14 - Dimensionality Reduction

Data Science Practicum 2021/22, Lesson 14

Marko Tkalčič

Univerza na Primorskem

Table of Contents

Machine Learning

Unsupervised Learning

- Data-driven
- Uses unlabeled data
- Goal: Structure recognition
- Types
  - Clustering (divide by similarity)
    - E.g. Targeted Marketing
  - Dimensionality Reduction
    - E.g. compacting data

Document	f1	f2	f3
It was the best of times	1	2	1
it was the worst of times	3	3	3
it was the age of wisdom	4	1	2
it was the age of foolishness	2	2	1

Supervised Learning

- Task-driven
- Uses pre-categorized data
- Goal: predictive modeling
- Types
  - Classification: labels are discrete classes
  - Regression: labels are continuous variables

Document	f1	f2	f3	Label
It was the best of times	1	2	1	C
it was the worst of times	3	3	3	S
it was the age of wisdom	4	1	2	C
it was the age of foolishness	2	2	1	?

Table of Contents

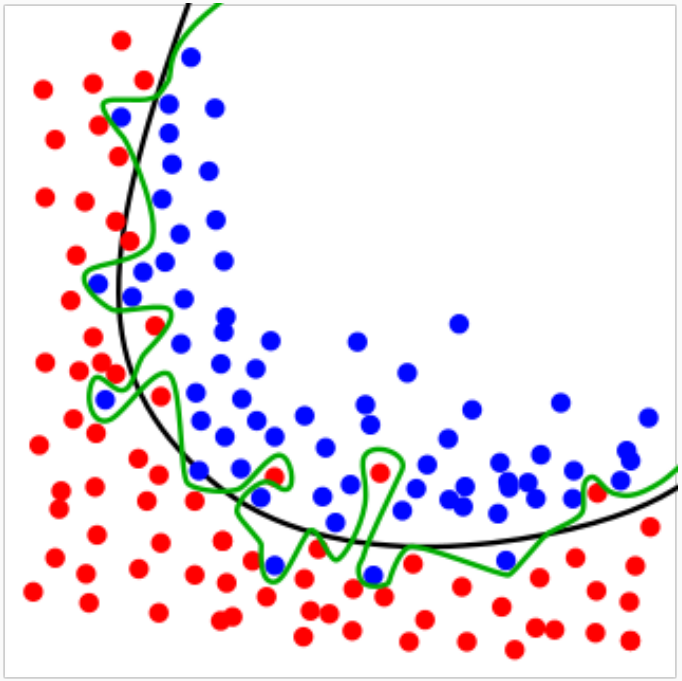
Dimensionality Reduction

- A high-dimensional dataset has many 100s, 1000s or even millions of variables (columns)
  - many variables are highly correlated among them
- We want less variables - a simpler representation of the data
- Fewer input dimensions often means correspondingly **fewer parameters of the machine learning model** (degrees of freedom)
- This linear regression model has two degrees of freedom (two parameters)

$$\hat{y} = x_1\beta_1 + x_2\beta_2$$

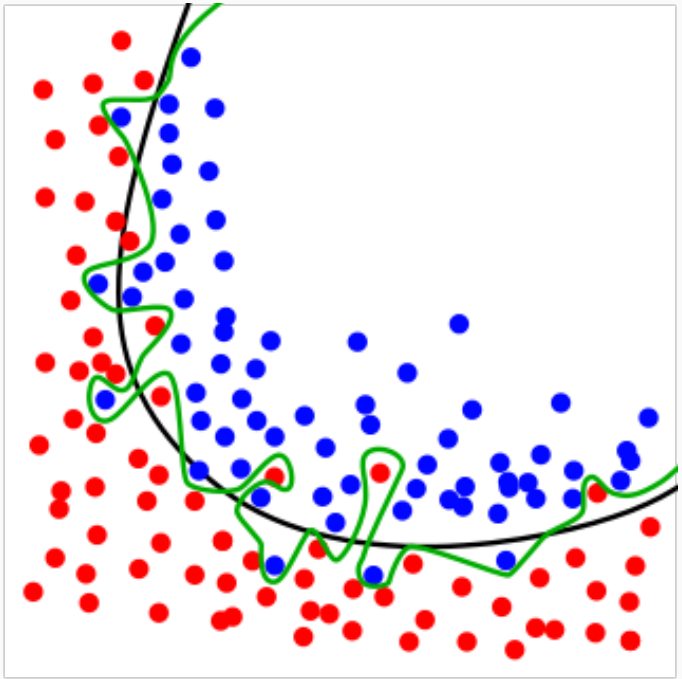
- More parameters than observations -> overfitting the training dataset.

