Análisis exploratorio de datos

Aprendizaje automático

Juan David Martínez
jdmartinev@eafit.edu.co

2023

UNIVERSIDAD EAFIT

Agenda

- Análisis exploratorio de datos
- Ingeniería de características

Análisis exploratorio de datos

UNIVERSIDAD EAFIT®

Análisis exploratorio de datos

Exploratory Data Analysis (EDA): Metodología para analizar bases de datos y encontrar sus principales características.

- **Coleccionar** o concatenar datos
- **Realizar investigaciones iniciales** para descubrir patrones, detectar anomalías, validar hipótesis y verificar suposiciones:
  - Estadísitcas descriptivas
  - Representaciones gráficas (histogramas, barras,…)
- **Procesar** los datos para obtener información relevante
- **¡Más un arte que una ciencia!**

Estadística descriptiva

Estadísticas generales - df.head(), df.shape, df.info()

- Número de muestras (filas)
- Número de características (columnas)

Estadística univariada (para cada característica)

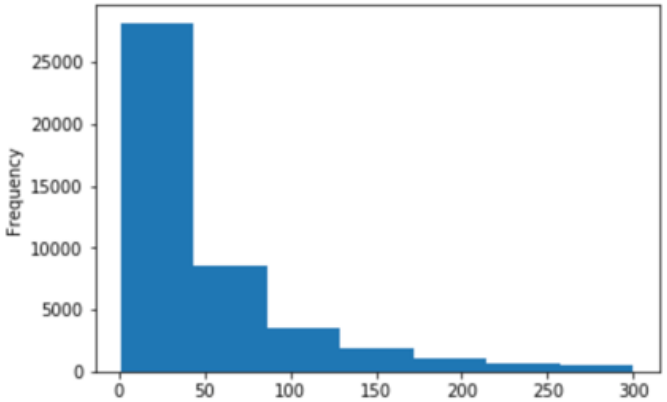
- Estadística para variables numéricas (media, varianza, histogramas)

  df.describe(), hist(df[feature])
- Estadística para variables categóricas (histogramas, moda, valores más/menos frecuentes, porcentajes, número de valores únicos)
  - Histograma de valores - df[feature].value_counts(), sns.distplot()
- Estadísticas de la variable objetivo
  - Distribución de clases – df[target].value_counts(), np.bincount(y)

Estadística descriptiva

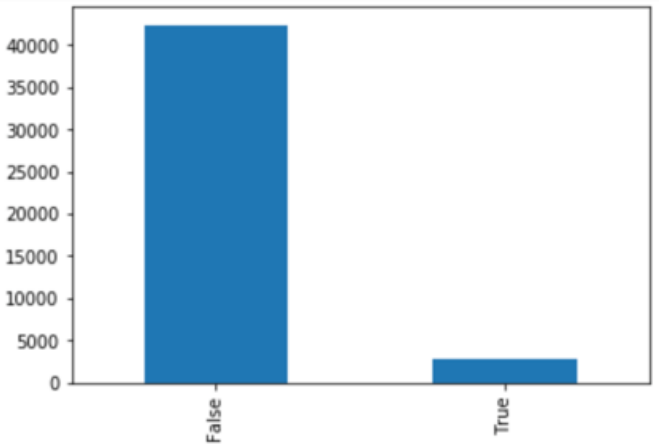
Características numéricas

```
import matplotlib.pyplot as plt

df[num_feature].plot.hist(bins = 7)
plt.show()
```



Características categóricas

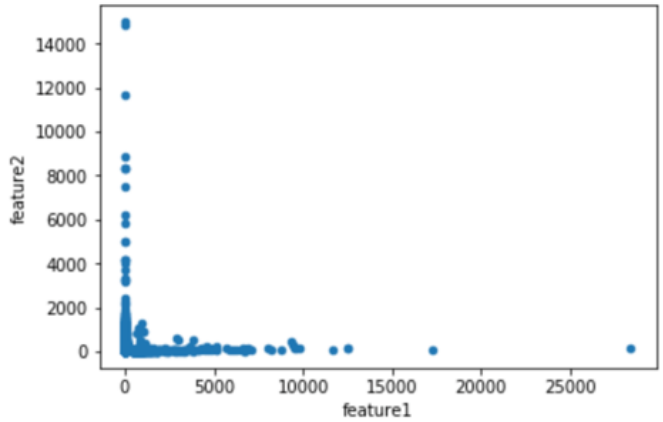
```
import matplotlib.pyplot as plt

df[cat_feature].value_counts().plot.bar()
plt.show()
```

