

Feature selection & parameter tuning

Ruxandra Stoean

Further bibliography

- M. Kuhn, K. Johnson, Feature Engineering and Selection (Chapman & Hall/CRC Data Science Series), Chapman & Hall/CRC Data Science Series, 2021
- V. Bolón-Canedo, A. Alonso-Betanzos, Recent Advances in Ensembles for Feature Selection, Intelligent Systems Reference Library, Springer, 2018
- E. Bartz, T. Bartz-Beielstein, M. Zaefnerer, O. Mersmann, Hyperparameter Tuning for Machine and Deep Learning with R: A Practical Guide, Springer, 2023
- L. Owen, Hyperparameter Tuning with Python: Boost your machine learning model's performance via hyperparameter tuning, Packt Publishing, 2022

Feature selection

Feature selection

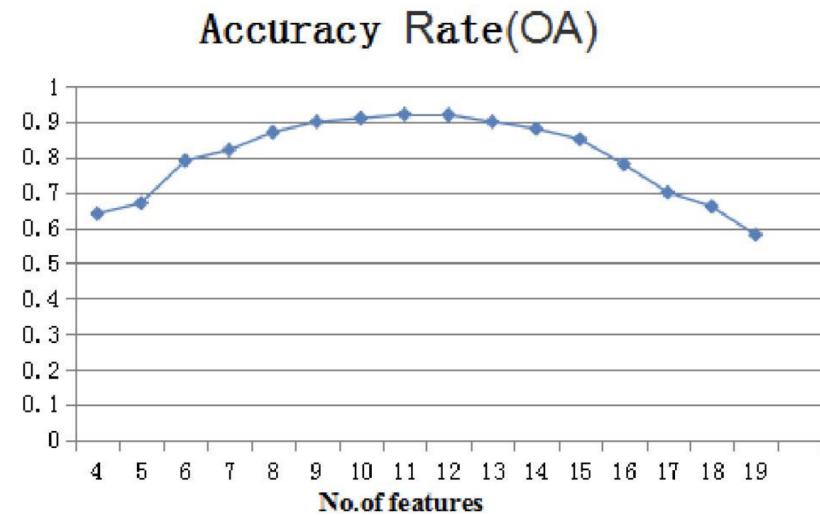
- Data could have 10 000 descriptive features
 - Their number should be reduced to 1000 (or 100) to apply a model
 - Which 1000 attributes should be kept?
- This is *feature selection*

The diagram illustrates the relationship between independent variables and a dependent variable. A horizontal bar at the top is labeled "Features (independent variables)" and contains the names of the variables: Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age, and Outcome. Below this bar is a table with nine rows of data. Red arrows point from each of the first eight columns (independent variables) to their corresponding values in the table. A red arrow also points from the "Outcome" column to its corresponding value in the table. The table has alternating light gray and white rows. The first row (light gray) has values: Pregnancies=6, Glucose=148, BloodPressure=72, SkinThickness=35, Insulin=0, BMI=33.6, DiabetesPedigreeFunction=0.627, Age=50, Outcome=1. The second row (white) has values: Pregnancies=1, Glucose=85, BloodPressure=66, SkinThickness=29, Insulin=0, BMI=26.6, DiabetesPedigreeFunction=0.351, Age=31, Outcome=0. The third row (light gray) has values: Pregnancies=8, Glucose=183, BloodPressure=64, SkinThickness=0, Insulin=0, BMI=23.3, DiabetesPedigreeFunction=0.672, Age=32, Outcome=1. The fourth row (white) has values: Pregnancies=1, Glucose=89, BloodPressure=66, SkinThickness=23, Insulin=94, BMI=28.1, DiabetesPedigreeFunction=0.167, Age=21, Outcome=0. The fifth row (light gray) has values: Pregnancies=0, Glucose=137, BloodPressure=40, SkinThickness=35, Insulin=168, BMI=43.1, DiabetesPedigreeFunction=2.288, Age=33, Outcome=1.

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1

Why needed?

- Sometimes performance decreases when the number of features is high
- The curse of dimensionality
 - The space volume increases rapidly, data become sparse
 - More data samples will be needed



Increase in noise

- Some additional features can only increase noise
 - E.g. other features in Pima diabetes: favorite music, hair color
- Models could guide learning towards the new attributes
 - They could find correlations only true for the training set, and not generalizable to new data
- More features imply more complex models
 - More weights for a neural network
 - More nodes in decision trees
 - More trees in random forests etc.
 - Increased search space => harder search

Reasons for feature selection

- To improve performance (in terms of accuracy, speed)
 - For data visualization, if possible
 - To reduce dimensionality and eliminate noise
 - Simpler models are easier to interpret
 - Better generalization by reducing overfitting
 - Variable redundancy
-
- Feature selection is the process of choosing an (almost) optimal subset of attributes with respect to a certain criterion.

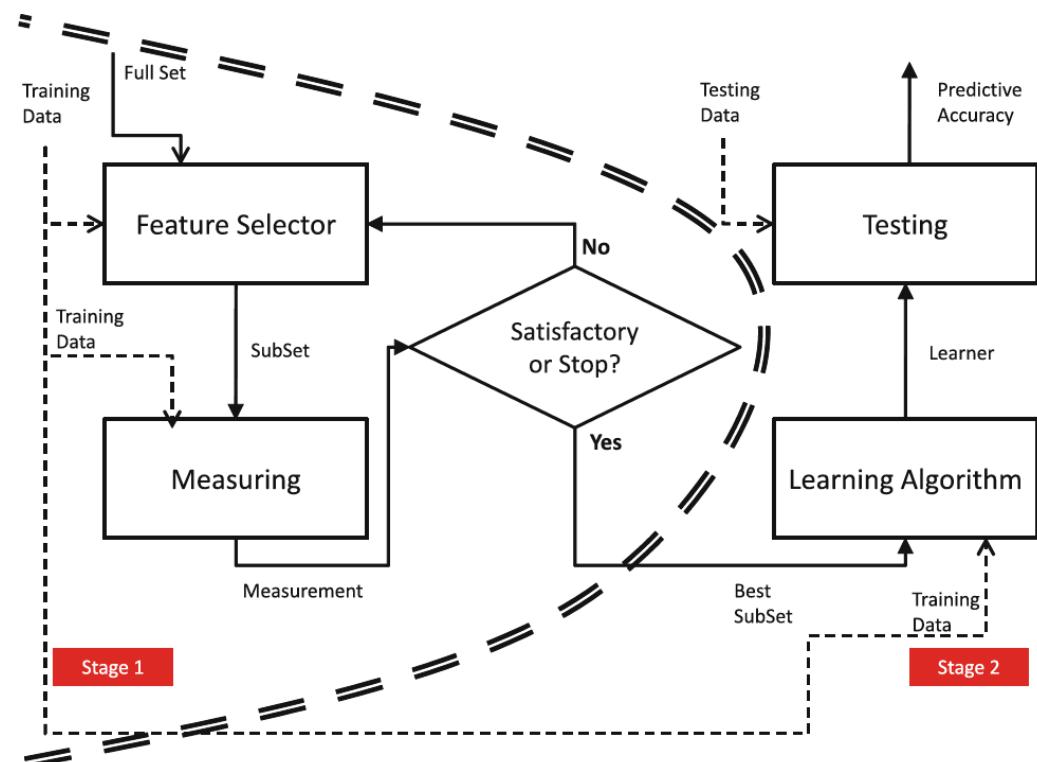
Feature selection approaches

- Filter
 - Base, univariate, based on information gain, Fisher score, correlations
- Wrapper
 - Forward and backward selection, exhaustive, meta-heuristics
- Embedded
 - Lasso, Random forest



Filter methods

- For preprocessing
- Model-agnostic
- Model-independent
- Computationally economic
- Faster than wrapper
- Eliminate irrelevant attributes
- Discover correlations between variables and class



<https://web.iitd.ac.in/~bspanda/fs.pdf>

Variance threshold

- Eliminates attributes with zero (very low) variance – constant values

```
# variance threshold

from sklearn.feature_selection import VarianceThreshold
from sklearn import datasets
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

ix, iy = datasets.load_iris(return_X_y=True)
ix_train, ix_test, iy_train, iy_test = train_test_split(ix, iy, test_size = 0.33)

sel = VarianceThreshold(threshold=0)
sel.fit_transform(ix_train)
sel.get_support() # see selected features

✓ 0.0s

array([ True,  True,  True,  True])
```

Python

R

```
1 # variance threshold
2 library(caret)
3
4 data(iris)
5 dat <- iris
6
7 index <- 1:nrow(dat)
8 testindex <- sample(index, trunc(length(index) / 3))
9 testset <- dat[testindex, ]
10 trainset <- dat[-testindex, ]
11
12 print(names(trainset)[nearZeroVar(trainset)])
```

Univariate

- Selects the best features based on statistical tests, such as ANOVA
- Estimates the degree of linear relationship between features and class
- Assumes features have a normal distribution

Univariate

- SelectKBest - the features with the highest k scores
 - e.g., Chi-square test on the relation between variables and target
- SelectPercentile – the features with respect to a percentile of the highest scores

```
# select K best
from sklearn.feature_selection import SelectKBest, chi2

ix_new = SelectKBest(chi2, k=2).fit_transform(ix_train, iy_train)
ix_new
```

```
array([[1.7,  0.3],
       [5.8,  1.8],
       [4. ,  1.3],
       [4.5,  1.5],
       [4.3,  1.3],
       [5. ,  1.7],
       [1.4,  0.2],
       [5.1,  1.6],
       [1.6,  0.2],
       [5.1,  2.4],
       [1.7,  0.4],
       [5.1,  1.5],
       [1.4,  0.2],
       [1.6,  0.2],
       [1.3,  0.3],
       [5. ,  1.5],
       [5.5,  2.1],
       [6.1,  1.9],
       [4.5,  1.6],
```

Python

Petal length and width for Iris

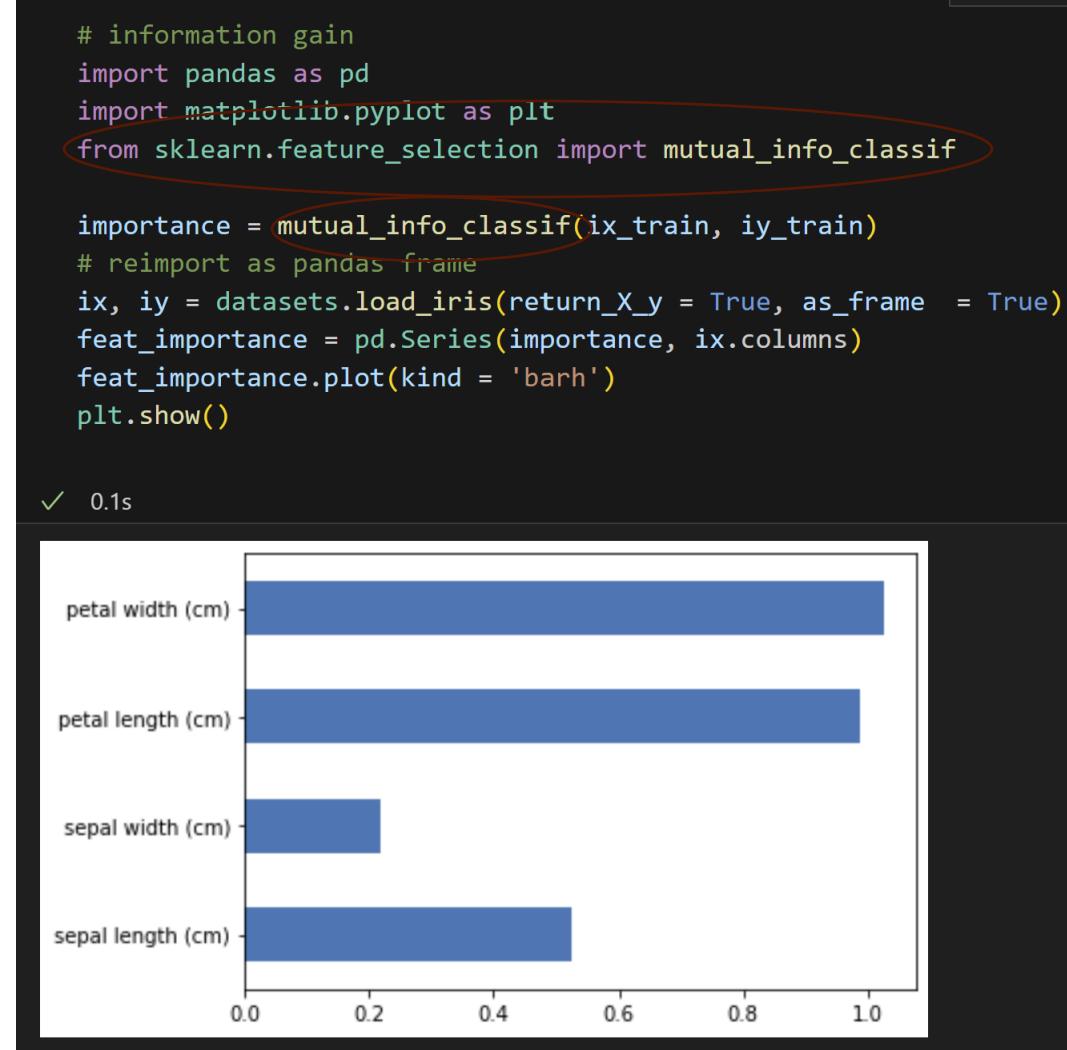
```
array([[1.7],
       [5.8],
       [4. ],
       [4.5],
       [4.3],
       [5. ],
       [1.4],
       [5.1],
       [1.6],
       [5.1],
       [1.7],
       [5.1],
       [1.4],
       [1.6],
       [1.3],
       [5. ],
       [5.5],
```

```
# select percentile
from sklearn.feature_selection import SelectPercentile, chi2

ix_new = SelectPercentile(chi2, percentile=10).fit_transform(ix_train, iy_train)
ix_new
```

Mutual information in Python

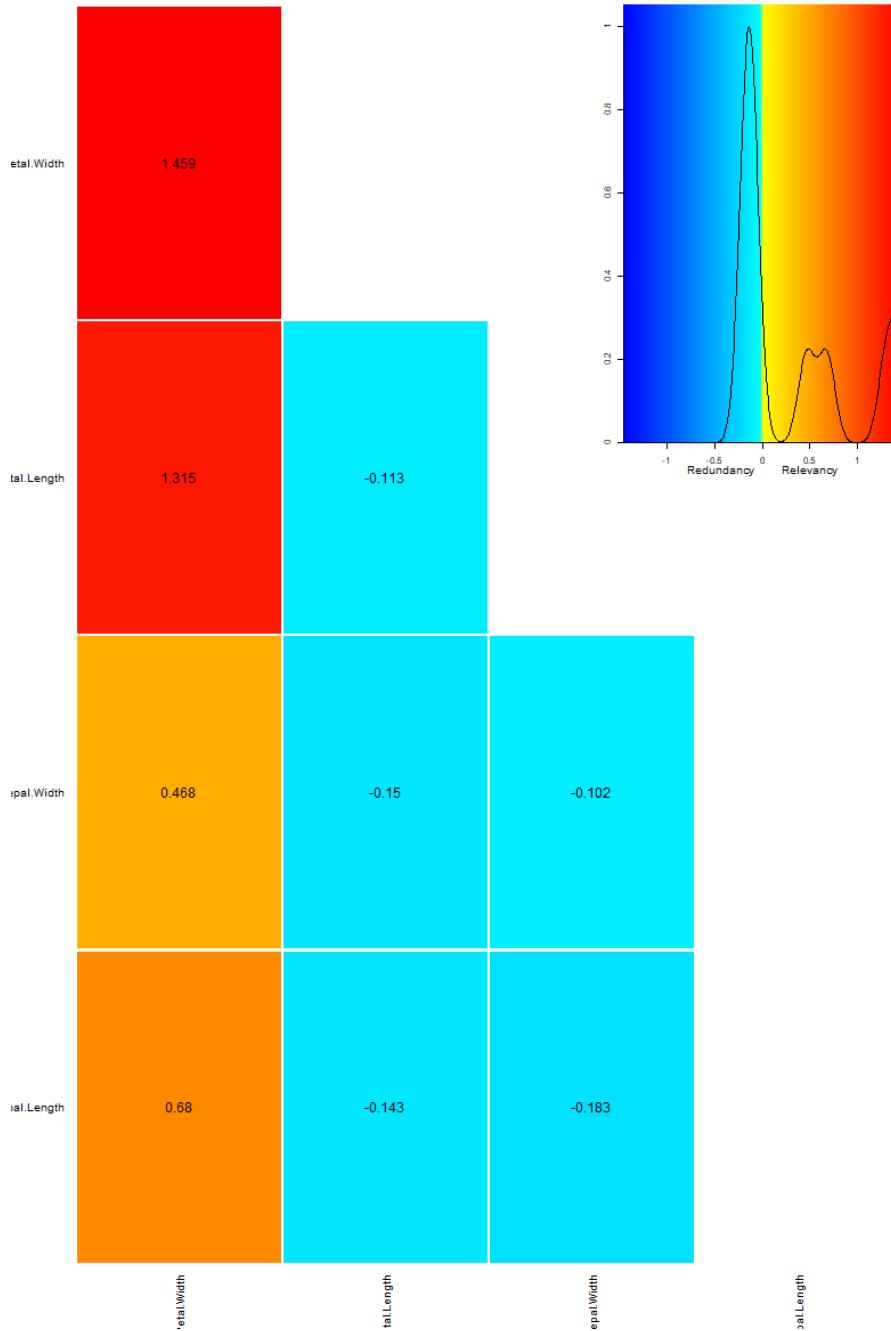
- Based on the gain achieved by reducing entropy
- Ranking is established based on the information gain of each feature in relation to the target



Mutual information in R - varrank

```
1 # Mutual information
2
3 library(varrank)
4
5 data(iris)
6 dat <- iris
7
8 index <- 1:nrow(dat)
9 testindex <- sample(index, trunc(length(index) / 3))
10 testset <- dat[testindex, ]
11 trainset <- dat[-testindex, ]
12
13 varMI <- varrank(trainset, variable.important = "Species", discretization.method = "sturges")
14 plot(varMI)
```

Variable relevance



Fisher's score in Python

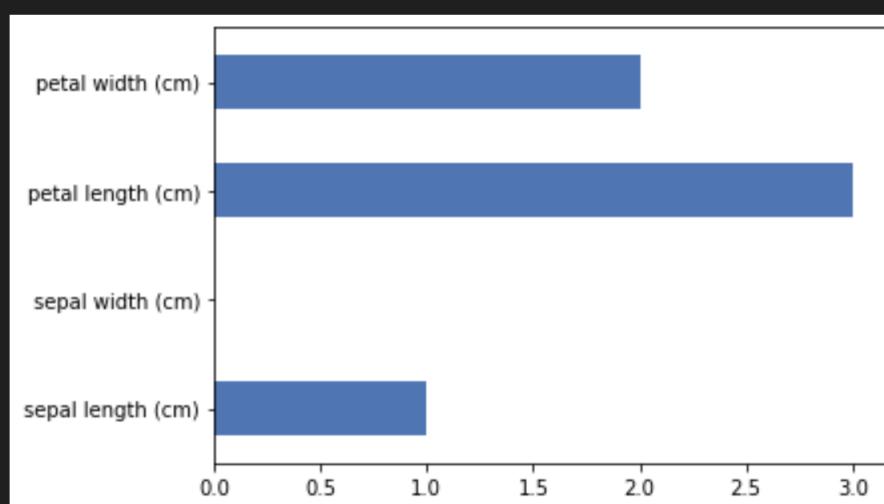
- Fisher's score computes the inter-class variance over intra-class variance
- Variables are ranked according to score
- pip install skfeature-chappers

```
# Fisher's score
import pandas as pd
import matplotlib.pyplot as plt
from skfeature.function.similarity_based import fisher_score

ranking = fisher_score.fisher_score(ix_train, iy_train)

# reimport as pandas frame
ix, iy = datasets.load_iris(return_X_y = True, as_frame = True)
feat_importance = pd.Series(ranking, ix.columns)
feat_importance.plot(kind = 'barh')
plt.show()
```

✓ 0.1s



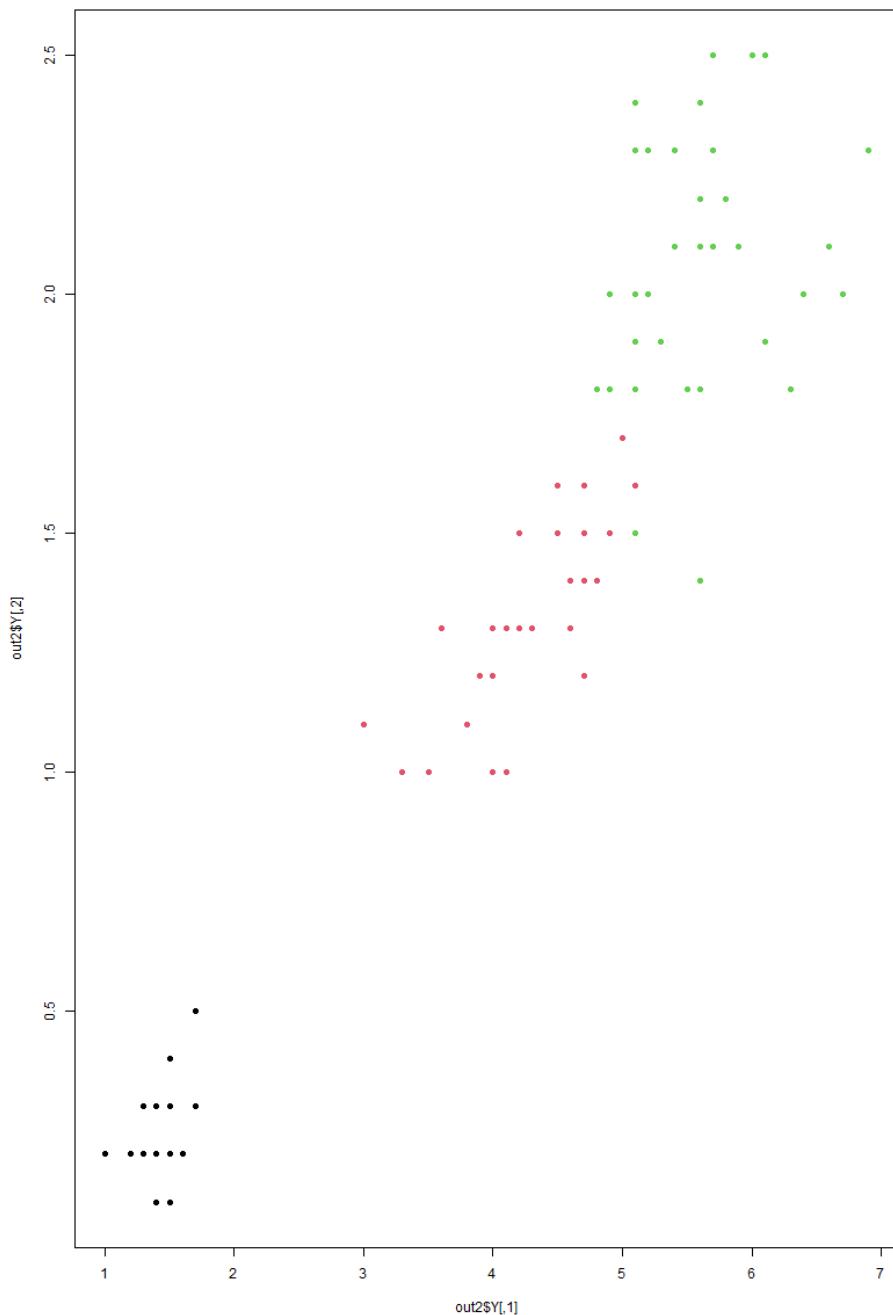
Feature	Fisher's Score
petal width (cm)	~2.0
petal length (cm)	~3.0
sepal width (cm)	~0.1
sepal length (cm)	~1.0

Fisher's score in R

```
1 # Fisher score
2 library(Rdimtools)
3
4 data(iris)
5 dat <- iris
6
7 index <- 1:nrow(dat)
8 testindex <- sample(index, trunc(length(index) / 3))
9 testset <- dat[testindex, ]
10 trainset <- dat[-testindex, ]
11
12 out2 <- do.fscores(as.matrix(trainset[, -classColumn]), as.factor(trainset[, classColumn]))
13 plot(out2$Y, pch=19, col=as.factor(trainset[, classColumn]), main="Fisher Score")
```

Plot of data with petal length and width

Fisher Score



Correlation

- Determines the correlation between the features
 - Select only one of the correlated variables
 - e.g., choose the one with higher importance from other measures
 - or the one most correlated to the target
- Pearson correlation can be the underlying statistics

Python

```
# correlation

import seaborn as sb
import matplotlib.pyplot as plt

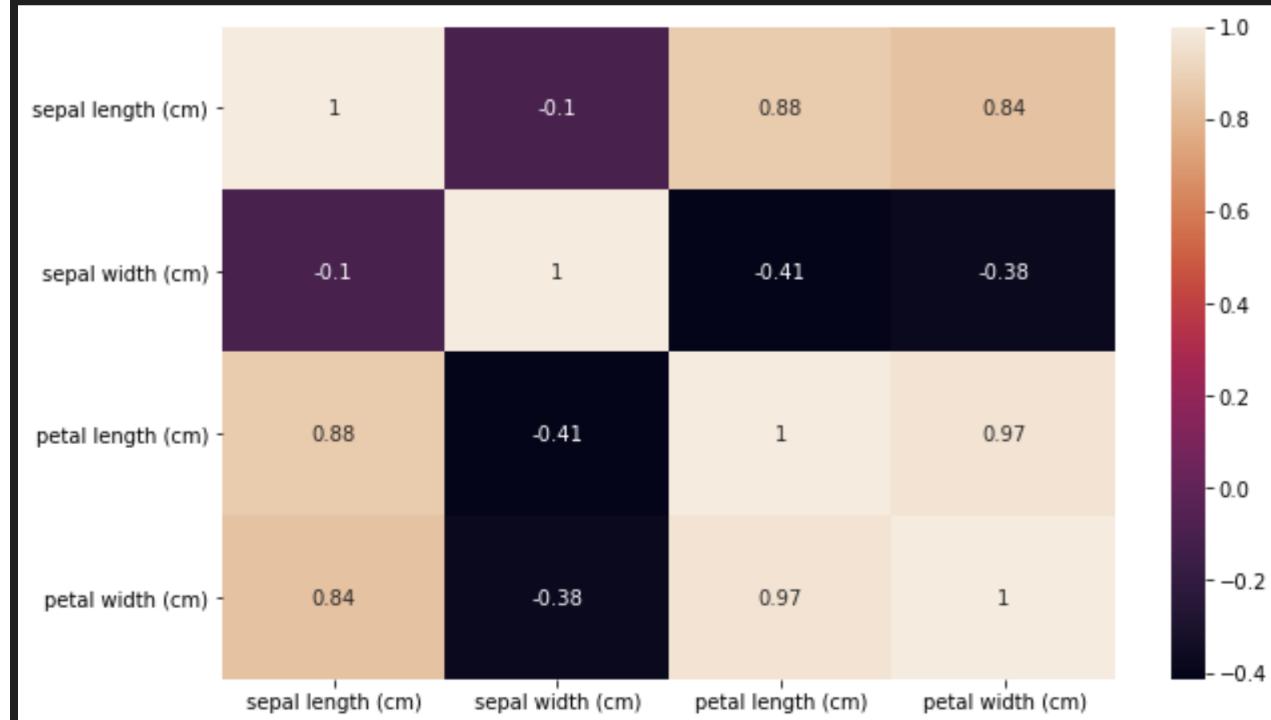
# import as pandas frame
ix, iy = datasets.load_iris(return_X_y = True, as_frame = True)
ix_train, ix_test, iy_train, iy_test = train_test_split(ix, iy, test_size = 0.33)

corr = ix_train.corr()

plt.figure(figsize = (10,6))
sb.heatmap(corr, annot = True)
```

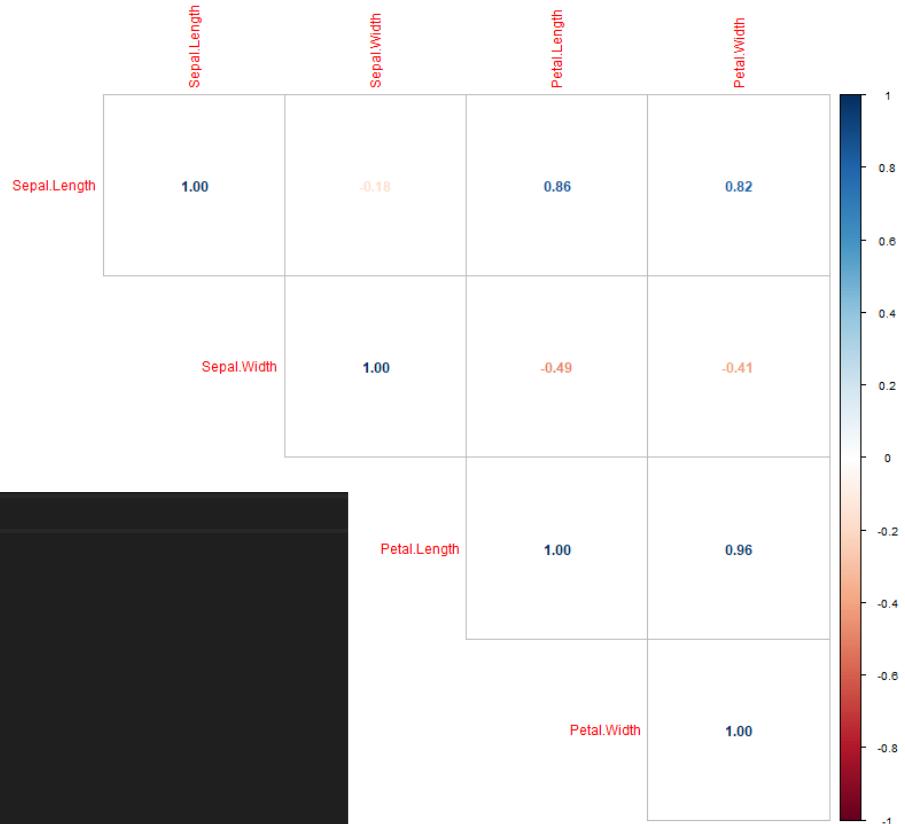
✓ 0.2s

<AxesSubplot:>



R cor and corrplot

```
1 # correlation
2
3 library(corrplot)
4
5 data(iris)
6 dat <- iris
7 classColumn <- 5
8
9 index <- 1:nrow(dat)
10 testindex <- sample(index, trunc(length(index) / 3))
11 testset <- dat[testindex, ]
12 trainset <- dat[-testindex, ]
13
14 # method = "pearson" by default
15 # only the upper half of the matrix is shown
16 corrplot(cor(trainset[, -classColumn]), method = "number", type = "upper")
```



Pima Diabetes feature selection with Python

```
# variance threshold
import pandas as pd
from sklearn.feature_selection import VarianceThreshold
from sklearn.model_selection import train_test_split

data = pd.read_csv("diabetes.csv")

#take first n-1 columns for x and last column for y
dx = data.iloc[:, :-1]
dy = data.iloc[:, -1]

dx_train, dx_test, dy_train, dy_test = train_test_split(dx, dy, test_size = 0.25)

sel = VarianceThreshold(threshold=0)
sel.fit_transform(dx_train)
sel.get_support() # see selected features

array([ True,  True,  True,  True,  True,  True,  True,  True])
```

```
# select K best

from sklearn.feature_selection import SelectKBest, chi2

dx_new = SelectKBest(chi2, k=2).fit_transform(dx_train, dy_train)
dx_new
```

```
array([[126.,    0.],
       [ 83.,    0.],
       [174., 194.],
       ...,
       [ 99.,   18.],
       [116., 105.],
       [118.,    0.]])
```

```
# select percentile

from sklearn.feature_selection import SelectPercentile, chi2

dx_new = SelectPercentile(chi2, percentile=10).fit_transform(dx_train, dy_train)
dx_new
```

```
array([[  0.],
       [  0.],
       [194.],
       [ 23.],
       [ 49.],
       [  0.],
       [145.],
       [110.],
       [237.],
       [  0.],
       [  0.],
       [ 57.]]]
```

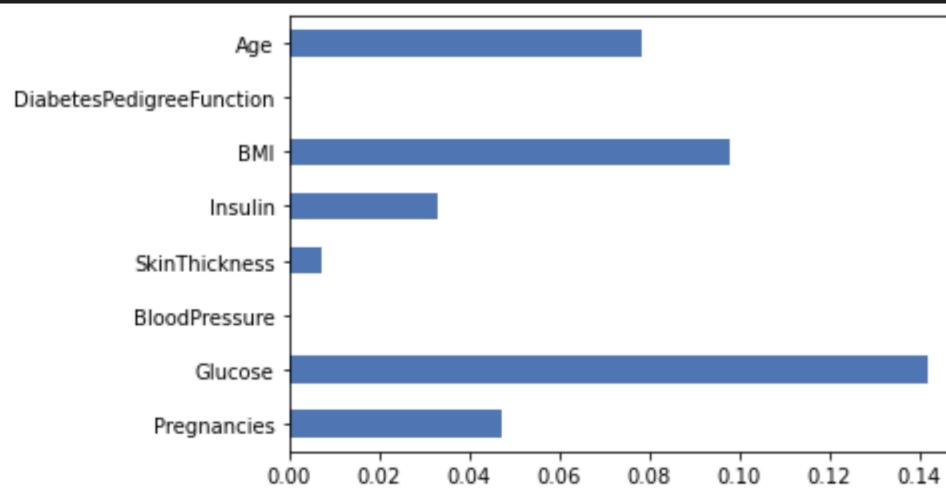
dx_train

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
766	1	126	60	0	0	30.1	0.349 47
720	4	83	86	19	0	29.3	0.317 34
611	3	174	58	22	194	32.9	0.593 36
182	1	0	74	20	23	27.7	0.299 21
134	2	96	68	13	49	21.1	0.647 26
...
57	0	100	88	60	110	46.8	0.962 31
333	12	106	80	0	0	23.6	0.137 44
566	1	99	72	30	18	38.6	0.412 21
527	3	116	74	15	105	26.3	0.107 24
577	2	118	80	0	0	42.9	0.693 21

576 rows × 8 columns

```
# information gain
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_selection import mutual_info_classif

importance = mutual_info_classif(dx_train, dy_train)
feat_importance = pd.Series(importance, dx.columns)
feat_importance.plot(kind = 'barh')
plt.show()
```

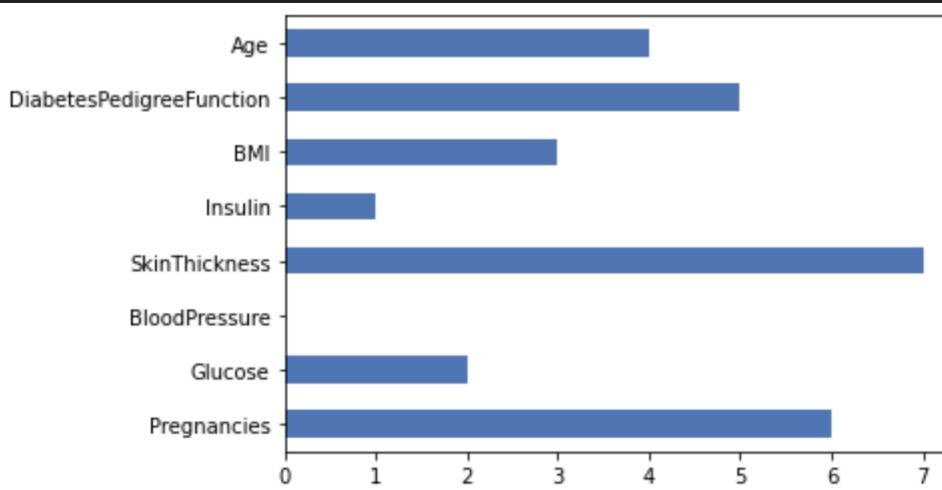


```
# Fisher's score
import pandas as pd
import matplotlib.pyplot as plt
from skfeature.function.similarity_based import fisher_score

dxn = dx_train.to_numpy()
dyn = dy_train.to_numpy()

ranking = fisher_score.fisher_score(dxn, dyn)

feat_importance = pd.Series(ranking, dx_train.columns)
feat_importance.plot(kind = 'barh')
plt.show()
```



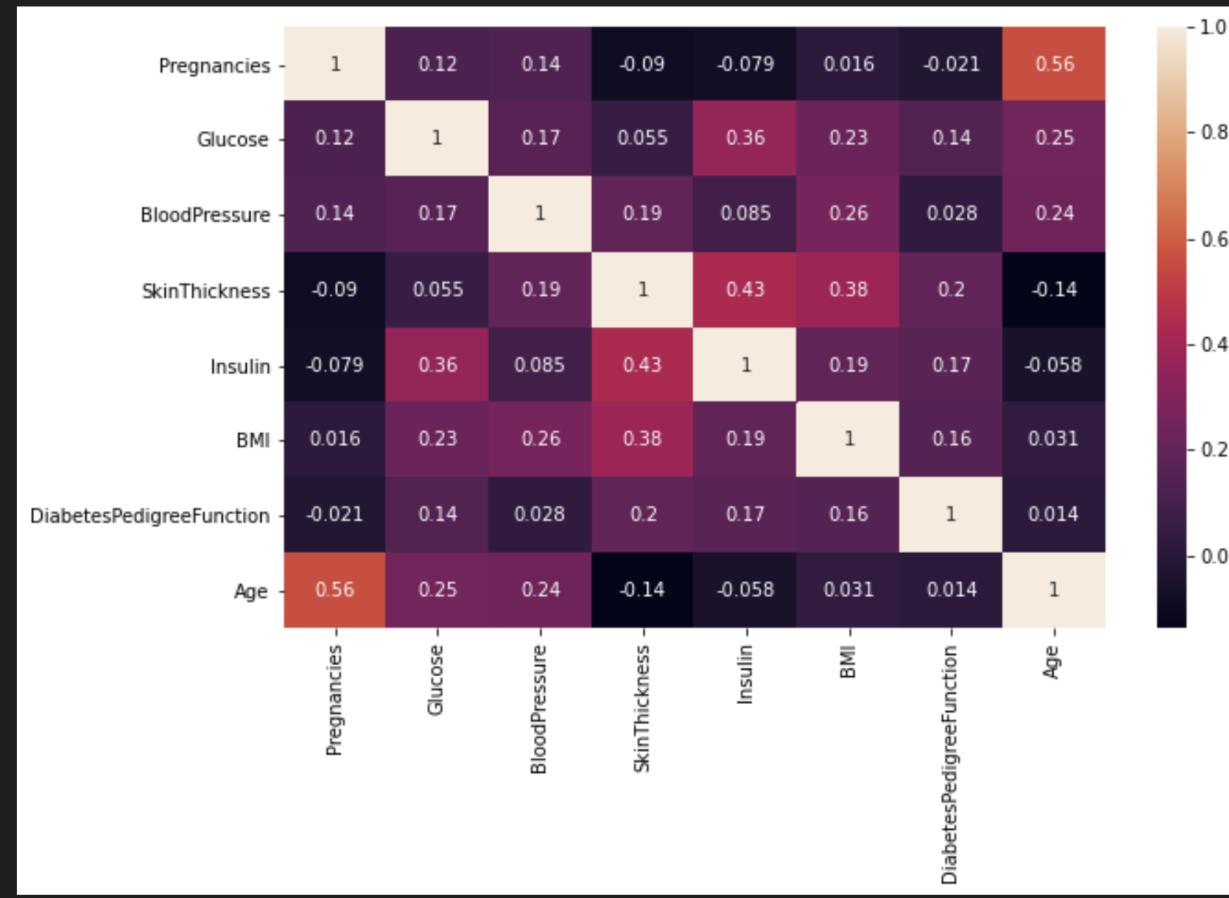
```
# correlation

import seaborn as sb
import matplotlib.pyplot as plt

corr = dx_train.corr()

plt.figure(figsize = (10,6))
sb.heatmap(corr, annot = True)
```

<AxesSubplot:>

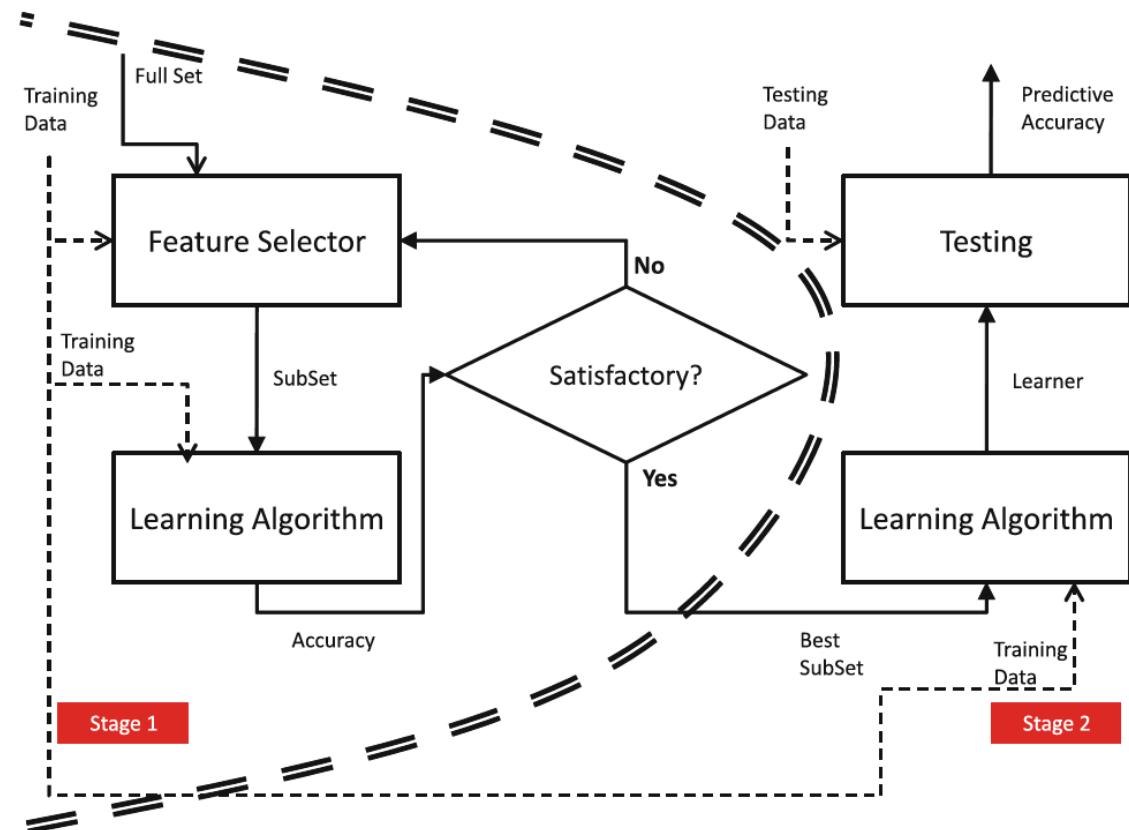


Filter methods - discussion

- The measurement of uncertainty, distances, dependencies or consistency is more economical than computing a model accuracy
 - Filter methods are faster
- They can be applied for very large data sets due to simplicity and low complexity in the evaluation measure

Metode Wrapper

- Use a feature subset and train a model on this selection
- Model-agnostic but model-dependent
- Are usually expensive computationally



<https://web.iitd.ac.in/~bspanda/fs.pdf>

Exhaustive models

- Suppose original data have N features
- If the aim is to select k attributes
- There are $N! / k!(N-k)!$ possibilities
 - How many for $N = 100$ and $k = 5$?
 - Over 75 million possibilities
 - While considered only $k = 5$
 - And this is for just one repetition of the model, but, since it is stochastic and cross-validation is also a must, repeated runs will be performed
- These methods are not practical

Forward feature selection

- Iterative method, begins with an empty set of attributes, or with one or two attributes
- At each iteration a feature is added on the base of the score (from k-fold cross-validation) for a selection criterion
 - Could be accuracy, F1-score, ROC-AUC for classification or MSE, R² for regression
- The selection procedure is Greedy because it evaluates every combination starting with the best ones
- Stopping criterion
 - reaching a given number of desired features
 - or when the model performance can no longer be increased

Python

```
# forward selection

from sklearn.feature_selection import SequentialFeatureSelector
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import svm

data = pd.read_csv("diabetes.csv")

#take first n-1 columns for x and last column for y
dx = data.iloc[:, :-1]
dy = data.iloc[:, -1]

dx_train, dx_test, dy_train, dy_test = train_test_split(dx, dy, test_size = 0.25)

svmm = svm.SVC(kernel='rbf')
svmm.fit(dx_train,dy_train)

dy_pred = svmm.predict(dx_test)
print(accuracy_score(dy_test, dy_pred))
```

8]

· 0.71875

```
# cv - k-fold cross-validation, default 5  
# n_features_to_select, default half of the features  
# default scorer - accuracy  
  
fwdfs = SequentialFeatureSelector(svmm, direction='forward', cv = 5, n_features_to_select = 3)  
dx_train = fwdfs.fit_transform(dx_train,dy_train)
```

```
features = fwdfs.support_  
features
```

```
array([False,  True, False, False, False,  True, False,  True])
```

```
dx_test = fwdfs.transform(dx_test)
```

```
svmm.fit(dx_train,dy_train)  
dy_pred = svmm.predict(dx_test)  
print(accuracy_score(dy_test, dy_pred))
```

```
0.7552083333333334
```

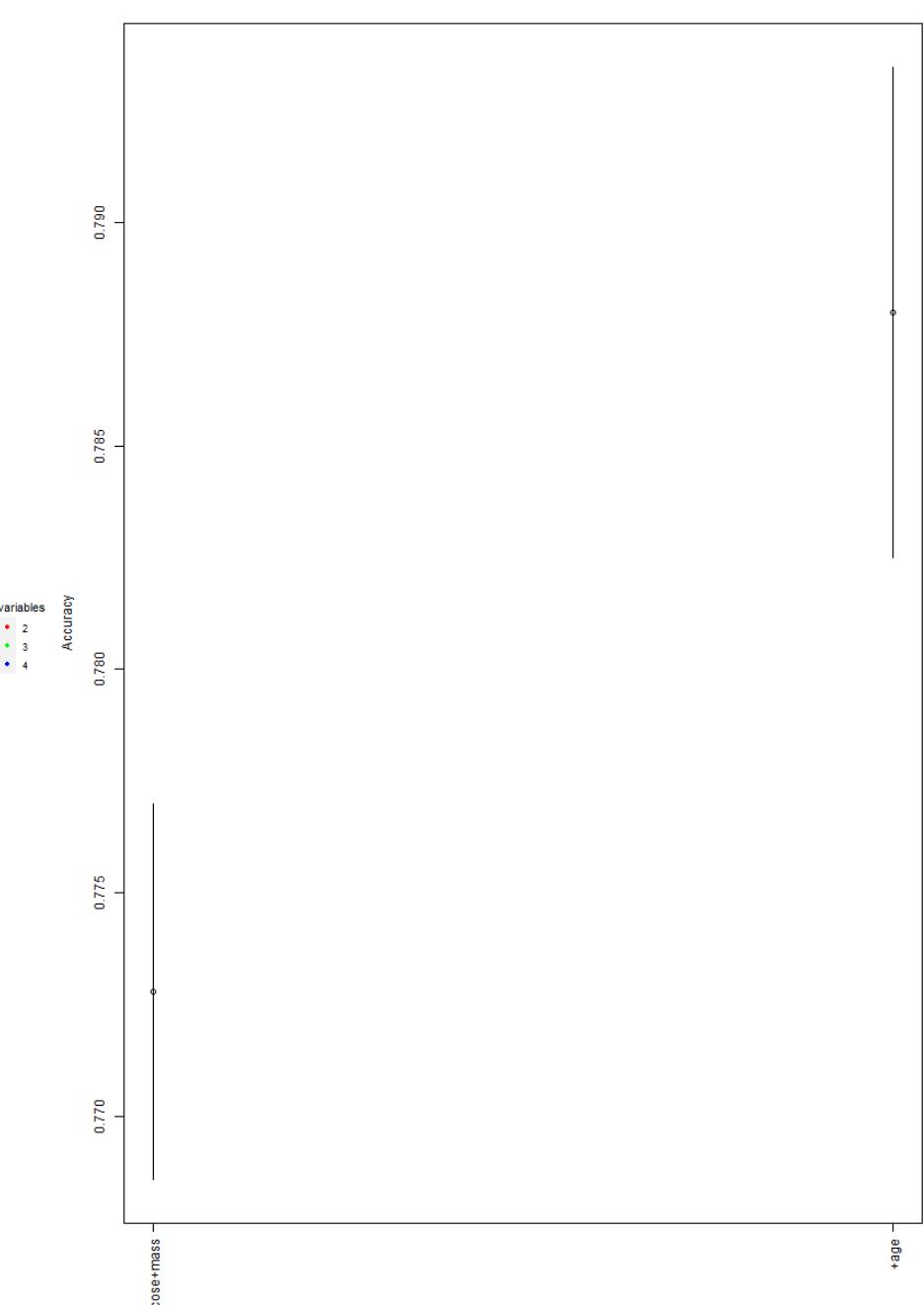
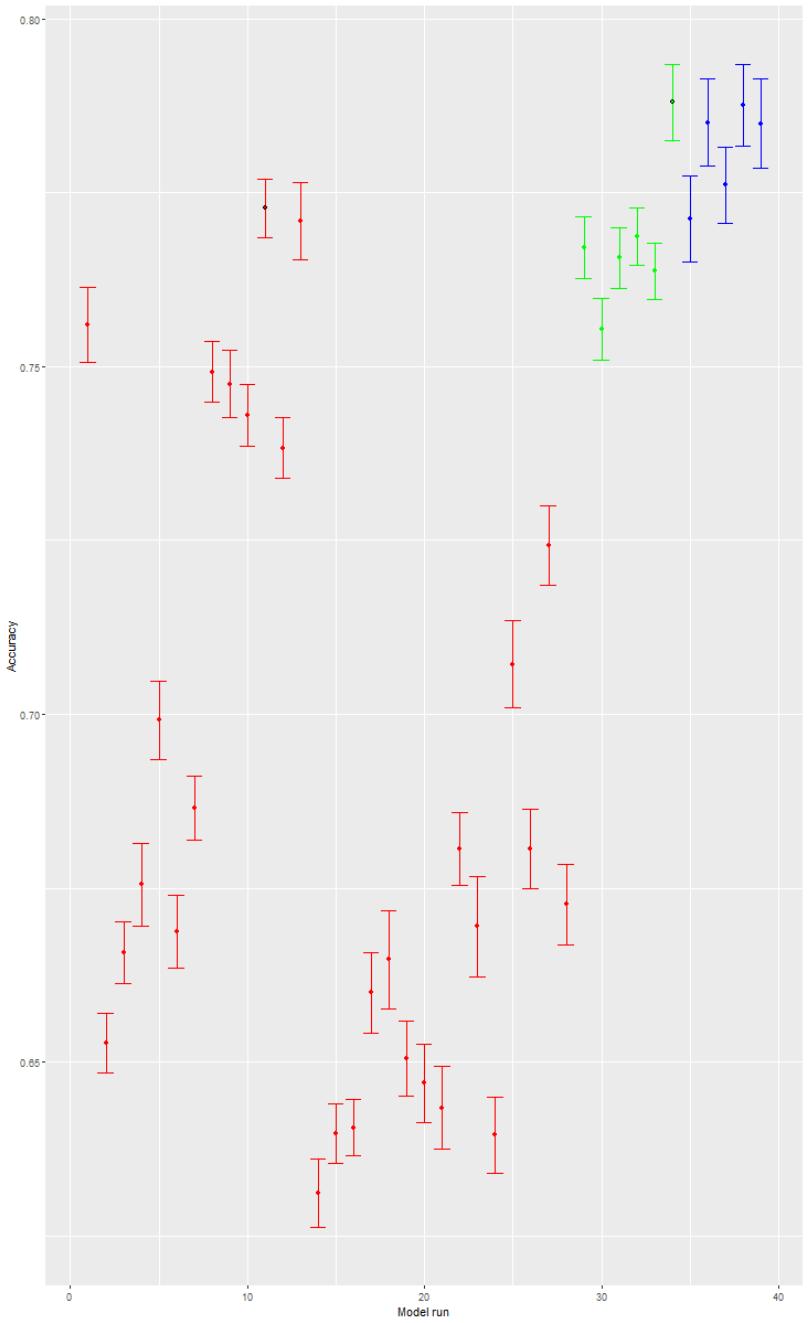
R

```
1 # Forward Feature Selection
2
3 library(mlbench)
4 library(e1071)
5 library(CAST)
6
7 data(PimaIndiansDiabetes)
8 dat <- PimaIndiansDiabetes
9
10 repeats <- 30
11 classColumn <- 9
12
13 index <- 1:nrow(dat)
14
15 testindex <- sample(index, trunc(length(index) / 4))
16 testset <- dat[testindex, ]
17 trainset <- dat[-testindex, ]
18
19 svm_model <- svm(diabetes ~ ., data = trainset, kernel = "radial", cost = 1)
20 svm_pred <- predict(svm_model, testset[, -classColumn])
21 contab <- table(pred = svm_pred, true = testset[, classColumn])
22 accuracyBefore <- classAgreement(contab)$diag
23 print("Accuracy before")
24 print(accuracyBefore)
25
26 ffsmodel <- ffs(trainset[, -classColumn], trainset[, classColumn], method="svmRadial")
27 ffsmodel
28
29 plot_ffs(ffsmodel)
30 plot_ffs(ffsmodel, plotType = "selected")
```

Accuracy and variables selected

```
[1] "Accuracy before"  
[1] 0.71875  
[1] "model using pregnant,glucose will be trained now..."  
[1] "maximum number of models that still need to be trained: 48"  
[1] "model using pregnant,pressure will be trained now..."  
[1] "maximum number of models that still need to be trained: 47"  
[1] "model using pregnant,triceps will be trained now..."  
[1] "maximum number of models that still need to be trained: 46"  
[1] "model using pregnant,insulin will be trained now..."  
[1] "maximum number of models that still need to be trained: 45"  
[1] "model using pregnant,mass will be trained now..."  
[1] "maximum number of models that still need to be trained: 44"  
[1] "model using pregnant,pedigree will be trained now..."  
[1] "maximum number of models that still need to be trained: 43"
```

```
[1] "model using additional variable pressure will be trained now..."  
[1] "maximum number of models that still need to be trained: 13"  
[1] "model using additional variable triceps will be trained now..."  
[1] "maximum number of models that still need to be trained: 12"  
[1] "model using additional variable insulin will be trained now..."  
[1] "maximum number of models that still need to be trained: 11"  
[1] "model using additional variable pedigree will be trained now..."  
[1] "maximum number of models that still need to be trained: 10"  
[1] "vars selected: glucose,mass,age with Accuracy 0.788"  
Note: No increase in performance found using more than 3 variables
```



Backward feature elimination

- Begins with all features and eliminates the least important attributes with respect to the score of the desired metric
- Stopping criterion:
 - score can no longer be improved
 - or one feature remains
 - or the desired number of attributes is attained
- Also a Greedy approach

Python

```
# backward selection

data = pd.read_csv("diabetes.csv")

#take first n-1 columns for x and last column for y
dx = data.iloc[:, :-1]
dy = data.iloc[:, -1]

dx_train, dx_test, dy_train, dy_test = train_test_split(dx, dy, test_size = 0.25)

svmm = svm.SVC(kernel='rbf')
svmm.fit(dx_train,dy_train)

dy_pred = svmm.predict(dx_test)
print(accuracy_score(dy_test, dy_pred))
```

0.7552083333333334

```
fw dbs = SequentialFeatureSelector(svmm, direction='backward', cv = 5)
dx_train = fw dbs.fit_transform(dx_train,dy_train)

features = fw dbs.support_
features
```

array([False, True, True, False, False, True, False, True])

```
svmm.fit(dx_train,dy_train)
dx_test = fw dbs.transform(dx_test)
dy_pred = svmm.predict(dx_test)
print(accuracy_score(dy_test, dy_pred))
```

0.7708333333333334

Stochastic search methods

- Hill climbing, simulated annealing, genetic algorithm
 - Representation can be binary, with the size of the individuals equal to the number of features:
 - 1 = feature is kept, 0 = feature is eliminated

0	1	1	0	1	0	1	0	
Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1

Stochastic search methods

- Hill climbing, simulated annealing, genetic algorithm
 - Representation can be binary, with the size of the individuals equal to the number of features:
 - 1 = feature is kept, 0 = feature is eliminated

0	1	1	0	1	0	1	0	
Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1

Stochastic search methods

- Hill climbing, simulated annealing, genetic algorithm
 - Evaluation can be given by the prediction accuracy on the validation set of a model trained only with the selected features

0	1	1	0	1	0	1	0	
Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1

Generic evolutionary algorithms for feature selection

- Initialization
 1. Generate a population P of N candidate solutions – each solution represents a feature subset
 2. Evaluate every candidate solution, e.g. apply an SVM or NN to be trained on samples with the given attributes
- Repeat until stopping criterion (e.g., a number of iterations)
 1. Generate new candidate solutions starting from P (selection, recombination, mutation etc)
 2. Evaluate each newly created individual
 3. Update P

Genetic algorithm for feature selection

- Every evaluation of a candidate solution of features selected means a training of the ML model
- An economic budget for the number of function evaluations of a GA is around 5000
- A hill climbing or simulated annealing algorithm uses only one candidate solution instead of a population of potential solutions
 - It is often preferred, even if it does not attain solutions as good as those of a GA

Embedded methods

- Similar to wrapper, in the sense that features are selected during model learning
- Reaches a solution faster by avoiding the retraining of the model for each extracted subset of features
- Among the most popular:
 - LASSO
 - Achieves regularization L1, e.g. penalizes the absolute values of the feature coefficients and thus selects the useful features
 - Random Forest
 - Good interpretability

Parameter tuning

Examples of parameters

- Support vector machines
 - Kernel
 - Penalization for errors C
- Neural networks
 - Number of hidden layers
 - Number of neurons in hidden layers
 - Optimizer, activation function
- Decision trees
 - Split criterion
 - Max depth
 - Ensembles: number of estimators, number of random attributes in each tree

Parameter selection techniques

- Manual search
- Random search
- Grid search
- Latin hypercube sampling
- Automatic tuning
 - E.g. based on evolutionary computation



Manual search

- Manual
 - Probably the most used approach to set parameters
 - Appropriate values are looked for empirically, following to improve performance results in validation



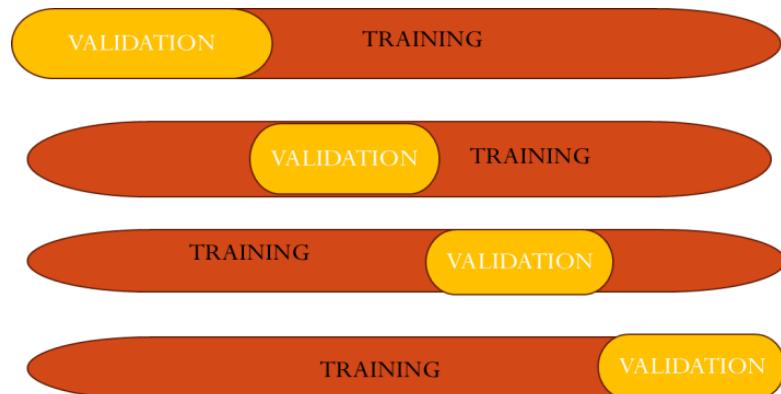
TRAINING

VALIDATION

TEST

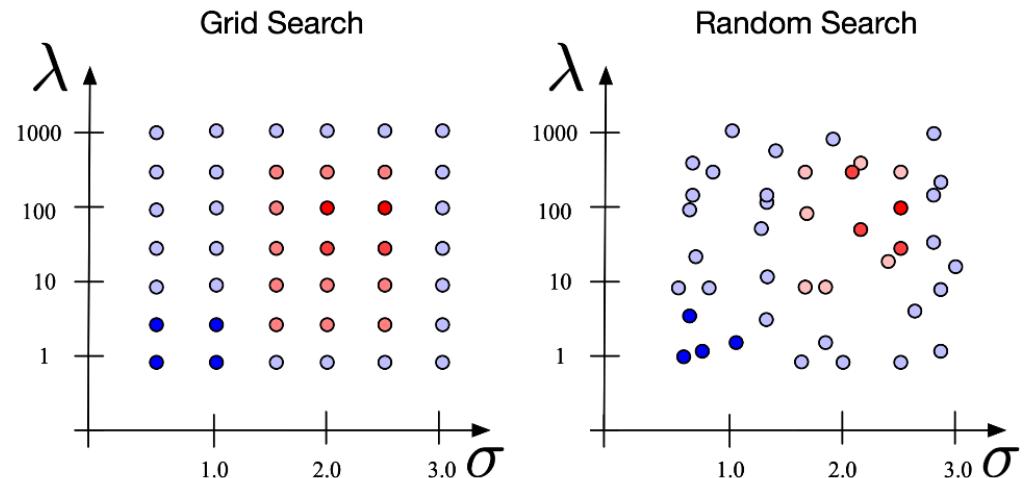
Grid vs random tuning

- Grid
 - Define a grid of values for the parameters to be tuned
 - Run the model with each parameter setting a number of times (e.g. k-fold)
 - Choose the setting with best validation performance



Grid vs random tuning

- Random
 - Generate random values for each parameter and test the validation accuracy as with grid
 - For each parameter there are several distinct values
 - With grid, the same value for a parameter is tested several times



Latin hypercube sampling (LHS)

- A controlled generation of parameter values
 - similar to grid
 - but without repeating values for a parameter
- Covers the search space
- Specify
 - number of configurations to be generated
 - number of dimensions (number of parameters)

```
from scipy.stats import qmc
l_bounds = [0, 2]
u_bounds = [10, 5]

# generate 5 pairs within the bounds
# [0, 10] and [2, 5]
sampler = qmc.LatinHypercube(d=2)
sample = sampler.random(n=5)
qmc.scale(sample, l_bounds, u_bounds)

✓ 0.0s

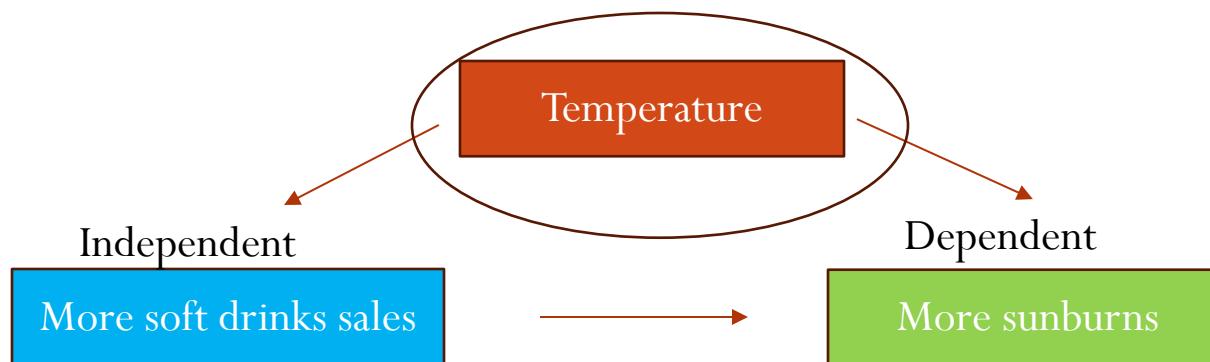
array([[7.79858189, 3.97507706],
       [4.62325307, 3.73304944],
       [8.31379531, 4.50975086],
       [0.89824804, 2.32420778],
       [2.14558817, 2.95678044]])
```

Parameter tuning with evolutionary algorithms

- Establish the options for each parameter in turn
 - Possible set of values or interval
- Choose an adequate representation for the candidate solutions
- Each fitness evaluation for an individual assumes training the model and recording the performance on validation
 - If we perform cross-validation, then the mean validation performance will be recorded
- The best individual will give the parameter configuration to be later used on the test set

Confounding variables

- A variable unaccounted for that affects the relationship between an independent and the dependent variables in the data set
 - Makes it appear that there is a cause-effect relationship between the two
- The confounding variable
 - Is correlated with the independent variable
 - Has a causal relationship with the dependent variable



Homework

- For one problem (Wisconsin, car or customer):
 - Apply 2 filter methods and 1 wrapper technique with one of the models (NN or Bagging)
 - Apply a grid search or LHS for the parameter tuning of the model (NN or Bagging)
- **(Optional)** Study the method Recursive Feature Elimination and implement it on a problem of your choice.
- **(Optional)** Study Lasso regularization and implement it on a problem of your choice.

DON'T MAKE ME
-CHOOSE

