DSC510: Introduction to Data Science and Analytics
Lab 5: Data Preparation II

University of Cyprus
Department of Computer Science

Pavlos Antoniou

Office: B109, FST01

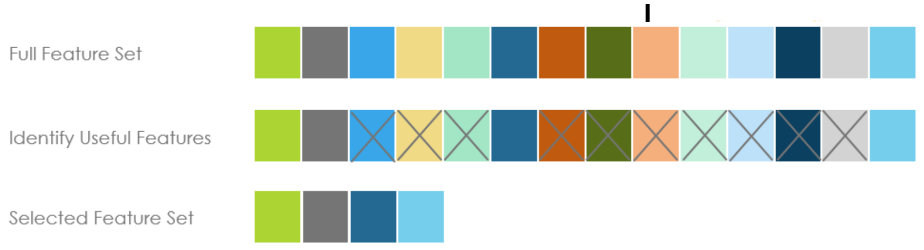
Recap from last lab

- Splitting the dataset is needed at the early stages of a data science project to **avoid data leakage** and ensure **fair model evaluation**
- Focus on pre-processing techniques
  - Data cleaning/imputation → deal with missing values
  - Data encoding → convert categorical data to numerical
    - Label / Ordinal / One-hot encoding
    - Cyclical encoding for datetime-based features
  - Data transformation
    - Scaling: Min-max scaler, Robust scaler
    - Standardization: Standard scaler
    - Unskewing → transform skewed distributions to be more symmetric: Sqrt, Log, Box-Cox, Yeo-Johnson
- When to use each technique

Feature selection & Feature extraction

- Used to eliminate the number of features (columns) leading to:
  - Lower computation time when training predictive modelling algorithms
  - Noise reduction by discarding irrelevant or redundant features
  - Easier to understand (interpretable) feature set, easier to visualize dataset
- Useful in datasets with large number of features that may not all contribute meaningfully to the prediction task.
- Feature selection: Choose a subset of existing features
- Feature extraction: Create new features from the original one
  - Dimensionality Reduction techniques: map high-dimensional (high number of features) data to fewer dimensions (features) while preserving structure, e.g pairwise distances

