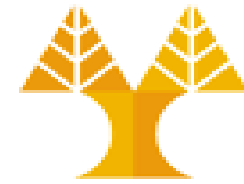


# EPL448: Data Mining on the Web – Lab 7



University of Cyprus  
Department of  
Computer Science

Παύλος Αντωνίου

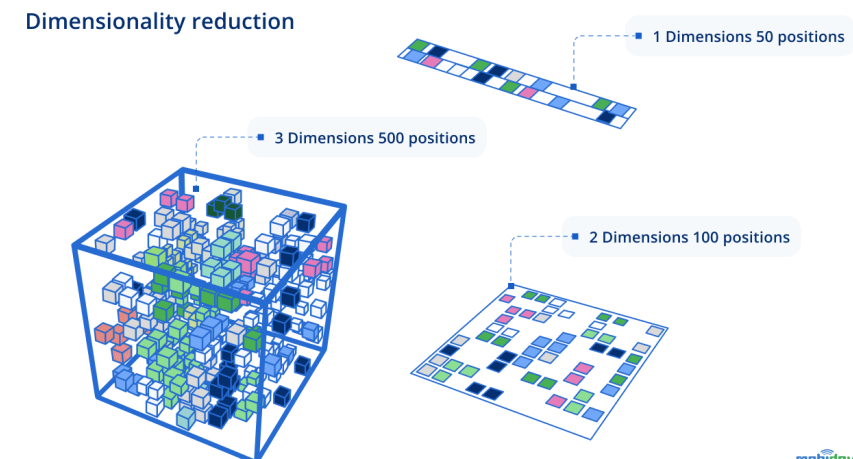
Γραφείο: B109, ΘΕΕ01

# Feature selection & Feature extraction



- Used to eliminate the number of features (columns) leading to:
  - Less computation time when training predictive modelling algorithms
  - Easier to understand, visualize a dataset
- Useful in datasets with large number of features that may not all contribute meaningfully to the prediction task.
- Feature selection: **Select a subset of the original feature set**
- Feature extraction: **Build a new set of features from the original feature set**

- Dimensionality Reduction techniques: used for mapping observations in high-dimensional (high number of features) space to lower number of dimensions (features) while preserving structure, e.g pairwise distances, between observations



# Feature selection



- **Select a subset of the original feature set**
  - **Feature selection using statistical techniques:** select features based on their statistical properties or statistical relationship with target variable (e.g., correlation, variance, chi-squared test)
    - fast but not accurate methods
  - **Feature selection using feature importance:** ensemble predictive modelling techniques (e.g., decision trees, random forest, gradient boosting) evaluate features importance during their training process
    - moderate speed and better accuracy
  - **Feature selection using the predictive performance of model:** iteratively select a subset of “important” features based on which the model is trained to achieve the highest predictive performance (e.g., forward/backward selection)
    - slow (computationally expensive) but accurate methods

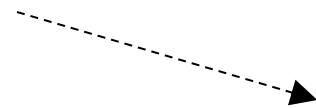
# Feature selection using correlation



- pandas corr() method to compute pairwise correlation between all dataset columns
  - available correlation methods: pearson, kendall, spearman

# Wine dataset: 178 wine observations by 13 features. Wines classified into 3 types.

```
df = pd.read_csv('wine.csv')
```



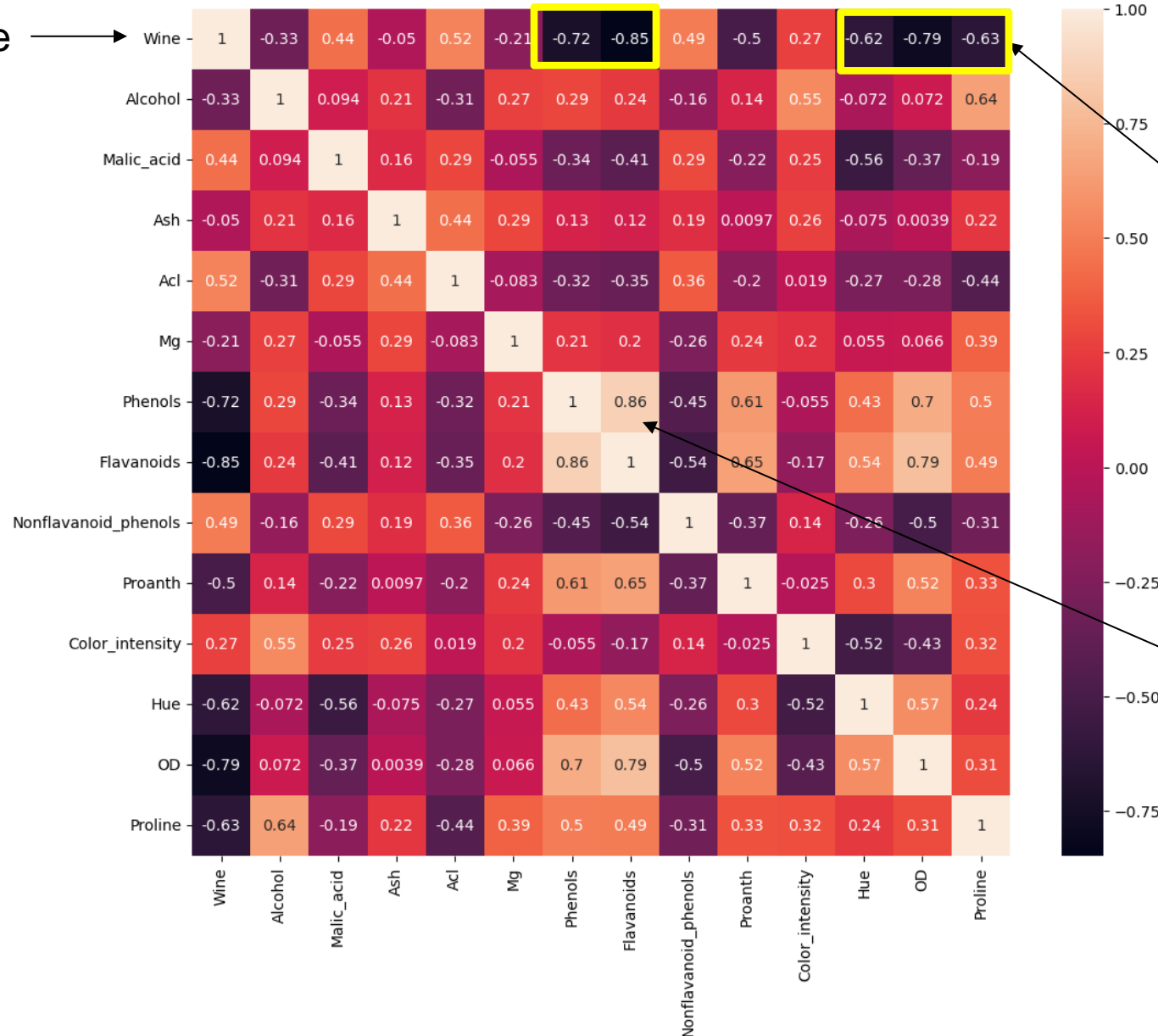
	Wine	Alcohol	Malic_acid	Ash	AcI	Mg	Phenols	Flavanoids	Nonflavanoid_phenols	Proanth	Color_intensity	Hue	OD	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	3	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740
174	3	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750
175	3	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835
176	3	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840
177	3	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560

```
fig, ax = plt.subplots( figsize = ( 12 , 10 ) )  
sns.heatmap(df.corr(method='pearson'), annot = True)
```

# Feature selection using correlation



Target variable →



Observations:

- Features Phenols, Flavanoids, Hue, OD, Proline are highly negatively correlated to the target value (Wine) ; see the first line of the heat map
- Features Phenols & Flavanoids are highly (positively) correlated to each other. One of them could be removed if the dataset had a large number of features. This is not the case so we can keep them.

# Feature selection using variance

---



- Quick and lightweight way of eliminating features with very low variance, i. e. features with not much useful information
  - *Variance* shows how spread out the feature distribution is (the average squared distance from the mean)

```
import numpy as np
np.std([2, 2, 2, 2, 2, 2, 2, 2]) # 0.0
```

- If a feature has 0 variance it is completely useless. Using a feature with zero variance only adds to model complexity, not to its predictive power.

```
np.std([5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6]) # 0.28747978728803447
```

- Features that go around a single constant are also useless. In other words, any feature with close to 0 variance should be dropped.
-

# Feature selection using variance



- Scikit-learn provides VarianceThreshold estimator that accepts a threshold cut-off and removes all features with variance below that threshold

```
X = df.drop(columns=['Wine']) # features dataframe
y = df['Wine'] # target dataframe
X.describe()
```

	Alcohol	Malic_acid	Ash	Acid	Mg	Phenols	Flavanoids	Nonflavanoid_phenols	Proanth	Color_intensity	Hue	OD	Proline
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.590899	5.058090	0.957449	2.611685	746.893258
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	0.572359	2.318286	0.228572	0.709990	314.907474
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	1.280000	0.480000	1.270000	278.000000
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	1.250000	3.220000	0.782500	1.937500	500.500000
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	1.555000	4.690000	0.965000	2.780000	673.500000
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.950000	6.200000	1.120000	3.170000	985.000000
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.000000	1.710000	4.000000	1680.000000

- Often, it is not fair to compare the variance of a feature to another. The reason is that as the values in the distribution get bigger, the variance grows exponentially. In other words, the variances will not be on the same scale.

# Feature selection using variance



- Scikit-learn provides VarianceThreshold estimator that accepts a threshold cut-off and removes all features with variance below that threshold

```
X = df.drop(columns=['Wine']) # features dataframe
y = df['Wine'] # target dataframe
X.describe()
```

	Alcohol	Malic_acid	Ash	Acid	Mg	Phenols	Flavanoids	Nonflavanoid_phenols	Proanth	Color_intensity	Hue	OD	Proline
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.590899	5.058090	0.957449	2.611685	746.893258
std	0.811827	1.117146	0.2743	1.117146	1.117146	1.117146	1.117146	1.117146	1.117146	1.117146	1.117146	1.117146	1.117146
min	11.030000	0.740000	1.360000	11.030000	11.030000	11.030000	11.030000	11.030000	11.030000	11.030000	11.030000	11.030000	11.030000
25%	12.362500	1.602500	2.210000	12.362500	12.362500	12.362500	12.362500	12.362500	12.362500	12.362500	12.362500	12.362500	12.362500
50%	13.050000	1.865000	2.360000	13.050000	13.050000	13.050000	13.050000	13.050000	13.050000	13.050000	13.050000	13.050000	13.050000
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.950000	6.200000	1.120000	3.170000	985.000000
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.000000	1.710000	4.000000	1680.000000

The above features all have different medians, quartiles, and ranges – completely different distributions. We cannot compare these features to each other.

- Often, it is not fair to compare the variance of a feature to another. The reason is that as the values in the distribution get bigger, the variance grows exponentially. In other words, the variances will not be on the same scale.



# Feature selection using variance



- One method we can use to scale all features is the Robust Scaler (see previous lab) which is not highly affected by outliers:

```
from sklearn.preprocessing import RobustScaler
transformer = RobustScaler().fit(X)
scaled_data = transformer.transform(X)
X_scaled = pd.DataFrame(scaled_data, columns=X.columns)
```

- We use the VarianceThreshold with a threshold 0.35 on the X\_scaled:

```
from sklearn.feature_selection import VarianceThreshold
selector = VarianceThreshold(threshold=0.35)
# Learn variances from X_scaled
_ = selector.fit(X_scaled)
# Get a mask (or integer index if indices=True is set) of the features selected
mask = selector.get_support()
print(mask)
```

```
[ True  True  True  True  True False  True  True  True  True  True False  True]
```

False if the corresponding feature is selected to be dropped: Phenols and OD have variance  $\leq 0.35$

*X\_scaled.std()*  
variances are on the same scale after transformation

Alcohol	0.381132
Malic_acid	0.569766
Ash	0.623277
Ac1	0.603174
Mg	0.565067
Phenols	0.350252
Flavanoids	0.357746
Nonflavanoid_phenols	0.552056
Proanth	0.668561
Color_intensity	0.605204
Hue	0.458666
OD	0.331842
Proline	0.422453
dtype:	float64

# Feature selection using feature importance

---



- A set of predictive techniques (**ensemble** methods) can be used to **assign scores to input features** as part of the training phase. Each score indicates the relative **importance** of each feature **when making a prediction**
    - Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model (see more [here](#))
  - **Feature importance** scores can be calculated both for problems that involve predicting a numerical value, called **regression**, and those problems that involve predicting a class label, called **classification** (studied thoroughly in Labs 8-9)
-

# Feature selection using feature importance

---



- The scores are useful and can be used in a range of situations in a predictive modeling problem, such as:
    - Better understanding the data (which feature(s) are important, i.e. influencing the decision-making process)
    - Reducing the number of input features (choosing the most important features of the dataset for training)
-

# Feature selection using feature importance



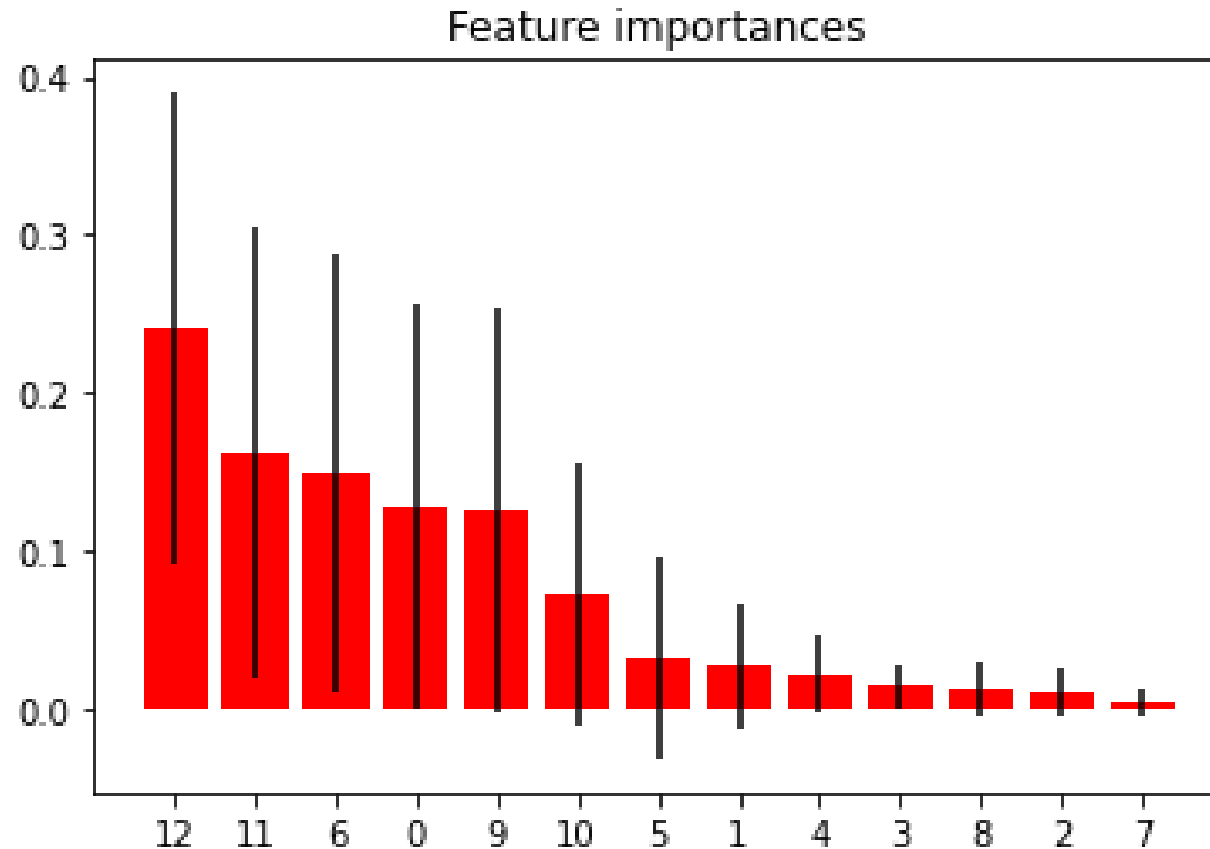
- Get **feature importance** by training an ensemble predictive technique (ensemble classifiers/regressors)
  - Fit (train) predictive technique on the whole set of features
  - Weights are assigned to each feature

```
# Feature Importance using ExtraTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier

# Build an estimator (forest of trees) and compute the feature importances
estimator = ExtraTreesClassifier(n_estimators=100, max_features= 13, random_state=0)
estimator.fit(X, y)

# Lets get the feature importances.
# Features with high importance score higher.
importances = estimator.feature_importances_
```

# Feature selection using feature importance



Feature ranking:

1. feature 12 - Proline (0.240954)
2. feature 11 - OD (0.162438)
3. feature 6 - Flavanoids (0.149339)
4. feature 0 - Alcohol (0.127384)
5. feature 9 - Color\_intensity (0.125520)
6. feature 10 - Hue (0.072550)
7. feature 5 - Phenols (0.031640)
8. feature 1 - Malic\_acid (0.027186)
9. feature 4 - Mg (0.021927)
10. feature 3 - Acl (0.013679)
11. feature 8 - Proanth (0.012320)
12. feature 2 - Ash (0.010575)
13. feature 7 - Nonflavanoid\_phenols (0.004489)

Note: It is recommended to evaluate various classifiers or regressors belonging to the [sklearn.ensemble](#) module. You may have to play with their input parameters for better understanding of the behavior of each model.

# Feature selection using feature importance



- Instead of training an ensemble method only once, we can run the training process multiple times.
- Recursive Feature Elimination (RFE) aims at selecting features by recursively eliminating the worst feature(s) – having lowest importance – at every iteration.

Current set of features = all features

Repeat

1. **Predictive ensemble technique trained** on **current set** of features, weights are assigned to each
2. Feature whose absolute weight is the smallest is pruned from current set features

Until desired number of features is reached

# Feature selection using feature importance



```
from sklearn.feature_selection import RFE
estimator = ExtraTreesClassifier(n_estimators=100,
random_state=0)
# keep the 5 most informative features
# step corresponds to the (integer) number
# of features to remove at each iteration
selector = RFE(estimator, n_features_to_select=5, step=1)
selector = selector.fit(X, y)
print(list(selector.support_))
print(list(selector.ranking_))
```



```
[True, False, False, False, False, False, True, False,
False, True, False, True, True]
[1, 3, 8, 5, 6, 4, 1, 9, 7, 1, 2, 1, 1]
```

```
0          6          9      11 12
```

Important features

# Feature selection using predictive performance of ML model

---



- Forward selection/Backward elimination are two repetitive methods of stepwise selecting important features:
    - Use a predictive technique (any ML model) and a criterion ([scoring](#)) function to measure performance (effectiveness in making predictions):
      - **Classification** problems: accuracy (% of correct predictions), f1, precision, recall
      - **Regression** problems: R2, Mean Squared Error (MSE), Root Mean Squared Error (RMSE)
    - Split dataset (train/test), train model on train data, make predictions on test data
    - Select features that maximize / minimize the criterion function
    - Termination point: reach desired number of features
-

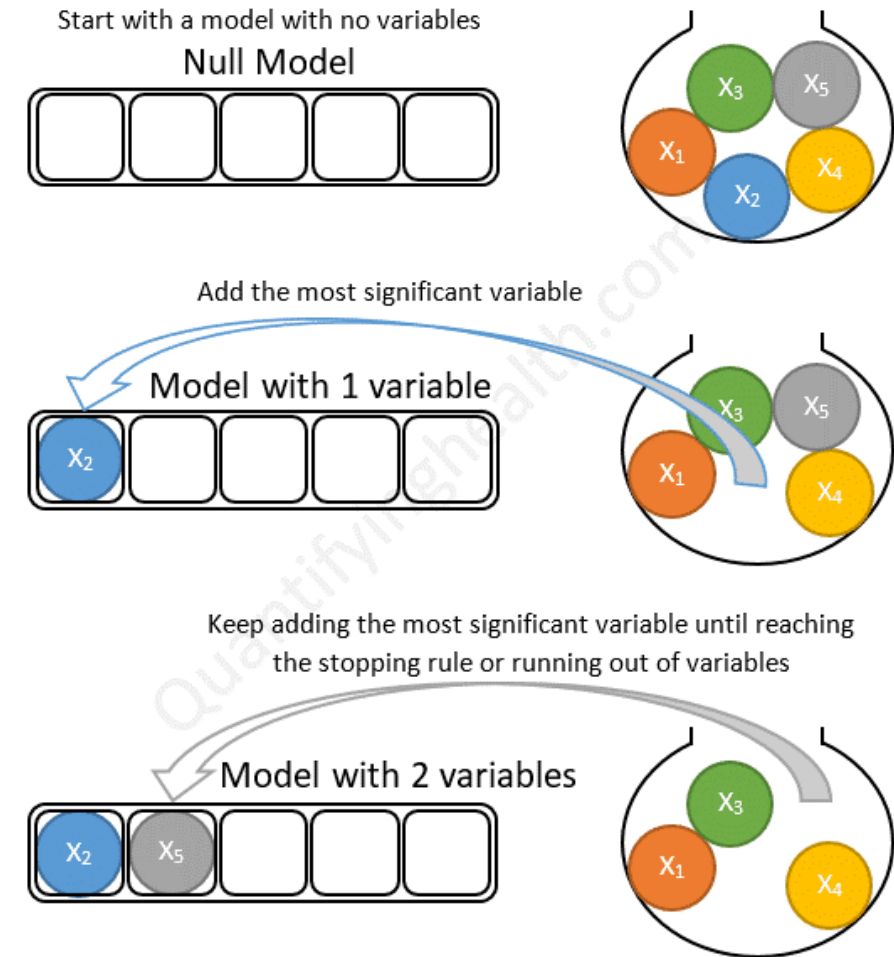


# Feature selection using predictive performance of ML model



- Forward selection:
  - Start with a null model (with no features)
  - Add a feature that maximizes criterion function upon insertion
  - Repeat procedure until termination criterion is satisfied

Forward stepwise selection example with 5 variables:

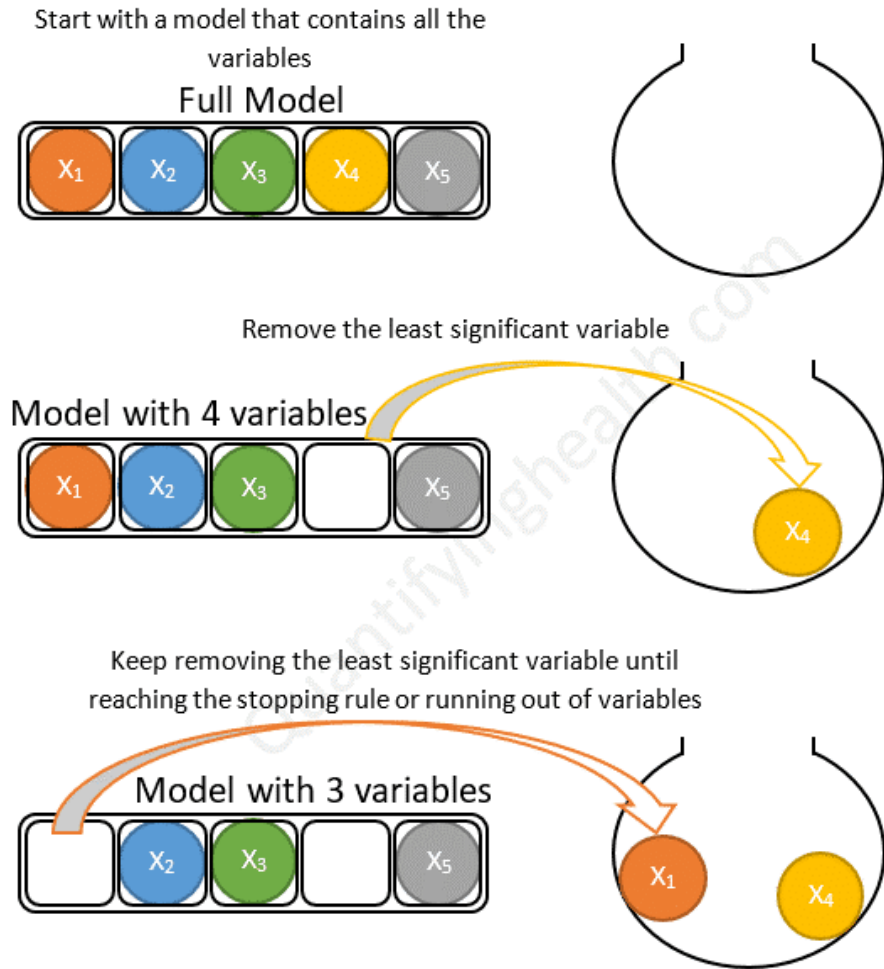


# Feature selection using predictive performance of ML model



- Backward elimination:
  - Start with all features in the model (full model)
  - Remove a feature that has the minimum impact (maximizes criterion function) upon removal
  - Repeat procedure until termination criterion is satisfied

Backward stepwise selection example with 5 variables:



# Examples

---



- **Example 1 – Forward Selection**

- Use the wine dataset to choose the “best” 5 (out of 13) features
- Classification method: k-nearest neighbors
  - Distance-based algorithm: achieves better results when input features are scaled
- Criterion (scoring) function: accuracy
- Initialize classifier

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier(n_neighbors=4)
```

---

# Examples

Install [mlxtend](#) library. Run  
conda install -c  
conda-forge mlxtend  
on Anaconda prompt prior  
running this example

- Initialize and fit [Sequential Forward Selection model](#)
  - Can be used for both classification and regression problems

```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
sfs = SFS(knn, # scikit-learn classifier
          k_features=5, # termination point
          forward=True, # forward selection
          floating=False,
          verbose=2, # logging level (messages printed when running)
          scoring='accuracy', # criterion function
          cv=10) # 10-fold cross validation: resampling method
# that uses different portions of the data to test and train a model on different iterations. Here, we have 10
# iterations per feature selection round (more details in the next labs).
# perform feature selection & learn model from training data
sfs = sfs.fit(X_scaled, y)
# Results
# Features: 1/5 -- score: 0.7810457516339869
# Features: 2/5 -- score: 0.9212418300653595
# Features: 3/5 -- score: 0.9493464052287581
# Features: 4/5 -- score: 0.9552287581699346
# Features: 5/5 -- score: 0.9663398692810456
```

mean scores (over 10 iterations)

# Examples



- We can access the indices of the 5 best features directly via the `k_feature_idx_` attribute and the prediction score via `k_score_`

```
print('\nSequential Forward Selection (k=5):')
print('Selected features:',sfs.k_feature_idx_) # (1, 4, 6, 9, 12)
print('Prediction score:',sfs.k_score_)      # 0.9663398692810456
```

- **Example 2 – Backward Elimination**

```
sbs = SFS(knn, # scikit-learn classifier
          k_features=5, # termination criterion
          forward=False, # backward elimination
          floating=False,
          scoring='accuracy', # criterion function
          cv=10) # 10-fold cross validation

sbs = sbs.fit(X_scaled, y)
print('\nSequential Backward Selection (k=5):')
print('Selected features:',sbs.k_feature_idx_)# (0, 2, 8, 9, 12)
print('Prediction (CV) score:',sbs.k_score_) # 0.9607843137254901
```

# Examples

---



- **Example 3 – Plotting the results**

