LISBON SCHOOL OF ECONOMICS & MANAGEMENT
UNIVERSIDADE DE LISBOA

Dimensionality Reduction Algorithms

CARLOS J. COSTA

Dimensional Reduction Algorithms

- dimensionality reduction seek and exploit the inherent structure in the data,
- unsupervised learning

Dimensional Reduction Algorithms

- Feature Extraction
- Feature Selection

Dimensional Reduction Algorithms

- Feature Extraction
  - PCA (principal Components analysis)
  - LDA (Linear Discriminant Analysis)
  - NMF (Non-negative Matrix Factorization)
  - TSVD (Truncate Singular Value Decomposition)

PCA

- Principal Component Analysis, or PCA, is a dimensionality-reduction method
- It is often used to reduce the dimensionality of large data sets
- The purpose is transforming a large set of variables into a smaller
- Containing most of the information in the large set.
- When data is linearly inseparable using PCA extension using kernels

PCA

- Standardization
- Covariance Matrix computation.
- Compute the eigenvectors and eigenvalues of the covariance matrix to identify the principal components
- Feature Vector
- Recast the Data Along the Principal Components Axes

```python
# Load libraries
from sklearn import datasets
from sklearn.decomposition import PCA
```

```python
# Load the Iris flower dataset:
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```python
# Create an PCA that will reduce the data down to 2 feature
PCAModel = PCA(n_components=2)

# run an PCA and use it to transform the features
XPCA = PCAModel.fit(X).transform(X)
```

```python
# Print the number of features
print('Original number of features:', X.shape[1])
print('Reduced number of features:', XPCA.shape[1])
```

```
Original number of features: 4
Reduced number of features: 2
```

```python
## View the ratio of explained variance
PCAModel.explained_variance_ratio_
```

```
array([0.92461621, 0.05301557])
```

LDA

- Linear Discriminant Analysis (LDA)
- Is a linear transformation techniques that is commonly used for dimensionality reduction (like PCA)
- Reducing features by maximizing class separation

```python
# Load libraries
from sklearn import datasets
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

```python
# Load the Iris flower dataset:
iris = datasets.load_iris()
X = iris.data
y = iris.target
```

```python
# Create an LDA that will reduce the data down to 1 feature
ldaModel = LinearDiscriminantAnalysis(n_components=2)

# run an LDA and use it to transform the features
XLda = ldaModel.fit(X, y).transform(X)
```

```python
# Print the number of features
print('Original number of features:', X.shape[1])
print('Reduced number of features:', XLda.shape[1])
```

```
Original number of features: 4
Reduced number of features: 2
```

```python
## View the ratio of explained variance
ldaModel.explained_variance_ratio_
```

```
array([0.99147248, 0.00852752])
```

NMF

- Non-negative Matrix Factorization
- Performs matrix factorization
- It can be applied for:
  - Recommender Systems,
  - Collaborative Filtering
  - topic modelling
  - dimensionality reduction.
- Does not provides the explained variance

NMF

```
# Load libraries
from sklearn import datasets
from sklearn.decomposition import NMF
```

```
# Load the Iris flower dataset:
iris = datasets.load_iris()
X = iris.data
```

```
# Create an NMF that will reduce the data down to 2 feature
NMFModel = NMF(n_components=2)

# run an LDA and use it to transform the features
XNMF = NMFModel.fit(X).transform(X)

# Print the number of features
print('Original number of features:', X.shape[1])
print('Reduced number of features:', XNMF.shape[1])
```

TSVD

- Truncate Singular Value Decomposition
- Used in sparce feature matrix

TSVD

```python
# Load libraries
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import TruncatedSVD
from scipy.sparse import csr_matrix
from sklearn import datasets
import numpy as np
```

```python
# Load the data
digits = datasets.load_digits()
# Standardize the feature matrix
X = StandardScaler().fit_transform(digits.data)
# Make sparse matrix
X_sparse = csr_matrix(X)
```

```python
# Create a TSVD
tsvdModel = TruncatedSVD(n_components=10)
```

```python
# Conduct TSVD on sparse matrix
X_sparse_tsvd = tsvdModel.fit(X_sparse).transform(X_sparse)
```

```python
# Show results
print('Original number of features:', X_sparse.shape[1])
print('Reduced number of features:', X_sparse_tsvd.shape[1])
```

```
Original number of features: 64
Reduced number of features: 10
```

```python
# Sum of first three components' explained variance ratios
tsvdModel.explained_variance_ratio_[0:3].sum()
```

```
0.3003938538627934
```

Dimensional Reduction Algorithms

- Feature Selection
  - Thresholding numerical features variance
  - Thresholding binary features variance
  - Handling high correlated features
  - Removing irrelevant features for Classification
  - RFEC

Thresholding numerical features variance

- The dataset has set of numerical features

Approach:

- Remove those with the low variance
- Low variance likely contains little information

Thresholding numerical features variance

```python
from sklearn import datasets
from sklearn.feature_selection import VarianceThreshold
```

```python
# Load iris data
iris = datasets.load_iris()

# Create features and target
X = iris.data
y = iris.target
```

```python
# Create VarianceThreshold object with a variance with a
#threshold of 0.5
thresholder = VarianceThreshold(threshold=.5)

# Conduct variance thresholding
XHighVariance = thresholder.fit_transform(X)
```

```python
# View first five rows with features with variances above
# threshold
XHighVariance[0:5]
```

```
array([[5.1, 1.4, 0.2],
       [4.9, 1.4, 0.2],
       [4.7, 1.3, 0.2],
       [4.6, 1.5, 0.2],
       [5. , 1.4, 0.2]])
```

Handling high correlated features

```python
# Load libraries
import pandas as pd
import numpy as np
```

```python
# Create feature matrix with two highly correlated features
X = np.array([[6, 12, 1],
              [5, 10, 0],
              [4, 8, 1],
              [3, 3, 0],
              [2, 5, 1],
              [1, 2, 0],
              [3, 6, 1],
              [5, 10, 0],
              [9, 19, 1]])

# Convert feature matrix into DataFrame
df = pd.DataFrame(X)
```

```python
# Create correlation matrix
corr_matrix = df.corr().abs()
# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
# Find index of feature columns with correlation greater than 0.95
to_drop = [column for column in upper.columns if any(upper[column] > 0.95)]
```

```python
# Drop features
df.drop(df[to_drop], axis=1)
```

Removing irrelevant features for Classification

Categorical features:

- Calculate Chi-square statistic between each feature and target

Quantitative features:

- Calculate ANOVA F-Value between each feature and target

Recursive Eliminating Feature

```
# Load libraries
from sklearn.datasets import make_regression
from sklearn.feature_selection import RFECV
from sklearn import datasets, linear_model
import warnings

# Suppress an annoying but harmless warning
warnings.filterwarnings(action="ignore", module="scipy", message="^internal gelsd")
```

```
# Generate features matrix, target vector, and the true coefficients
X, y = make_regression(n_samples = 10000,
                       n_features = 100,
                       n_informative = 2,
                       random_state = 1)
```

```
# Create a linear regression
olsModel = linear_model.LinearRegression()
```

```
# Create recursive feature eliminator that scores features by mean squared errors
rfecvModel = RFECV(estimator=olsModel, step=1, scoring='neg_mean_squared_error')

# Fit recursive feature eliminator
rfecvModel.fit(X, y)

# Recursive feature elimination
rfecvModel.transform(X)
```

```
# Number of best features
rfecvModel.n_features_
```