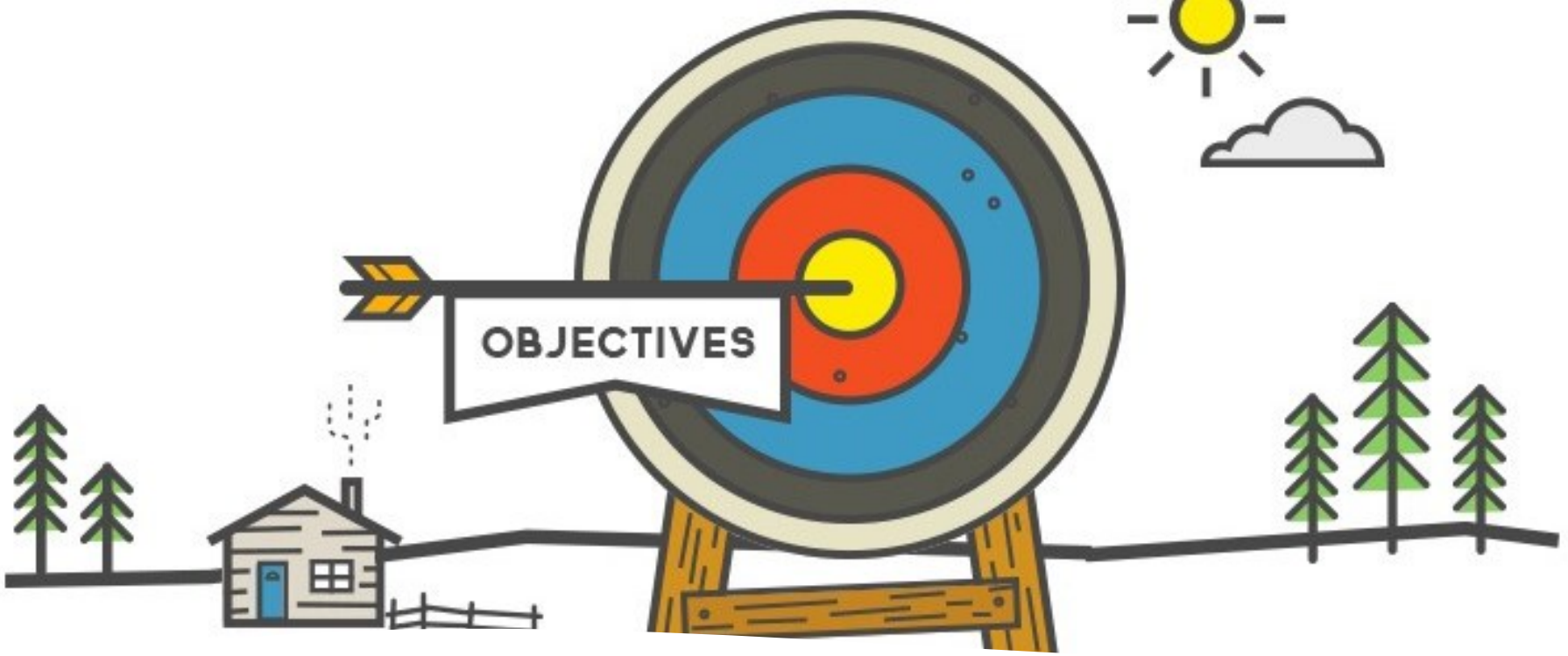
Lisbon School
of Economics
& Management
Universidade de Lisboa