```
from mlxtend.plotting import
plot_sequential_feature_selection as plot_sfs
import matplotlib.pyplot as plt
sfs = SFS(knn,
          k_features=5,
          forward=True,
          floating=False,
          scoring='accuracy',
          verbose=2,
          cv=10)

sfs = sfs.fit(X_scaled, y)
fig1 = plot_sfs(sfs.get_metric_dict(), kind='std_dev')
plt.ylim([0.8, 1])
plt.title('Sequential Forward Selection (w. StdDev)')
plt.grid()
plt.show()
```

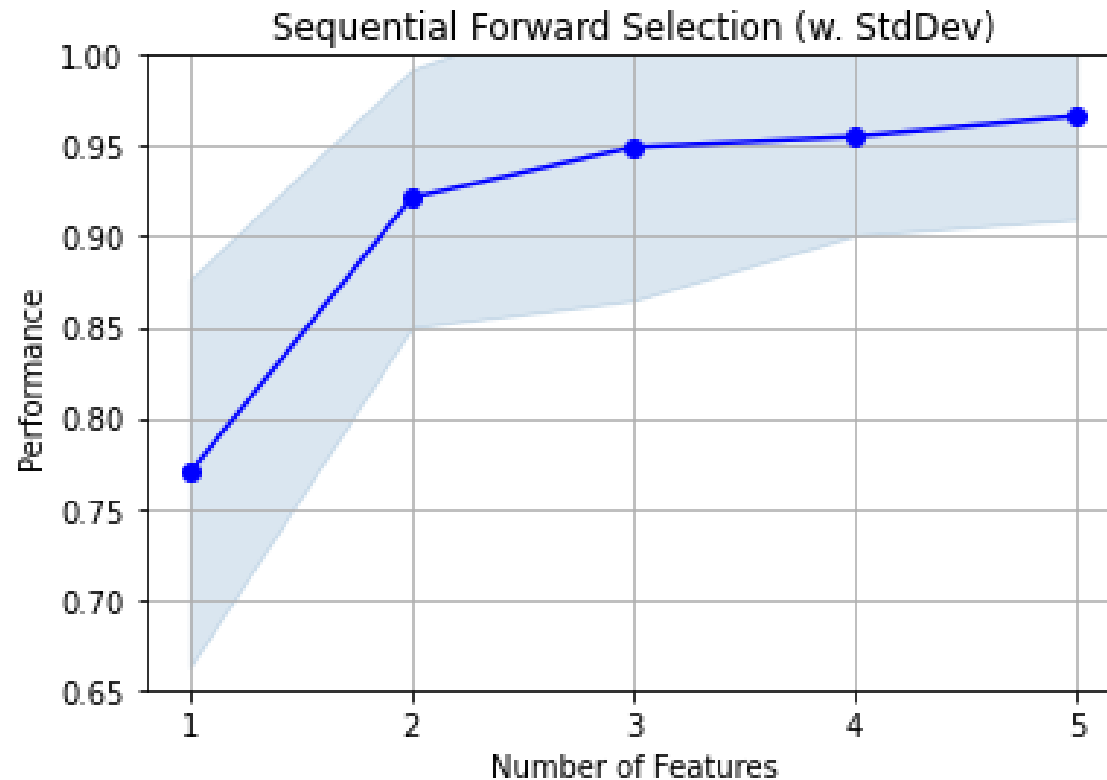
---

# Examples



- **Example 3 – Plotting the results**

```
Features: 1/5 -- score: 0.7810457516339869
Features: 2/5 -- score: 0.9212418300653595
Features: 3/5 -- score: 0.9493464052287581
Features: 4/5 -- score: 0.9552287581699346
Features: 5/5 -- score: 0.9663398692810456
```





- **Example 4 – Selecting the "best" feature combination in k-range**
    - Set `k_features` to a tuple (`min_k`, `max_k`)
    - SFS selects the best feature combination of size `min_k` to `max_k` inclusive that scored best during cross validation
    - In forward selection
      - It also returns the best score achieved for every feature subset from 1 feature to `max_k` features, i.e. for `k_features=(5,9)` it returns the best score achieved for 1 feature, 2 features, ... up to 9 features
    - In backward selection
      - It also returns the best score achieved for every feature subset from all features down to `min_k` features, i.e. `k_features=(5,9)` the best score achieved for 13 features (for the wine dataset), 12 features, ..., down to 5 features
-



# Examples



- **Example 4 – Selecting the "best" feature combination in k-range**

```
X, y = wine_data()
```

```
knn = KNeighborsClassifier(n_neighbors=4)
sfs_range = SFS(estimator=knn,
                k_features=(2, 13),
                forward=True,
                floating=False,
                scoring='accuracy',
                cv=10)
```

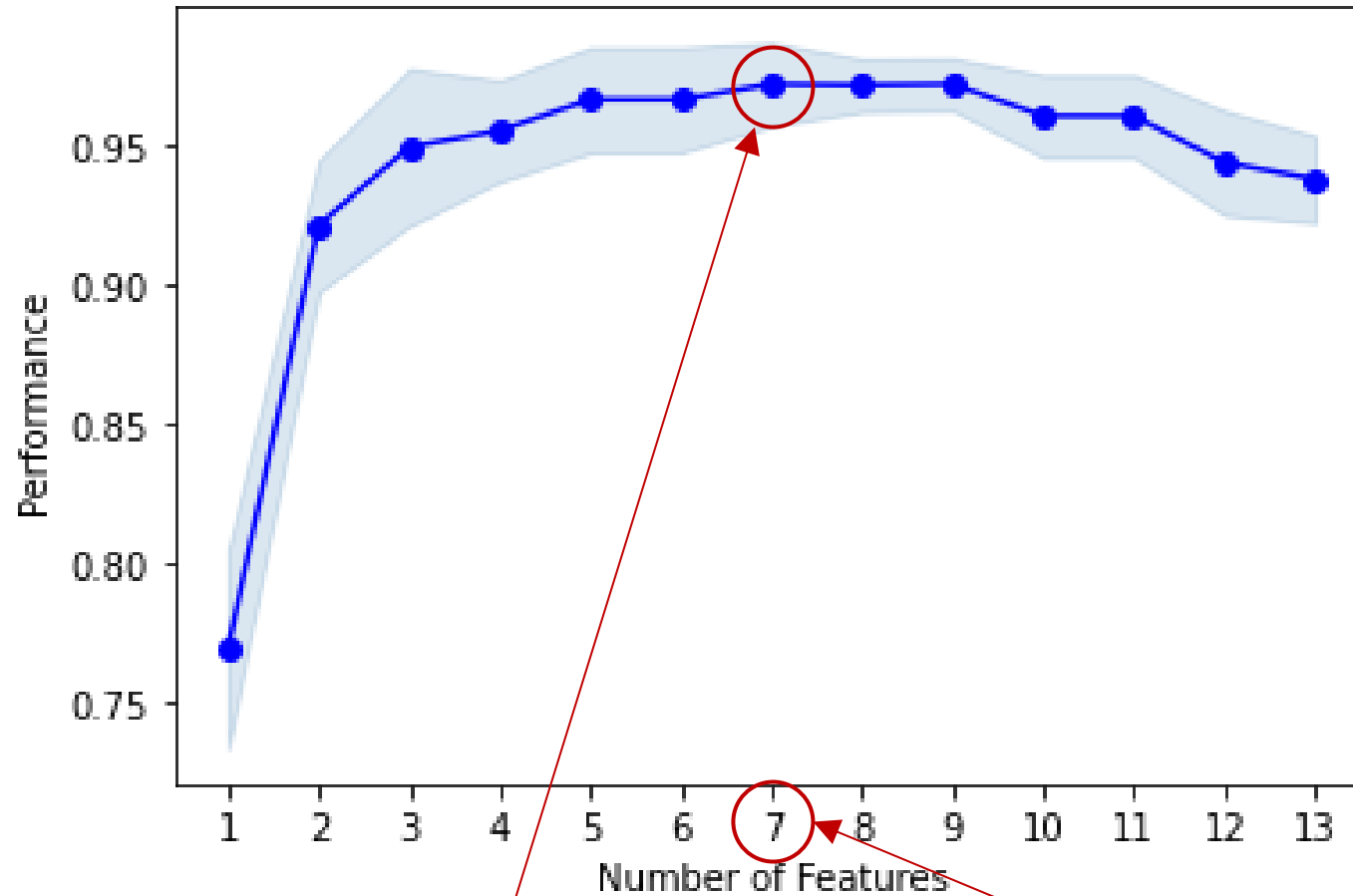
```
sfs_range = sfs_range.fit(X_scaled, y)
```

```
print('best combination (ACC: %.3f): %s\n' % (sfs_range.k_score_,
sfs_range.k_feature_idx_))
print('all subsets:\n', sfs_range.subsets_)
plot_sfs(sfs_range.get_metric_dict(), kind='std_err');
```

# Examples



- **Example 4 – Selecting the "best" feature combination in k-range**



best combination (ACC: 0.972): (1, 4, 6, 9, 10, 11, 12)

```
X_sfs = sfs_range.transform(X) # extract the columns from X dataset
```

# SFS with regression problems



- Use appropriate estimator (regressor) and scoring function (e.g. R2, RMSE etc.)

```
rf = RandomForestRegressor()
```

```
sfs_range = SFS(estimator=rf,  
                k_features=(2, 13),  
                forward=True,  
                floating=False,  
                scoring='r2', # or 'neg_root_mean_squared_error'  
                cv=10)
```

```
sfs_range = sfs_range.fit(X, y) # no need for scaled features in tree-based models
```

```
print('best combination (R2: %.3f): %s\n' % (sfs_range.k_score_,  
sfs_range.k_feature_idx_))  
print('all subsets:\n', sfs_range.subsets_)  
plot_sfs(sfs_range.get_metric_dict(), kind='std_err');
```

# Feature extraction

---



- Build a new set of features from the original feature set
  - Differs from feature selection in two ways:
    - Instead of choosing subset of features
    - Create new feature set (dimensions)
-

# Feature extraction

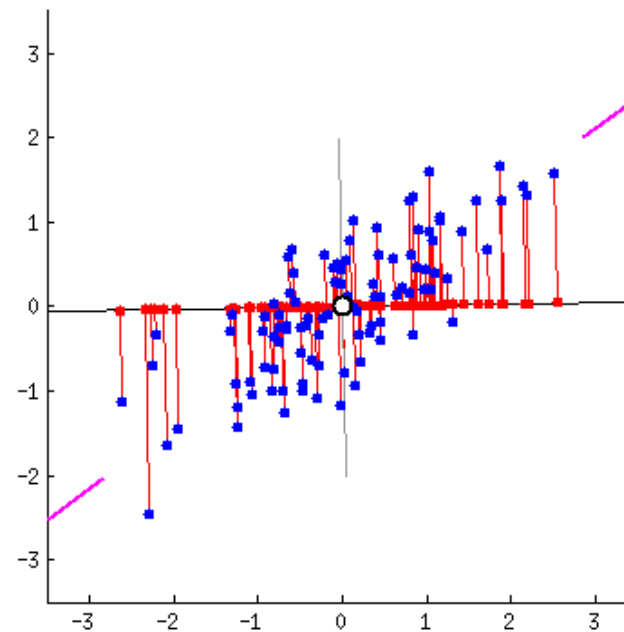


- Idea:
  - Given data points in  $d$ -dimensional space,
  - Project into lower  $k$ -dimensional space ( $k < d$ ) while preserving as much information as possible
  - In particular, choose projection that minimizes the squared error in reconstructing original data
- Methods:
  - Principal Component Analysis (PCA)
    - <http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html>
  - Singular Vector Decomposition (SVD)
    - <https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.svds.html>
    - <http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html>
  - Linear Discriminant Analysis (LDA)
    - [http://scikit-learn.org/stable/modules/generated/sklearn.discriminant\\_analysis.LinearDiscriminantAnalysis.html](http://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html)

# PCA



- **PCA** tries to identify a set of new directions (new features) called **principal components** that account for the **most variance (information)**
- Principal components (new directions/features) are the linear combinations of the old directions (old features)



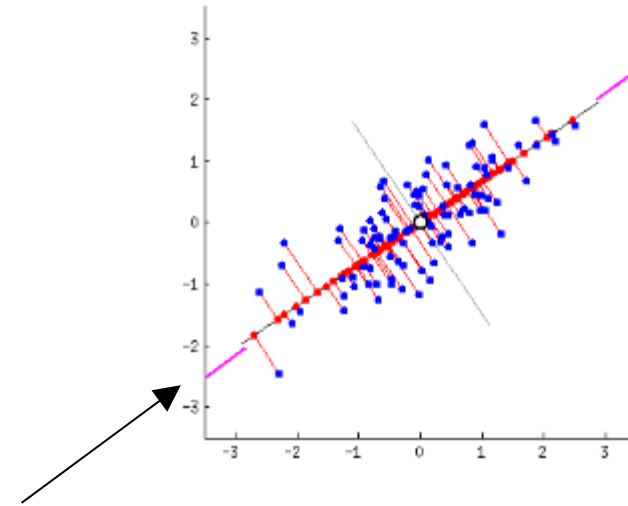
The eigenvectors and eigenvalues of a covariance (or correlation) matrix represent the “core” of a PCA: The eigenvectors (principal components) determine the directions of the new feature space, and the eigenvalues determine their magnitude. In other words, the eigenvalues explain the variance of the data along the new feature axes.

Excellent explanation about PCA: <http://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues/140579#140579>

# PCA Example



- Dataset: 2-D observations
  - blue dots
