Optimization Techniques for Machine Learning

Alassane KONE
(ISE 2019 – Senior Data Scientist)

February 2023

Chapter 6

Features Selection Optimization

Overview

Feature selection is the process of reducing the number of input variables when developing a predictive model.

Having irrelevant features in your data can decrease the accuracy of many models, especially linear algorithms like linear and logistic regression.

Three benefits of performing feature selection before modeling your data are:

- **Reduces Overfitting**: Less redundant data means less opportunity to make decisions based on noise.
- **Improves Accuracy**: Less misleading data means modeling accuracy improves.
- **Reduces Training Time**: Less data means that algorithms train faster.

Overview

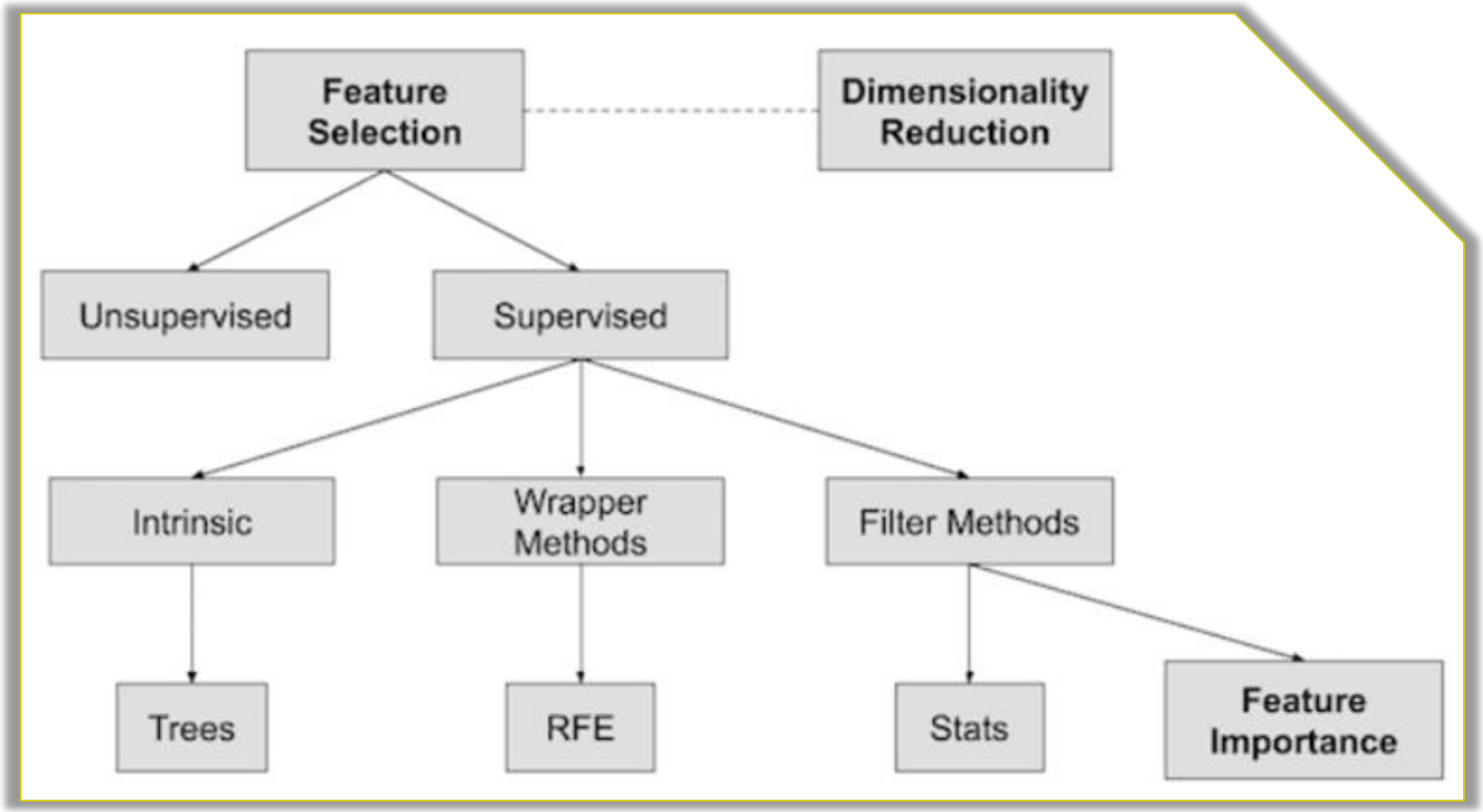
We can summarize feature selection as follows:

- **Feature Selection: Select a subset of input features from the dataset.**
  - Unsupervised: Do not use the target variable (e.g. remove redundant variables).
    - Correlation
  - Supervised: Use the target variable (e.g. remove irrelevant variables).
    - Wrapper: Search for well-performing subsets of features.
      - RFE
    - Filter: Select subsets of features based on their relationship with the target.
      - Statistical Methods
      - Feature Importance Methods
  - Intrinsic: Algorithms that perform automatic feature selection during training.
      - Decision Trees
- **Dimensionality Reduction: Project input data into a lower-dimensional feature space.**

The following image provides a summary of this hierarchy of feature selection techniques:

Overview

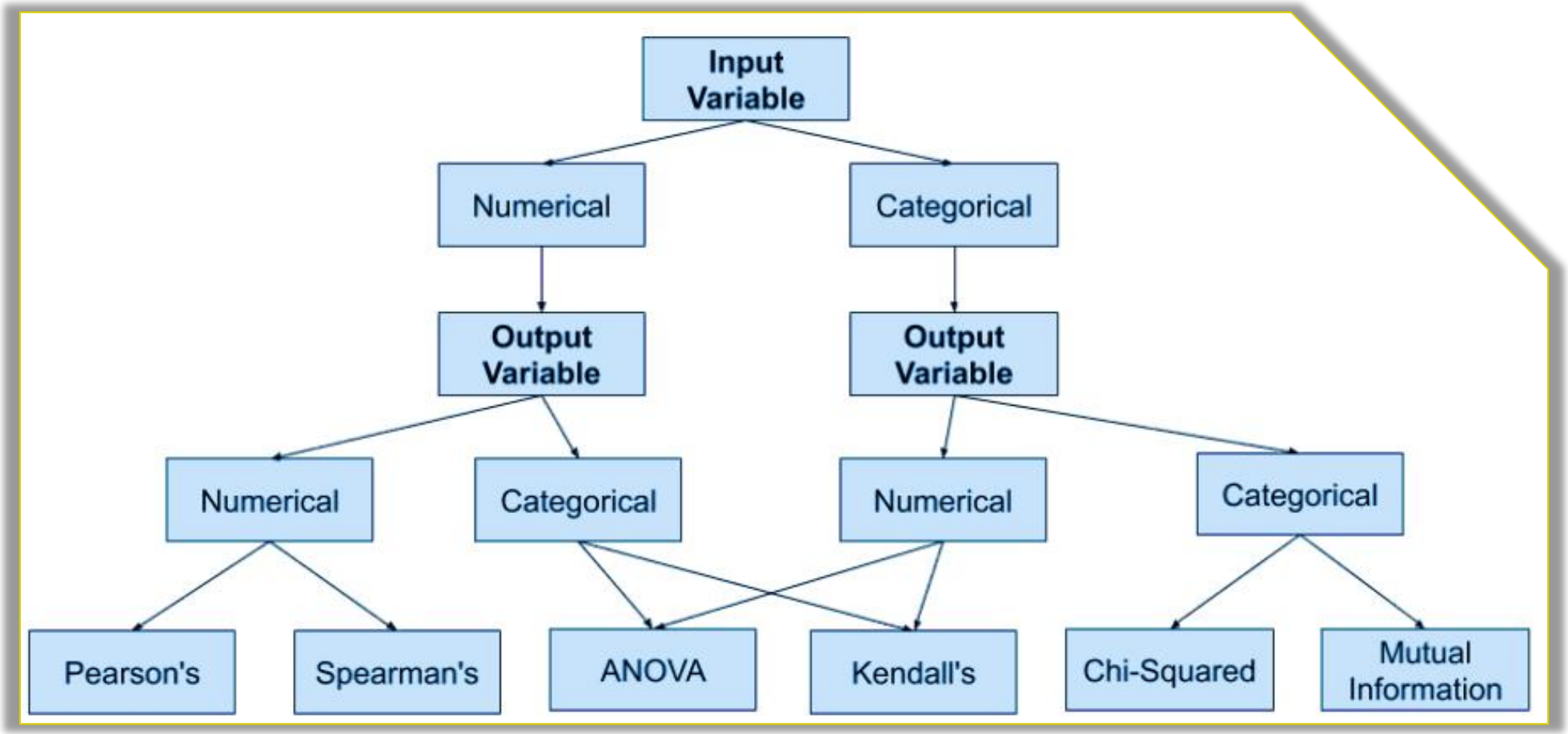
Feature selection Techniques



1. Statistics for Filter-Based Feature Selection Methods

Statisticals Methods

How to choose a statistical method ?



2. Grid Search For Feature Selection

Enumerate All Feature Subsets

When the number of input variables is relatively small and the model evaluation is relatively fast, then it may be possible to enumerate all possible subsets of input variables.

This means evaluating the performance of a model using a test harness given every possible unique group of input variables.

Baseline

```
# evaluate a decision tree on the entire small dataset
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.tree import DecisionTreeClassifier
# define dataset
X, y = make_classification(n_samples=1000, n_features=3, n_informative=2, n_redundant=1, random_state=1)
# define model
model = DecisionTreeClassifier()
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report result
print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

Enumerate All Feature Subsets

Grid Search

```
# determine the number of columns
n_cols = X.shape[1]
best_subset, best_score = None, 0.0
# enumerate all combinations of input features
for subset in product([True, False], repeat=n_cols):
    # convert into column indexes
    ix = [i for i, x in enumerate(subset) if x]
    # check for now column (all False)
    if len(ix) == 0:
        continue
    # select columns
    X_new = X[:, ix]
    # define model
    model = DecisionTreeClassifier()
    # define evaluation procedure
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    # evaluate model
    scores = cross_val_score(model, X_new, y, scoring='accuracy', cv=cv, n_jobs=-1)
    # summarize scores
    result = mean(scores)
    # report progress
    print('>f(%s) = %f ' % (ix, result))
    # check if it is better than the best so far
    if best_score is None or result >= best_score:
        # better result
        best_subset, best_score = ix, result
# report best
print('Done!')
print('f(%s) = %f' % (best_subset, best_score))
```

3. Wrappers Methods: RFE

RFE : Overview

Recursive Feature Elimination, or RFE for short, is a popular feature selection algorithm.

RFE is popular because it is easy to configure and use and because it is effective at selecting those features (columns) in a training dataset that are more or most relevant in predicting the target variable.

There are two important configuration options when using RFE: the choice in the number of features to select and the choice of the algorithm used to help choose features..

RFE works by searching for a subset of features by starting with all features in the training dataset and successfully removing features until the desired number remains.

This is achieved by fitting the given machine learning algorithm used in the core of the model, ranking features by importance, discarding the least important features, and re-fitting the model. This process is repeated until a specified number of features remains.

RFE: Example 1

```
# Feature Extraction with RFE
import pandas as pd
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

# load data

df = pd.read_csv("data_0006.csv")
array = df.values
X = array[:,0:8]
Y = array[:,8]

# feature extraction
model = LogisticRegression(solver='lbfgs')
rfe = RFE(model, 3)
fit = rfe.fit(X, Y)
print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
```

RFE: In a Classification Project

```
# evaluate RFE for classification
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline

# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant=5, random_state=1)
# create pipeline
rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=5)
model = DecisionTreeClassifier()
pipeline = Pipeline(steps=[('s',rfe),('m',model)])
# evaluate model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(pipeline, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
# Prediction with RFE

# fit the model on all available data
pipeline.fit(X, y)
# make a prediction for one example
data =
[[2.56999479,-0.13019997,3.16075093,-4.35936352,-1.61271951,-1.39352057,-2.48924933,-1.93094078,3.2613036
6,2.05692145]]
yhat = pipeline.predict(data)
print('Predicted Class: %d' % (yhat))
```

RFE Hyperparameters: Numbers of features

```
# explore the number of selected features for RFE
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from matplotlib import pyplot

# get the dataset
def get_dataset():
    X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant=5,
random_state=1)
    return X, y

# get a list of models to evaluate
def get_models():
    models = dict()
    for i in range(2, 10):
        rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=i)
        model = DecisionTreeClassifier()
        models[str(i)] = Pipeline(steps=[('s',rfe),('m',model)])
    return models

# evaluate a give model using cross-validation
def evaluate_model(model, X, y):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
    return scores

# define dataset
X, y = get_dataset()
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X, y)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```

RFE Hyperparameters: Base Algorithm

```
# get a list of models to evaluate
def get_models():
    models = dict()
    # lr
    rfe = RFE(estimator=LogisticRegression(), n_features_to_select=5)
    model = DecisionTreeClassifier()
    models['lr'] = Pipeline(steps=[('s',rfe),('m',model)])
    # perceptron
    rfe = RFE(estimator=Perceptron(), n_features_to_select=5)
    model = DecisionTreeClassifier()
    models['per'] = Pipeline(steps=[('s',rfe),('m',model)])
    # cart
    rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=5)
    model = DecisionTreeClassifier()
    models['cart'] = Pipeline(steps=[('s',rfe),('m',model)])
    # rf
    rfe = RFE(estimator=RandomForestClassifier(), n_features_to_select=5)
    model = DecisionTreeClassifier()
    models['rf'] = Pipeline(steps=[('s',rfe),('m',model)])
    # gbm
    rfe = RFE(estimator=GradientBoostingClassifier(), n_features_to_select=5)
    model = DecisionTreeClassifier()
    models['gbm'] = Pipeline(steps=[('s',rfe),('m',model)])
    return models
```

RFE: Automatically Select the Number of Features

It is also possible to automatically select the number of features chosen by RFE.

This can be achieved by performing cross-validation evaluation of different numbers of features as we did in the previous section and automatically selecting the number of features that resulted in the best mean score.

The **RFECV class** implements this for us.

The RFECV is configured just like the RFE class regarding the choice of the algorithm that is wrapped. Additionally, the minimum number of features to be considered can be specified via the "min_features_to_select" argument (defaults to 1).

```
from sklearn.feature_selection import RFECV

# automatically choose the number of features
rfe = RFECV(estimator=DecisionTreeClassifier())

##
```