Dimension Reduction Algorithms

Carlos J. Costa



Learning Goals

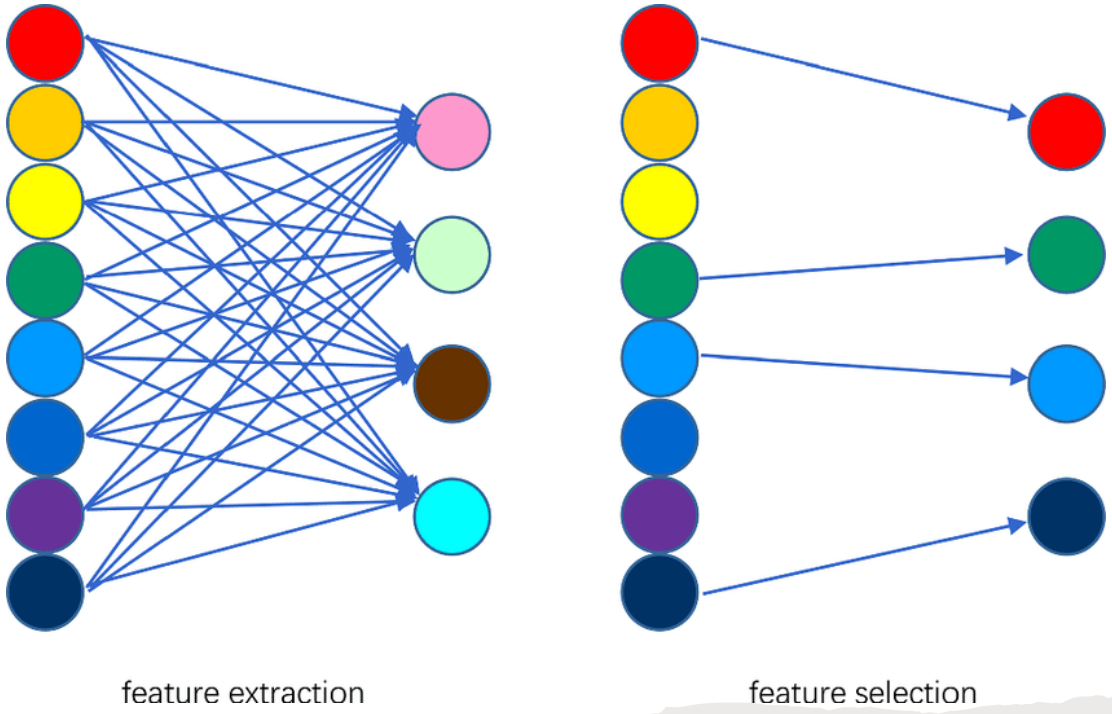
- Know concept of Dimension Reduction Algorithms
- Distinguish between Feature Extraction and Feature Selection
- Distinguish between main algorithms
- Apply algorithms by using python libraries

Summary

- Dimension reduction
- Feature Extraction vs Feature Selection
- Feature Extraction (PCA, LDA, NMF, TSVD)
- Feature Selection

Dimension Reduction Algorithms

- dimensionality reduction
- seek and exploit the inherent structure in the data
- Unsupervised learning
- Feature extraction
- Feature selection



1. Exploratory Factor Analysis (EFA)

- Exploratory Factor Analysis is a latent variable model whose goal is to discover the underlying structure that explains correlations among observed variables.
- There is no prior hypothesis about which variables load on which factors.
- EFA assumes:
  - observed variables are linear combinations of latent factors
  - plus measurement error
  - only common variance is modeled (not total variance)

1. Exploratory Factor Analysis (EFA)

- EFA is used when:
  - theory is weak or absent
  - scales are being developed
  - the dimensional structure is unknown

- In Python, sklearn provides FactorAnalysis, which corresponds to maximum-likelihood EFA (*without the rich diagnostics of psychometrics packages, but conceptually correct*).

1. Exploratory Factor Analysis (EFA)

```
rom sklearn.datasets
import load_iris
from sklearn.preprocessing
import StandardScaler
from sklearn.decomposition import FactorAnalysis
X, _ = load_iris(return_X_y=True)
X = StandardScaler().fit_transform(X)
efa = FactorAnalysis(
          n_components=2,
          rotation="varimax",
          random_state=0)
X_efa = efa.fit_transform(X)
print("Factor loadings shape:", efa.components_.shape)
```

2. Confirmatory Factor Analysis (CFA)

- Confirmatory Factor Analysis is also a latent variable model, but the structure is imposed a priori.
- Before fitting the model, the researcher specifies:
  - number of factors
  - which observed variables load on which factors
  - which loadings are fixed to zero
  - whether factors correlate

- CFA is not exploratory. It is a hypothesis-testing model, usually evaluated using global fit indices (CFI, TLI, RMSEA).

2. Confirmatory Factor Analysis (CFA)

- CFA is not implemented in sklearn, because sklearn is not a Structural Equation Modeling framework.
- In Python, CFA is performed with libraries that are sklearn-compatible, such as factor_analyzer.
- Needs:

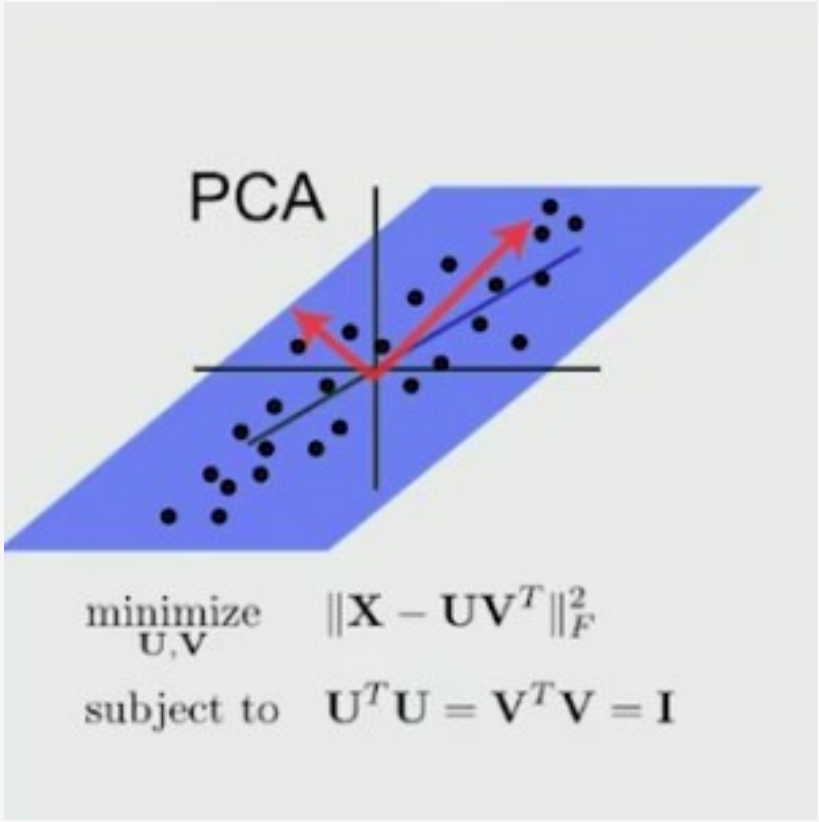
```
pip install factor_analyzer
```

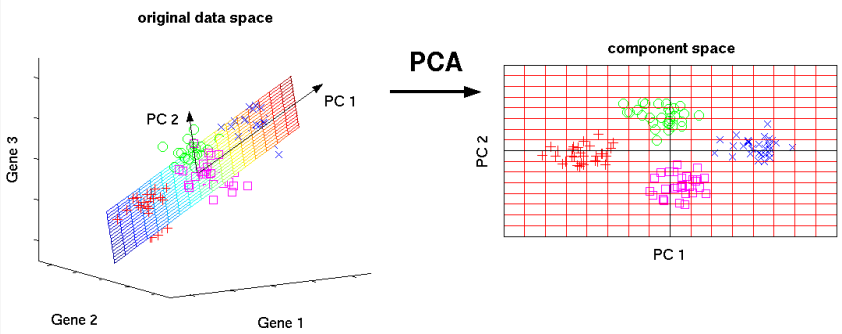
2. Confirmatory Factor Analysis (CFA)

```
import pandas as pd
from factor_analyzer import ConfirmatoryFactorAnalyzer, ModelSpecificationParser
import numpy as np
# Create a sample DataFrame for demonstration
# In a real scenario, you would load your data, e.g., df = pd.read_csv("items.csv")
np.random.seed(42)
data = np.random.rand(100, 8) * 10 # 100 samples, 8 variables
columns = [f"V{i}" for i in range(1, 9)]
df = pd.DataFrame(data, columns=columns)


# df must contain only the observed variables
# Example column names: V1 ... V8
model_dict = {
   "F1": ["V1", "V2", "V3", "V4"],
   "F2": ["V5", "V6", "V7", "V8"]
}
model_spec = ModelSpecificationParser.parse_model_specification_from_dict(
df, model_dict)
cfa = ConfirmatoryFactorAnalyzer(model_spec, disp=False, max_iter=1000) # Increased
max_iter
cfa.fit(df.values)
print("CFA loadings:\n", cfa.loadings_)
```

3. PCA (Principal Component Analysis)

- **Principal Component Analysis** is not a latent variable model.
- PCA:
  - finds **orthogonal directions** that maximize total variance
  - treats all variance (common + unique + error) as signal
  - produces deterministic linear combinations of variables
- Components are **mathematical constructs**, not theoretical factors.
- PCA is used for:
  - dimensionality reduction
  - visualization
  - preprocessing
  - noise reduction