Overfitting



- Underfitting: model is too simple
- Overfitting: model is too complex
  - the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably
  - An **overfitted model** is a statistical model that contains more parameters than can be justified by the data

Overfitting



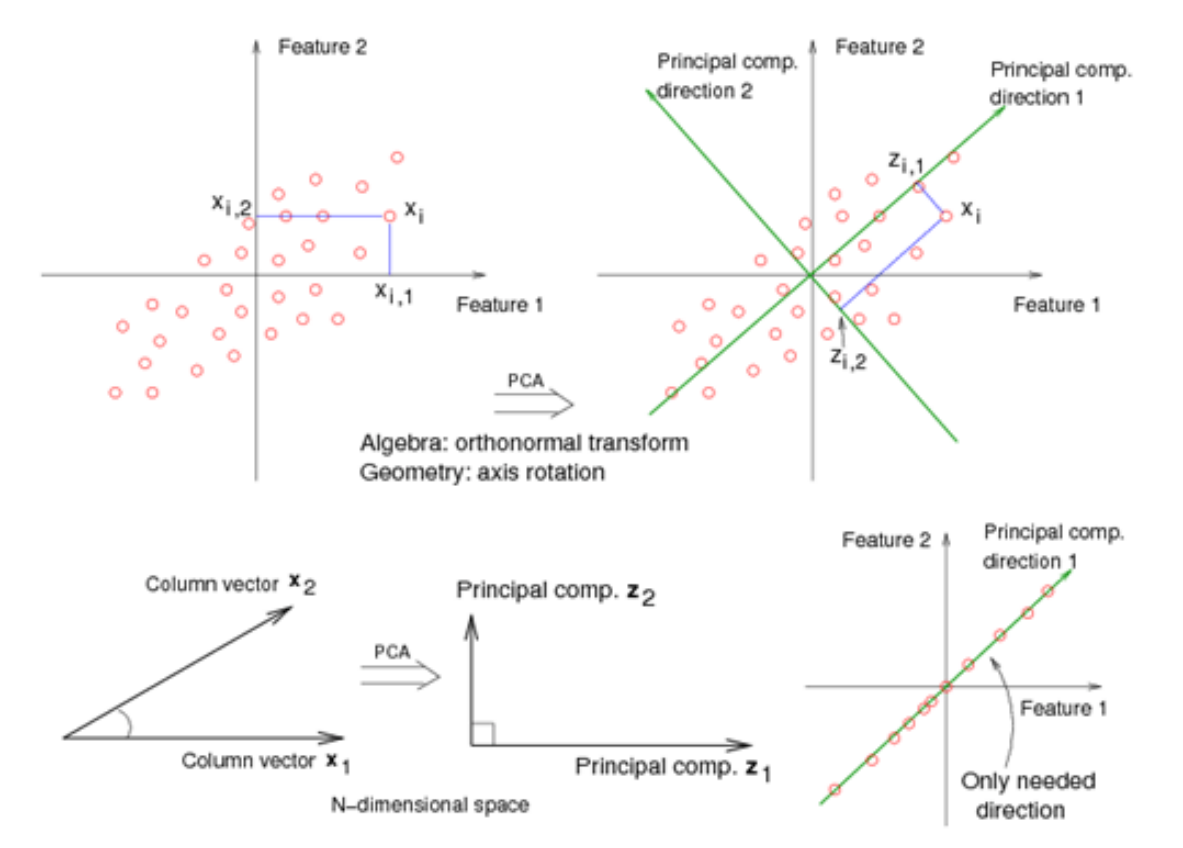
- Underfitting: model is too simple
- Overfitting: model is too complex
  - the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably
  - An **overfitted model** is a statistical model that contains more parameters than can be justified by the data

*As a rule of thumb, use **the rule of 10**, namely the amount of training data you need for a well performing model is 10x the number of parameters in the model.*

Table of Contents

Principal Component Analysis - PCA

- Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. [Wikipedia]



Create a dataset

- `make_classification` - Generate a random n-class classification problem.

