Untitled43

November 4, 2025

[1]:
```
from sklearn.preprocessing import StandardScaler
import numpy as np
X_train = np.array([[ 1., -1., 2.], [ 2., 0., 0.], [ 0., 1., -1.]])
scaler = StandardScaler().fit_transform(X_train)
print(scaler)
```

```
[[ 0.         -1.22474487  1.33630621]
 [ 1.22474487  0.         -0.26726124]
 [-1.22474487  1.22474487 -1.06904497]]
```

[10]:
```
from sklearn import preprocessing
import numpy as np

# Example data matrix (you can replace this with your own)
X_train = np.array([
    [1.0, -1.0, 2.0],
    [2.0, 0.0, 0.0],
    [0.0, 1.0, -1.0]
])

# Scaling to [0, 1] range
print("[0, 1] scaling:\n")
min_0_max_1_scaler = preprocessing.MinMaxScaler()
X_train_min_0_max_1 = min_0_max_1_scaler.fit_transform(X_train)
print(X_train_min_0_max_1)

# Scaling between a given minimum and maximum value (e.g., [0, 10])
print("\n[0, 10] scaling:\n")
min_max_scaler = preprocessing.MinMaxScaler(feature_range=(0, 10))
X_train_minmax = min_max_scaler.fit_transform(X_train)
print(X_train_minmax)

# Scaling so that the training data lies within the range [-1, 1]
print("\n[-1, 1] scaling:\n")
max_abs_scaler = preprocessing.MaxAbsScaler()
X_train_maxabs = max_abs_scaler.fit_transform(X_train)
print(X_train_maxabs)
```

```
[0, 1] scaling:
```

```
[[0.5        0.         1.        ]
 [1.         0.5        0.33333333]
 [0.         1.         0.        ]]

[0, 10] scaling:

[[ 5.          0.         10.        ]
 [10.          5.          3.33333333]
 [ 0.         10.          0.        ]]

[-1, 1] scaling:

[[ 0.5 -1.   1. ]
 [ 1.   0.   0. ]
 [ 0.   1.  -0.5]]
```

[12]:
```
import numpy as np
from sklearn import preprocessing
X_train = np.array([[ 1., -1., 2.], [ 2., 0., 0.], [ 0., 1., -1.]])
scaler = preprocessing.RobustScaler()
X_train_rob_scal = scaler.fit_transform(X_train)
print(X_train_rob_scal)
```

```
[[ 0.         -1.          1.33333333]
 [ 1.          0.          0.        ]
 [-1.          1.         -0.66666667]]
```

[16]:
```
from sklearn import preprocessing
import numpy as np
X = np.array([[ 1., -1., 2.], [ 2., 0., 0.], [ 0., 1., -1.]])
X_normalized = preprocessing.normalize(X)
print(X_normalized)
```

```
[[ 0.40824829 -0.40824829  0.81649658]
 [ 1.          0.          0.        ]
 [ 0.          0.70710678 -0.70710678]]
```

[18]:
```
from sklearn import preprocessing

# Define possible categories
genders = ['female', 'male']
locations = ['from Africa', 'from Asia', 'from Europe', 'from US']
browsers = ['uses Chrome', 'uses Firefox', 'uses Safari']

# Sample data
X = [
    ['male', 'from US', 'uses Safari'],
```

```
    ['female', 'from Europe', 'uses Safari'],
    ['female', 'from Asia', 'uses Firefox'],
    ['male', 'from Africa', 'uses Chrome']
]

# Create and fit an OrdinalEncoder
enc = preprocessing.OrdinalEncoder()
X_enc = enc.fit_transform(X)

print("Encoded data:\n", X_enc)
print("\nCategories used by encoder:\n", enc.categories_)
```

```
Encoded data:
 [[1. 3. 2.]
 [0. 2. 2.]
 [0. 1. 1.]
 [1. 0. 0.]]

Categories used by encoder:
 [array(['female', 'male'], dtype=object), array(['from Africa', 'from Asia',
'from Europe', 'from US'], dtype=object), array(['uses Chrome', 'uses Firefox',
'uses Safari'], dtype=object)]
```