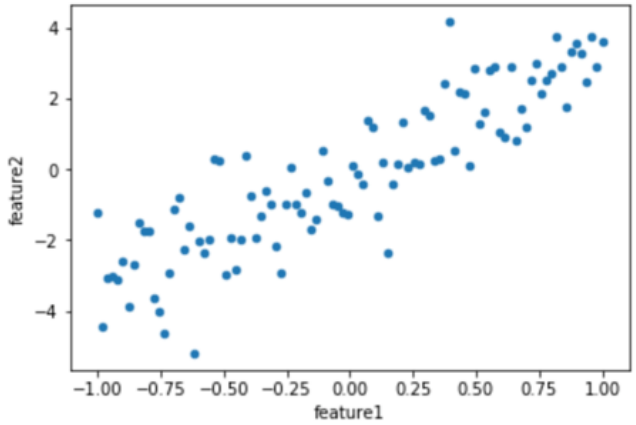


Estadística multivariada

Estadísticas multivariadas (más de una característica)

- Correlaciones – df.plot.scatter(feature1, feature2), df[[feature1, feature2]]





Relaciones lineales entre pares de características o una característica y la variable objetivo

Estadística multivariada

Estadísticas multivariadas (más de una característica)

Correlaciones: Cómo se relacionan (linealmente) pares de características

```
[feature1, feature2]
df[cols].corr()
```

	feature1	feature2
feature1	1	0.0128493
feature2	0.0128493	1

	feature1	feature2
feature1	1	0.882106
feature2	0.882106	1

Los valores de la correlación están entre -1 y 1: -1 significa correlación negativa perfecta, 1 significa correlación positiva perfecta, 0 significa que no hay correlación entre las variables.

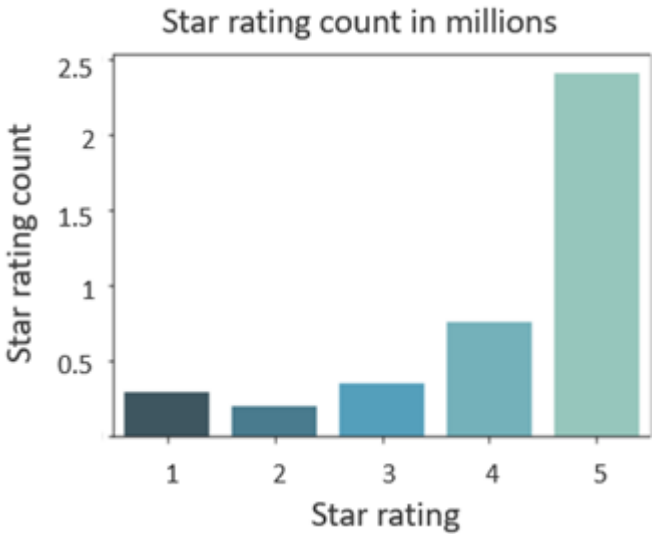
Estadística multivariada

Las características altamente correlacionadas pueden afectar el rendimiento de algunos modelos de aprendizaje de máquina, como la regresión lineal o logística, cuando su implementación involucra el cálculo de inversas de matrices. Además, puede esconder la relevancia de alguna característica.

- Seleccione una de las características correlacionadas y descarte las demás
- Otros modelos, como los árboles de decisión, son inmunes a este problema

Adicionalmente, características altamente correlacionadas con la variable objetivo podrían mejorar el rendimiento de modelos lineales.

Bases de datos desbalanceadas



- El número de muestras por clase no está distribuido equitativamente
- El modelo de ML puede no reconocer adecuadamente la clase poco frecuente

Ejemplos:

- Detección de fraudes
- Detección de anomalías
- Diagnóstico médico

Amazon review dataset: El número de calificaciones con 5 estrellas es casi igual a la suma del número de calificaciones de los otros tipos

Bases de datos desbalanceadas

¿Cómo manejar el problema de desbalance de clases?

Submuestreo	Remuestreo	Generación de datos	Pesos para cada muestra
Reducir el número de muestras de la clase dominante	Aumentar el tamaño de la o las clases con pocas muestras	Crear nuevas muestras, similares, pero no idénticas	Para modelos que tienen función de costo, asignar mayores pesos a la clase rara

El ajuste se debe hacer únicamente en los datos de entrenamiento. Los conjuntos de validación y prueba deben conservar la distribución original de los datos

https://imbalanced-learn.org/stable/

Imputación de datos faltantes

Descartar filas y/o columnas con valores faltantes: Remover esas filas y/o columnas de la base de datos.

Una menor cantidad de datos puede generar modelos sobre/sub entrenados

Imputar (completar) los valores faltantes:

- **Reemplazar** los valores faltantes de características numéricas con el promedio o de características categóricas con la moda `df['col'].fillna((df['col'].mean())), df['col'].fillna((df['col'].mode()))`
- **Placeholder:** Asignar un valor común a los valores faltantes
- **Técnicas avanzadas:** Predecir los valores faltantes usando ML (AWS Datawig usa NNs)

  https://github.com/awslabs/datawig

Imputación de datos faltantes

SimpleImputer: **Clase** de **sklearn** para utilizar métodos de imputación, implementa los métodos .fit() y .transform()

SimpleImputer(missing_values=nan, strategy='mean', fill_value=None)

- **numerical data:**

  Strategy = “mean”, replace missing values using the mean along each column

  Strategy = “median”, replace missing values using the median along each column

- **numerical or categorical data:**

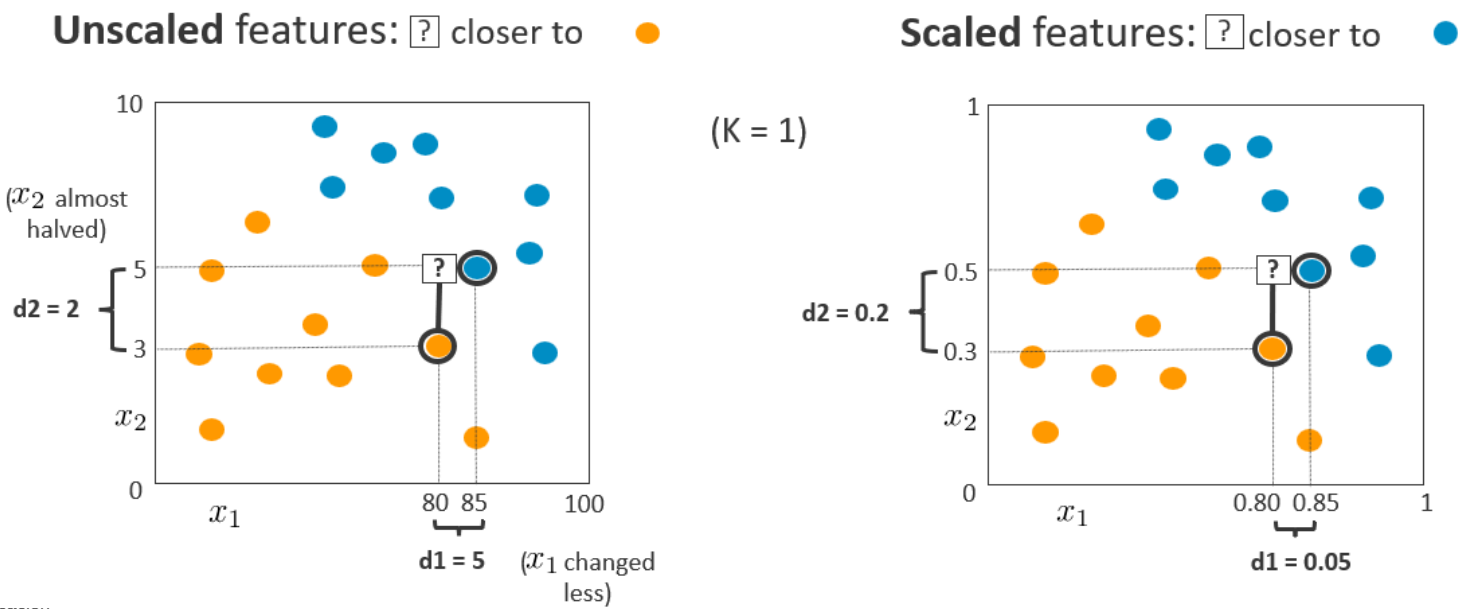
  Strategy = “most_frequent”, replace missing using the most frequent value along each column

Strategy = “constant”, replace missing values with fill_value

Estandarización de características

Motivación: Algoritmos como kNN y redes neuronales cambian su comportamiento cuando las características no están en la misma escala.

Solución: Llevar las características a la misma escala (mean/variance, MinMax)



UNIVERSIDAD EAFIT

Estandarización de características

StandardScaler **Clase** de **sklearn**, implementa los métodos .fit() y .transform() para escalar cada característica (columna) de forma que quede con media y desviación estándar 1.

$$x_{scaled} = \frac{x - x_{mean}}{x_{std}}$$

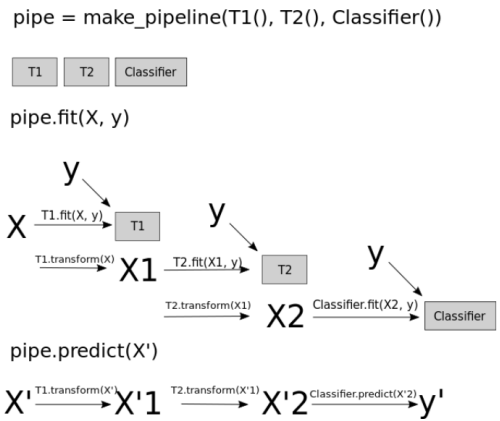
MinMAxScaler **Clase** de **sklearn**, implementa los métodos .fit() y .transform() para escalar cada característica (columna) de forma que los valores mínimo y máximo sean 0 y 1.

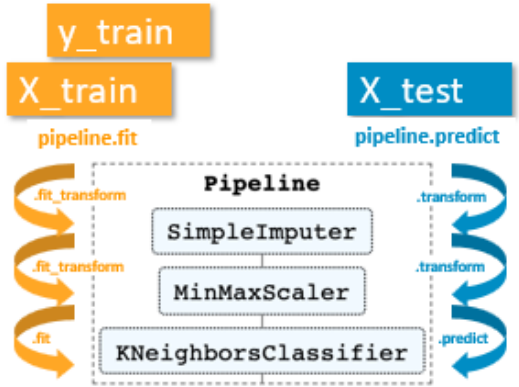
$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Pipeline (sklearn)

Pipeline: **secuencia** de transformaciones de datos que, generalmente, finaliza con un estimador, implementa los métodos .fit() y .predict(). Previene el sesgo a los datos de entrenamiento.

```
pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', MinMaxScaler()),
    ('clf', KNeighborsClassifier(n_neighbors = 3))
    ])

pipeline.fit(X_train, y_train)
predictions = pipeline.predict(X_test)
```