Feature selection

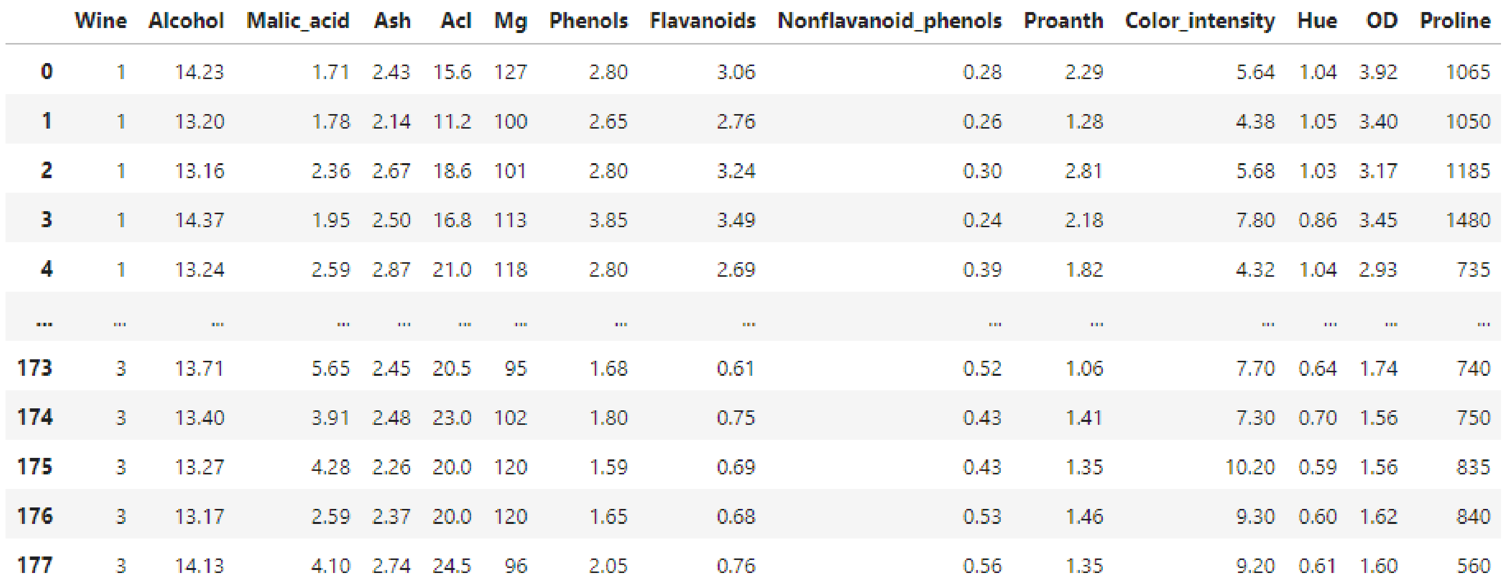


- Select a **subset** of the original feature set
  - Feature selection using statistical techniques: select features based on their statistical properties or statistical relationship with target variable (e.g., correlation, variance, chi-squared test)
    - fast but not accurate methods
  - Feature 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 computes 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')
```

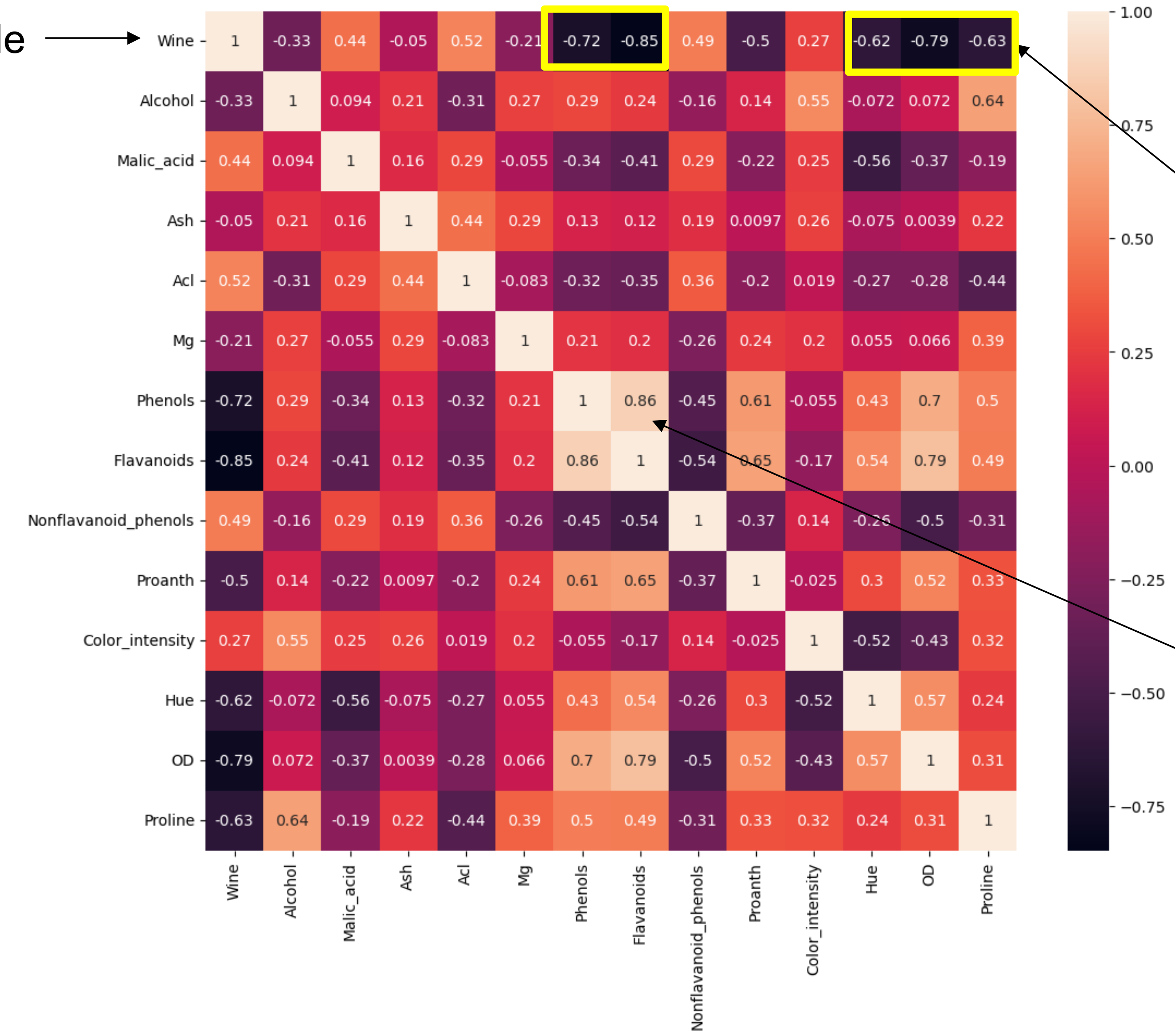


	Wine	Alcohol	Malic_acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid_phenols	Proanth	Color_intensity	Hue	OD	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	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



	Wine	Alcohol	Malic_acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid_phenols	Proanth	Color_intensity	Hue	OD	Proline
Wine	1	-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
Alcohol	-0.33	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
Malic_acid	0.44	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
Ash	-0.05	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
Acl	0.52	-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
Mg	-0.21	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
Phenols	-0.72	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
Flavanoids	-0.85	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
Nonflavanoid_phenols	0.49	-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
Proanth	-0.5	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
Color_intensity	0.27	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
Hue	-0.62	-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
OD	-0.79	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
Proline	-0.63	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

Observations:

- Features Phenoids, 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 correlation

- Using a correlation matrix on features created through **one-hot encoding** or **label encoding** has some considerations and limitations
- Correlation matrix on **one-hot** encoded features
  - One-hot encoded features are often highly correlated due to mutual exclusivity
    - Each row has **exactly one 1 and the rest 0s** → this is what we call **mutual exclusivity**
    - So, if you know the value of one column (e.g. 1), you automatically know the others (0).
    - Statistically, this causes negative correlations between the columns.
  - Correlations between certain one-hot encoded features are often an artifact of the encoding, not real relationships in the data.
  - Therefore, high correlation between one-hot features doesn't necessarily indicate meaningful dependencies or useful associations.

Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

Feature selection using correlation

- Correlation Matrix on **Ordinal/Label** Encoded Features
  - Ordinal Data: If the label encoding represents an ordinal variable (e.g., "low," "medium," "high" encoded as 1, 2, 3), a correlation matrix may offer useful insights about linear relationships since the encoding reflects an order.
  - Nominal Data: If the label encoding is applied to nominal (non-ordinal) data, the numeric labels do not inherently represent distance or order, so correlations may be misleading. For example, encoding "red," "blue," "green" as 1, 2, 3 would imply relationships between colors that don’t exist, leading to incorrect conclusions in a correlation matrix.
- One-hot encoded data and label-encoded nominal data should not be used in a correlation matrix.

Feature selection using correlation

- Correlation with the Target Variable
  - If target is categorical, do not include in correlation matrix
    - Correlation works only on numeric, continuous data.
    - Encoding the target (e.g. 0 and 1 for classes) is arbitrary and can produce misleading correlations.
  - If target is continuous numerical, include in correlation matrix
    - Seeing how strongly each feature correlates with the target helps identify potentially useful/important features

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	Acl	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	Acl	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	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.228572	0.709990	314.907474
min	11.030000	0.740000	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.480000	1.270000	278.000000
25%	12.362500	1.602500	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0.782500	1.937500	500.500000
50%	13.050000	1.865000	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	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

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)
```