[20]:
```
import pandas as pd

# Example data
df = pd.DataFrame({
    "Date": ["2024-12-01", "2025-01-15", "2025-03-20"]
})

# Convert 'Date' to datetime
df["Date"] = pd.to_datetime(df["Date"])

# Extract features
df["year"] = df["Date"].dt.year
df["month"] = df["Date"].dt.month
df["day"] = df["Date"].dt.day
df["week_of_year"] = df["Date"].dt.isocalendar().week  # modern replacement
df["day_of_year"] = df["Date"].dt.dayofyear

# Drop original Date column
df = df.drop(["Date"], axis=1)

# Show result
print(df.info())
print(df)
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 3 entries, 0 to 2
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   year          3 non-null      int32
 1   month         3 non-null      int32
 2   day           3 non-null      int32
 3   week_of_year  3 non-null      UInt32
 4   day_of_year   3 non-null      int32
dtypes: UInt32(1), int32(4)
memory usage: 195.0 bytes
None
```

	year	month	day	week_of_year	day_of_year
0	2024	12	1	48	336
1	2025	1	15	3	15
2	2025	3	20	12	79

[24]:
```python
from sklearn import preprocessing

# Original categorical labels
labels = ["India", "US", "Japan"]

# Create the LabelEncoder
le = preprocessing.LabelEncoder()

# Fit and transform labels into numbers
new_labels = le.fit_transform(labels)

# Show encoded labels
print("Encoded labels:\n", new_labels)

# Decode some numeric labels back to original text
print("\nInverse transform:\n", le.inverse_transform([2, 0, 1]))
```

```
Encoded labels:
 [0 2 1]

Inverse transform:
 ['US' 'India' 'Japan']
```

[32]:
```python
# Import OneHotEncoder from scikit-learn
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

df = pd.read_csv('cars.csv')
# Create a OneHotEncoder object
onehotencoder = OneHotEncoder()
```

```
# Reshape the 1-D 'Country' array to 2-D as fit_transform expects 2-D
# Then fit and transform the data
X = onehotencoder.fit_transform(df['model'].values.reshape(-1, 1)).toarray()

# Print the result
print(X)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 1. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

[34]:
```
from sklearn import preprocessing
import numpy as np

# Sample data
X = np.array([
    [-3., 5., 15.],
    [0., 6., 14.],
    [6., 3., 11.]
])

# Create KBinsDiscretizer object
# n_bins specifies the number of bins per feature
# encode='ordinal' means each bin is encoded as an integer
kbd = preprocessing.KBinsDiscretizer(n_bins=[3, 2, 2], encode='ordinal',␣
 ↪strategy='uniform')

# Fit and transform the data
X_kbd = kbd.fit_transform(X)

# Print the discretized output
print(X_kbd)
```

```
[[0. 1. 1.]
 [1. 1. 1.]
 [2. 0. 0.]]
```

```
C:\Users\Thinkpad\anaconda3\Lib\site-
packages\sklearn\preprocessing\_discretization.py:248: FutureWarning: In version
1.5 onwards, subsample=200_000 will be used by default. Set subsample explicitly
to silence this warning in the mean time. Set subsample=None to disable
subsampling explicitly.
  warnings.warn(
```

[36]:
```
from sklearn import preprocessing
import numpy as np

# Sample data
X = [[1., -1., 2.],
     [2., 0., 0.],
     [0., 1., -1.]]

# Default Binarizer (threshold=0)
binarizer = preprocessing.Binarizer()
X_bin = binarizer.fit_transform(X)
print("Binarized with threshold=0:\n", X_bin)

# Binarizer with custom threshold
binarizer_1 = preprocessing.Binarizer(threshold=1.1)
X_bin_1 = binarizer_1.fit_transform(X)
print("\nBinarized with threshold=1.1:\n", X_bin_1)
```

```
Binarized with threshold=0:
 [[1. 0. 1.]
 [1. 0. 0.]
 [0. 1. 0.]]

Binarized with threshold=1.1:
 [[0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 0.]]
```

[2]:
```
from sklearn import preprocessing
import numpy as np
X=np.arange(9).reshape(3,3)
print(X)
poly=preprocessing.PolynomialFeatures(degree=3,
 interaction_only=True)
X_poly=poly.fit_transform(X)
print(X_poly)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[[  1.   0.   1.   2.   0.   0.   2.   0.]
 [  1.   3.   4.   5.  12.  15.  20.  60.]
 [  1.   6.   7.   8.  42.  48.  56. 336.]]
```

[4]:
```
from sklearn import preprocessing
import numpy as np
transformer = preprocessing.FunctionTransformer(np.log1p, validate=True)
X = np.array([[0, 1], [2, 3]])
```