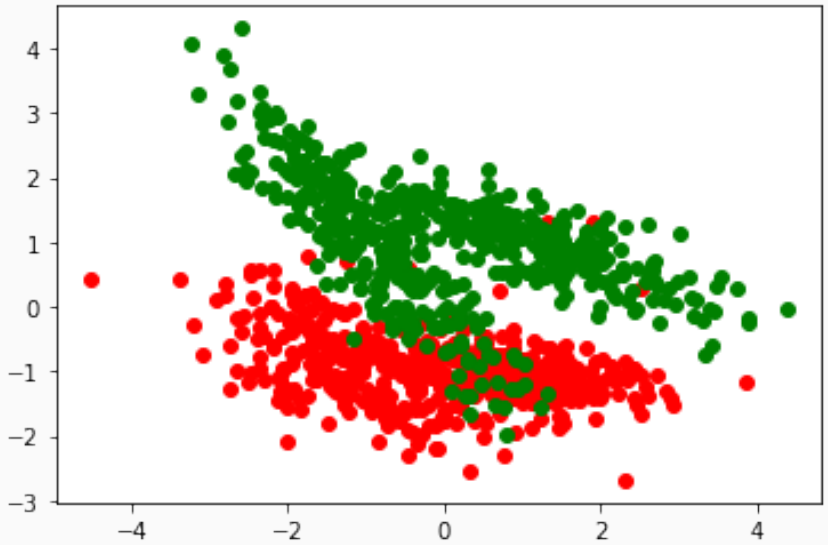
```
def make_classification(n_samples=100, n_features=20, *, n_informative=2,
                        n_redundant=2, n_repeated=0, n_classes=2,
                        n_clusters_per_class=2, weights=None, flip_y=0.01,
                        class_sep=1.0, hypercube=True, shift=0.0, scale=1.0,
                        shuffle=True, random_state=None):
```

Create a dataset

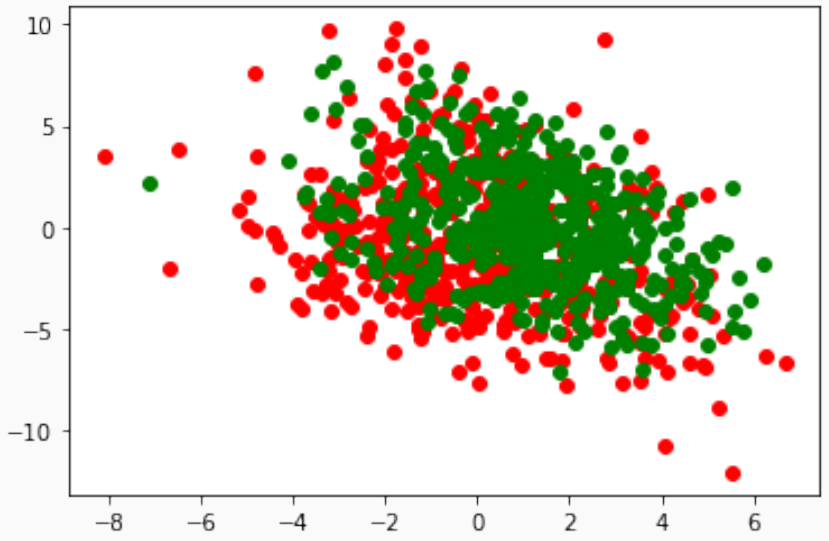
- `make_classification` - Generate a random n-class classification problem.

```
def make_classification(n_samples=100, n_features=20, *, n_informative=2,
                        n_redundant=2, n_repeated=0, n_classes=2,
                        n_clusters_per_class=2, weights=None, flip_y=0.01,
                        class_sep=1.0, hypercube=True, shift=0.0, scale=1.0,
                        shuffle=True, random_state=None):
```

```
X, y = make_classification(n_samples=1000, n_features=2,
    n_informative=2, n_redundant=0, random_state=7)
from matplotlib import pyplot as plt
plt.scatter(X[y==0,0],X[y==0,1],color="red")
plt.scatter(X[y==1,0],X[y==1,1],color="green")
plt.show()
```




```
X, y = make_classification(n_samples=1000, n_features=20,
    n_informative=10, n_redundant=10, random_state=7)
```



