Feature Selection

Tushar B. Kute,
http://tusharkute.com

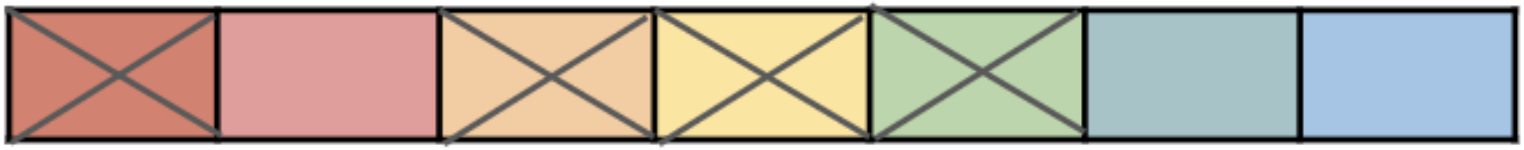
Feature Selection

- The training time and performance of a machine learning algorithm depends heavily on the features in the dataset. Ideally, we should only retain those features in the dataset that actually help our machine learning model learn something.
- Unnecessary and redundant features not only slow down the training time of an algorithm, but they also affect the performance of the algorithm. The process of selecting the most suitable features for training the machine learning model is called "feature selection".

Feature Selection

All Features

Feature Selection

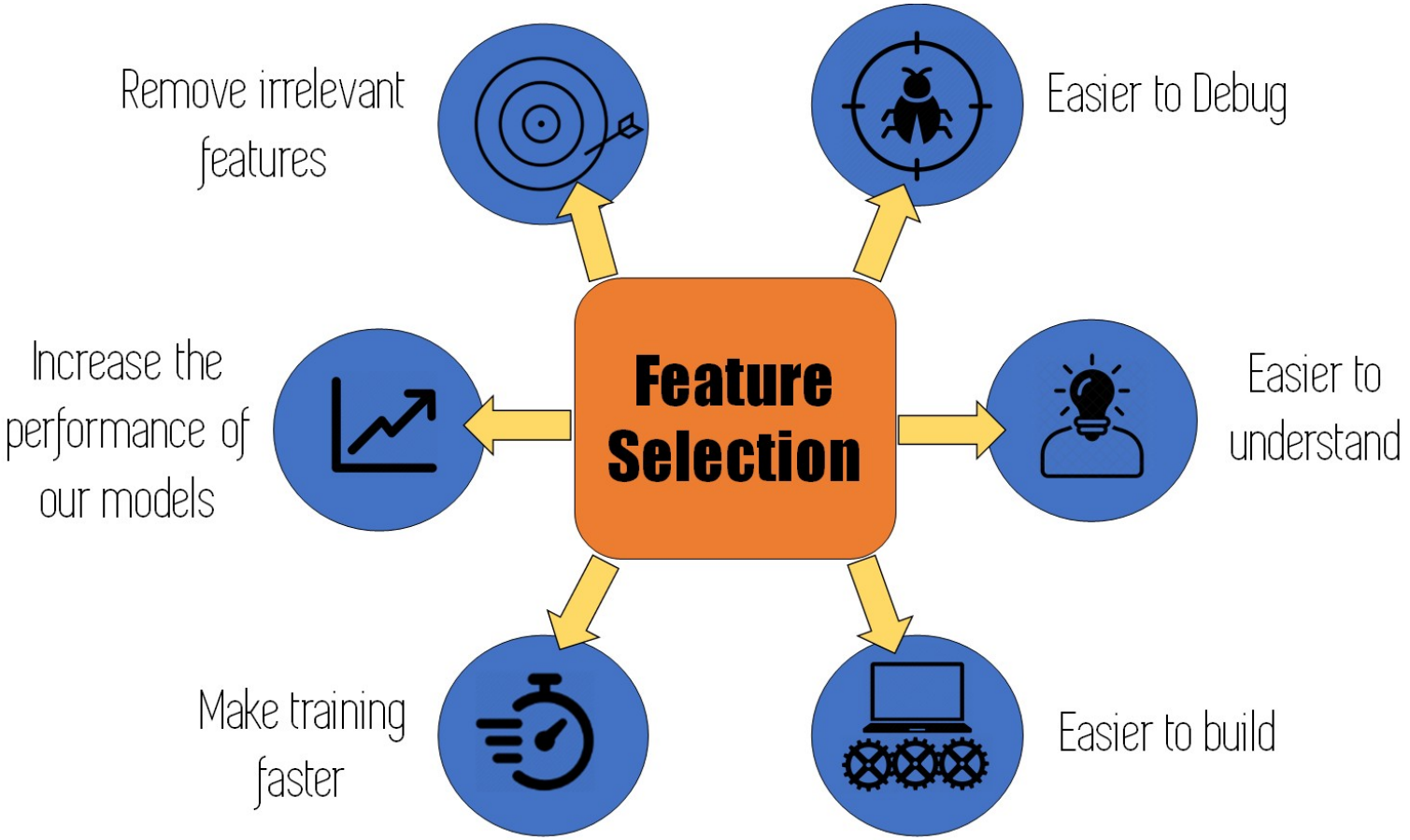


Final Features

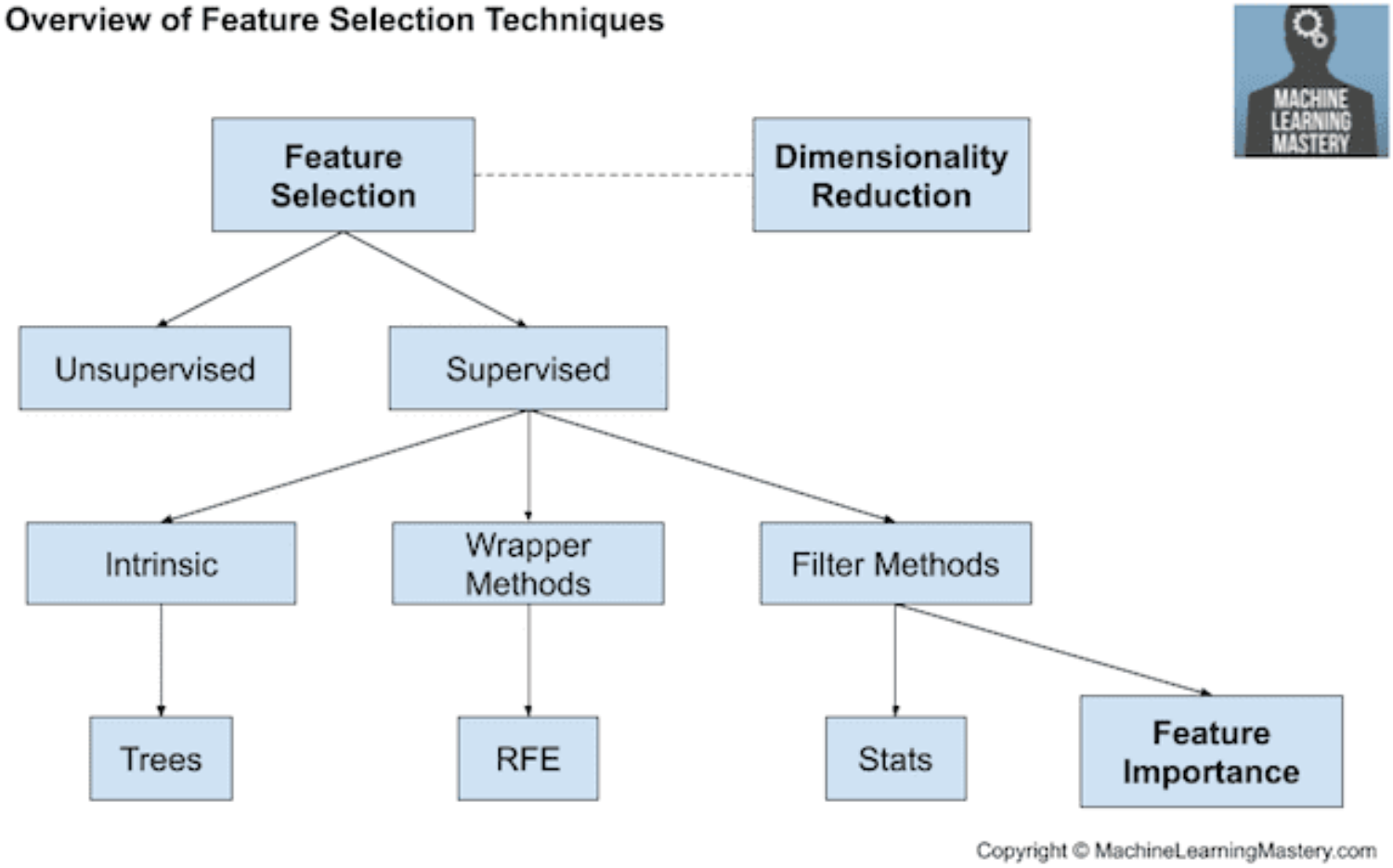
Why feature selection ?

- There are several advantages of performing feature selection before training machine learning models, some of which have been enlisted below:
  - Models with less number of features have higher explainability
  - It is easier to implement machine learning models with reduced features
  - Fewer features lead to enhanced generalization which in turn reduces overfitting
  - Feature selection removes data redundancy
  - Training time of models with fewer features is significantly lower
  - Models with fewer features are less prone to errors

Why feature selection ?



Types


Overview of Feature Selection Techniques
Feature Selection
Dimensionality Reduction
Unsupervised
Supervised
Intrinsic
Wrapper Methods
Filter Methods
Trees
RFE
Stats
Feature Importance
Copyright © MachineLearningMastery.com

Filter Methods

- Filter method relies on the general uniqueness of the data to be evaluated and pick feature subset, not including any mining algorithm.
- Filter method uses the exact assessment criterion which includes distance, information, dependency, and consistency.
- The filter method uses the principal criteria of ranking technique and uses the rank ordering method for variable selection.

Filter Methods

- Filters methods belong to the category of feature selection methods that select features independently of the machine learning algorithm model. This is one of the biggest advantages of filter methods.
- Features selected using filter methods can be used as an input to any machine learning models.
- Another advantage of filter methods is that they are very fast. Filter methods are generally the first step in any feature selection pipeline.
- They are broadly categorized into two categories:
  - Univariate Filter Methods
  - Multivariate Filter Methods.

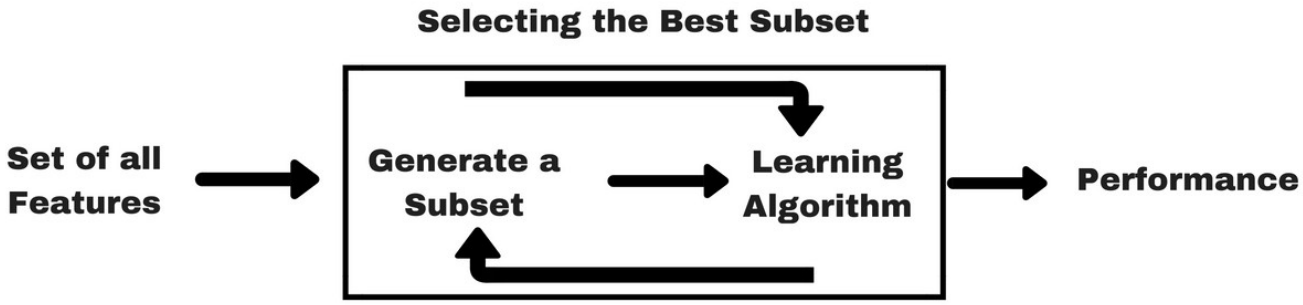
Univariate Filter Methods

- The univariate filter methods are the type of methods where individual features are ranked according to specific criteria.
- The top N features are then selected. Different types of ranking criteria are used for univariate filter methods, for example fisher score, mutual information, and variance of the feature.
- One of the major disadvantage of univariate filter methods is that they may select redundant features because the relationship between individual features is not taken into account while making decisions.
- Univariate filter methods are ideal for removing constant and quasi-constant features from the data.

Multivariate Filter Methods

- Multivariate filter methods are capable of removing redundant features from the data since they take the mutual relationship between the features into account.
- Multivariate filter methods can be used to remove duplicate and correlated features from the data.

Wrapper Methods



- A wrapper method needs one machine learning algorithm and uses its performance as evaluation criteria.
- This method searches for a feature which is best-suited for the machine learning algorithm and aims to improve the mining performance.
- To evaluate the features, the predictive accuracy used for classification tasks and goodness of cluster is evaluated using clustering.

Common Filter Methods

- Remove constant features
- Remove quasi-constant features
- Remove duplicate features and
- Remove correlated features

Remove Constant Features

- Constant features are the type of features that contain only one value for all the outputs in the dataset.
- Constant features provide no information that can help in classification of the record at hand. Therefore, it is advisable to remove all the constant features from the dataset.
- Let's see how we can remove constant features from a dataset. The dataset that we are going to use for this example is the Santandar Customer Satisfaction dataset, that can be downloaded from Kaggle.

Let's Code

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import VarianceThreshold
```

```
df = pd.read_csv('satandar.csv', nrows=40000)
df.shape
```

```
(40000, 371)
```

```
df.columns;
```

```
X_train, X_test, y_train, y_test =train_test_split(
    df.drop(labels=['TARGET'], axis=1),
    df['TARGET'],
    test_size=0.2,
    random_state=0)
```

Variance Threshold

- VarianceThreshold(threshold=0.0)
  - Feature selector that removes all low-variance features.
  - threshold
    - Features with a training-set variance lower than this threshold will be removed.
    - The default is to keep all features with non-zero variance, i.e. remove the features that have the same value in all samples.

Variance Threshold

```
constant_filter = VarianceThreshold(threshold=0)
```

```
constant_filter.fit(X_train)
```

```
VarianceThreshold(threshold=0)
```

```
len(X_train.columns[constant_filter.get_support()])
```

```
314
```

```
constant_columns = [column for column in X_train.columns
   if column not in X_train.columns[constant_filter.get_support()]]

print(len(constant_columns))
```

```
56
```

Constant Columns

```
for column in constant_columns:
    print(column)
```

```
ind_var2_0
ind_var2
ind_var18_0
ind_var18
ind_var27_0
ind_var28_0
ind_var28
ind_var27
ind_var34_0
ind_var34
ind_var41
ind_var46_0
ind_var46
```

Extracted Features

```
X_train = constant_filter.transform(X_train)
X_test = constant_filter.transform(X_test)

X_train.shape, X_test.shape
```

```
((32000, 314), (8000, 314))
```

Removing Quasi-Constant features

- Quasi-constant features, as the name suggests, are the features that are almost constant. In other words, these features have the same values for a very large subset of the outputs.
- Such features are not very useful for making predictions. There is no rule as to what should be the threshold for the variance of quasi-constant features.
- However, as a rule of thumb, remove those quasi-constant features that have more than 99% similar values for the output observations.
- We create a quasi-constant filter with the help of VarianceThreshold function. However, instead of passing 0 as the value for the threshold parameter, we will pass 0.01, which means that if the variance of the values in a column is less than 0.01, remove that column.
- In other words, remove feature column where approximately 99% of the values are similar.

Applying Quasi-constant feature

```
qconstant_filter = VarianceThreshold(threshold=0.01)
```

```
qconstant_filter.fit(X_train)
```

```
VarianceThreshold(threshold=0.01)
```

```
len(X_train.columns[qconstant_filter.get_support()])
```

```
260
```

```
qconstant_columns = [column for column in X_train.columns
    if column not in X_train.columns[qconstant_filter.get_support()]]
```

```
print(len(qconstant_columns))
```

```
54
```

Quasi-Constant Columns

```
for column in qconstant_columns:
    print(column)
```

```
ind_var1
ind_var6_0
ind_var6
ind_var13_largo
ind_var13_medio_0
ind_var13_medio
ind_var14
ind_var17_0
ind_var17
ind_var19
ind_var20_0
ind_var20
ind_var29_0
ind_var29
ind_var30_0
ind_var31_0
ind_var31
```

```
X_train = qconstant_filter.transform(X_train)
X_test = qconstant_filter.transform(X_test)

X_train.shape, X_test.shape
```

```
((32000, 260), (8000, 260))
```

Remove duplicate features

- Duplicate features are the features that have similar values.
- Duplicate features do not add any value to algorithm training, rather they add overhead and unnecessary delay to the training time.
- Therefore, it is always recommended to remove the duplicate features from the dataset before training.
- Removing duplicate columns can be computationally costly since we have to take the transpose of the data matrix before we can remove duplicate features.

Let's Code

```
# Remove duplicates
df = pd.read_csv('satandar.csv', nrows=40000)
df.shape
```

```
(40000, 371)
```

```
X_train, X_test, y_train, y_test =train_test_split(
    df.drop(labels=['TARGET'], axis=1),
    df['TARGET'],
    test_size=0.2,
    random_state=0)
```

```
train_features_T = train_features.T
train_features_T.shape
```

```
(370, 32000)
```

Unique and duplicated features

```
X_train_T = X_train.T
X_train_T.shape
```

```
(370, 32000)
```

```
print(X_train_T.duplicated().sum())
```

```
85
```

```
unique_features = X_train_T.drop_duplicates(keep='first').T
```

```
unique_features.shape
```

```
(32000, 285)
```

Get the duplicated features

```
duplicated_features = [dup_col for dup_col in X_train.columns
                       if dup_col not in unique_features.columns]
duplicated_features
```

```
['ind_var2',
 'ind_var13_medio',
 'ind_var18_0',
 'ind_var18',
 'ind_var26',
 'ind_var25',
 'ind_var27_0',
 'ind_var28_0',
 'ind_var28',
 'ind_var27',
 'ind_var29_0',
 'ind_var29',
 'ind_var32',
 'ind_var34_0',
 'ind_var34',
```

Remove correlated features

- A dataset can also contain correlated features. Two or more than two features are correlated if they are close to each other in the linear space.
- Take the example of the feature set for a fruit basket, the weight of the fruit basket is normally correlated with the price. The more the weight, the higher the price.
- Correlation between the output observations and the input features is very important and such features should be retained.
- However, if two or more than two features are mutually correlated, they convey redundant information to the model and hence only one of the correlated features should be retained to reduce the number of features.

Let's Code

```
# Remove correlated features
df = pd.read_csv('paribas.csv', nrows=20000)
# Downloaded from Kaggle
df.shape
```

```
(20000, 133)
```

```
cols = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
# Retain only numeric columns
numerical_columns = list(df.select_dtypes(include=cols).columns)
df = df[numerical_columns]
```

```
df.shape
```

```
(20000, 114)
```

Find Correlations

```
X_train, X_test, y_train, y_test =train_test_split(
    df.drop(['target','ID'], axis=1),
    df['target'],
    test_size=0.2,
    random_state=0)
```

```
correlated_features = set()
correlation_matrix = df.corr()
```

```
correlation_matrix.head()
```

	ID	target	v1	v2	v4	v5	v6
ID	1.000000	0.005830	0.004817	0.001134	-0.003499	0.002015	-0.010675
target	0.005830	1.000000	-0.016746	0.042634	0.054891	0.009453	0.043471
v1	0.004817	-0.016746	1.000000	-0.205826	-0.145037	-0.049337	-0.020251

Correlation above 80%

```
# Column name will correlation > 80%
for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > 0.8:
            colname = correlation_matrix.columns[i]
            correlated_features.add(colname)
```

```
len(correlated_features)
```

```
55
```

```
print(correlated_features)
```

```
{'v67', 'v46', 'v73', 'v118', 'v77', 'v121', 'v64', 'v12', 'v114'
03', 'v87', 'v100', 'v109', 'v76', 'v84', 'v124', 'v44', 'v63', '
'v25', 'v95', 'v104', 'v96', 'v122', 'v123', 'v101', 'v60', 'v68'
```

Drop correlated features

```
X_train.drop(labels=correlated_features, axis=1, inplace=True)
X_test.drop(labels=correlated_features, axis=1, inplace=True)
```

```
X_train.shape
```

```
(16000, 57)
```

```
X_train.head()
```

	v1	v2	v4	v5	v6	v7	v8	v9
17815	1.573723	4.744721	3.731608	10.831683	1.474480	1.720226	1.852197	8.846154
18370	0.896505	10.448345	6.167674	9.769273	2.146071	2.969885	1.450758	9.478909
1379	1.867116	10.866761	3.853017	9.191264	2.169892	3.061395	2.301629	7.830188

Select K best

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

```
# Feature extraction
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, y)
```

```
# Summarize scores
np.set_printoptions(precision=3)
print(fit.scores_)
```

```
[ 111.52   1411.887    17.605    53.108 2175.565   127.669     5.393
04]
```

```
features = fit.transform(X)
# Summarize selected features
print(features[0:5,:])
```

Recursive Feature Elimination

- The Recursive Feature Elimination (or RFE) works by recursively removing attributes and building a model on those attributes that remain.
- It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute.

RFE

```
# Wrapper Method
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
```

```
model = LogisticRegression()
```

```
rfe = RFE(model, 3)
fit = rfe.fit(X, y)
```

RFE

```
print("Num Features: %s" % (fit.n_features_))
print("Selected Features: %s" % (fit.support_))
print("Feature Ranking: %s" % (fit.ranking_))
```

```
Num Features: 3
Selected Features: [ True False False False False  True
Feature Ranking: [1 2 4 5 6 1 1 3]
```

```
X.iloc[:,fit.support_].head()
```

	TimesPregnant	BMI	DiabetesFunct
0	6	33.6	0.627
1	1	26.6	0.351

Comparing

Filter methods	Wrapper methods	Embedded methods
Generic set of methods which do not incorporate a **specific machine learning algorithm.**	Evaluates on a **specific machine learning algorithm** to find optimal features.	Embeds (fix) features during **model building process.** Feature selection is done by observing each iteration of model training phase.
Much **faster** compared to Wrapper methods in terms of time complexity	**High computation time** for a dataset with many features	Sits **between Filter methods and Wrapper methods** in terms of time complexity
Less prone to **over-fitting**	High chances of **over-fitting** because it involves training of machine learning models with different combination of features	Generally used to reduce **over-fitting** by **penalizing** the coefficients of a model being too large.
Examples – **Correlation, Chi-Square test, ANOVA, Information gain** etc.	Examples - **Forward Selection, Backward elimination, Stepwise selection** etc.	Examples - **LASSO, Elastic Net, Ridge Regression** etc.

Useful resources

- https://stackabuse.com
- https://datacamp.com
- https://scikit-learn.org
- www.towardsdatascience.com
- www.medium.com

Thank you

This presentation is created using LibreOffice Impress 5.1.6.2, can be used freely as per GNU General Public License

@mitu_skillologies

/mITuSkillologies

@mitu_group

/company/mitu-skillologies

MITUSkillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

contact@mitu.co.in

tushar@tusharkute.com