`X_scaled.std()`

```
Alcohol                 0.381132
Malic_acid              0.569766
Ash                     0.623277
Acl                     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
```

variances are on the same scale after transformation

- 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 <= 0.35

Feature selection using feature importance

- Ensemble methods can be used to assign scores to input features during training phase.
  - ML methods that combine multiple base models to produce stronger predictive model (see more here)
- These scores show how much each feature contributes to prediction
- Feature importance can be calculated for both regression problems (predicting numbers) and classification problems (predicting categories) – studied thoroughly in Labs 6-9

(*) Short list of common Ensemble Learning methods: ExtraTrees, Random Forest, AdaBoost, Gradient Boosting, XGBoost, LightGBM, CatBoost

Feature selection using feature importance

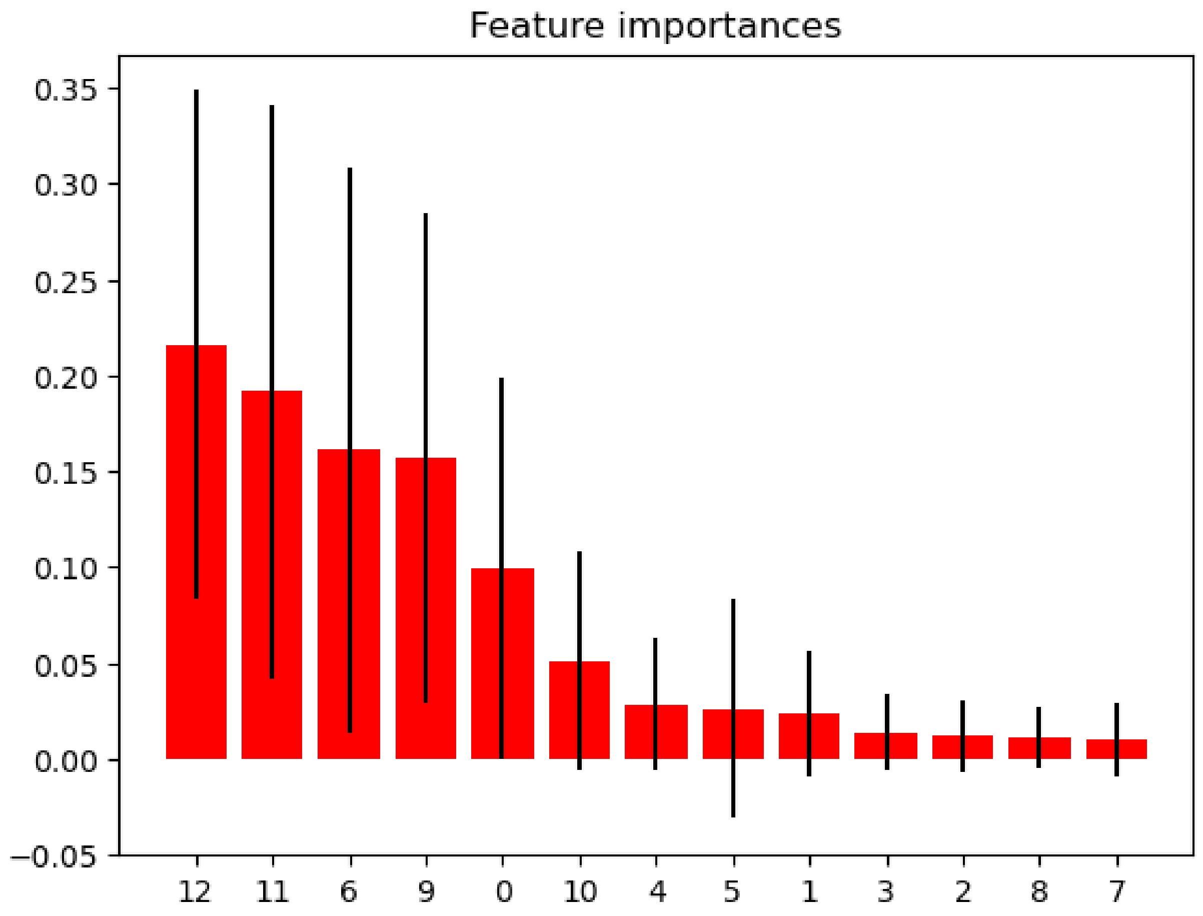
- The scores are useful and can be used in a range of situations in a predictive modeling problem, such as:
  - Better understanding the data (which feature(s) are important, i.e. influencing the decision-making process)
  - Reducing the number of input features (choosing the most important features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

# 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 (0.216120)
2. feature 11 - OD (0.191357)
3. feature 6 - Flavanoids (0.161096)
4. feature 9 - Color_intensity (0.157190)
5. feature 0 - Alcohol (0.098776)
6. feature 10 - Hue (0.051089)
7. feature 4 - Mg (0.028590)
8. feature 5 - Phenols (0.026197)
9. feature 1 - Malic_acid (0.023425)
10. feature 3 - Acl (0.013834)
11. feature 2 - Ash (0.011700)
12. feature 8 - Proanth (0.010689)
13. feature 7 - Nonflavanoid_phenols (0.0
09936)
```

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_train, y_train)
print(list(selector.support_))
print(list(selector.ranking_))
```

```
[True, False, False, False, False, False, True, False, False
, True, False, True, True]
[1, 3, 9, 5, 6, 4, 1, 8, 7, 1, 2, 1, 1]
 0                 6        9     11  12
```

Important features

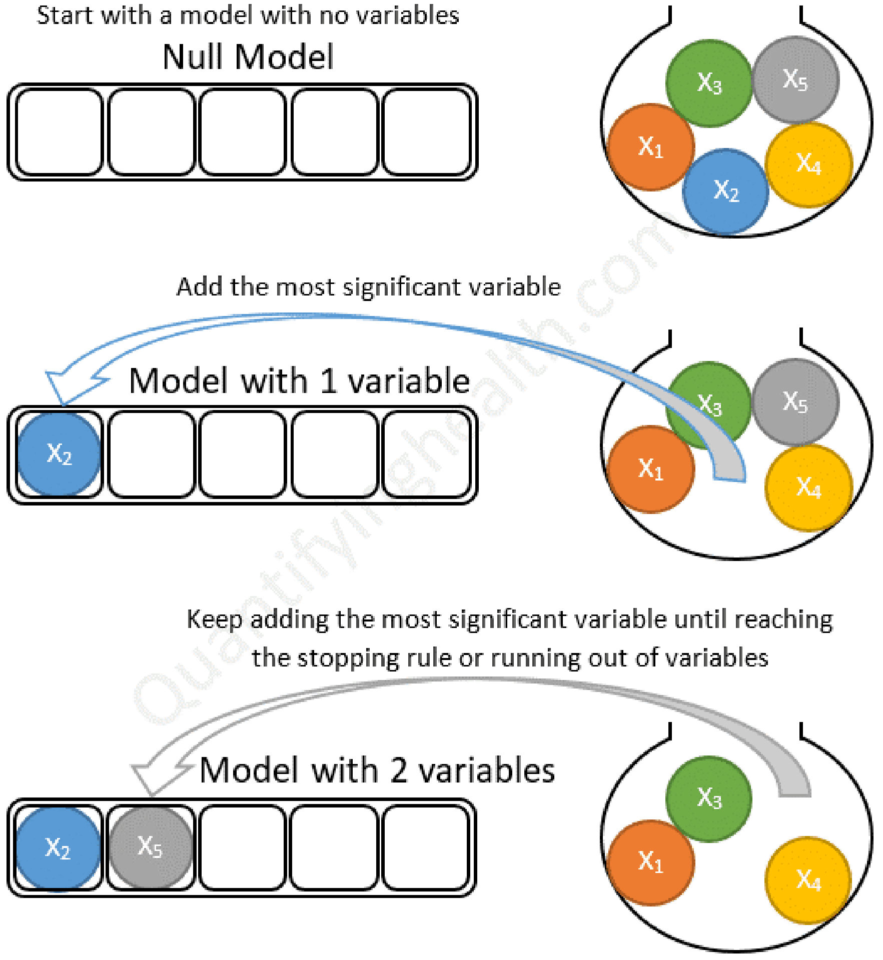
Feature selection using predictive performance of ML model

- 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)
  - Perform training and validation of the model using the Cross-Validation (CV) process
  - 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

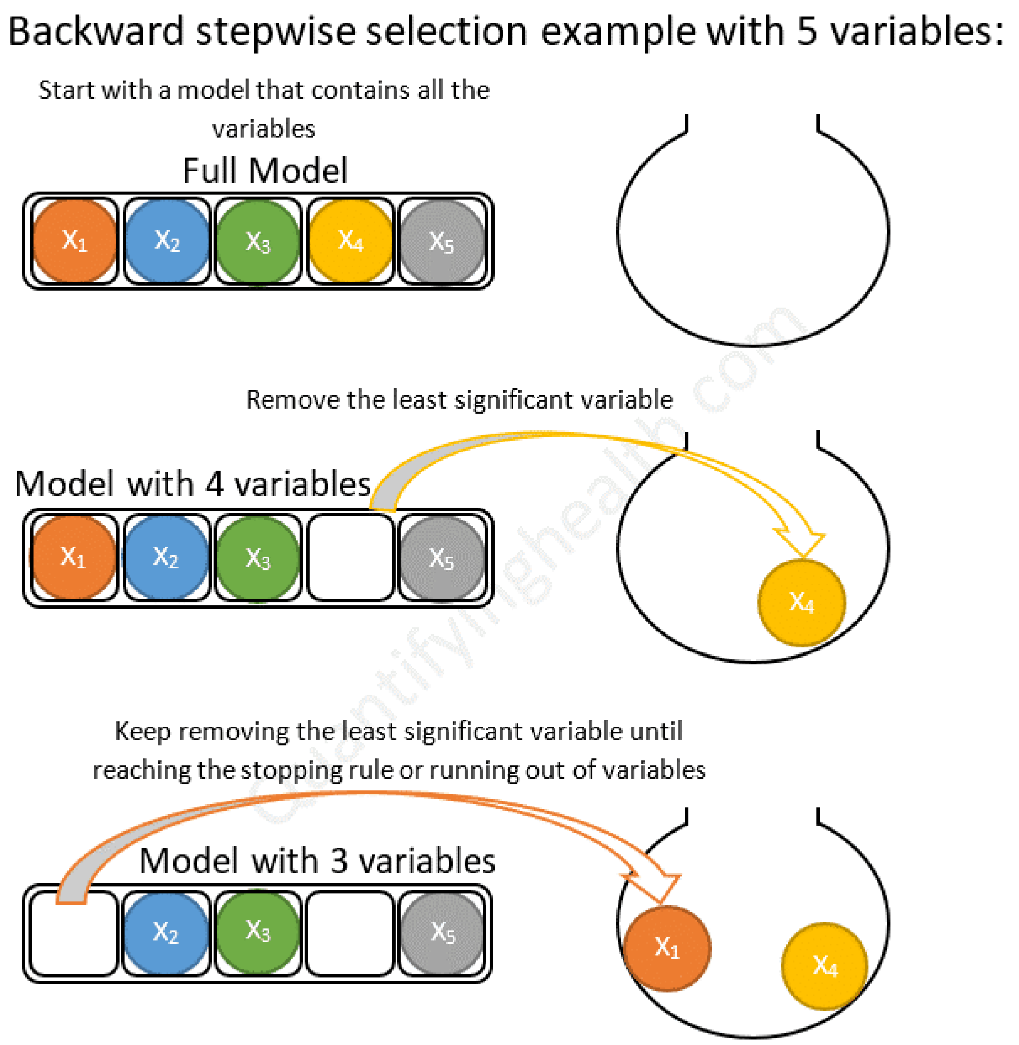
Forward stepwise selection example with 5 variables:



Start with a model with no variables
Null Model
X_3
X_5
X_1
X_4
X_2
Add the most significant variable
Model with 1 variable
X_2
Keep adding the most significant variable until reaching the stopping rule or running out of variables
Model with 2 variables
X_2
X_5