```
X_tr = transformer.fit_transform(X)
print(X_tr)
```

```
[[0.         0.69314718]
 [1.09861229 1.38629436]]
```

[6]:
```
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd

texts = [
    "blue car and blue window",
    "black crow in the window",
    "i see my reflection in the window"
]

# Create a Count Vectorizer that encodes presence/absence of words
vec = CountVectorizer(binary=True)

# Fit the vectorizer to the texts
vec.fit(texts)

# Print the sorted vocabulary
print([w for w in sorted(vec.vocabulary_.keys())])

# Transform texts into binary vectors
X = vec.transform(texts).toarray()
print(X)

# Create a DataFrame for better visualization
df = pd.DataFrame(X, columns=sorted(vec.vocabulary_.keys()))
print(df)
```

```
['and', 'black', 'blue', 'car', 'crow', 'in', 'my', 'reflection', 'see', 'the',
'window']
[[1 0 1 1 0 0 0 0 0 0 1]
 [0 1 0 0 1 1 0 0 0 1 1]
 [0 0 0 0 0 1 1 1 1 1 1]]
   and  black  blue  car  crow  in  my  reflection  see  the  window
0    1      0     1    1     0   0   0           0    0    0       1
1    0      1     0    0     1   1   0           0    0    1       1
2    0      0     0    0     0   1   1           1    1    1       1
```

[12]:
```
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd

texts = [
    "blue car and blue window",
    "black crow in the window",
```

```
    "i see my reflection in the window"
]

# Create a Count Vectorizer that encodes presence/absence of words
vec = CountVectorizer(binary=False)

# Fit the vectorizer to the texts
vec.fit(texts)

# Print the sorted vocabulary
print([w for w in sorted(vec.vocabulary_.keys())])

# Transform texts into binary vectors
X = vec.transform(texts).toarray()
print(X)

# Create a DataFrame for better visualization
df = pd.DataFrame(X, columns=sorted(vec.vocabulary_.keys()))
print(df)
```

```
['and', 'black', 'blue', 'car', 'crow', 'in', 'my', 'reflection', 'see', 'the',
'window']
[[1 0 2 1 0 0 0 0 0 0 1]
 [0 1 0 0 1 1 0 0 0 1 1]
 [0 0 0 0 0 1 1 1 1 1 1]]
   and  black  blue  car  crow  in  my  reflection  see  the  window
0    1      0     2    1     0   0   0           0    0    0       1
1    0      1     0    0     1   1   0           0    0    1       1
2    0      0     0    0     0   1   1           1    1    1       1
```

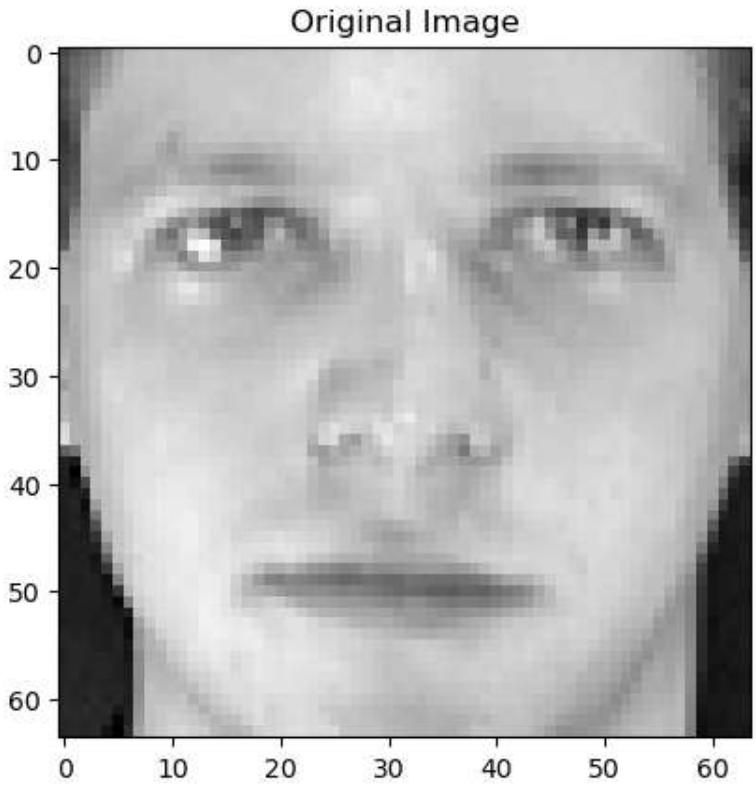
[14]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd

texts = [
    "blue car and blue window",
    "black crow in the window",
    "i see my reflection in the window"
]

# Create a TF-IDF vectorizer
vec = TfidfVectorizer()

# Fit the vectorizer to the texts
vec.fit(texts)

# Print the sorted vocabulary
print([w for w in sorted(vec.vocabulary_.keys())])
```

```
# Transform texts into TF-IDF vectors
X = vec.transform(texts).toarray()
print(X)

# Create a DataFrame for better visualization
df = pd.DataFrame(X, columns=sorted(vec.vocabulary_.keys()))
print(df)
```

```
['and', 'black', 'blue', 'car', 'crow', 'in', 'my', 'reflection', 'see', 'the',
'window']
[[0.39687454 0.         0.79374908 0.39687454 0.         0.
  0.         0.         0.         0.         0.2344005 ]
 [0.         0.53409337 0.         0.         0.53409337 0.40619178
  0.         0.         0.         0.40619178 0.31544415]
 [0.         0.         0.         0.         0.         0.35829137
  0.4711101  0.4711101  0.4711101  0.35829137 0.27824521]]
        and     black      blue       car      crow        in       my  \
0  0.396875  0.000000  0.793749  0.396875  0.000000  0.000000  0.00000
1  0.000000  0.534093  0.000000  0.000000  0.534093  0.406192  0.00000
2  0.000000  0.000000  0.000000  0.000000  0.000000  0.358291  0.47111

   reflection      see       the    window
0     0.00000  0.00000  0.000000  0.234400
1     0.00000  0.00000  0.406192  0.315444
2     0.47111  0.47111  0.358291  0.278245
```