- Find the best one dimension that converts dataset to 1-D observations
- Best dimension:
  - Line that points to the magenta ticks
    - Red dots are projections of the blue dots
    - Projection position is the new value of the (1-D) observation on the new dimension
  - Maximizes variance (spread of red dots)
    - Increased differentiation among new 1-D observations
  - Minimizes reconstruction error (red line)
    - Error = |position of blue dot – projection position of blue dot|



# PCA considerations

---

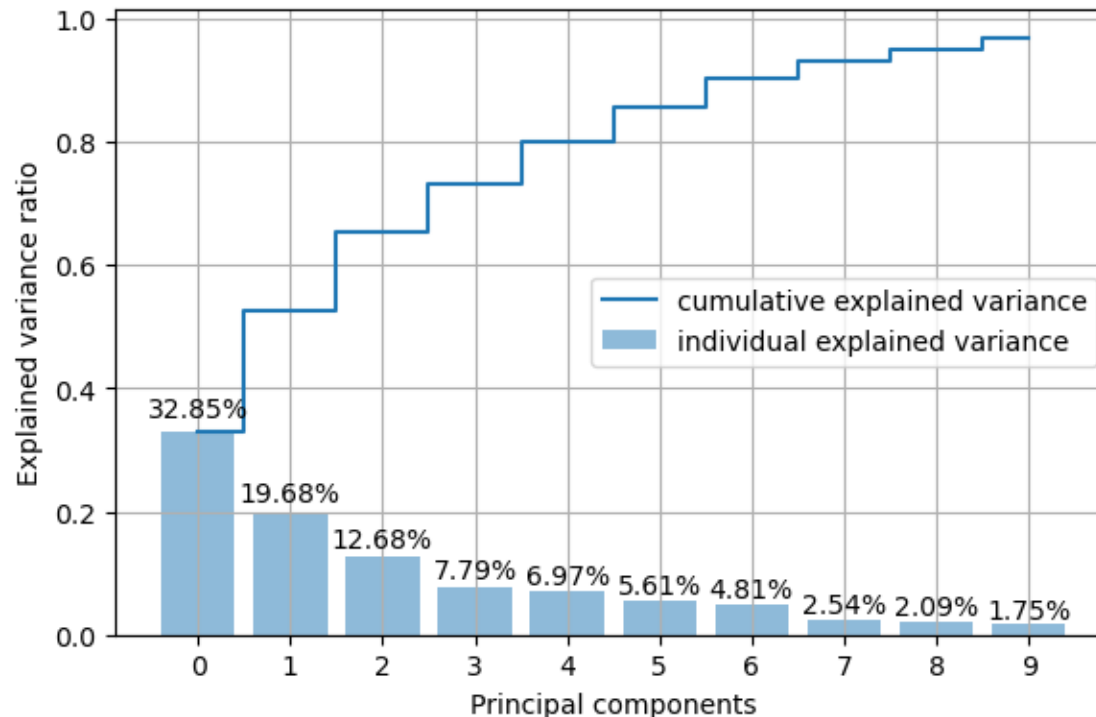


- PCA assumes that the data to be transformed is linearly separable and **works well** only when the **relationship between features is linear**
  - Can be previewed using scatterplots (or pairplot) that visualizes the relationships between any two features of the dataset
- PCA **requires** that the data (**features**) is mean-centered (mean=0)
  - for a single feature, this is done by subtracting the mean of that feature from each data point → done by PCA in advance for all features
- PCA is **sensitive** to the **scale** of the features
  - variance of features with larger scales will dominate the principal components unless the data is standardized → use standard scaler prior PCA (Lab 4)
- PCA **helps when** the **features** in your dataset are **correlated** and you want to remove the redundancy in the data by **transforming it into uncorrelated** principal components





- Is there a rule of thumb for finding the “best” number of PCA components (features)?
- A useful measure is to pick the  $k$  features that explain a high percentage of the total data variance
  - can be done by plotting the explained variance ratio  $r_k$  as a function of  $k$



Example:

Perform PCA on the SCALED wine dataset with 13 features to extract 10 new features (10 principal components)

Some observations:

- First 3 new features explain together 62% of the total variance
- Each of the last 3 new features explain around 2% of the total variance (can be omitted)

# SVD



- SVD is a generic way of breaking down a big matrix (dataset with features) into 3 smaller, more useful pieces
- SVD breaks a matrix ( $A$ ) into three matrices:  $A=U\Sigma V^T$ 
  - Matrices  $U$  and  $V$  contain information about the "directions" or "features" of the original dataset –  $U$  contains info about rows,  $V$  info about columns
  - $\Sigma$  (Sigma) is diagonal matrix with singular values. These are like "weights" that tell us how important certain directions (in  $U$  and  $V$ ) are.
- So if your original matrix (dataset) is too big and complicated, you can use just the most important singular values (the biggest ones in  $\Sigma$ ) and their corresponding vectors from  $U$  and  $V$ 
  - This helps you simplify the data while keeping most of its essential information.

# SVD considerations

---



- SVD is a generic way of decomposing a matrix for purposes like dimensionality reduction, latent semantic analysis (LSA\*) in text processing without necessarily focusing on variance
- It **works** well with **sparse matrices**, where many of the entries are zero (e.g., document-term frequency matrices in text processing)
- SVD **does not require data to be mean-centered**

(\*) LSA uses SVD to uncover hidden structures in text data by reducing the dimensionality of the document-term matrix and finding relationships between terms and documents that may not be immediately apparent from the raw data

---

# Supervised vs Unsupervised

---



- **SVD** and **PCA** are unsupervised methods
    - Both ignore the target variable (e.g. class labels)
  - **LDA** is a supervised method
    - Takes into account **class** labels (target variable), suitable for classification problems
    - identifies new (directions) features that best **separate two or more classes**
    - Note: the maximum number of new features = number of classes – 1
      - Example: if the dataset contains observations belonging to 3 classes (i.e. 3 unique values in the target variable) the maximum number of new features can be 2.
-

# Python-implemented algorithms

---



- Scikit-learn PCA (centers data, does not support sparse matrices)
  - SciPy SVD (works for sparse matrices with many zeros)
  - Scikit-learn TruncatedSVD: (works for large sparse matrices efficiently without making memory explode)
-

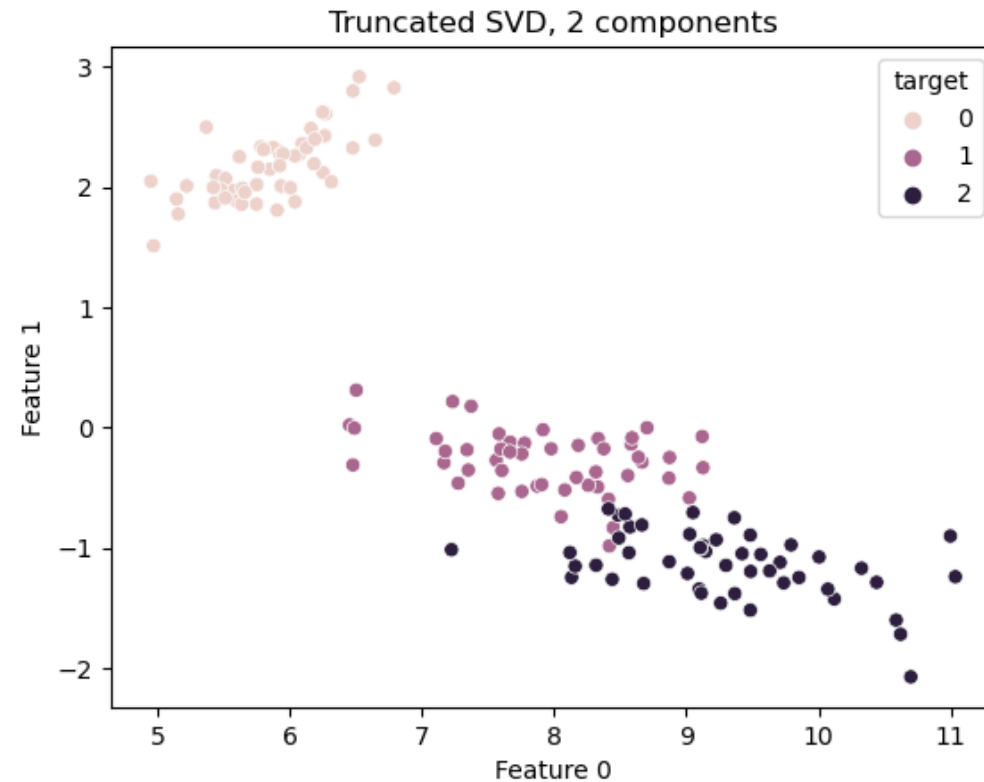
# Feature Extraction in Python

---



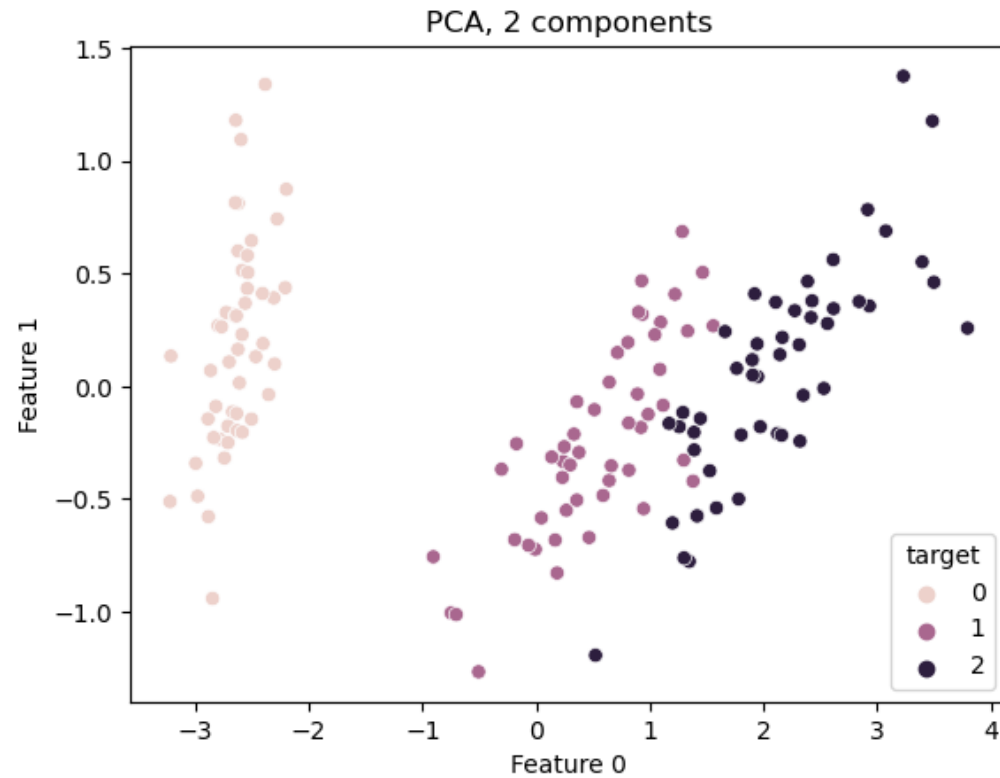
- Dataset: Iris dataset
    - 150 flower observations
    - 4 features
      - sepal length, sepal width, petal length, petal width
    - class variable
      - 0 (setosa), 1 (versicolor), 2 (virginica)
  - Perform dimensionality reduction using TruncatedSVD, PCA and LDA
    - 4 to 2 features
-

# Results – TruncatedSVD



TruncatedSVD explained  
variance ratio (first two  
components):  
[0.52875361 0.44845576]

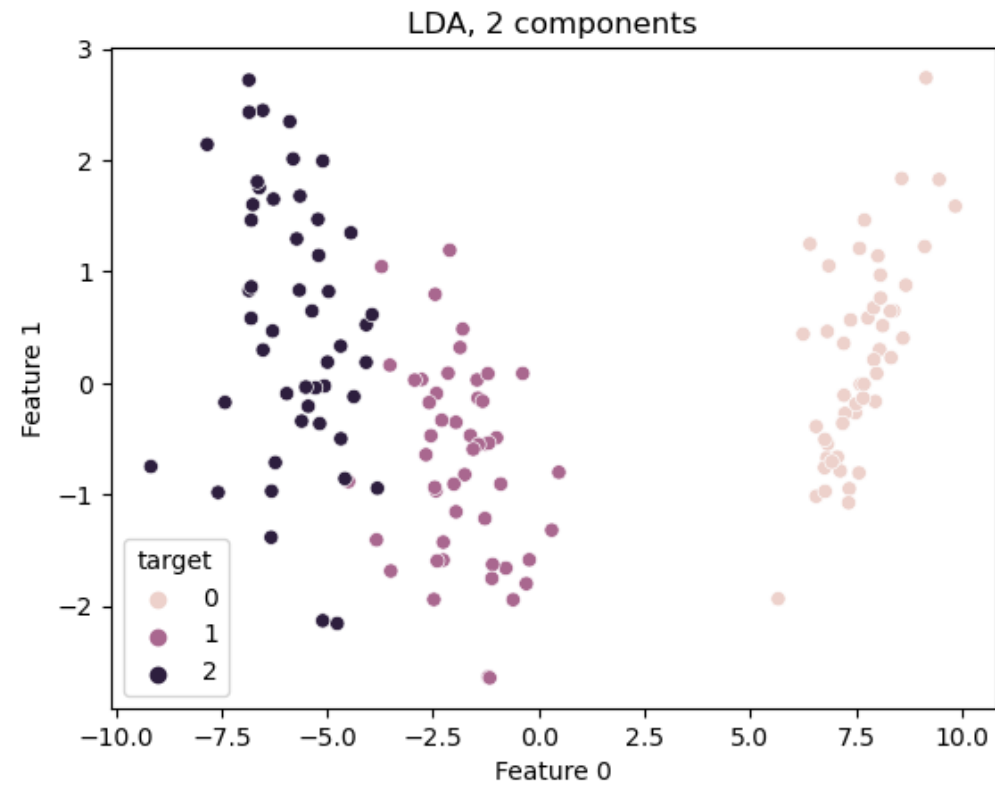
# Results – PCA



PCA explained variance  
ratio (first two  
components):  
[0.92461872 0.05306648]



# Results – LDA



LDA explained variance  
ratio (first two  
components) :  
[0.9912126 0.0087874]

# Importance evaluation in estimators

---



- There are several ways to get feature "importances". As often, there is no strict consensus about what this word means.
- In scikit-learn, the importance is implemented as described in [1] (often cited, but unfortunately rarely read...). It is sometimes called "gini importance" or "mean decrease impurity" and is defined as the total decrease in node impurity (weighted by the probability of reaching that node (which is approximated by the proportion of samples reaching that node)) averaged over all trees of the ensemble.
- In the literature or in some other packages, you can also find feature importances implemented as the "mean decrease accuracy". Basically, the idea is to measure the decrease in accuracy on OOB data when you randomly permute the values for that feature. If the decrease is low, then the feature is not important, and vice-versa.
- [1]: Breiman, Friedman, "Classification and regression trees", 1984.

