Politecnico di Torino

Data Science Lab

Scikit-learn

preprocessing

Andrea Pasini
Flavio Giobergia
Elena Baralis

DataBase and Data Mining Group

- **Data preprocessing with scikit-learn**

- **Summary**
  - **Normalization**
  - **Feature extraction (examples)**
    - **Handling nominal data**
    - **Computing TF-IDF**
  - **Dimensionality reduction**
    - **PCA**

Normalization

- Examples:
  - min-max normalization: MinMaxScaler
  - z-score normalization: StandardScaler

In [1]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler


minmax_s = MinMaxScaler()
zscore_s = StandardScaler()
```

- Applying normalization to training and test set

In [1]:
```
X_train = [[0, 10], [0, 20], [2, 10], [2, 20]]
X_test = [[1, 15]]

minmax_s.fit(X_train) # NOTE: “learning” on training data only!
X_train_norm = minmax_s.transform(X_train)
X_test_norm = minmax_s.transform(X_test) # correct
X_test_wrong = minmax_s.fit_transform(X_test) # do not fit on test
print(X_test_norm)
print(X_test_wrong)
```

Out[1]:
```
[[0.5 0.5]]
[[0, 0]]
```

Feature extraction

- Necessary when a datasets presents samples that:
  - Are **not numerical vectors**
    - Example: nominal data, text, images
  - The **model** has a low capacity/can't extract enough knowledge from the row features
    - Example: extraction of polynomial features

- Nominal data
  - Two nominal values can only be compared with the equality operator (**cannot be ordered**)
  - For this reason it is **incorrect** to map them to integer features:
    - E.g. 'red', 'green', 'blue' → [0, 1, 2]
    - Colors have no ordering
    - The model could infer ordering properties that do not describe correctly our data

- Nominal data
  - One of the simplest solutions is to use **one-hot encoding**:
    - `Red   → 0, 0, 1`
    - `Green → 0, 1, 0`
    - `Blue  → 1, 0, 0`
  - Pay attention: the **size of the output vector** is linear with the number of distinct values for the attribute
    - Some models (e.g. KNN, clustering) may have problems while working with high dimensional data

- Nominal data: 1-Hot vectors from dictionaries

In [1]:

```
from sklearn.feature_extraction import DictVectorizer
vect = DictVectorizer(sparse=False, dtype=int)

data = [{'model' : 'a', 'price' : 20000},
       {'model' : 'b', 'price' : 10000},
       {'model' : 'c', 'price' : 8000},
       {'model' : 'a', 'price' : 40000},
       {'model' : 'c', 'price' : 8500}]

print(vect.fit_transform(data))
```

Out[1]:

- Nominal data: 1-Hot vectors from dictionaries

In [1]:
```
...
print(vect.fit_transform(data))
```

```
data = [{'model' : 'a', 'price' : 20000},
        {'model' : 'b', 'price' : 10000},
        {'model' : 'c', 'price' : 8000},
        {'model' : 'a', 'price' : 40000},
        {'model' : 'c', 'price' : 8500}]
```

Out[1]:
```
[[     1      0      0 20000]
 [     0      1      0 10000]
 [     0      0      1  8000]
 [     1      0      0 40000]
 [     0      0      1  8500]]
```

a b c

- Nominal data: 1-Hot vectors from dictionaries
  - If you have training and test data use fit and transform separately:

In [1]:

```
train = data[:3]
test = data[3:]

vect = DictVectorizer(sparse=False, dtype=int)
vect.fit(train) # Learn vocabulary from training set
test_transformed = vect.transform(test)
```

Feature extraction

- 1-Hot encoding with *OneHotEncoder*
  - Allows passing data in tabular form (“feature matrix”)
  - Numerical values are also encoded – **beware!**
    - Some matrix manipulation is required to only encode categorical features

In [1]:

```
from sklearn.preprocessing import OneHotEncoder

X = np.array([
    ["a", 20000],
    ["b", 10000],
    ["c", 8000],
    ["a", 40000],
    ["c", 8500]
])

ohe = OneHotEncoder(sparse=False, dtype=int)
X_ohe = ohe.fit_transform(X[:,:1])

# same result as DictVectorizer (from before)
X_enc = np.hstack([ X_ohe, X[:,1:].astype(int)])
```

Feature extraction

- Textual data
  - Convert textual documents to count vectors
    - 1 feature for each word of the vocabulary that count the number of occurrences in the document
    - Scikit-learn transformer: *CountVectorizer*
    - Example:
      - "My cat. My dog. My cat."
      - "My dog. My house."

cat	dog	house	my
2	1	0	3
0	1	1	2

Feature extraction

- Textual data
  - Convert textual documents to count vectors
    - Drawback: frequent words have high scores for almost all documents
  - Solution: **TF-IDF** (Term Freq. Inverse Document Freq.)
    - Penalizes words that are common in all documents
    - Boosts words that are frequent in a document, but not in the others

- Textual data: TF-IDF

In [1]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vect = TfidfVectorizer(stop_words="english")

data = ["dog bites cat", "cat bites dog", "cat and dog house"]
print(vect.fit_transform(data).toarray())
```

convert to Numpy array

Out[1]:
```
[[0.67325467 0.52284231 0.52284231 0.        ]
 [0.67325467 0.52284231 0.52284231 0.        ]
 [0.         0.45329466 0.45329466 0.76749457]]
```

PoliTo DBMG

- Textual data: TF-IDF

In [1]:

```
...
data = ["dog bites cat", "cat bites dog", "cat and dog house"]
print(vect.fit_transform(data).toarray())
# Print the learned vocabulary
print(vect.vocabulary_)
```

bites cat dog house

stopword "and" has been removed

Out[1]:

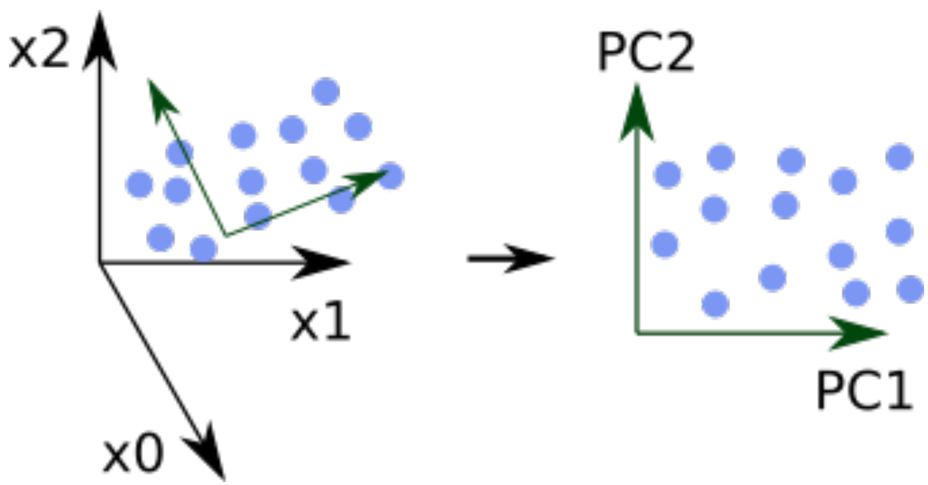
```
[[0.67325467 0.52284231 0.52284231 0.        ]   Doc 1
 [0.67325467 0.52284231 0.52284231 0.        ]   Doc 2
 [0.         0.45329466 0.45329466 0.76749457]]  Doc 3

{'dog': 2, 'bites': 0, 'cat': 1, 'house': 3}
```

specific of this document

Dimensionality reduction

- Useful when you want to reduce the number of features for high-dimensional data
  - For **graphical** representations
  - Before applying **classification** and **clustering** to give the features matrix a more **compact** representation

- Example: **PCA**
  - Reduces the dimensionality by finding the directions in the space where data has more variance