[16]:
```
from sklearn.feature_extraction import image
from sklearn.datasets import fetch_olivetti_faces
import matplotlib.pyplot as plt

# Load the Olivetti faces dataset
data = fetch_olivetti_faces()

# Show the first image
plt.imshow(data.images[0], cmap='gray')
plt.title("Original Image")
plt.show()

# Extract 3x3 patches from the first image
patches = image.extract_patches_2d(data.images[0], (3, 3))

print("Image shape:", data.images[0].shape)
print("Patches shape:", patches.shape)
print("Number of elements in all patches:", len(patches.flatten()))
```

```
downloading Olivetti faces from https://ndownloader.figshare.com/files/5976027
to C:\Users\Thinkpad\scikit_learn_data
```




```
Image shape: (64, 64)
Patches shape: (3844, 3, 3)
Number of elements in all patches: 34596
```

[28]:
```python
import cv2
import numpy as np

# Function to compute Hu Moments
def hu_moments(image):
    # Convert image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Compute moments
    moments = cv2.moments(gray)
    # Compute Hu Moments and flatten into a 1D array
    hu = cv2.HuMoments(moments).flatten()
    return hu

# Function to compute histogram
def histogram(image, mask=None):
    # Convert image to HSV
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

```
    # Compute histogram for the Hue channel
    hist = cv2.calcHist([hsv], [0], mask, [256], [0, 256])
    # Normalize the histogram
    cv2.normalize(hist, hist)
    return hist.flatten()

import cv2
import mahotas
import numpy as np

# Function to compute Haralick texture features
def haralick_moments(image):
    # Convert image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Convert to integer type as required by mahotas
    gray = gray.astype(np.uint8)
    # Compute Haralick features and take the mean across directions
    haralick = mahotas.features.haralick(gray).mean(axis=0)
    return haralick

# Class to compute Zernike moments
class ZernikeMoments:
    def __init__(self, radius):
        """
        Store the radius that will be used when computing Zernike moments.
        """
        self.radius = radius

    def describe(self, image):
        """
        Return the Zernike moments for the image.
        Note: image should be grayscale and ideally centered within a square␣
 ↪mask.
        """
        # Ensure the image is grayscale
        if len(image.shape) == 3:
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        # Convert to uint8 as required by mahotas
        image = image.astype(np.uint8)
        return mahotas.features.zernike_moments(image, self.radius)
```

[30]:
```
import cv2
import mahotas
import numpy as np
from sklearn.datasets import fetch_olivetti_faces
import matplotlib.pyplot as plt
from mahotas.features import zernike_moments, haralick
```