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:
Start with a model that contains all the variables
Full Model
X_1
X_2
X_3
X_4
X_5
Remove the least significant variable
Model with 4 variables
Keep removing the least significant variable until reaching the stopping rule or running out of variables
Model with 3 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
          n_jobs=-1,            # number of CPUs to use, -1 → all CPUs
          cv=10)                # 10-fold cross validation: resampling method
that uses different portions of the data to train and validate the model on different iterations. Here, we
have 10 iterations per feature selection round (more details in the next labs).

sfs = sfs.fit(X_train_scaled, y_train)
# Results
# Features: 1/5 -- score: 0.7809523809523811
# Features: 2/5 -- score: 0.9223809523809525
# Features: 3/5 -- score: 0.9642857142857142
# Features: 4/5 -- score: 0.9580952380952381
# Features: 5/5 -- score: 0.9785714285714286
```

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_) # (4, 5, 6, 9, 11)
print('Prediction score:',sfs.k_score_)        # 0.9785714285714286
```

- **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
          n_jobs=-1)

sbs = sbs.fit(X_train_scaled, y_train)
print('\nSequential Backward Selection (k=5):')
print('Selected features:',sbs.k_feature_idx_)# (0, 8, 9, 10, 12)
print('Prediction (CV) score:',sbs.k_score_)  # 0.9647619047619047
```

Examples

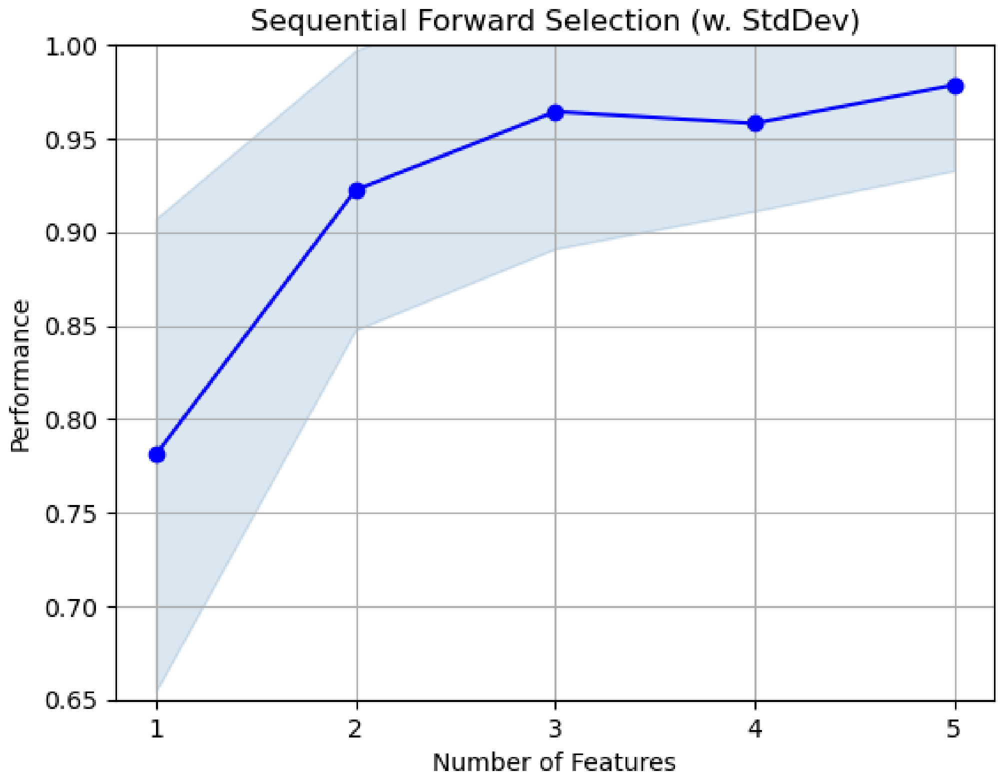
- **Example 3 – Plotting the results**

```
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,
          n_jobs=-1)