3. PCA (Principal Component Analysis)

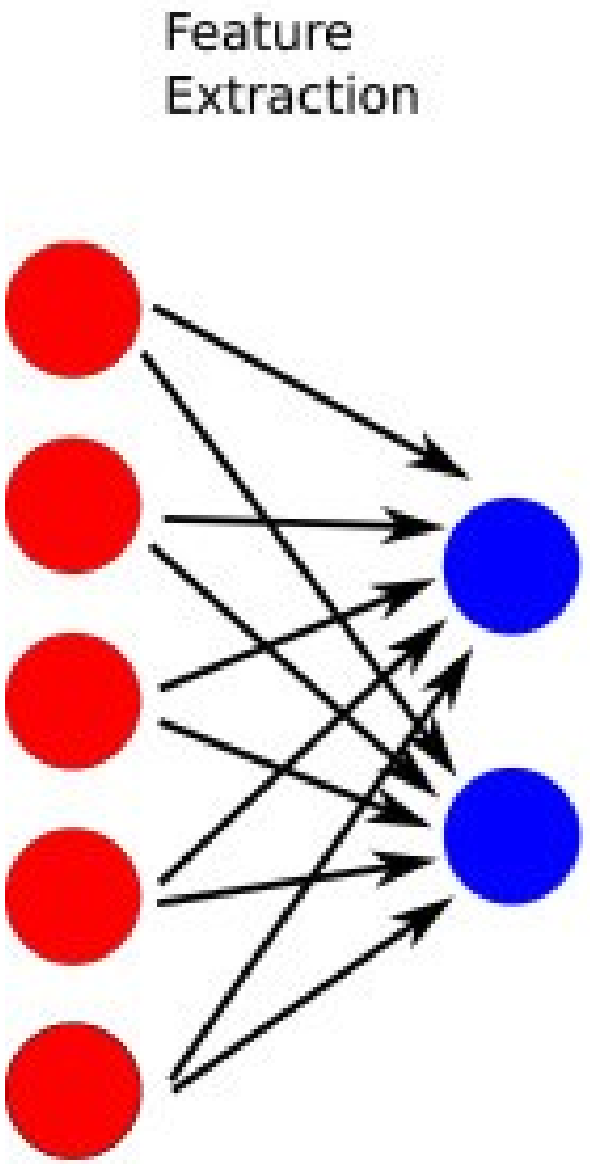
- Principal Component Analysis
- Data is first centered around its mean
- then finding the eigenvectors and eigenvalues of the covariance matrix.
- The eigenvectors represent the directions of maximum variance, while the eigenvalues represent the amount of variance explained by each eigenvector.
- The eigenvectors are then used to project the data onto a lower-dimensional space.
- The number of principal components to keep is determined by the amount of variance we want to retain.

3. PCA (Principal Component Analysis)

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA


X, _ = load_iris(return_X_y=True)
X = StandardScaler().fit_transform(X)


pca = PCA(n_components=2, random_state=0)
X_pca = pca.fit_transform(X)


print("Explained variance ratio:", pca.explained_variance_ratio_)
```

Dimension Reduction Algorithms

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



PCA

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

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

```
# 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)
```

```
# 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
```

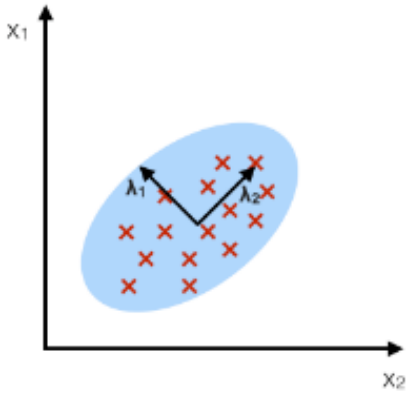
```
## View the ratio of explained variance
PCAModel.explained_variance_ratio_
```

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

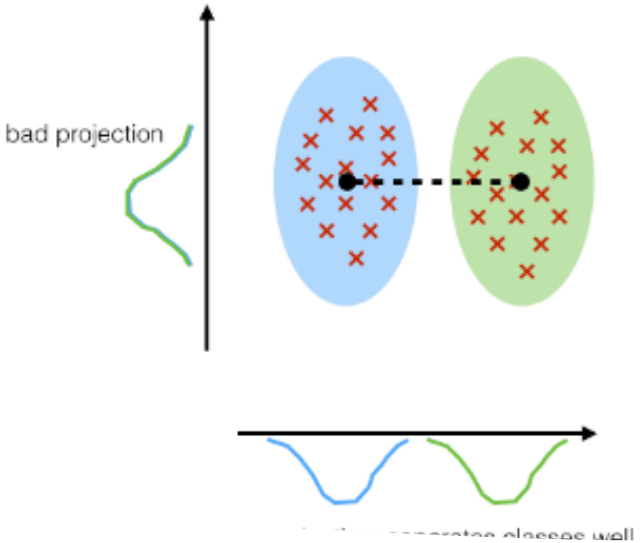
4. LDA (Linear Discriminant Analysis)

- Linear Discriminant Analysis is a supervised method.
- Unlike PCA:
  - it uses class labels
  - it maximizes between-class variance / within-class variance
  - the goal is class separation, not structure discovery
- LDA is both:
  - a classifier
  - a discriminative dimensionality reduction technique
- LDA is inappropriate when:
  - there are no labels
  - the goal is latent construct measurement

PCA:
component axes that maximize the variance



LDA:
maximizing the component axes for class-separation



LDA

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

4. LDA (Linear Discriminant Analysis)

```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])
```

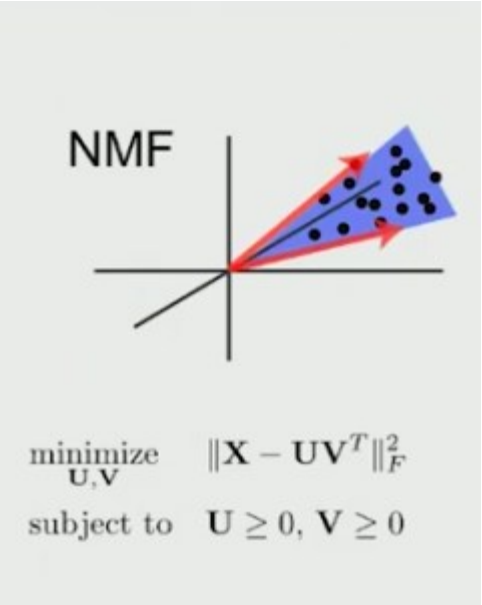
5. NMF (Non-negative Matrix Factorization)

- NMF is a matrix factorization technique, not a covariance or latent-variable model.
- It decomposes:
- X≈WHX \approx W HX≈WH
- under the constraint that all values are non-negative.
- This produces:
  - additive
  - parts-based
  - often interpretable representations
- NMF is common in:
  - topic modeling
  - recommender systems
  - image decomposition

5. NMF (Non-negative Matrix Factorization)

- Non-negative Matrix Factorization
- where a matrix is factorized into (usually) two matrices, with the property that all three matrices have no negative elements.

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



5. NMF (Non-negative Matrix Factorization)

```
: 1 # Load libraries
  2 from sklearn import datasets
  3 from sklearn.decomposition import NMF
```

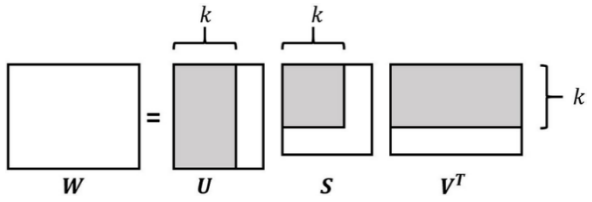
```
: 1 # Load the Iris flower dataset:
  2 iris = datasets.load_iris()
  3 X = iris.data
```