UNIVERSIDAD EAFIT

Ingeniería de características

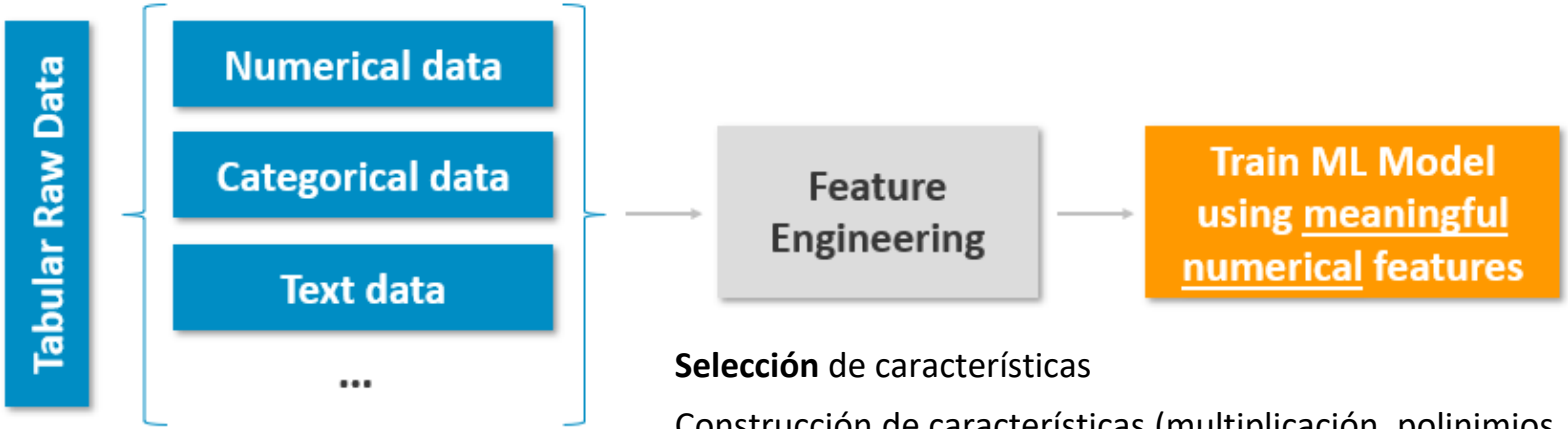
UNIVERSIDAD EAFIT®

Ingeniería de características

Utiliza conocimiento del **dominio y de los datos** para crear **nuevas características a partir de los datos crudos** como entradas para los modelos de ML.

Intuición: ¿Qué información utilizaría un humano para hacer predicciones?

¡A menudo, este proceso es más un arte que una ciencia!



Selección de características

Construcción de características (multiplicación, polinimios, logaritmos, kernels,…)

Codificación de características

Características categóricas: No tienen una representación numérica natural.

Ejemplo: color Є {green, red, blue}, fraude Є {falso, verdadero}

La mayoría de modelos de ML requieren convertir estas categorías a números.

Codificación (encoding): Asignar un número a cada categoría.

Ordinal: Las categorías están ordenadas, ejemplo, tamaño Є {L>M>S}. Podemos asignar los valores L->3, M->2, S->1.

Nominal: Las categorías no tienen un orden específico, ejemplo, color Є {green, red, blue}. Podemos asignar números de forma aleatoria.

Codificación de características

LabelEncoder: Clase de sklearn, codifica variables categóricas con valores entre 0 y n_classes – 1. Implementa los métodos .fit() y .transform().

- Codifica las variables objetivo o una sola característica, no la matriz completa **X.**

	color	size	price	classlabel
0	green	S	10.1	shirt
1	red	M	13.5	pants
2	blue	L	15.3	shirt

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['color'] = le.fit_transform(df['color'])
```

```
   color size  price classlabel
0      1    S   10.1      shirt
1      2    M   13.5      pants
2      0    L   15.3      shirt
```

Codificación de características

OrdinalEncoder: Clase de sklearn, codifica variables categóricas con valores entre 0 y n_classes – 1. Implementa los métodos .fit() y .transform().

- Codifica dos o más variables categóricas. No funciona para una sola característica.
- Devuelve una sola columna de enteros por característica.

	color	size	price	classlabel
0	green	S	10.1	shirt
1	red	M	13.5	pants
2	blue	L	15.3	shirt

```
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()