PCA with sklearn

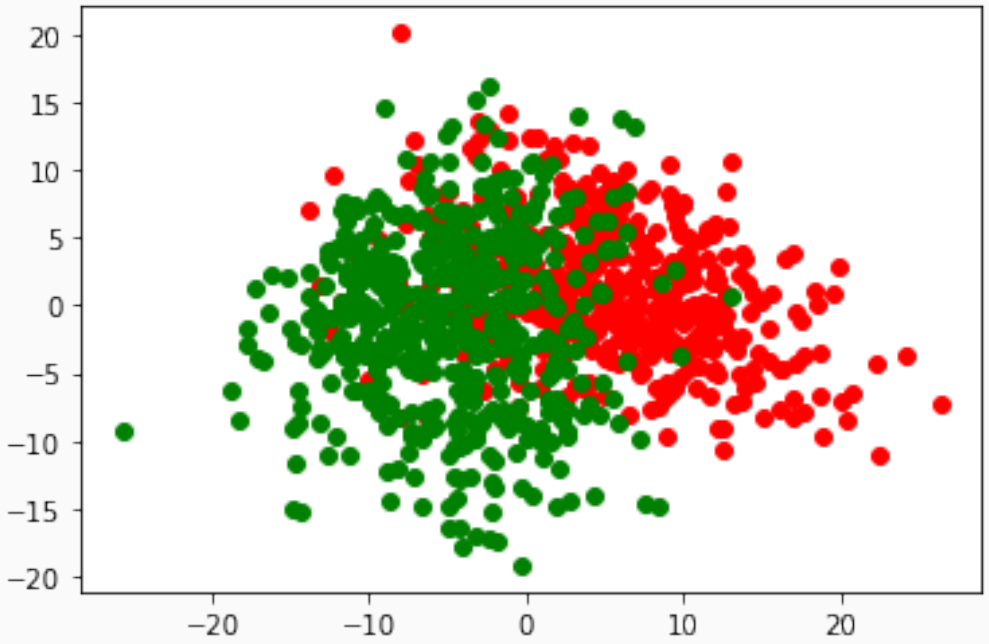
```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
model = pca.fit(X)
X_pca = pca.transform(X)
print(("Explained Variance: %s") % (model.explained_variance_ratio_))
print("Model components: ",model.components_)
```

```
Explained Variance: [0.34089892 0.19205661]
Model components:  [[-0.05066492 -0.14766907  0.0216404  -0.47059144 -0.0386959   0.32999151
   0.23428329 -0.07611129  0.07248979 -0.08817398  0.10508724 -0.28333493
  -0.12270876  0.19630055 -0.33842973  0.49929036 -0.0575997   0.14178668
  -0.06617724  0.18182412]
 [-0.21796152  0.24837002  0.10364418  0.01506274  0.28600193  0.22637386
  -0.36603144  0.07988956 -0.46225593 -0.09073186  0.10167718  0.17383909
   0.05715123  0.03754531 -0.34564107  0.09525623  0.29350637 -0.33008571
  -0.0913769   0.08539152]]
```

Visualize

```
import pandas as pd
import numpy as np

# visualize
df = pd.DataFrame(data=X_pca)
print(df.shape)
plt.scatter(df.iloc[y==0,0],df.iloc[y==0,1],color="red")
plt.scatter(df.iloc[y==1,0],df.iloc[y==1,1],color="green")
plt.show()
```