```
: 1 # Create an NMF that will reduce the data down to 2 feature
  2 NMFModel = NMF(n_components=2)
  3
  4 # run an LDA and use it to transform the features
  5 XNMF = NMFModel.fit(X).transform(X)
  6
  7 # Print the number of features
  8 print('Original number of features:', X.shape[1])
  9 print('Reduced number of features:', XNMF.shape[1])
```

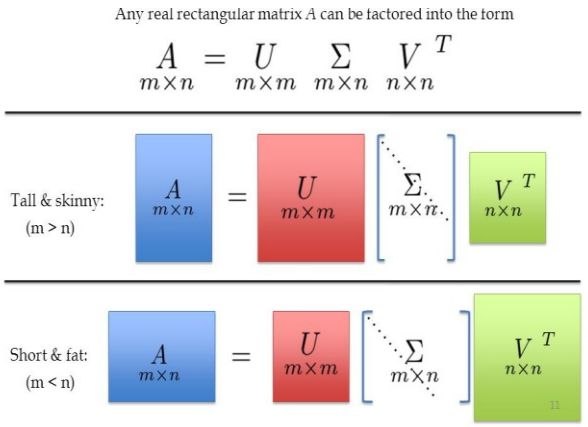
6. TSVD (Truncated Singular Value Decomposition)

- Explanation
- Truncated SVD is a purely algebraic low-rank approximation.
- Key characteristics:
    - equivalent to PCA only if data are centered
    - does not center data in sklearn
    - works efficiently on sparse matrices
- In text mining, TSVD is known as Latent Semantic Analysis (LSA).

6. TSVD (Truncated Singular Value Decomposition)



- Truncated Singular Value Decomposition
- Used in sparce feature matrix
- Contrary to PCA, this estimator does not center the data before computing the singular value decomposition.



6. TSVD (Truncated Singular Value Decomposition)

```
# 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
```

```
# 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)
```

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

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

```
# Show results
print('Original number of features:', X_sparse.shape
print('Reduced number of features:', X_sparse_tsvd.s
```

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

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

```
0.3003938538627934
```

Comparing

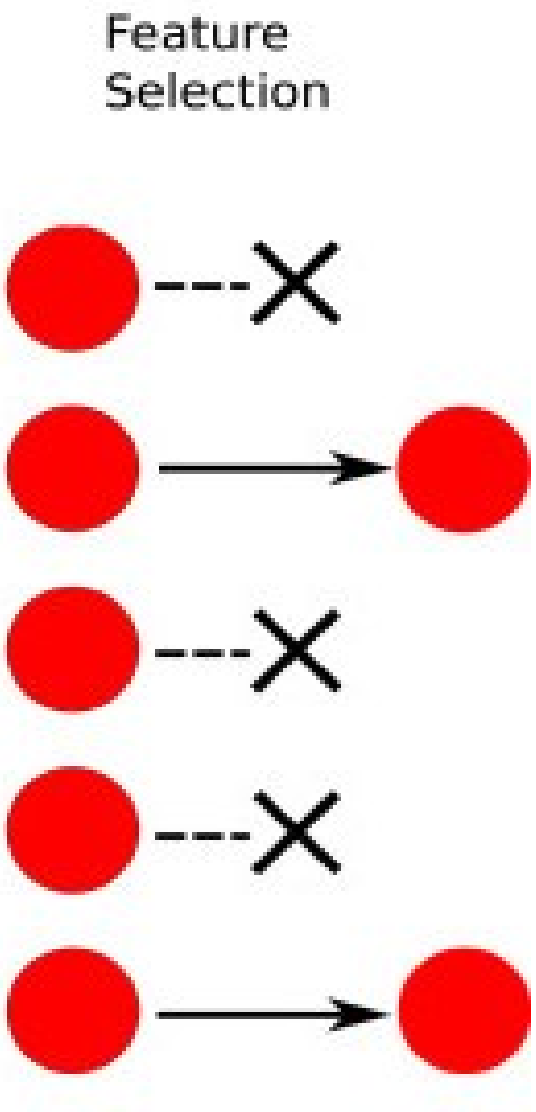
Method	Supervised	Latent variables	Models error	Uses labels	Main purpose
EFA	No	Yes	Yes	No	Discover latent structure
CFA	No	Yes	Yes	No	Test measurement model
PCA	No	No	No	No	Maximize total variance
LDA	Yes	No	Implicit	Yes	Class separation
NMF	No	No	No	No	Parts-based representation
TSVD	No	No	No	No	Low-rank approximation

Comparing

- measurement theory (EFA, CFA)
- variance-based projections (PCA)
- discriminative learning (LDA)
- matrix factorizations (NMF, TSVD)
- PCA decomposes the total covariance matrix; EFA decomposes the covariance matrix into common and unique variance under a probabilistic latent-variable model.

Dimension Reduction Algorithms

- Feature Selection
  - Thresholding numerical features variance
  - Thresholding binary features variance
  - Handling high correlated features
  - Removing irrelevant features for Classification
  - RFE (Recursive Feature Elimination)



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

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

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

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

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

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

```
# 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 Feature Elimination

```
# 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_
```

Conclusion

- Dimensionality reduction
- Feature Extraction and Feature Selection
- Feature Extraction (PCA, LDA, NMF, TSVD)
- Feature Selection