and which contain the *fit* and *transform* methods. **This allows us to customize pipelines with features that Sklearn does not offer by default.**

We will talk about *transformers*, objects that apply a transformation on an input. The class we will inherit from is TransformerMixin, but it is also

possible to extend from ClassifierMixin, RegressionMixin, ClusterMixin and others to create a custom estimator. Read here for all available options.

We will work with a dataset of textual data, on which we want to apply transformations such as:

- text preprocessing
- TF-IDF vectorization
- creation of additional features, such as sentiment, number of characters and number of sentences

This will be done through the use of Pipeline and FeatureUnion, a Sklearn class that combines feature sets from different sources.

The Dataset

We will use the dataset provided by Sklearn, *20newsgroups*, to have quick access to a body of textual data. For demonstration purposes, I will only use a sample of 10 texts but the example can be extended to any number of texts.

Let's import the dataset

```python
# essential libs for our task
import numpy as np
import pandas as pd

from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import FeatureUnion, Pipeline

import nltk
from nltk.corpus import stopwords
from nltk.sentiment.vader import SentimentIntensityAnalyzer
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('vader_lexicon')

import re

# 20newsgroups categories used to filter the dataset
categories = [
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'alt.atheism',
 'soc.religion.christian',
]

dataset = fetch_20newsgroups(subset='train', categories=categories, shuffle=True, remove

df = pd.DataFrame(dataset.data, columns=["corpus"]).sample(10) # <-- we only take 10 doc
```

transformer_mixin1_en.py hosted with ❤ by GitHub view raw

Creation of Textual Features

We will build the feature set containing this information

- vectorization with TF-IDF after applying preprocessing
- sentiment with NLTK.Vader
- number of characters in the text
- number of sentences in the text

Our process, without using pipelines, would be to sequentially apply all these steps with separate code blocks. The beauty of pipelines is that sequencing is maintained in a single block of code — **the pipeline itself becomes an estimator, capable of performing all operations in a single statement.**

Let's create our functions

```python
def preprocess_text(text: str, remove_stopwords: bool) -> str:
    """This utility function sanitizes a string by:
    - removing links
    - removing special characters
    - removing numbers
    - removing stopwords
    - transforming in lowercase
    - removing excessive whitespaces
    Args:
        text (str): the input text you want to clean
        remove_stopwords (bool): whether or not to remove stopwords
    Returns:
        str: the cleaned text
    """

    # remove links
    text = re.sub(r"http\S+", "", text)
    # remove special chars and numbers
    text = re.sub("[^A-Za-z]+", " ", text)
    # remove stopwords
    if remove_stopwords:
        # 1. tokenize
        tokens = nltk.word_tokenize(text)
        # 2. check if stopword
        tokens = [w for w in tokens if not w.lower() in stopwords.words("english")]
        # 3. join back together
        text = " ".join(tokens)
    # return text in lower case and stripped of whitespaces
    text = text.lower().strip()
    return text


def get_sentiment(text: str):
  """
```