Test with ML

- We will train a simple logistic regression model (details do not matter)
  - 10x fold validation
  - Accuracy of classification into `[0,1]`
- The better the features the higher the accuracy

```
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression
```

```
model = LogisticRegression()
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
    random_state=1)
n_scores = cross_val_score(model, X, y, scoring='accuracy',
    cv=cv, n_jobs=-1)
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy: 0.824 (0.034)
```

```
model = LogisticRegression()
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
    random_state=1)
n_scores = cross_val_score(model, df, y, scoring='accuracy',
    cv=cv, n_jobs=-1)
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

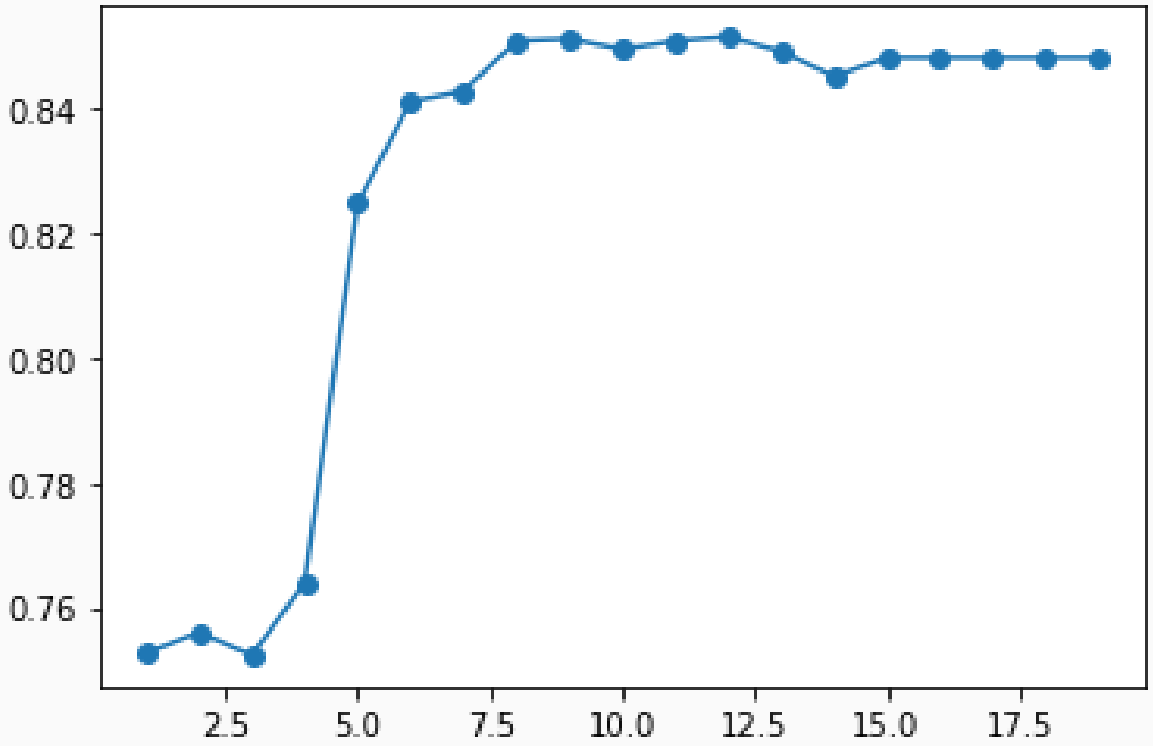
```
Accuracy: 0.697 (0.035)
```

Exercise

- Create a dataset with 1000 samples, 20 features, 8 informative, 5 redundant
- Apply PCA by setting the number of components between 1 and 20
- For each number of component calculate accuracy
- Plot the accuracy vs number of components

Exercise

- Create a dataset with 1000 samples, 20 features, 8 informative, 5 redundant
- Apply PCA by setting the number of components between 1 and 20
- For each number of component calculate accuracy
- Plot the accuracy vs number of components



Exercise

```
X, y = make_classification(n_samples=1000, n_features=20, n_informative=8, n_redundant=5, random_state=4)

# eval function
def eval_model(X,y):
    model = LogisticRegression()
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
    print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
    return mean(n_scores)

acc = []
n_comp = range(0,20)
for i in n_comp:
    print("Num of components = ",i)
    pca = PCA(n_components = i)
    model = pca.fit(X)
    X_pca = pca.transform(X)
    df = pd.DataFrame(data=X_pca)
    acc.append( eval_model(df,y))

plt.plot(n_comp,acc,"-o")
plt.show()
```

Table of Contents

Singular Value Decomposition

- works well on sparse data
  - Recsys data
  - bag of words

$$A = U\Sigma V^T$$

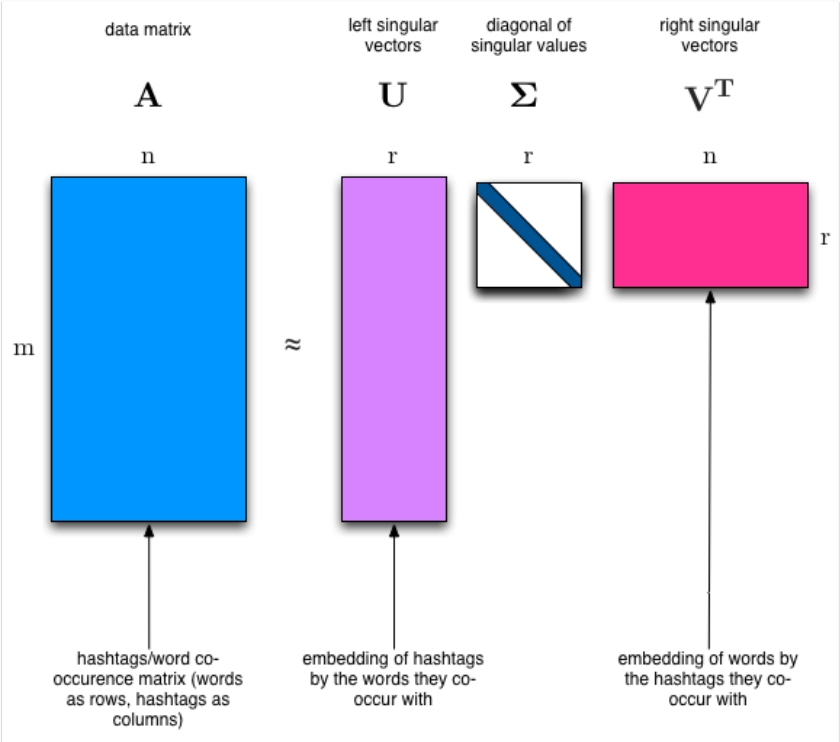
- A: m x n
- U: m x m
- Σ: m x n
- V: n x n

Truncated SVD - intuition

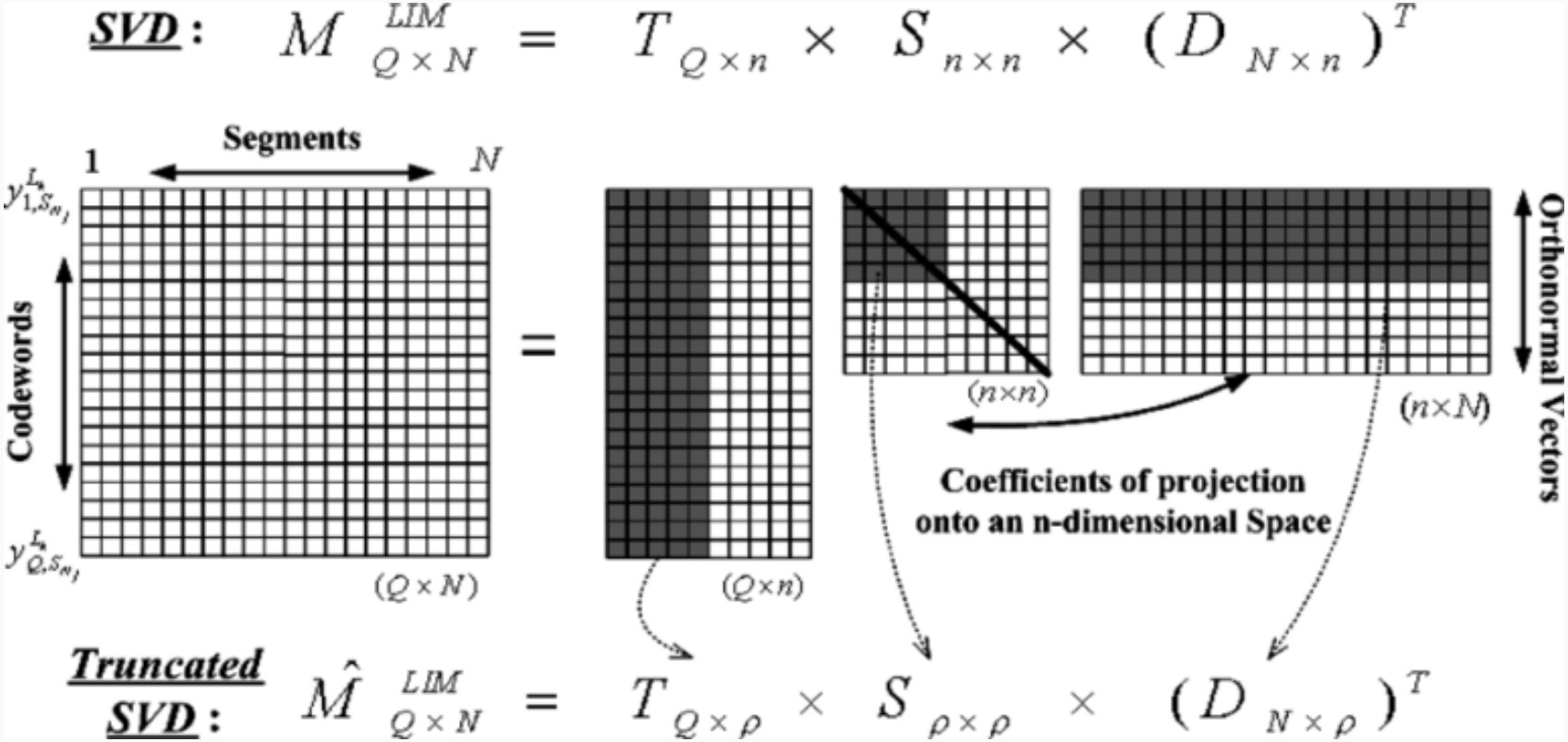
$$A = U\Sigma V^T$$

- A: m x n
- U: m x m
- Σ: m x n
- V: n x n

- U, V are reduced:
  - U: m x r
  - V: r x n



Truncated SVD



Truncated SVD - sklearn

- training set X: m rows (data points), n columns (variables/components/features)
- X_k is low rank approximation of $X

$$X \approx X_k = U_k \Sigma_k V_k^T$$

- new data with less columns (m rows, k columns = n_components in sklearn):

$$U_k \Sigma_k$$