sfs = sfs.fit(X_train_scaled, y_train)
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: 0.7809523809523811
Features: 2/5 -- score: 0.9223809523809525
Features: 3/5 -- score: 0.9642857142857142
Features: 4/5 -- score: 0.9580952380952381
Features: 5/5 -- score: 0.9785714285714286
```



Examples

- **Example 4 – Selecting the "best" feature combination in k-range**
  - Set `k_features` to a tuple (min_k, max_k)
  - In forward selection
    - It 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 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

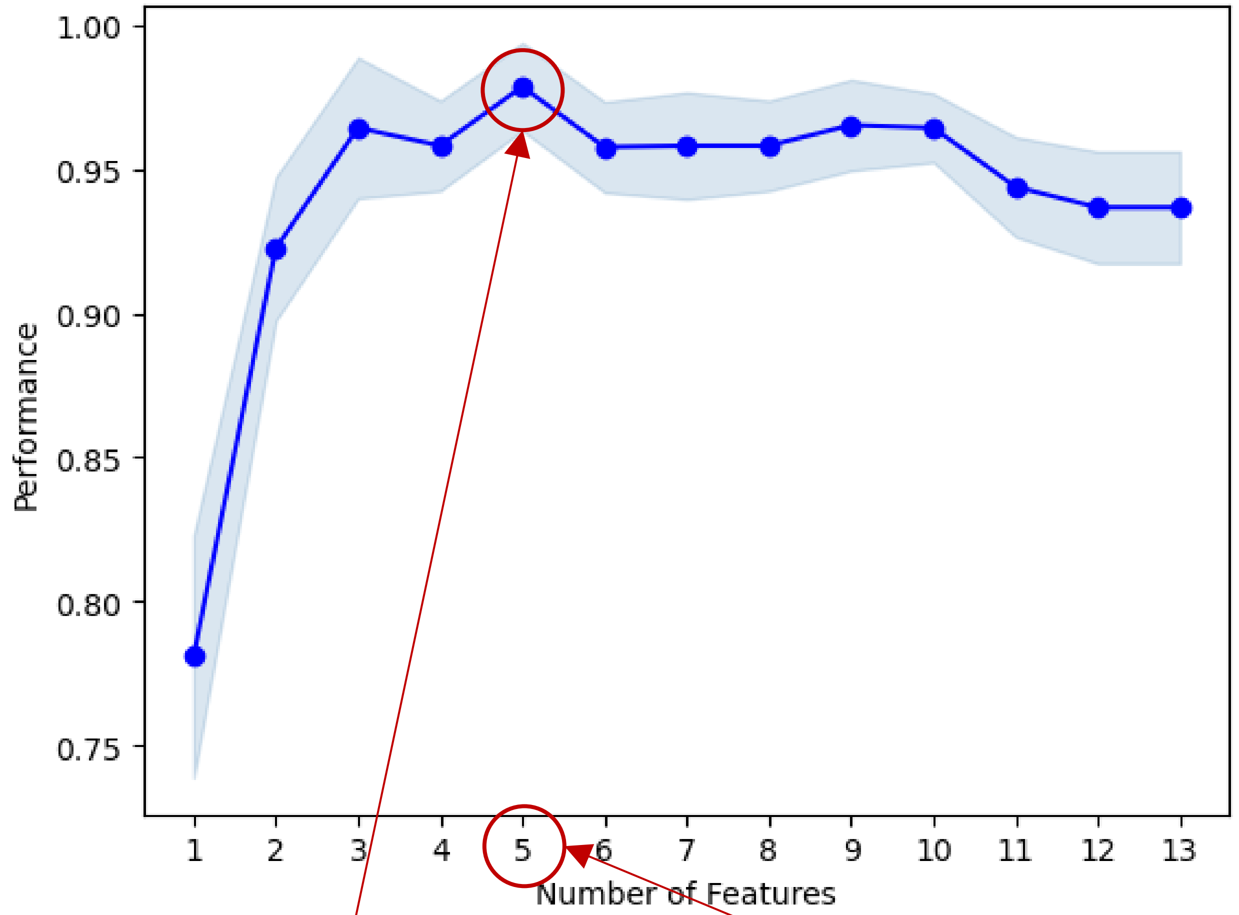
Examples

- **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,
           n_jobs=-1)

sfs_range = sfs_range.fit(X_train_scaled, y_train)

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: 0.979): (4, 5, 6, 9, 11)
X_train_scaled_selected = sfs_range.transform(X_train_scaled) # extract selected columns
```

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,
           n_jobs=-1)

# no need for scaled features in tree-based models
sfs_range = sfs_range.fit(X_train, y_train)

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

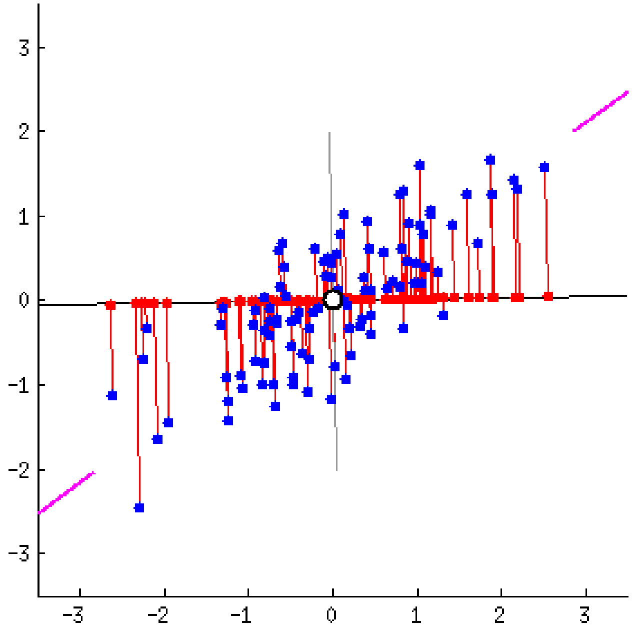
- Differs from feature selection in two ways:
  - Instead of choosing subset of features
  - Create new feature set (dimensions)

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.**svd**s.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

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)

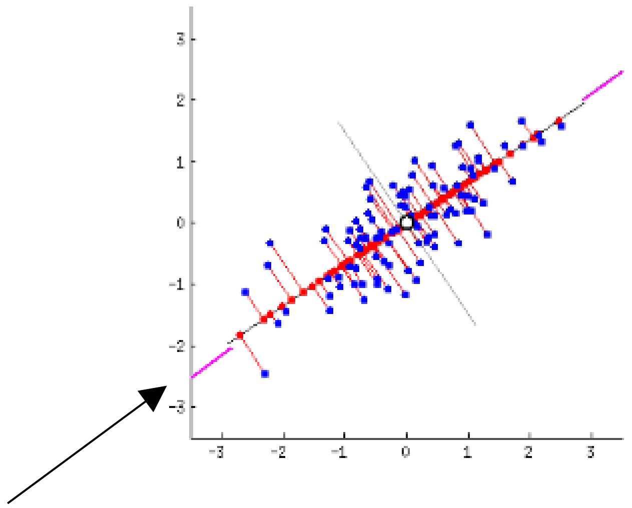


The eigenvectors and eigenvalues of a covariance (or correlation) matrix represent the “core” of a PCA: The eigenvectors (principal components) determine the directions of the new feature space, and the eigenvalues determine their magnitude. In other words, the eigenvalues explain the variance of the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

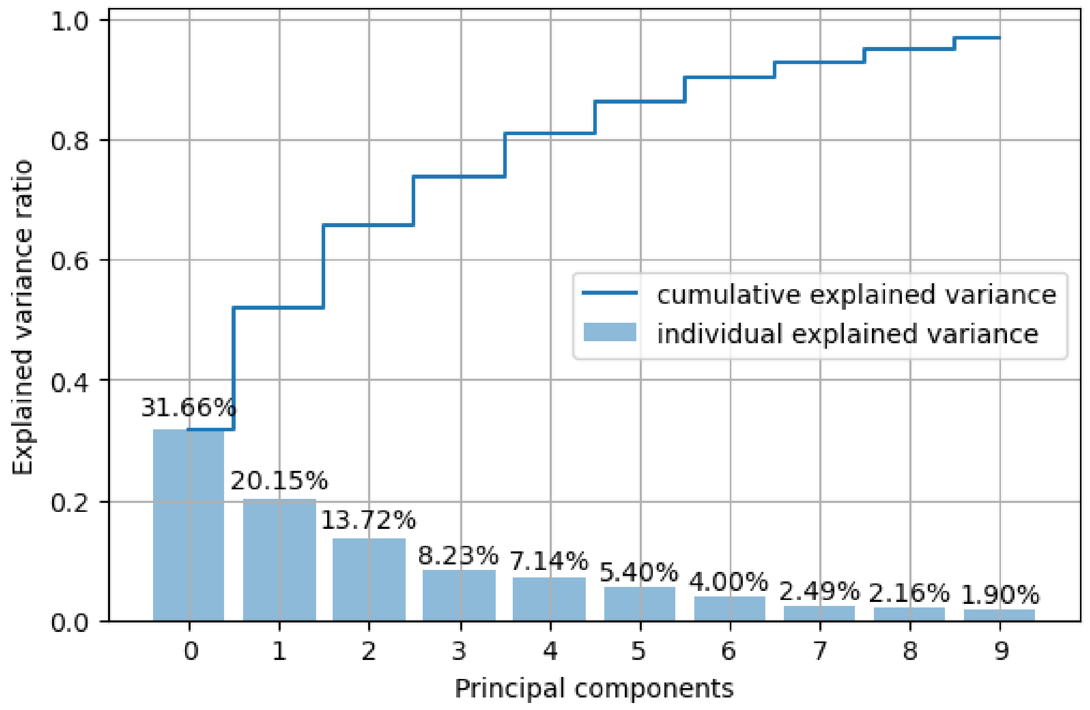


PCA considerations

- PCA works well when relationships between features are linear
  - Inappropriate for non-normal data (i.e. skewed or discrete data) → use unskewing techniques on skewed features prior PCA (Lab4)
- 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 requires that features are mean-centered (mean=0)
  - for each feature, this is done by subtracting the mean of that feature from each data point → done by PCA in advance for all features
- 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

PCA

- 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 65% 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
  - Σ (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 **Σ**) 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.

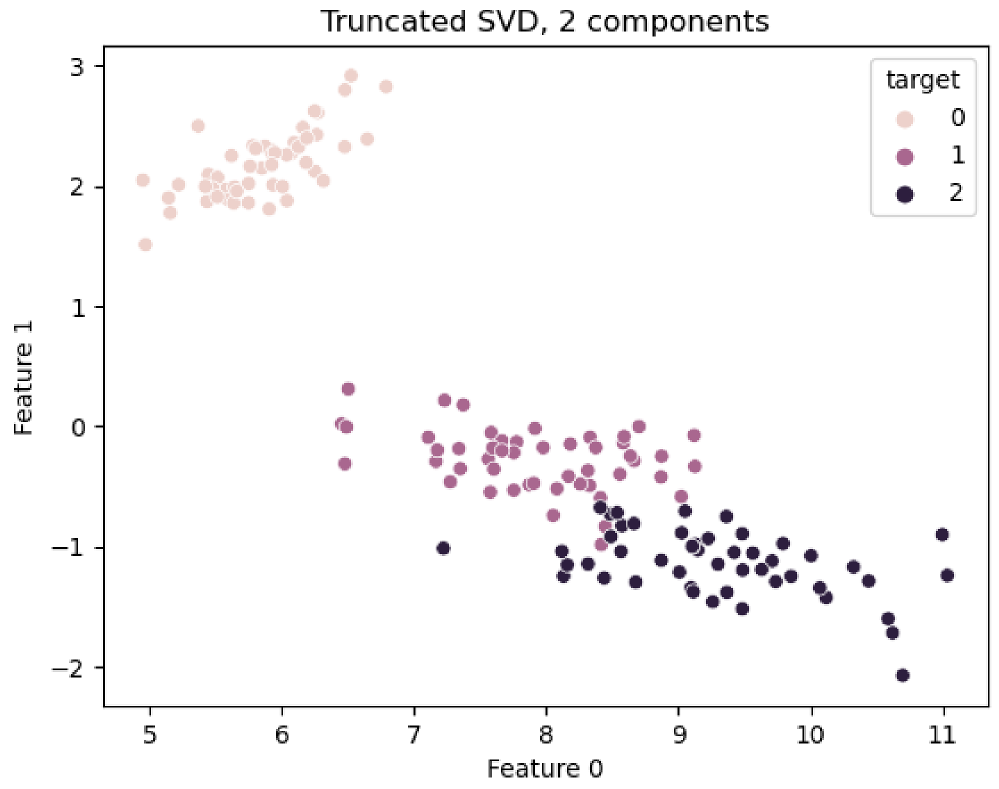
Python-implemented algorithms

- 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

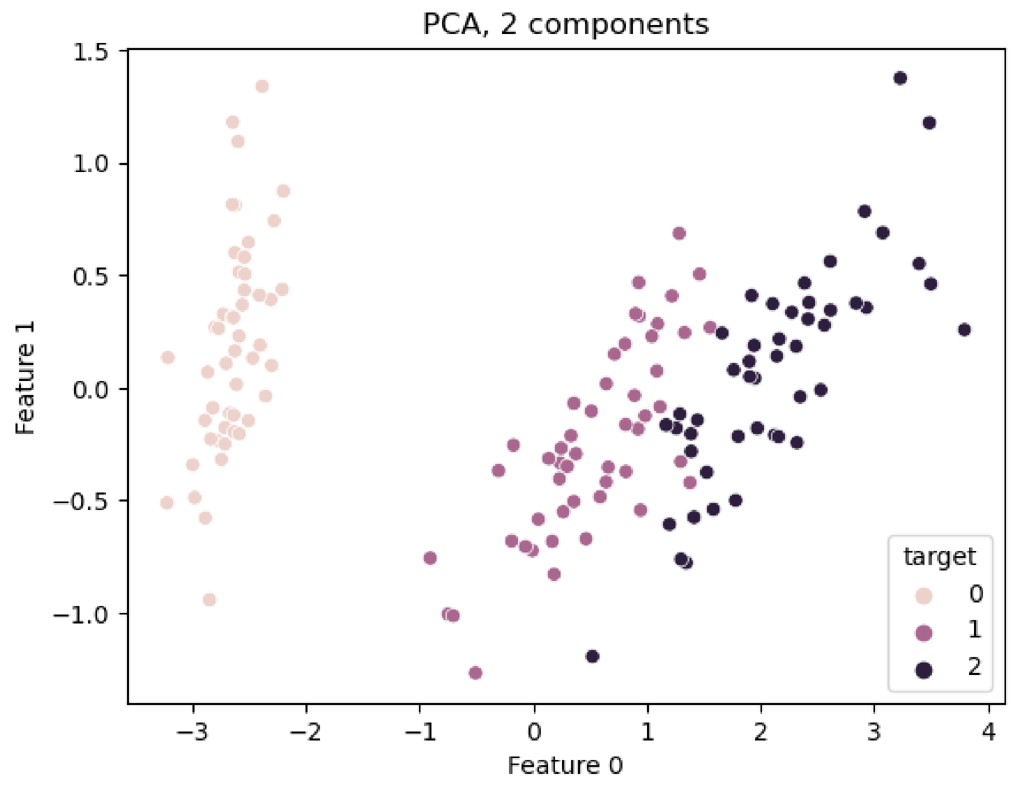
- Dataset: Iris dataset
  - 150 flower observations
  - 4 features
    - sepal length, sepal width, petal length, petal width
  - class variable
    - 0 (setosa), 1 (versicolor), 2 (virginica)
- Perform dimensionality reduction using TruncatedSVD, PCA and LDA
  - 4 to 2 features

Results – TruncatedSVD



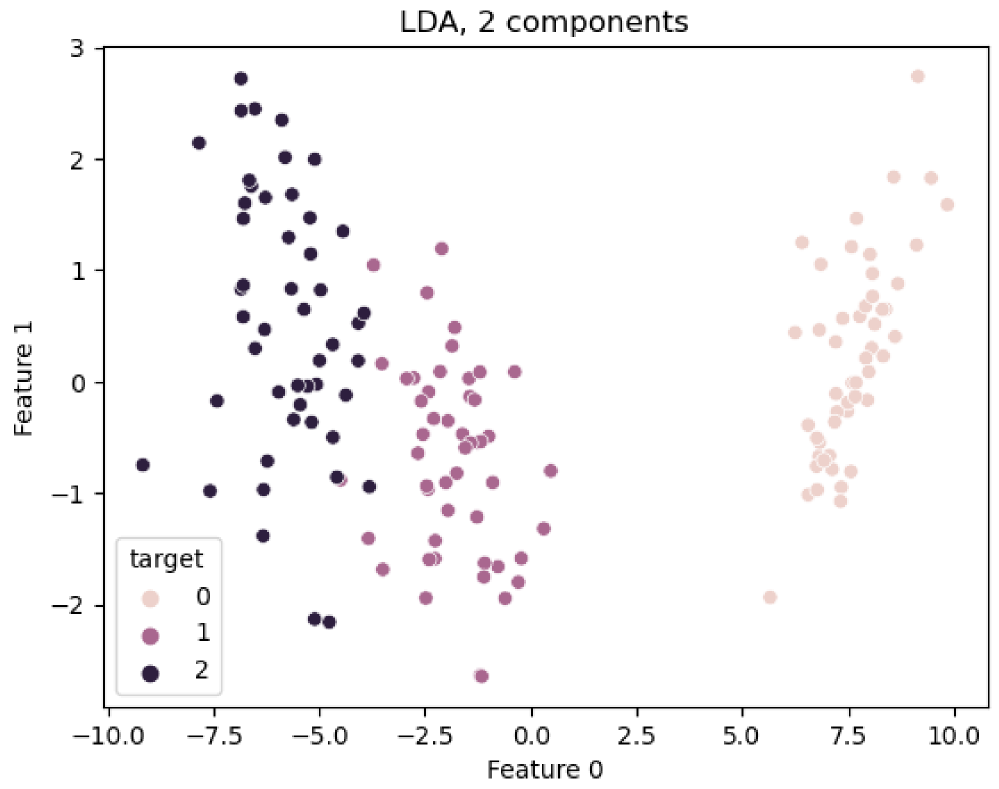
```
TruncatedSVD explained
variance ratio (first two
components):
[0.52875361 0.44845576]
```

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]
```

Best practice: Keep different versions of the dataset !!!

- Beneficial to keep the various versions of dataset at each stage, as it gives you the flexibility to try different approaches without redoing pre-processing steps. Here’s a breakdown of why retaining each version could be helpful:
  - **Original Dataset**: Keeping the raw data allows you to revisit it if you need to apply new techniques (e.g. imputing, scaling, encoding) in the future.
  - **Cleaned Data**: Keeping a dataset with just the basic cleaning (imputation, drop useless columns or rows with large number of missing values) lets you experiment with different scaling (min-max, standard, robust) and encoding (label, one hot, cyclical) techniques without starting over.
  - **Scaled and Encoded Data**: Keeping different datasets with various scaling or encoding techniques lets you be compatible with different ML techniques e.g. distance-based perform better with scaled data.
  - **Feature-Selected Data**: Retaining a version of your dataset after feature selection can be helpful to evaluate if selected features improve model performance compared to using all features.
  - **PCA or Other Extracted Features**: Keeping a transformed dataset with dimensionality reduction techniques allows you to compare models trained on reduced feature sets versus the full feature set.
- **Why Try Different Versions?**
  - **Model Flexibility**: Some models benefit from standardized scaling, while others don’t, and models like tree-based algorithms may perform better with the original (unscaled) features.
  - **Experimentation**: Comparing the results from different versions helps identify the most effective feature engineering and transformation steps for each algorithm.
  - **Efficient Experimentation**: You can re-use pre-processed datasets for quicker experimentation, avoiding the time required to apply transformations again.

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.