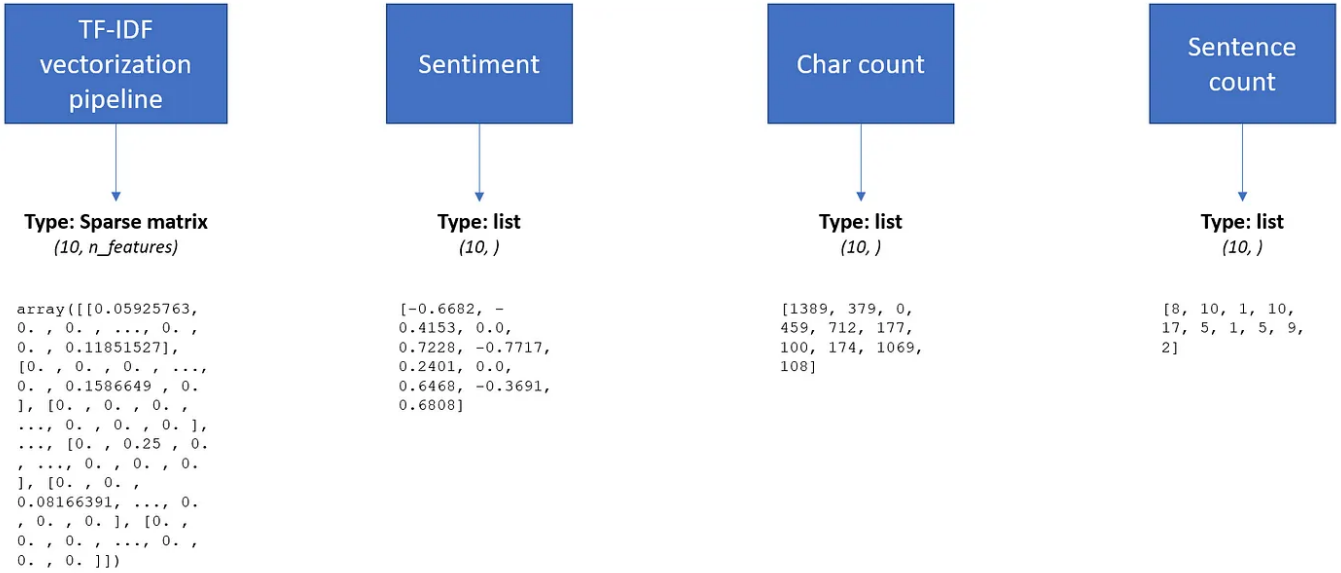
```
    Function that uses NLTK.Vader to extract sentiment.
    Sentiment is a score that expresses how positive or negative a text is.
    The value ranges from -1 to 1, where 1 is the most positive value.
    Args:
         text (str): text to parse
    Returns:
         sentiment (float): polarity of the text
     """
    vader = SentimentIntensityAnalyzer()
    return vader.polarity_scores(text)['compound']

def get_nchars(text: str):
    """
    Function that returns the number of characters in a text.
    Args:
         text (str): text to parse
    Returns:
         n_chars (int): number of characters
    """
    return len(text)

def get_nsentences(text: str):
    """
    Function that returns the number of sentences in a text.
    Args:
         text (str): text to parse
    Returns:
         n_chars (int): number of sentences
    """
    return len(text.split("."))
```

transformer_mixin2_eng.py hosted with ❤️ by **GitHub** view raw

Our goal is to create a unique feature set in order to train a model for some task. We will use Pipelines and FeatureUnion to put our matrices together.

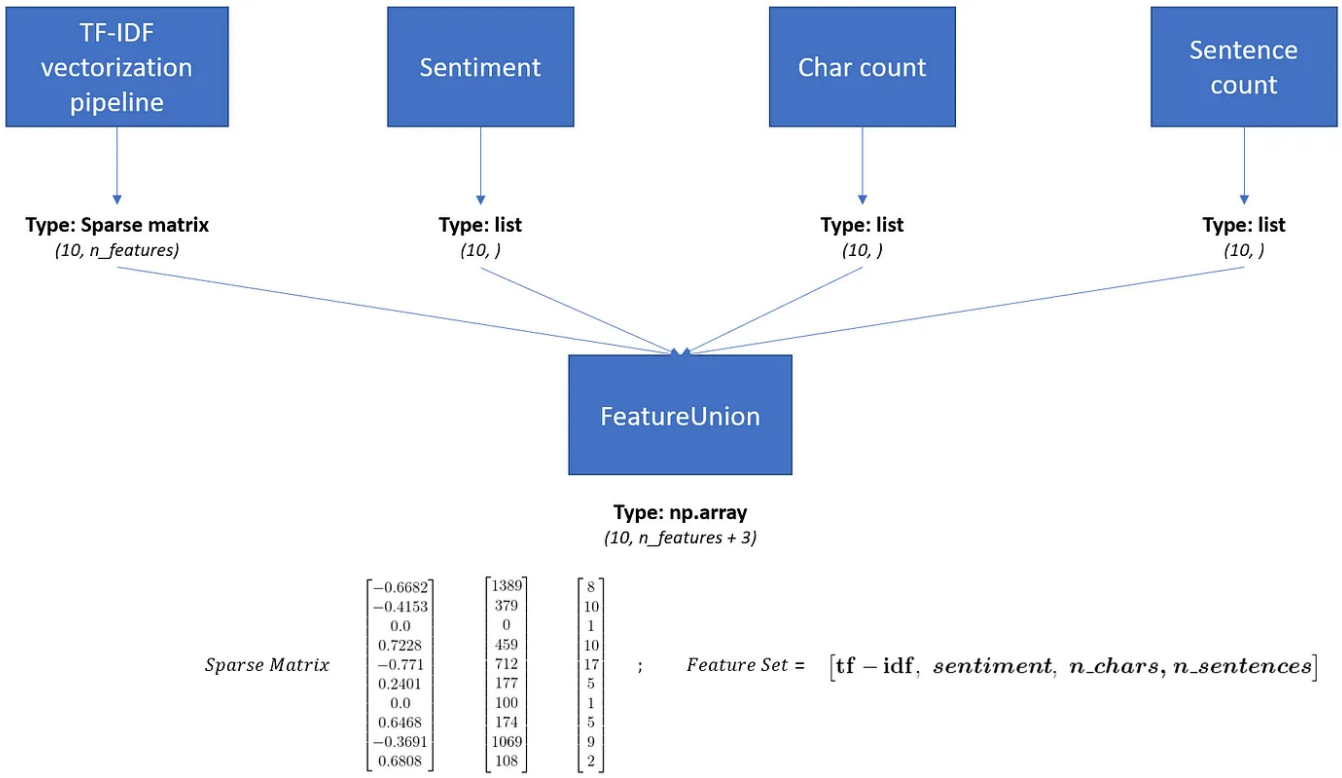
How to Merge Features from Different Sources

TF-IDF vectorization will create a sparse matrix which will have dimensions *n_documents_in_corpus*n_features*, sentiment will be a single number, as well as the output of n_chars and n_sentences. **We are going to take the outputs of each of these steps and create a single matrix that will contain them all, so that we can train a model on all the features we have engineered.** We will start from such schema



Inital state of things. Image by Author.

and reach this final representation



How FeatureUnion works. Image by Author.

The feature set will be used as a training vector for our model.

Classes that Inherit from BaseEstimator and TransformerMixin

In order to get our process going, we need to define the classes and what they will do in the pipeline. Let's start by creating a DummyEstimator, from

which we will inherit init, fit and transform. The DummyEstimator is a handy class that saves us from writing redundant code.

```python
class DummyTransformer(BaseEstimator, TransformerMixin):
  """
  Dummy class that allows us to modify only the methods that interest us,
  avoiding redudancy.
  """
  def __init__(self):
    return None

  def fit(self, X=None, y=None):
    return self

  def transform(self, X=None):
    return self
```

transformer_mixin_3_eng.py hosted with ❤️ by **GitHub** view raw

DummyEstimator will be inherited from four classes, *Preprocessor*, *SentimentAnalysis*, *NChars*, *NSentences* and *FromSparseToArray*.

```
class Preprocessor(DummyTransformer):
  """
  Class used to preprocess text
  """
  def __init__(self, remove_stopwords: bool):
    self.remove_stopwords = remove_stopwords
    return None

  def transform(self, X=None):
    preprocessed = X.apply(lambda x: preprocess_text(x, self.remove_stopwords)).values
    return preprocessed

class SentimentAnalysis(DummyTransformer):
  """
  Class used to generate sentiment
  """
  def transform(self, X=None):
    sentiment = X.apply(lambda x: get_sentiment(x)).values
    return sentiment.reshape(-1, 1) # <-- note the reshape to transform a row vector int

class NChars(DummyTransformer):
  """
  Class used to count the number of characters
  """
  def transform(self, X=None):
    n_chars = X.apply(lambda x: get_nchars(x)).values
    return n_chars.reshape(-1, 1)

class NSententences(DummyTransformer):
  """
  Class used to count the number of sentences
  """
  def transform(self, X=None):
    n_sentences = X.apply(lambda x: get_nsentences(x)).values
```

```
35      return n_sentences.reshape(-1, 1)
36
37  class FromSparseToArray(DummyTransformer):
38    """
39    Class used to transform a sparse matrix in a numpy array
40    """
41    def transform(self, X=None):
42      arr = X.toarray()
43      return arr
```

transformer_mixin_4_eng.py hosted with ❤ by GitHub view raw

As you can see, DummyEstimator allows us to define only the transform function, since every other class inherits its *init* and *fit* from DummyEstimator.

Now let's see how to implement the vectorization pipeline, **which will take into account the preprocessing of our texts.**

```
1  vectorization_pipeline = Pipeline(steps=[
2      ('preprocess', Preprocessor(remove_stopwords=True)), # the first step is to preproces
3      ('tfidf_vectorization', TfidfVectorizer()), # the second step applies vectorization c
4      ('arr', FromSparseToArray()), # the third step converts a sparse matrix into a numpy
5  ])
```

transformer_mixin_6eng.py hosted with ❤ by GitHub view raw

All that remains is to apply FeatureUnion to put the pieces together

```python
features = [
  ('vectorization', vectorization_pipeline),
  ('sentiment', SentimentAnalysis()),
  ('n_chars', NChars()),
  ('n_sentences', NSententences())
]
combined = FeatureUnion(features) # this is where we merge the features together
combined
```

transformer_mixin_7_eng.py hosted with ❤️ by **GitHub** view raw

```
FeatureUnion(transformer_list=[('vectorization',
                                Pipeline(steps=[('preprocess',
                                                 Preprocessor(remove_stopwords=True)),
                                                ('tfidf_vectorization',
                                                 TfidfVectorizer()),
                                                ('arr', FromSparseToArray())])),
                               ('sentiment', SentimentAnalysis()),
                               ('n_chars', NChars()),
                               ('n_sentences', NSententences())])
```

Steps and estimators of FeatureUnion. Image by Author.

Let's check the output of *fit_transform* on our corpus

```
[60]  1 combined.fit_transform(X=df['corpus'])

array([[ 5.92576343e-02,  0.00000000e+00,  0.00000000e+00, ...,
        -6.68200000e-01,  1.38900000e+03,  8.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00, ...,
        -4.15300000e-01,  3.79000000e+02,  1.00000000e+01],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00, ...,
         0.00000000e+00,  0.00000000e+00,  1.00000000e+00],
       ...,
       [ 0.00000000e+00,  2.50000000e-01,  0.00000000e+00, ...,
         6.46800000e-01,  1.74000000e+02,  5.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  8.16639089e-02, ...,
        -3.69100000e-01,  1.06900000e+03,  9.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00, ...,
         6.80800000e-01,  1.08000000e+02,  2.00000000e+00]])
```

Output of FeatureUnion — The four data sources are merged together in a single matrix. Image by Author.

The output appears to be correct! It's not very clear, though. We conclude the tutorial by inserting the combined feature set into the dataframe

```
# here we point to the second step of the second object in the vectorization_pipeline to
# to which we then add the other three columns
cols = vectorization_pipeline.steps[1][1].get_feature_names() + ["sentiment", "n_chars",
features_df = pd.DataFrame(combined.transform(df['corpus']), columns=cols)
```

transformer_mixin_&eng.py hosted with ❤ by GitHub view raw

The result is the following (here I have truncated the result for readability reasons)

```
1 features_df.iloc[:,-6:]
```

	yankee	yankees	zakat	sentiment	n_chars	n_sentences
0	0.000000	0.000000	0.118515	-0.6682	1389.0	8.0
1	0.000000	0.158665	0.000000	-0.4153	379.0	10.0
2	0.000000	0.000000	0.000000	0.0000	0.0	1.0
3	0.000000	0.000000	0.000000	0.7228	459.0	10.0
4	0.000000	0.000000	0.000000	-0.7717	712.0	17.0
5	0.000000	0.000000	0.000000	0.2401	177.0	5.0
6	0.305386	0.000000	0.000000	0.0000	100.0	1.0
7	0.000000	0.000000	0.000000	0.6468	174.0	5.0
8	0.000000	0.000000	0.000000	-0.3691	1069.0	9.0
9	0.000000	0.000000	0.000000	0.6808	108.0	2.0

Our joined feature set in a Pandas DataFrame. Image by Author.

We now have a dataset ready to be provided to any training model. It would be useful to experiment with *sklearn.preprocessing.StandardScaler* or similar and normalize n_chars and n_sentences. I will leave this exercise to the reader.

Conclusion

We have seen what a basic manipulation of the **BaseEstimator**, **TransformerMixin** and **FeatureUnion** classes in Sklearn can do for our custom project.

It enables us to create custom preprocessing classes and concatenate feature sets together while remaining in the Pipeline.

This kind of approach boosts productivity and clean code and can help a whole team in structuring scripts and notebooks.

I hope you learned something new today and that you apply these principles in your codebase! 😀

Recommended Reads

For the interested, here are a list of books that I recommended for each ML-related topic. There are ESSENTIAL books in my opinion and have greatly impacted my professional career.
Disclaimer: these are Amazon affiliate links. I will receive a small commission from Amazon for referring you these items. Your experience won't change and you

won't be charged more, but it will help me scale my business and produce even more content around AI.