df[['color','size','classlabel']] =
oe.fit_transform(df[['color','size','classlabel']])
```

	color	size	price	classlabel
0	1.0	2.0	10.1	1.0
1	2.0	1.0	13.5	0.0
2	0.0	0.0	15.3	1.0

Codificación de características

Problema: La codificación ordinal tiene problemas en algunos modelos de ML porque el orden y tamaño de los enteros no debería importar.

One-hot-encoding expande cada característica categórica en muchas características binarias (tantas como categorías en cada característica).

OneHotEncoder: Clase de sklearn, codifica variables categóricas como un arreglo numérico. Implementa los métodos .fit() y .transform().

- No asigna nombre a las nuevas variables. Funciona para más de dos características, para una sola se debe usar LabelBinarizer().

Codificación de características

get_dummies: versión de **Pandas** para **one-hot-encoding.**

	color	size	price	classlabel
0	green	S	10.1	shirt
1	red	M	13.5	pants
2	blue	L	15.3	shirt

	size	price	classlabel	color_blue	color_green	color_red
0	S	10.1	shirt	0	1	0
1	M	13.5	pants	0	0	1
2	L	15.3	shirt	1	0	0

Codificación de características

Problema: Demasiadas categorías. Para esto se puede:

- Definir una estructura jerárquica: códigos zip

  regiones -> estados -> ciudades

  Escoger un nivel específico para codificar esta característica
- Agrupar las categorías en diferentes grupos: edades

  grupos de edades 1-15, 16-22, 23-30

Codificación de características

Codificación usando la variable objetivo: Codificar utilizando valores que puedan explicar la variable objetivo.

Ejemplo: Promediar la variable objetivo para cada categoría. Después, reemplazar los valores categóricos con los valores promediados.

x_1	x_2	y
a	c	1
a	d	1
b	c	0
a	d	0
a	d	0
a	d	1
b	d	0

x_1 -> cat a -> 3/5 = 0.6
x_1 -> cat b -> 0/2 = 0
x_2 -> cat c -> 1/2 = 0.5
x_2 -> cat d -> 2/5 = 0.4

x_1	x_2	y
0.6	0.5	1
0.6	0.4	1
0	0.5	0
0.6	0.4	0
0.6	0.4	0
0.6	0.4	1
0	0.4	0

Procesamiento de datos

UNIVERSIDAD EAFIT®

Transformaciones en sklearn

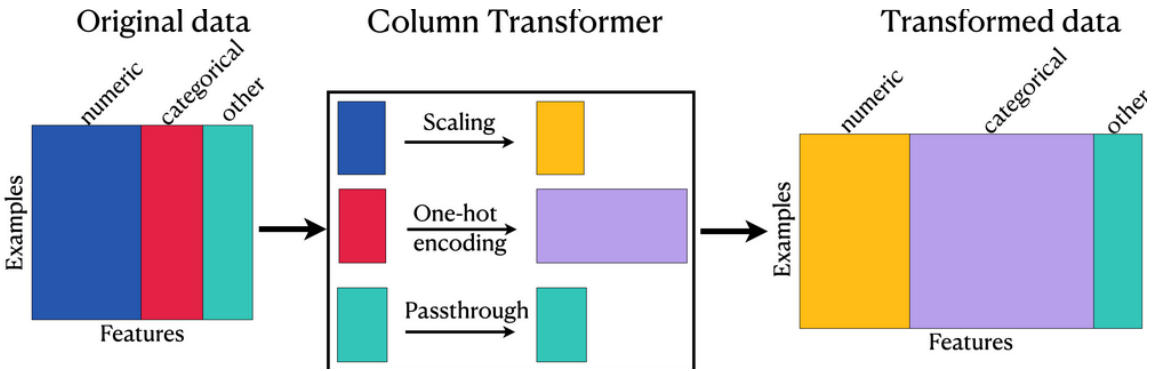
SimpleImputer, StandardScaler, MinMaxScaler, LabelEncoder, OrdinalEncoder, OneHotEncoder, CountVectorizer pertenecen a la clase **Transformers** de **sklearn.** Todos tienen los métodos:

- .fit(): aprende la transformación a partir de los datos de **entrenamiento**
- .transform() aplica la transformación a cualquier conjunto (entrenamiento, validación, prueba).
- En el conjunto de entrenamiento, también se puede aplicar .fit_transform()
-

ColumnTansformer en sklearn

ColumnTransformer aplica las transformaciones a las columnas de un arreglo de Numpy o a un DataFrame de Pandas. Implementa los métodos .fit() y .transform().

- Permite aplicar transformaciones a subconjuntos de características (numérica, categóricas, texto), de forma individual.
- Las características generadas por cada transformación se concatenarán para conformar un solo grupo de características



ColumnTranformer y Pipeline

```
numerical_processing = Pipeline([
   ('num_imputer', SimpleImputer(strategy='mean')),
   ('num_scaler', MinMaxScaler())])

categorical_processing = Pipeline([
   ('cat_imputer', Imputer(strategy='constant', fill_value='missing')),
   ('cat_encoder', OneHotEncoder(handle_unknown='ignore'))])

processor = ColumnTransformer(transformers =[
    ('num_processing', numerical_processing, ('feature1', 'feature3')),
    ('cat_processing', categorical_processing, ('feature0', 'feature2'))])

pipeline = Pipeline([('data_processing', processor),
                     ('estimator', KNeighborsClassifier())])

pipeline.fit(X_train, y_train)
predictions = pipeline.predict(X_test)
```