```
sklearn.decomposition.TruncatedSVD(n_components=2, *, algorithm='randomized', n_iter=5, random_state=None, tol=0.0
```

SVD with sklearn

```
X, y = make_classification(n_samples=1000, n_features=20, n_informative=10, n_redundant=10, random_state=7)
eval_model(X,y)
```

```
Accuracy: 0.824 (0.034)
```

SVD with sklearn

```
X, y = make_classification(n_samples=1000, n_features=20, n_informative=10, n_redundant=10, random_state=7)
eval_model(X,y)
```

```
Accuracy: 0.824 (0.034)
```

```
# define transform
svd = TruncatedSVD(n_components=2, n_iter=7, random_state=42)
# prepare transform on dataset
svd.fit(X)
# apply transform to dataset
transformed = svd.transform(X)
eval_model(transformed,y)
```

```
0.7916666666666666
```

Inverse transform

```
reconstructed = svd.inverse_transform(transformed)
```

$$X \approx X_k = U_k \Sigma_k V_k^T$$

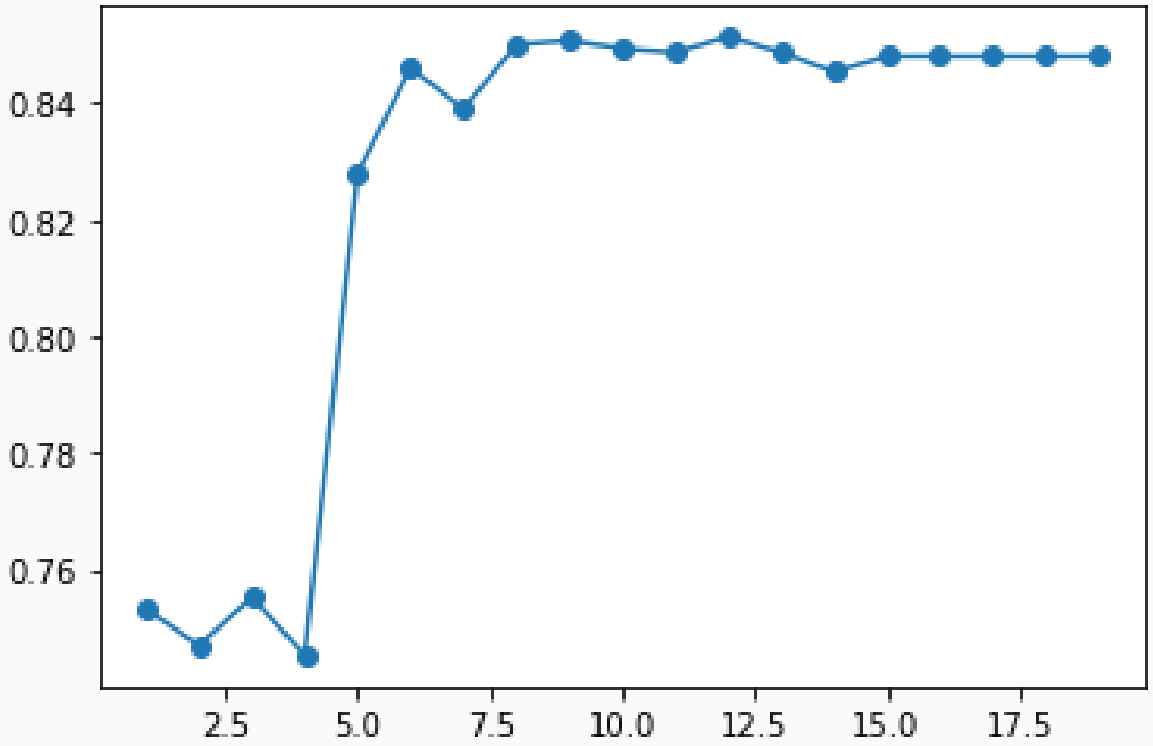
- Image compression
- Matrix factorization for recommender systems

Exercise

- Create a dataset with 1000 samples, 20 features, 8 informative, 5 redundant
- Apply SVD by setting the number of components between 1 and 20
- For each number of components calculate accuracy
- Plot the accuracy vs number of components

Exercise

- Create a dataset with 1000 samples, 20 features, 8 informative, 5 redundant
- Apply SVD by setting the number of components between 1 and 20
- For each number of components calculate accuracy
- Plot the accuracy vs number of components



Exercise

```
X, y = make_classification(n_samples=1000, n_features=20, n_informative=8, n_redundant=5, random_state=4)

# eval function
def eval_model(X,y):
    model = LogisticRegression()
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
    print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
    return mean(n_scores)

acc = []
n_comp = range(0,20)
for i in n_comp:
    print("Num of components = ",i)
    svd = TruncatedSVD(n_components=i, n_iter=7, random_state=42)
    svd.fit(X)
    transformed = svd.transform(X)
    df = pd.DataFrame(data=transformed)
    acc.append( eval_model(df,y))

plt.plot(n_comp,acc,"-o")
plt.show()
```

Table of Contents

References

Part of the material has been taken from the following sources. The usage of the referenced copyrighted work is in line with fair use since it is for nonprofit educational purposes.

- https://machinelearningmastery.com/dimensionality-reduction-algorithms-with-python/
- https://en.wikipedia.org/wiki/Overfitting
- https://medium.com/@malay.haldar/how-much-training-data-do-you-need-da8ec091e956
- https://www.bogotobogo.com/python/scikit-learn/scikit_machine_learning_Data_Compresssion_via_Dimensionality_Reduction_1_Pri
- https://scikit-learn.org/stable/modules/decomposition.html#lsa
- https://www.researchgate.net/figure/Diagram-of-SVD-and-truncated-SVD-for-feature-transformation_fig3_3457515
- https://laptrinhx.com/nlp-tutorial-topic-modeling-with-singular-value-decomposition-svd-and-truncated-svd-fbpca-and-sklearn-python-libraries-used-3042841100/