- **Intro to ML:** *Confident Data Skills: Master the Fundamentals of Working with Data and Supercharge Your Career* by Kirill Eremenko
- **Sklearn / TensorFlow:** *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by Aurelien Géron
- **NLP:** *Text as Data: A New Framework for Machine Learning and the Social Sciences* by Justin Grimmer
- **Sklearn / PyTorch:** *Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python* by Sebastian Raschka
- **Data Viz:** *Storytelling with Data: A Data Visualization Guide for Business Professionals* by Cole Knaflic

Useful Links (written by me)

- **Learn how to perform a top-tier Exploratory Data Analysis in Python:** *Exploratory Data Analysis in Python — A Step-by-Step Process*
- **Learn the basics of TensorFlow:** *Get started with TensorFlow 2.0 — Introduction to deep learning*

- **Perform text clustering with TF-IDF in Python:** *Text Clustering with TF-IDF in Python*

If you want to support my content creation activity, feel free to follow my referral link below and join Medium's membership program. I will receive a portion of your investment and you'll be able to access Medium's plethora of articles on data science and more in a seamless way.

Code

Data Science Pipeline Transformers Python Machine Learning

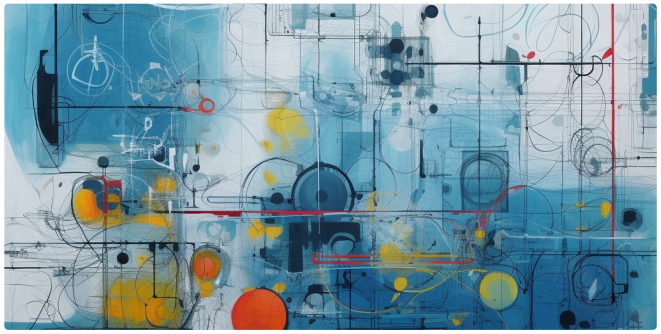
Written by Andrea D'Agostino

Follow

975 Followers · Writer for Towards Data Science

Data scientist. I write about data science, machine learning and analytics. I also write about career and productivity tips to help you thrive in the field.

More from Andrea D'Agostino and Towards Data Science



Andrea D'Agostino in Towards Data Science

Exploratory Data Analysis in Python — A Step-by-Step Process

What is exploratory analysis, how it is structured and how to apply it in Python with...

· 13 min read · Jul 7, 2022

497 2

John Adeojo in Towards Data Science

Building AI Strategies for Businesses

The art of crafting an AI strategy through Wardley Maps

· 13 min read · Jun 6

809 9

Avi Chawla in Towards Data Science

Andrea D'Agostino in Towards Data Science

6 Things That You Probably Didn't Know You Could Do With Pandas

Some hidden treasures of Pandas library.

6 min read · May 17

356 5

Time series prediction with LSTM in Tensorflow

Learn how to create an LSTM-based neural network to predict an univariate time series

· 8 min read · Jun 22, 2022

237 6

See all from Andrea D'Agostino

See all from Towards Data Science

Recommended from Medium

Andras Gefferth in Towards Data Science

One Hot Encoding scikit vs pandas

You can safely use pandas.get_dummies for machine learning applications, just need to ...

8 min read · Mar 13

8

Shreya Goyal in Geek Culture

Finding Optimal epochs using K-Fold for Transformer Models.

Usually fine-tuning a pre-trained Transformer model on some downstream task such as Te...

3 min read · Feb 9

10

Lists

What is ChatGPT?

9 stories · 108 saves

Stories to Help You Level-Up at Work

19 stories · 102 saves

Staff Picks

352 stories · 112 saves

Serafeim Loukas, PhD in Towards AI

How To Estimate FP, FN, TP, TN, TPR, TNR, FPR, FNR & Accuracy f...

In this post, I explain how someone can read a confusion matrix and how to extract several...

✨ · 6 min read · Feb 5

64

DeepThinke... in SFU Professional Computer Scie...

MLflow—a modern MLOps tool for data project collaboration

A comprehensive introduction to MLflow and its capabilities

16 min read · Feb 11

3.8K 3

Max Bade in Dev Genius

Ali Soleymani

Automate Feature Selection for a Fraud Machine Learning Model in...

Let's get one thing straight—I am not an author or journalist or blogger, and I feel like ...

7 min read · Mar 4

42 2

Grid search and random search are outdated. This approach...

If you're a data scientist, there is a good chance you have used "Grid Search" to fine-...

· 7 min read · Feb 8

343 5

See more recommendations