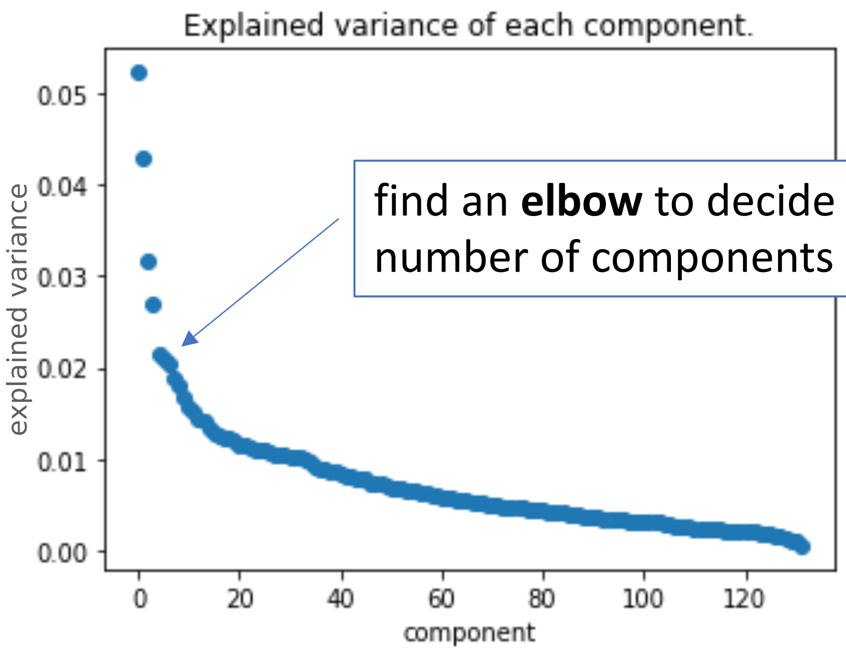
- PCA with Scikit-learn

```
from sklearn.decomposition import PCA


pca = PCA(n_components=5)
X_projection = pca.fit_transform(X)
```

- **n_components** specify the number of components that you want to keep after applying PCA
  - Should be <= the number of initial features
- The result is a features matrix with the specified number of features

- Choosing the correct number of components

```
pca = PCA(n_components=130)
X_projection = pca.fit_transform(X)
plt.plot(pca.explained_variance_ratio_, marker='o', linestyle='')
```



find an **elbow** to decide the number of components

- Applying the transformation and a classifier

```
pca = PCA(n_components=6)
X_projection = pca.fit_transform(X_train)
my_classifier.train(X_projection, y_train)

# PCA is already fit on training data: do not fit it on test set!
X_test_proj = pca.transform(X_test)
y_test_pred = my_classifier.predict(X_test_proj)
```

Missing values imputation

- We can fill missing values based on:
  - Univariate approaches
    - Using naive strategies (e.g. “0”, mean value)
  - Multivariate approaches
    - Infer values from known data
- Both approaches are provided in scikit-learn

Univariate imputation

`sklearn.impute.SimpleImputer`

- Imputation based on:
  - Constant values
    - `strategy=“constant”, fill_value=value`
  - Statistics
    - Mean, median, most frequent
    - `strategy=“mean”|”median”|”most_frequent”`

Multivariate imputation

e.g., `sklearn.impute.SimpleImputer`

- Indentify k nearest neighbors
  - Distance defined as distance over non-missing features
  - By default, NaN-aware Euclidean distance used, defined as:
    - $d^2(a,b) = w(a,b) \sum_i^n \mathbf{1}(a_i \neq \perp \wedge b_i \neq \perp)(a_i - b_i)^2$
    - $w(a,b) = \dfrac{n}{\sum \mathbf{1}(a_i \neq \perp \wedge b_i \neq \perp)}$
    - i.e., distance over non-missing dimensions
    - weighted by w
      - (larger w = more missing dimensions, the ones available are weighted more)
- Impute value based on mean over neighbors

- **Other preprocessing methods**
  - https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing