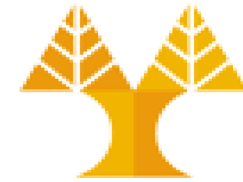
EPL448: Data Mining on the Web – Lab 7



University of Cyprus
Department of
Computer Science

Παύλος Αντωνίου

Γραφείο: B109, ΘΕΕ01

Feature selection & Feature extraction

- Used to eliminate the number of features (columns)
- Useful for further processing:
  - Less computation time when running predictive modelling algorithms
  - Easier to understand, visualize a dataset
- 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 being 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

- Carried out using:
  - Feature variance: need for features with high variance
  - Feature importance: need for features with high importance
  - Feature correlation: need for features with low correlation among them

Feature selection – 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 – variance

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

```
from sklearn.datasets import load_wine
data = load_wine()
X = data.data
y = data.target
```

Wine dataset: 178 wine observations by 13 features. X is 2D array of features. y is the array of classes (target value). Wines classified into 3 types.

  - 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.

```
df = pd.DataFrame(X)
df.describe()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
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

Feature selection – variance

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

```
from sklearn.datasets import load_wine
data = load_wine()
X = data.data
y = data.target
```

Wine dataset: 178 wine observations by 13 features. X is 2D array of features. y is the array of classes (target value). Wines classified into 3 types.

  – 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.

```
df = pd.DataFrame(X)
df.describe()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
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								0.228572	0.709990	314.907474
min	11.030000	0.740000	1.36000								0.480000	1.270000	278.000000
25%	12.362500	1.602500	2.21000								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

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

Feature selection – 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)
X_scaled = transformer.transform(X)
```

removes the median and scales the data according to the inter quantile range (IQR)

```
X_scaled.std()
```

variances are on the same scale after transformation

0	0.381132
1	0.569766
2	0.623277
3	0.603174
4	0.565067
5	0.350252
6	0.357746
7	0.552056
8	0.668561
9	0.605204
10	0.458666
11	0.331842
12	0.422453

- 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: 5 and 11 have variance <= 0.35

Feature Importance

- A class 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 for problems that involve predicting a numerical value, called regression, and those problems that involve predicting a class label, called classification (studied in Labs 8-9)

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)
  - Better understanding a model (understand the evaluated model parameters' values)
  - Reducing the number of input features (choosing the most important features of the dataset for training)

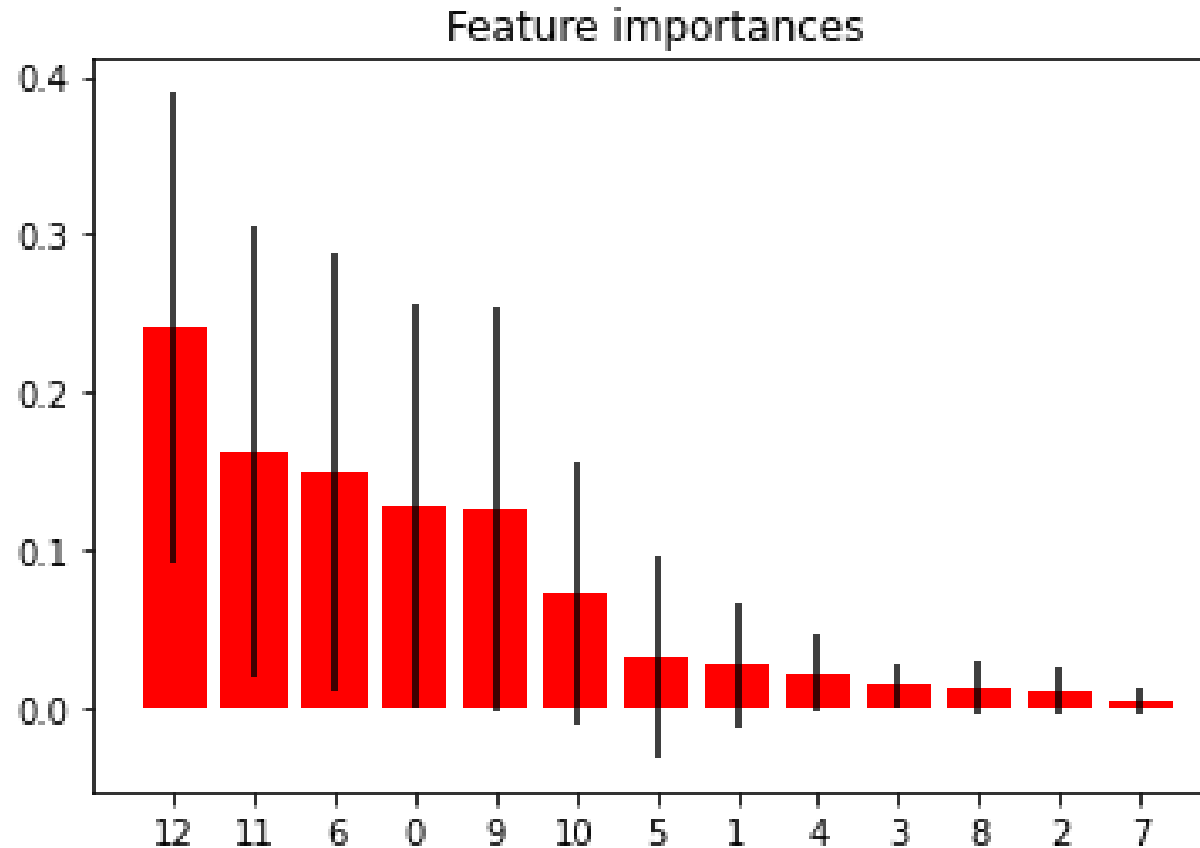
Feature selection – importance

- Remove features with low importance
- Get feature importance by training a predictive technique
- Use ensemble classifiers/regressors
  - Fit (train) predictive technique on whole set of features
  - Weights are assigned to each feature

ExtraTreeClassifier is applied on the wine dataset, taking into account scaled feature values from the previous example.

```
# 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 – importance



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 – importance

- Recursive Feature Elimination (RFE) aims at selecting features by recursively considering smaller and smaller sets of features as input to a predictive technique

```
Current set of features = all features
Repeat
   1. Predictive 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 – 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 – correlation

- Use the pandas corr() method to compute pairwise correlation of columns
  - available correlation methods: pearson, kendall, spearman

```
import seaborn as sns
from matplotlib import cm as cm

plt.figure()
# convert list of wine observations to a pandas dataframe
# (num of columns = num of features)
# find pairwise correlation of columns
# default correlation method = pearson
df = pd.DataFrame(X)
df['y'] = y
corr = df.corr(method='pearson')
cmap = cm.get_cmap('jet', 30)
_, ax = plt.subplots( figsize = ( 12 , 10 ) )
sns.heatmap(corr, cmap = cmap, square=True, cbar_kws={ 'shrink' : .9 }, ax=ax,
annot = True, annot_kws = { 'fontsize' : 12 }, xticklabels=corr.columns,
yticklabels=corr.columns)
```

Feature selection – correlation

	0	1	2	3	4	5	6	7	8	9	10	11	12	y
0	1	0.094	0.21	-0.31	0.27	0.29	0.24	-0.16	0.14	0.55	-0.072	0.072	0.64	-0.33
1	0.094	1	0.16	0.29	-0.055	-0.34	-0.41	0.29	-0.22	0.25	-0.56	-0.37	-0.19	0.44
2	0.21	0.16	1	0.44	0.29	0.13	0.12	0.19	0.0097	0.26	-0.075	0.0039	0.22	-0.05
3	-0.31	0.29	0.44	1	-0.083	-0.32	-0.35	0.36	-0.2	0.019	-0.27	-0.28	-0.44	0.52
4	0.27	-0.055	0.29	-0.083	1	0.21	0.2	-0.26	0.24	0.2	0.055	0.066	0.39	-0.21
5	0.29	-0.34	0.13	-0.32	0.21	1	0.86	-0.45	0.61	-0.055	0.43	0.7	0.5	-0.72
6	0.24	-0.41	0.12	-0.35	0.2	0.86	1	-0.54	0.65	-0.17	0.54	0.79	0.49	-0.85
7	-0.16	0.29	0.19	0.36	-0.26	-0.45	-0.54	1	-0.37	0.14	-0.26	-0.5	-0.31	0.49
8	0.14	-0.22	0.0097	-0.2	0.24	0.61	0.65	-0.37	1	-0.025	0.3	0.52	0.33	-0.5
9	0.55	0.25	0.26	0.019	0.2	-0.055	-0.17	0.14	-0.025	1	-0.52	-0.43	0.32	0.27
10	-0.072	-0.56	-0.075	-0.27	0.055	0.43	0.54	-0.26	0.3	-0.52	1	0.57	0.24	-0.62
11	0.072	-0.37	0.0039	-0.28	0.066	0.7	0.79	-0.5	0.52	-0.43	0.57	1	0.31	-0.79
12	0.64	-0.19	0.22	-0.44	0.39	0.5	0.49	-0.31	0.33	0.32	0.24	0.31	1	-0.63
y	-0.33	0.44	-0.05	0.52	-0.21	-0.72	-0.85	0.49	-0.5	0.27	-0.62	-0.79	-0.63	1

1.00 0.75 0.50 0.25 0.00 −0.25 −0.50 −0.75

Observations:

- Features 5 & 6 are highly (positively) correlated to each other
- Features 5, 6,10, 11, 12 are highly negatively correlated to the target value (y) ; see the last line of the heat map

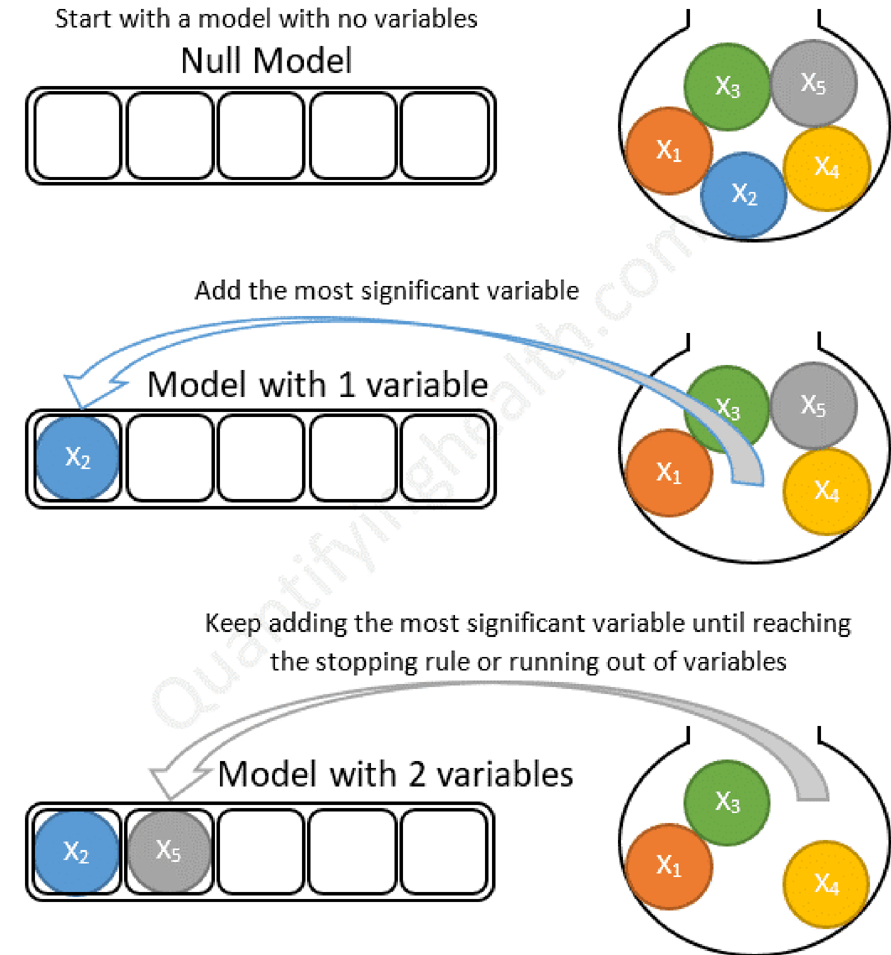
Feature selection – mlxtend library

- Forward selection/Backward elimination are two repetitive methods of stepwise selecting important features:
  - Use a predictive technique (not necessary an ensemble method) and a criterion (scoring) function:
    - Classification problems: accuracy, 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 – mlxtend library

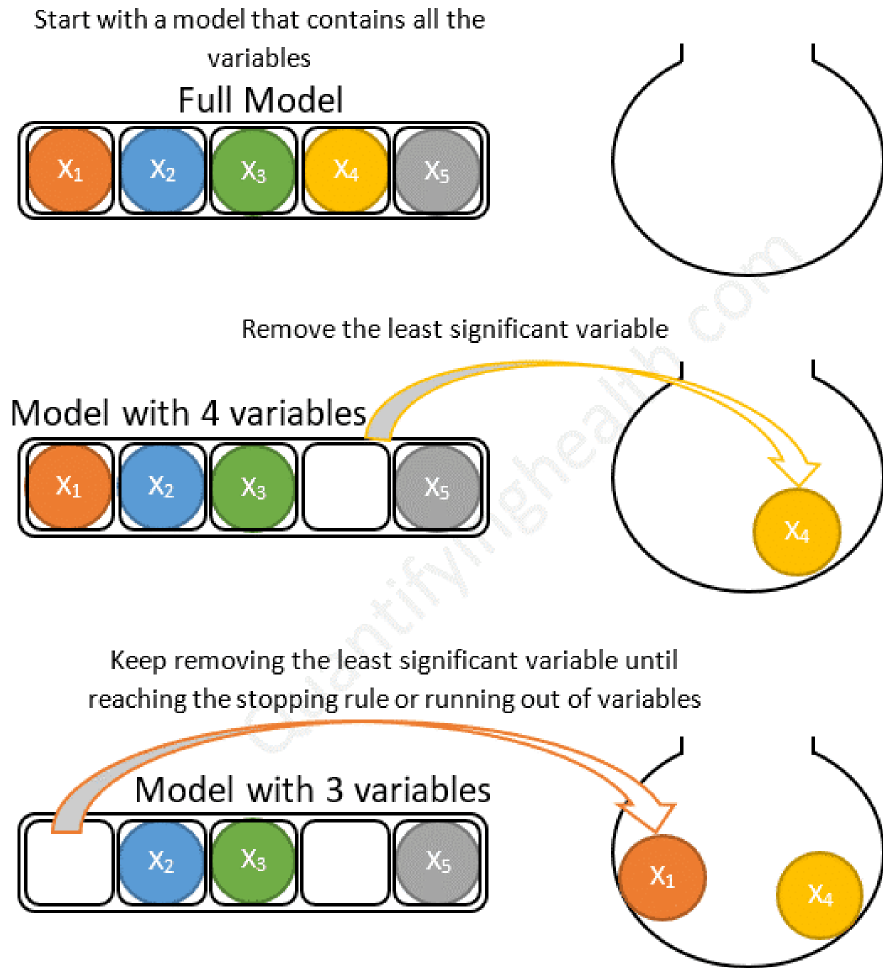
- Forward selection
  - Start with a null model (with no features)
  - Add a feature that maximizes criterion function one after the other
  - Repeat procedure until termination criterion is satisfied

Forward stepwise selection example with 5 variables:



Feature selection – mlxtend library

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



Examples

- **Example 1 – Forward Selection**
  - Use the wine dataset to choose the “best” 5 (out of 13) features
  - Classification method: k-nearest neighbors
  - 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
          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, y)
# Results
# Features: 1/5 -- score: 0.7696078431372548
# 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, 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

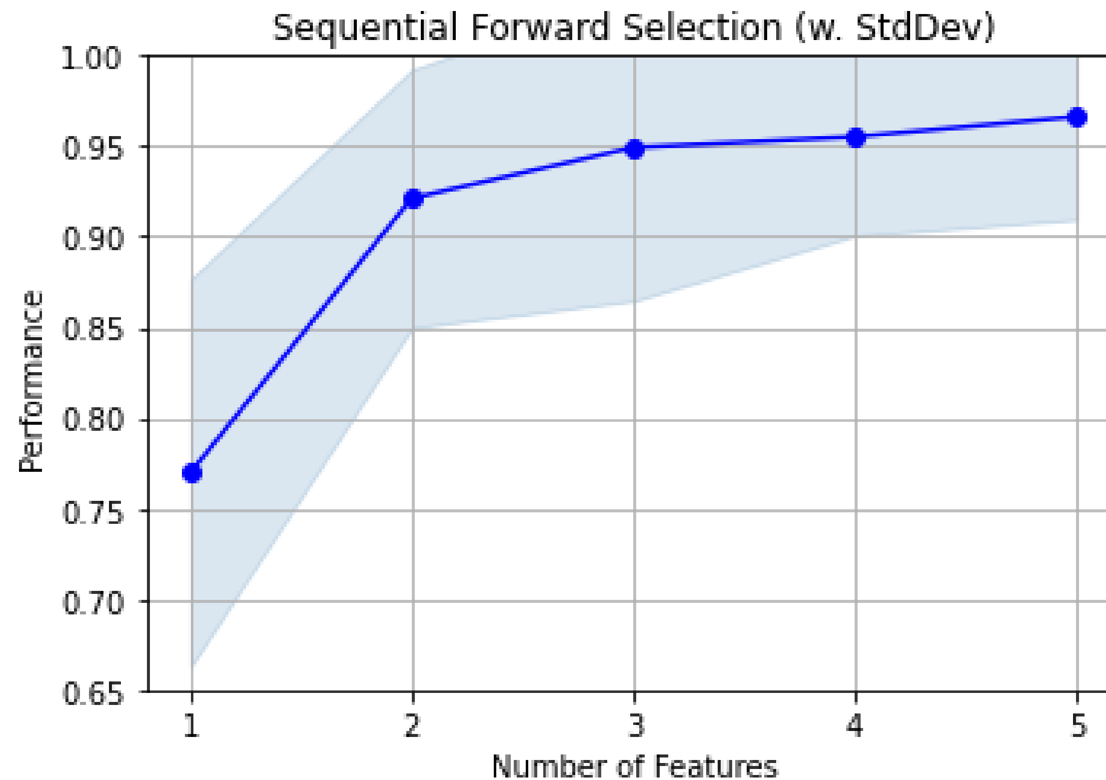
- **Example 3 – Plotting the results**

```
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, 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: 0.7696078431372548
Features: 2/5 -- score: 0.9212418300653595
Features: 3/5 -- score: 0.9493464052287581
Features: 4/5 -- score: 0.9552287581699346
Features: 5/5 -- score: 0.9663398692810456
```



Examples

- **Example 4 – Selecting the "best" feature combination in k-range**
  - Set `k_features` to a tuple (min_k, max_k)
  - Let SFS select the best feature combination that it discovered by iterating:
    - k=[min_k, max_k] (forward)
    - k=[max_k, min_k] (backward)
  - The size of the returned feature subset is then within min_k to max_k, depending on which combination scored best during cross validation.

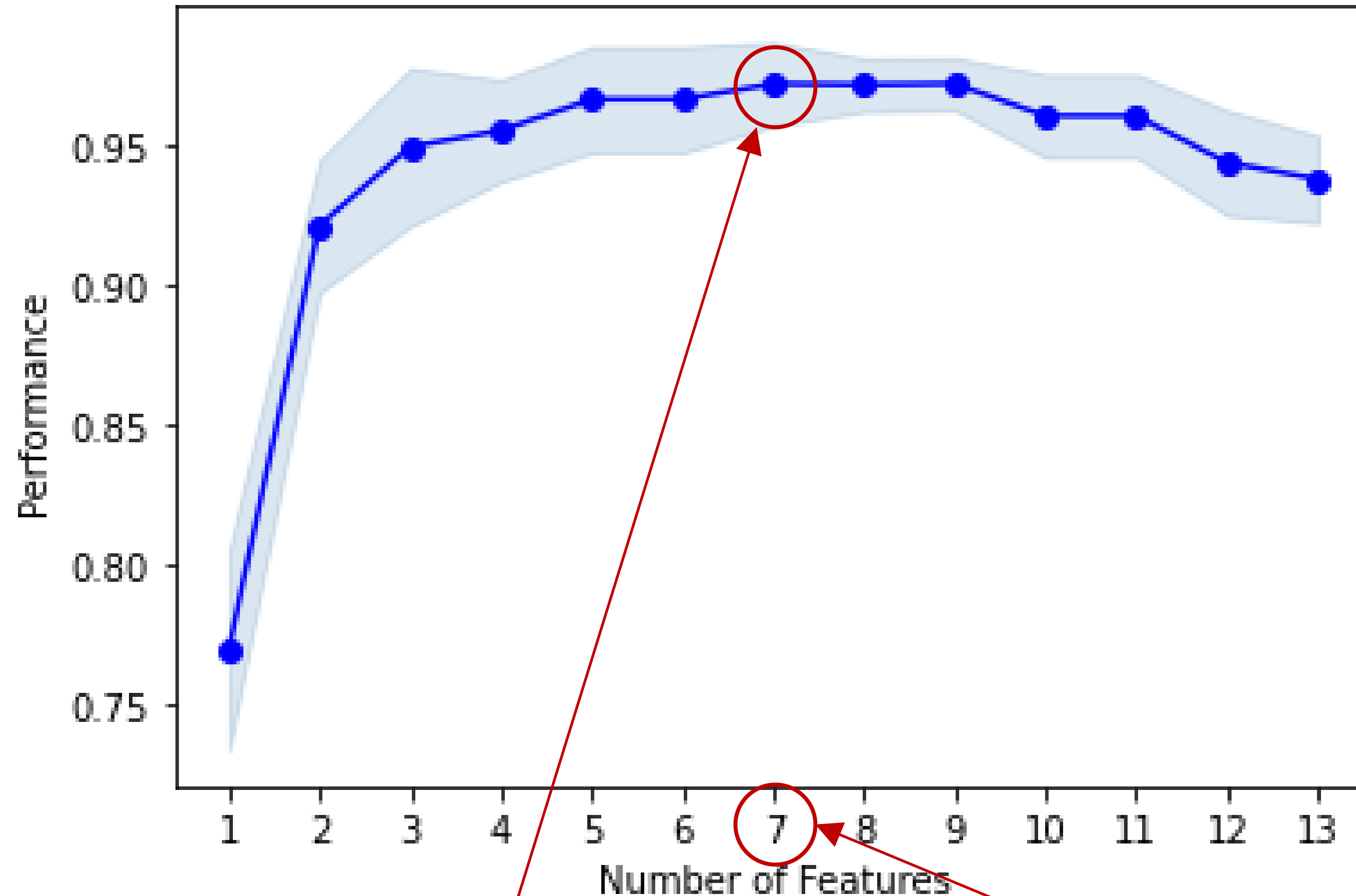
Examples

- **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),    # TRY FROM 2 TO 13 FEATURES
           forward=True,
           floating=False,
           scoring='accuracy',
           cv=10)

sfs_range = sfs_range.fit(X, 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)

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

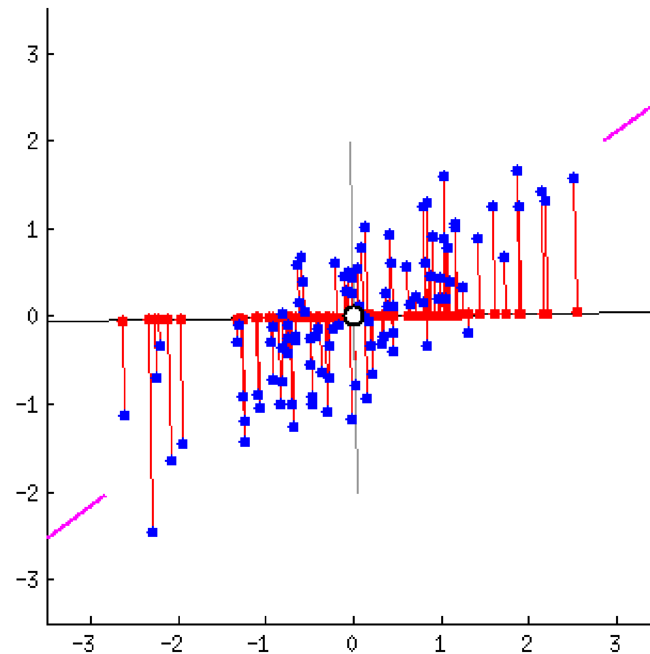
- Differers from feature selection in two ways:
  - Instead of choosing subset of features,
  - Create new features (dimensions) defined as functions over all features

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:
  - Singular Vector Decomposition (SVD)
    - https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.**svd**s.html
    - http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.**TruncatedSVD**.html
  - Principal Component Analysis (PCA)
    - http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.**PCA**.html
  - Linear Discriminant Analysis (LDA)
    - http://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.**LinearDiscriminantAnalysis**.html

PCA & SVD

- SVD and PCA try to identify combination of new features (or dimensions) called singular vectors or components respectively, that account for the most variance in the data

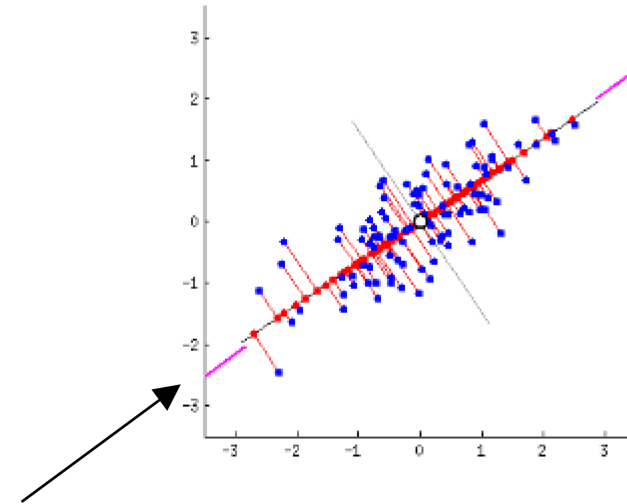


The eigenvectors and eigenvalues of a covariance (or correlation) matrix represent the “core” of a PCA: The eigenvectors (principal components) determine the directions of the new feature space, and the eigenvalues determine their magnitude. In other words, the eigenvalues explain the variance of the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



Supervised vs Unsupervised

- SVD and PCA are unsupervised methods
  - Both ignore class labels (if any)
  - Identify features that account for the most variance in **data**

- LDA is a supervised method
  - Takes into account **class** labels (target values)
  - identifies new features that account for the most variance between **classes**

PCA & SVD

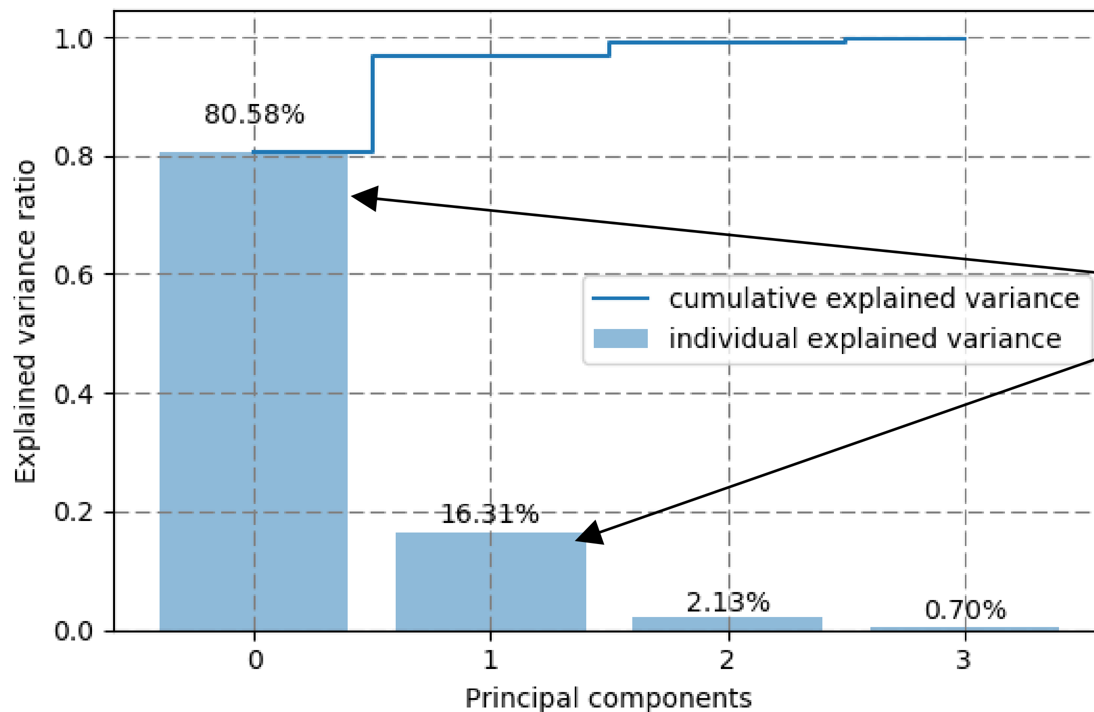
- PCA and SVD not optimal for classification:
  - note that there is no mention of the class label in the definition of PCA
  - keeping the dimensions of largest energy (variance) is a good idea but not always enough
  - certainly improves the density estimation, since space has smaller dimension
  - but could be unwise from a classification point of view
  - the discriminant dimensions for each class could be thrown out

PCA & SVD

- Does this mean you should never use PCA/SVD?
  - no, typically it is a good method to find a suitable subset of features, as long as you are not too greedy
- Is there a rule of thumb for finding the “best” number of SVD/PCA components (features)?

PCA & SVD

- 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
PCA with k = 4
(4 principal components) on a dataset with 13 features.

- First 2 components explain together 96.89% of the total variance
- The rest components explain a quite small percentage of variance; can be omitted
- Number of PCA components to consider: k = 2

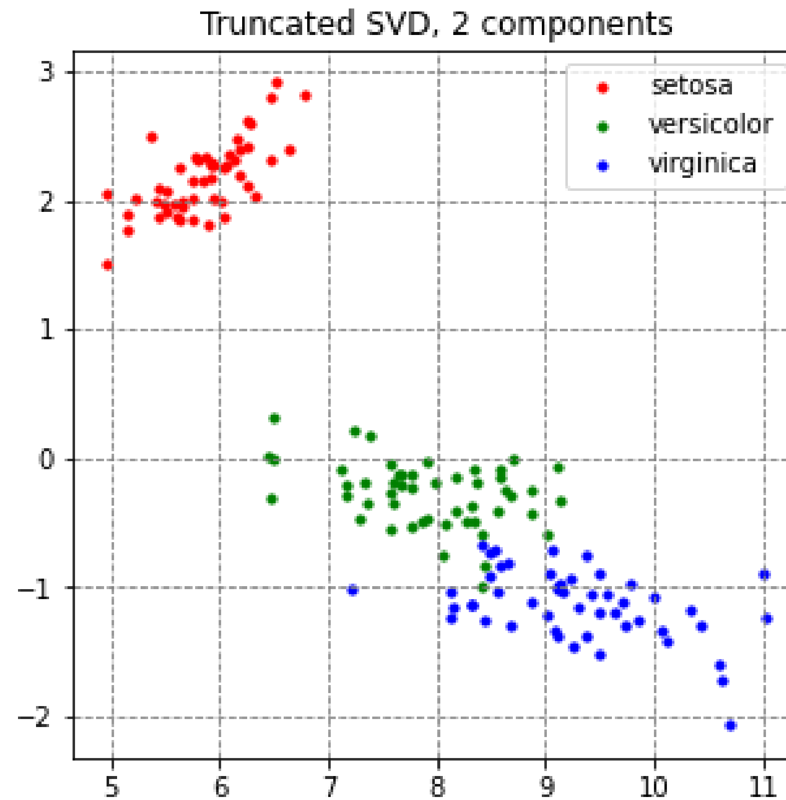
Python-implemented algorithms

- SCiPy SVD
- Scikit-learn TruncatedSVD: works with sparse (with many zeros) matrices efficiently
- Scikit-learn PCA

Feature Extraction in Python

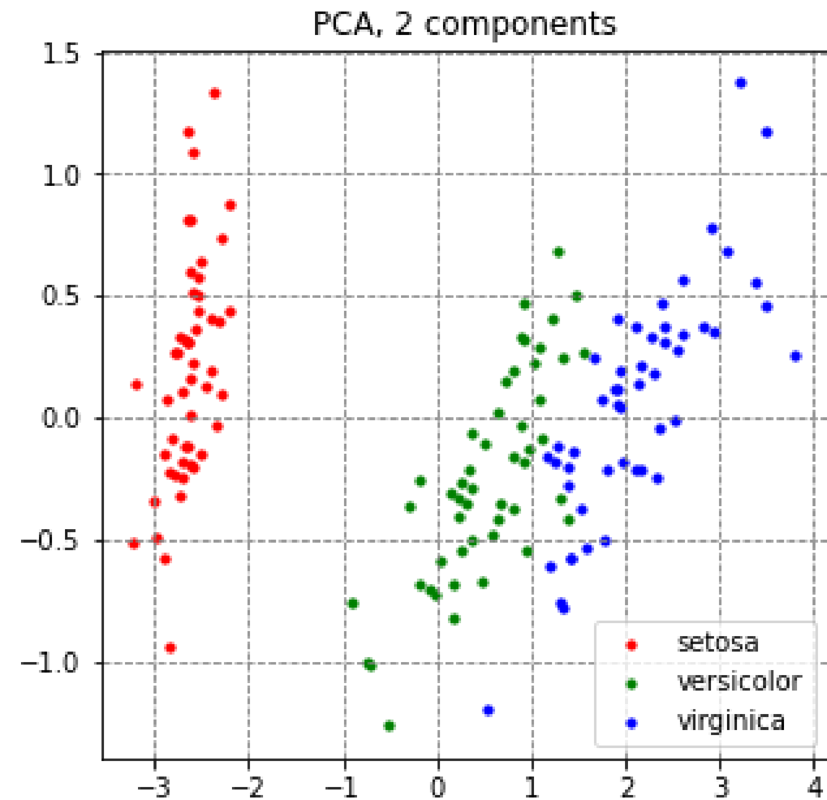
- Dataset: Iris dataset
  - 150 flower observations
  - 4 features
    - sepal length, sepal width, petal length, petal width
  - class variable
    - 0 (setosa), 1 (versicolor), 2 (virginica)
- Perform dimensionality reduction using TruncatedSVD, PCA and LDA
  - 4 to 2 features

Results – TruncatedSVD



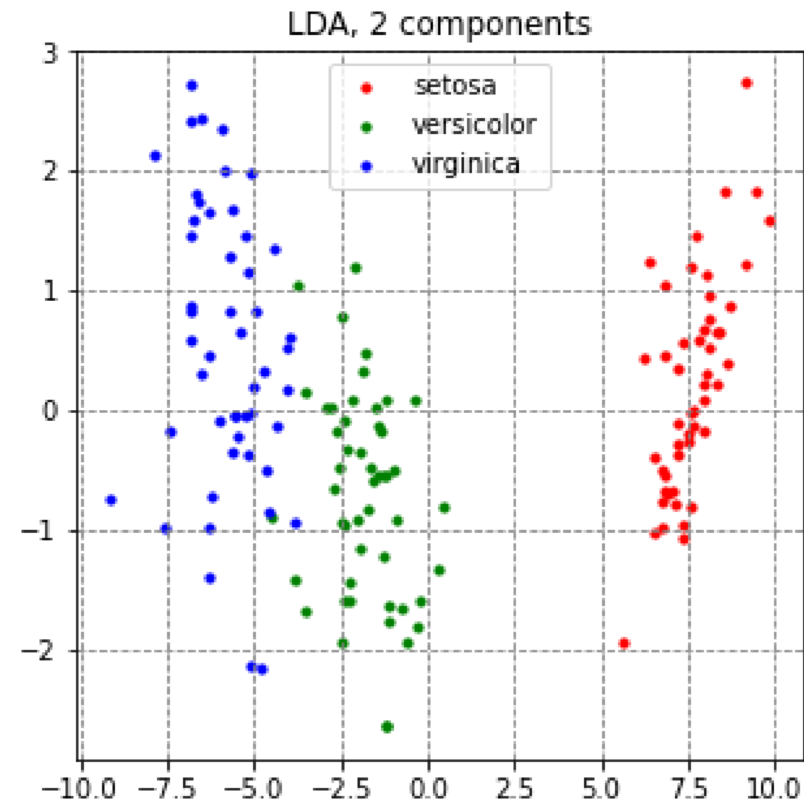
TruncatedSVD explained variance ratio (first two components):
[0.52875361 0.44845576]

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.