```
# Load the Olivetti faces dataset
data = fetch_olivetti_faces()
image = data.images[0]

plt.imshow(image, cmap='gray')
plt.title("Olivetti Face Sample")
plt.axis('off')
plt.show()

# --- Hu Moments ---
def hu_moments(image):
    moments = cv2.moments(image)
    hu = cv2.HuMoments(moments).flatten()
    return hu

hu_mot = hu_moments(image)
print("Hu Moments ({}):\n".format(len(hu_mot)), hu_mot)

# --- Histogram (normalized 256-bin) ---
def histogram(image):
    hist = cv2.calcHist([image.astype(np.uint8)], [0], None, [256], [0, 256])
    hist = cv2.normalize(hist, hist).flatten()
    return hist

hist = histogram(image)
print("\nHistogram ({}):\n".format(len(hist)), hist)

# --- Haralick Features ---
def haralick_moments(image):
    har = mahotas.features.haralick(image.astype(np.uint8)).mean(axis=0)
    return har

haralick_feats = haralick_moments(image)
print("\nHaralick ({}):\n".format(len(haralick_feats)), haralick_feats)

# --- Zernike Moments ---
def zernike_moments(image, radius=21):
    # Convert to binary for Zernike computation
    thresh = image > image.mean()
    zern = mahotas.features.zernike_moments(thresh.astype(np.uint8), radius)
    return zern

zernike_feats = zernike_moments(image)
print("\nZernike ({}):\n".format(len(zernike_feats)), zernike_feats)
```

Olivetti Face Sample

```
Hu Moments (7):
 [2.44346111e-01 1.09110661e-04 5.17020161e-05 3.76399586e-06
 1.40892052e-11 1.16310230e-08 5.05826981e-11]

Histogram (256):
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

Haralick (13):
 [1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]

Zernike (25):
 [0.31830989 0.01890163 0.06481972 0.0633957  0.02584877 0.12966706
 0.03554216 0.01836268 0.05732032 0.04925423 0.04822797 0.07821339
```

```
0.03389148 0.0193667  0.07146486 0.01171312 0.03826596 0.03583074
0.07708602 0.04074665 0.02787281 0.01661538 0.06716495 0.02689876
0.01687837]
```

[40]:
```
from silx.opencl import sift
sift_ocl=sift.SiftPlan(template=image)
keypoints=sift_ocl.keypoints(image)
```

```
C:\Users\Thinkpad\anaconda3\Lib\site-packages\pyopencl\cache.py:420:
CompilerWarning: Non-empty compiler output encountered. Set the environment
variable PYOPENCL_COMPILER_OUTPUT=1 to see more.
  prg.build(options_bytes, [devices[i] for i in to_be_built_indices])
```

[44]:
```
from sklearn import datasets
from sklearn.decomposition import PCA

# Load the breast cancer dataset
dat = datasets.load_breast_cancer()
X, Y = dat.data, dat.target

# Show dataset shapes
print("Examples =", X.shape, "Labels =", Y.shape)

# Apply PCA (reduce to 5 components)
pca = PCA(n_components=5)
X_pca = pca.fit_transform(X)

# Show transformed shapes
print("Examples_PCA =", X_pca.shape, "Labels =", Y.shape)
```

```
Examples = (569, 30) Labels = (569,)
Examples_PCA = (569, 5) Labels = (569,)
```

[46]:
```
from sklearn import datasets
from sklearn.decomposition import KernelPCA

# Load breast cancer dataset
dat = datasets.load_breast_cancer()
X, Y = dat.data, dat.target

# Show original data shapes
print("Examples =", X.shape, "Labels =", Y.shape)

# Kernel options: "linear", "poly", "rbf", "sigmoid", "cosine", "precomputed"
kpca = KernelPCA(n_components=7, kernel='rbf')

# Fit and transform the data
X_kpca = kpca.fit_transform(X)
```

```
# Show transformed data shapes
print("Examples_KPCA =", X_kpca.shape, "Labels =", Y.shape)
```

```
Examples = (569, 30) Labels = (569,)
Examples_KPCA = (569, 7) Labels = (569,)
```

[52]:

[54]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis(n_components=1)  # max allowed
X_lda = lda.fit_transform(X, Y)
print(X_lda.shape)
```

```
(569, 1)
```

[ ]: