AASD 4000
Machine Learning - I

Applied AI Solutions Developer Program

Module 06
Feature Selection

Vejey Gandyer

Agenda

Feature Selection

Importance of Feature Selection

Feature Selection Checklist

Filter Feature Selection Methods

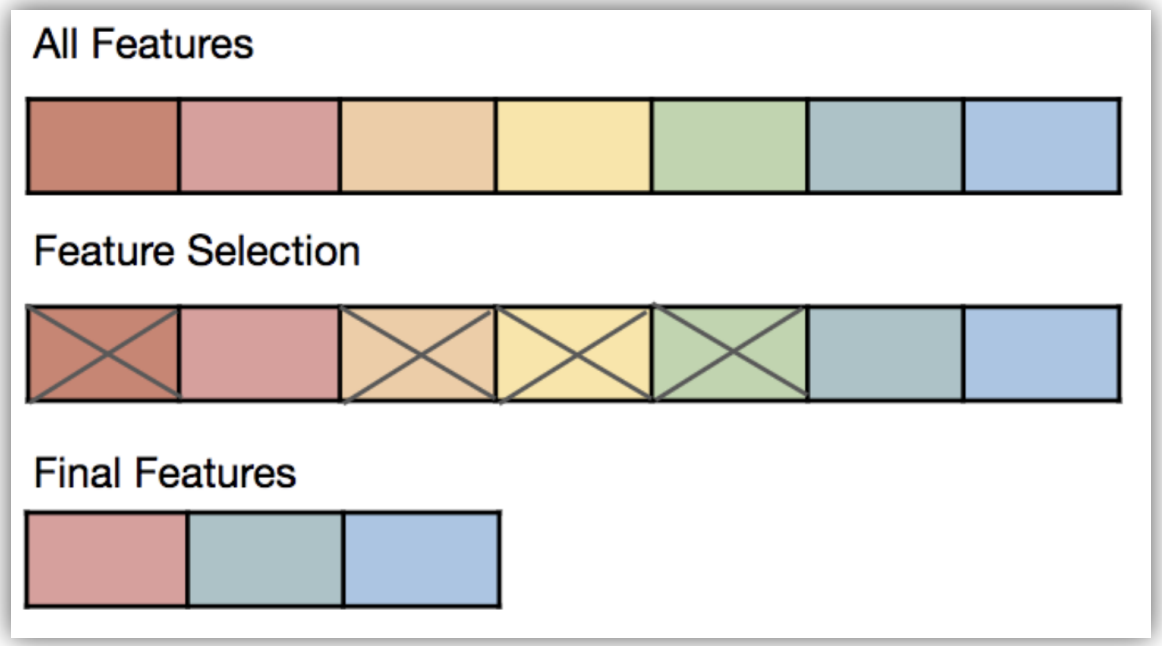
Wrapper Feature Selection Methods

Embedded Feature Selection Methods

Feature Selection

What is it?

What is Feature Selection ?



When building a machine learning model in real-life, it's almost rare that all the variables in the dataset are useful to build a model.

Feature selection is the process of finding and selecting the **most useful features** in a dataset.

Importance of Feature Selection

Why it is needed?

Importance of Feature Selection

Faster Training

Fewer features enables the machine learning algorithm to train **faster**

Occam's Razor

Fewer features reduces the **complexity** of a model and makes it easier to **interpret**

Garbage-in Garbage-out

Fewer and important features improves the **accuracy** of a model if the right subset is chosen

Curse of dimensionality

Fewer features reduces **overfitting**

Feature Selection Checklist

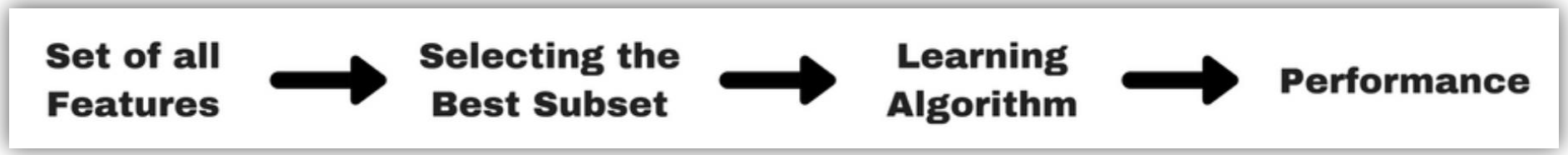
Source: https://jmlr.csail.mit.edu/papers/volume3/guyon03a/guyon03a.pdf

1. **Do you have domain knowledge**? If yes, construct a better set of ad hoc”” features
2. **Are your features commensurate**? If no, consider normalizing them.
3. **Do you suspect interdependence of features**? If yes, expand your feature set by constructing conjunctive features or products of features, as much as your computer resources allow you.
4. **Do you need to prune the input variables** (e.g. for cost, speed or data understanding reasons)? If no, construct disjunctive features or weighted sums of feature
5. **Do you need to assess features individually** (e.g. to understand their influence on the system or because their number is so large that you need to do a first filtering)? If yes, use a variable ranking method; else, do it anyway to get baseline results.
6. **Do you need a predictor**? If no, stop

1. **Do you suspect your data is “dirty”** (has a few meaningless input patterns and/or noisy outputs or wrong class labels)? If yes, detect the outlier examples using the top ranking variables obtained in step 5 as representation; check and/or discard them.
2. **Do you know what to try first**? If no, use a linear predictor. Use a forward selection method with the “probe” method as a stopping criterion or use the 0-norm embedded method for comparison, following the ranking of step 5, construct a sequence of predictors of same nature using increasing subsets of features. Can you match or improve performance with a smaller subset? If yes, try a non-linear predictor with that subset.
3. **Do you have new ideas, time, computational resources, and enough examples**? If yes, compare several feature selection methods, including your new idea, correlation coefficients, backward selection and embedded methods. Use linear and non-linear predictors. Select the best approach with model selection
4. **Do you want a stable solution** (to improve performance and/or understanding)? If yes, subsample your data and redo your analysis for several “bootstrap”.

Filter Feature Selection Methods

Filter Methods



Relies only on the **data** to be evaluated and not on the machine learning algorithm

Assessment criterion : distance, information, dependency, and consistency

Uses **ranking** technique and uses the rank ordering method for variable selection

Features give rank based on statistical scores

- determine features' correlation with the outcome variable

Filter Feature Selection Methods

Pearson Correlation

Pearson Correlation

$$r = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sqrt{\sum (x - \bar{x})^2 \sum (y - \bar{y})^2}}$$

Check the **absolute value** of Pearson's correlation between the **target** and **numerical features** in the dataset

Keep the **top n** features that satisfies this criterion

Pearson Correlation

```
def cor_selector(X, y,num_feats):
    cor_list = []
    feature_name = X.columns.tolist()
    # calculate the correlation with y for each feature
    for i in X.columns.tolist():
        cor = np.corrcoef(X[i], y)[0, 1]
        cor_list.append(cor)
    # replace NaN with 0
    cor_list = [0 if np.isnan(i) else i for i in cor_list]
    # feature name
    cor_feature = X.iloc[:,np.argsort(np.abs(cor_list))[-num_feats:]].columns.tolist()
    # feature selection? 0 for not select, 1 for select
    cor_support = [True if i in cor_feature else False for i in feature_name]
    return cor_support, cor_feature
cor_support, cor_feature = cor_selector(X, y,num_feats)
print(str(len(cor_feature)), 'selected features')
```

Filter Feature Selection Methods

Chi-Squared

Chi-Squared

$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Check the **chi-square metric** between the **target** and **numerical features** in the dataset

Select the feature that has **maximum** chi-squared values

Categorical features

Expected Frequency > 5

Chi-Squared

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.preprocessing import MinMaxScaler
X_norm = MinMaxScaler().fit_transform(X)
chi_selector = SelectKBest(chi2, k=num_feats)
chi_selector.fit(X_norm, y)
chi_support = chi_selector.get_support()
chi_feature = X.loc[:,chi_support].columns.tolist()
print(str(len(chi_feature)), 'selected features')
```

Case Study: Mobile Price Prediction

https://www.kaggle.com/iabhishekofficial/mobile-price-classification#train.csv

```python
import pandas as pd
import numpy as np
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
data = pd.read_csv("D://Blogs//train.csv")
X = data.iloc[:,0:20]  #independent columns
y = data.iloc[:,-1]    #target column i.e price range
#apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores.nlargest(10,'Score'))  #print 10 best features
```

	Specs	Score
13	ram	931267.519053
11	px_height	17363.569536
0	battery_power	14129.866576
12	px_width	9810.586750
8	mobile_wt	95.972863
6	int_memory	89.839124
15	sc_w	16.480319
16	talk_time	13.236400
4	fc	10.135166
14	sc_h	9.614878

Filter Feature Selection Methods

Mutual Information Gain

Mutual Information Gain

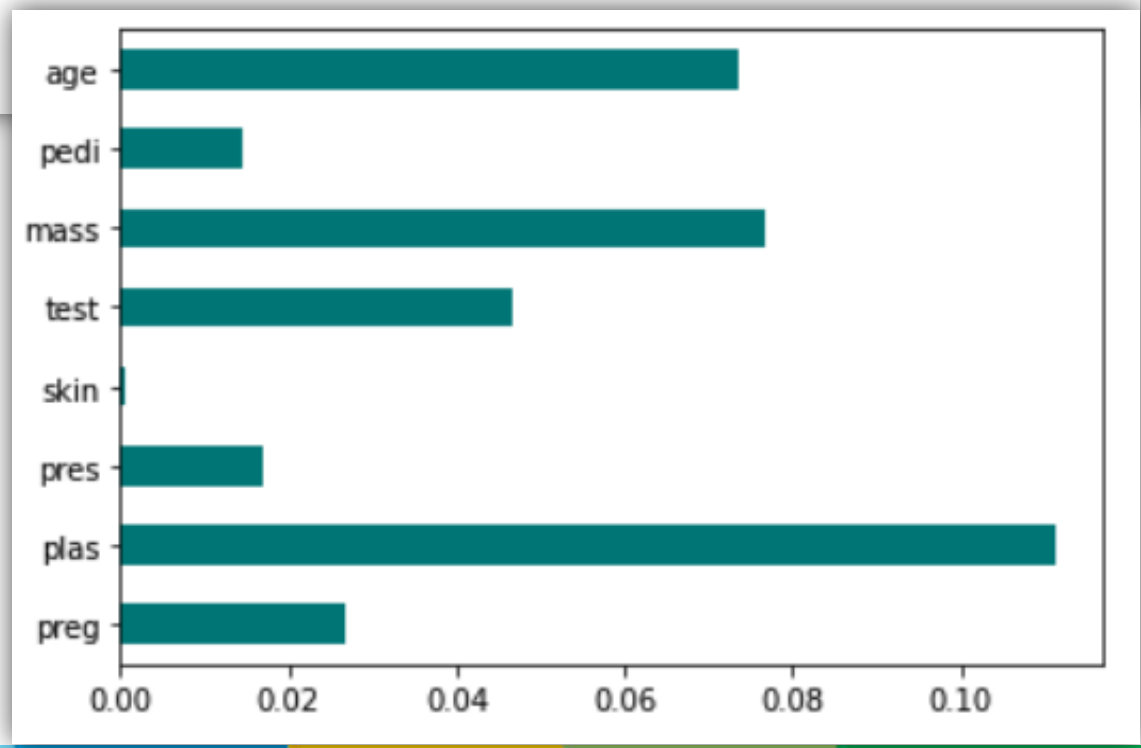
Calculates the reduction in **entropy** from the transformation of a dataset

Choose the feature with best **information gain** value with respect to target variable

Mutual Information Gain

```
from sklearn.feature_selection import mutual_info_classif
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

importances = mutual_info_classif(X, Y)
feat_importances = pd.Series(importances, dataframe.columns[0:len(dataframe.columns)-1])
feat_importances.plot(kind='barh', color='teal')
plt.show()
```



Filter Feature Selection Methods

Fisher's Score

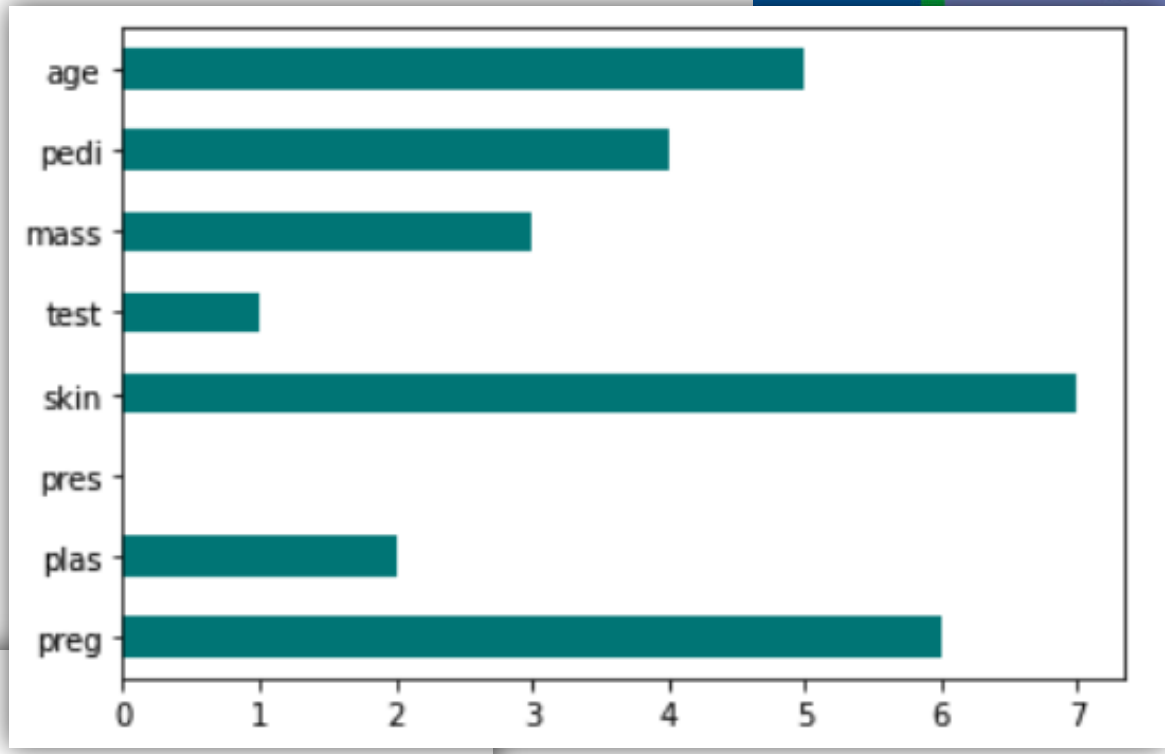
Fisher's score

Calculates Fisher's score between **target** and **feature**

Returns the **ranks** of features based on the fisher's score

Select **top n** features

Fisher's score




```
from skfeature.function.similarity_based import fisher_score
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

# Calculating Scores
ranks = fisher_score.fisher_score(X, Y)

# Plotting the ranks
feat_importances = pd.Series(ranks, dataframe.columns[0:len(dataframe.columns)-1])
feat_importances.plot(kind='barh', color='teal')
plt.show()
```

Filter Feature Selection Methods

Correlation Coefficient with Heat Map

Correlation Coefficient

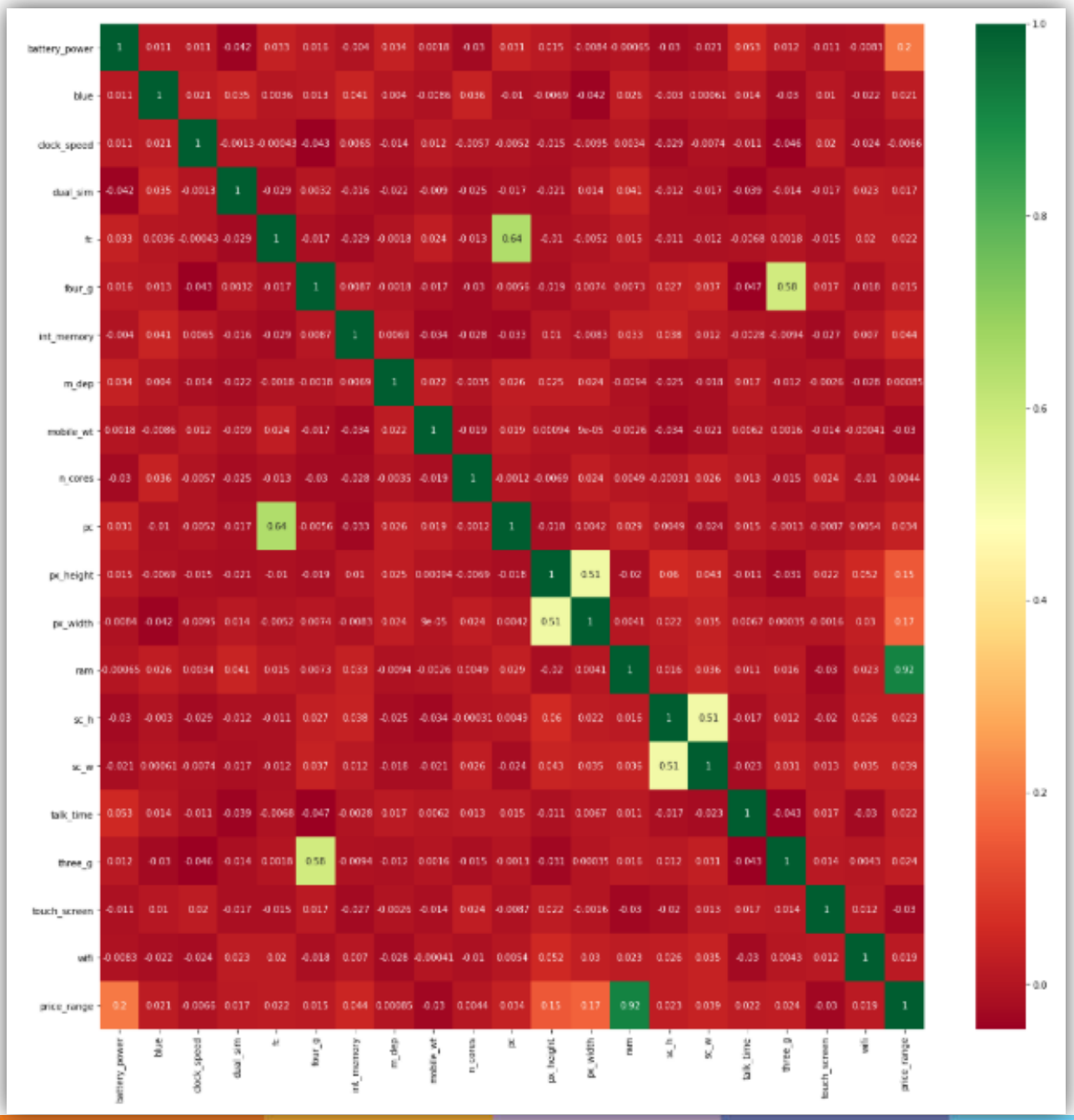
Correlation: Measure of linear relationship of 2 or more features

Good features – highly correlated with target, uncorrelated amongst other features

Heatmap - easy visualization

Correlation Coefficient

```
import pandas as pd
import numpy as np
import seaborn as sns
data = pd.read_csv("train.csv")
X = data.iloc[:,0:20]  #independent columns
y = data.iloc[:,-1]    #target column i.e price range
#get correlations of each features in dataset
corrmat = data.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
#plot heat map
g=sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```


battery_power
blue
clock_speed
dual_sim
fc
four_g
int_memory
m_dep
mobile_wt
n_cores
pc
px_height
px_width
ram
sc_h
sc_w
talk_time
three_g
touch_screen
wifi
price_range
1.0
0.8
0.6
0.4
0.2
0.0

Filter Feature Selection Methods

Variance Threshold

Variance Threshold

Removes all features whose **variance** doesn't meet a **threshold**

Removes all **zero-variance** features

get_support() returns Boolean value – True means variable does not have zero variance

Variance Threshold

```
from sklearn.feature_selection import VarianceThreshold

# Resetting the value of X to make it non-categorical
X = array[:, 0:8]

v_threshold = VarianceThreshold(threshold=0)
v_threshold.fit(X) # fit finds the features with zero variance
v_threshold.get_support()
```

Filter Feature Selection Methods

Mean Absolute Difference (MAD)

Mean Absolute Difference (MAD)

Computes the absolute difference from the mean value

```
mean_abs_diff = np.sum(np.abs(X - np.mean(X, axis=0)), axis=0) / X.shape[0]
```

Higher the MAD, higher the discriminatory power, so good feature

Filter Feature Selection Methods

Dispersion Ratio

Dispersion Ratio

$$AM_i = \overline{X_i} = \frac{1}{n}\sum_{j=1}^{n} X_{ij}$$

$$GM_i = \left(\prod_{j=1}^{n} X_{ij}\right)^{\frac{1}{n}}$$

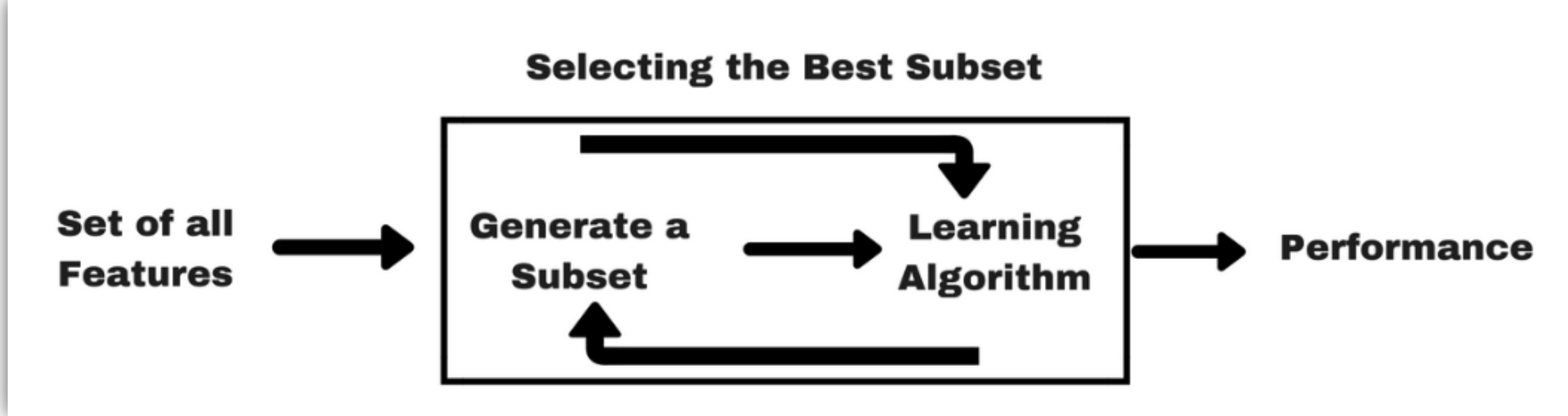
$$RM_i = \frac{AM_i}{GM_i} \in [1, +\infty)$$

Measure of dispersion applies the Arithmetic Mean and Geometric Mean

Higher dispersion, more relevant feature

Wrapper Feature Selection Methods

Wrapper Methods



Use a subset of features to train a model

Add or remove features depending on the inference accuracy

Search through a feature space to identify the best features

Wrapper Feature Selection Methods

Forward Feature Selection

Forward Feature Selection

Iterative method

No feature at the start

At each iteration, keep adding feature that best improves the performance of model

Repeat until no improvement is observed

Forward Feature Selection

```
from mlxtend.feature_selection import SequentialFeatureSelector
ffs = SequentialFeatureSelector(lr, k_features='best', forward=True, n_jobs=-1)
ffs.fit(X, Y)
features = list(ffs.k_feature_names_)
features = list(map(int, features))
lr.fit(x_train[features], y_train)
y_pred = lr.predict(x_train[features])
```

Wrapper Feature Selection Methods

Backward Feature Elimination

Backward Feature Elimination

Iterative method

All features at the start and build a model

At each iteration, keep removing feature that does not affect performance of model

Repeat until preset criterion is achieved

Backward Feature Elimination

```
from sklearn.linear_model import LogisticRegression
from mlxtend.feature_selection import SequentialFeatureSelector
lr = LogisticRegression(class_weight='balanced',
                        solver='lbfgs',
                        random_state=42,
                        n_jobs=-1,
                        max_iter=500
)
lr.fit(X, Y)
bfs = SequentialFeatureSelector(lr, k_features='best', forward=False, n_jobs=-1)
bfs.fit(X, Y)
features = list(bfs.k_feature_names_)
features = list(map(int, features))
lr.fit(x_train[features], y_train)
y_pred = lr.predict(x_train[features])
```

Wrapper Feature Selection Methods

Exhaustive Feature Selection

Exhaustive Feature Selection

Iterative method

Brute-force evaluation of each feature subset

At each iteration, it tries every possible combination of features and measures the performance

Returns the best performing feature subset

Exhaustive Feature Selection

```python
# Algorithm you want to evaluate on your features
from sklearn.linear_model import RandomForestClassifier

# Exhaustive Feature Selection
from mlxtend.feature_selection import ExhaustiveFeatureSelector

# Create the ExhaustiveFeatureSelector object
efs = ExhaustiveFeatureSelector(RandomForestClassifier(),
                                min_features=4,
                                max_features=8,
                                scoring='roc_auc',
                                cv=2
)

# Fit the object to the training data
efs = efs.fit(X, Y)

# Print the selected features
selected_features = x_train.columns[list(efs.best_idx_)]
print(selected_features)

# Print the final prediction score
print(efs.best_score_)
```

```
Int64Index([0, 1, 2, 3, 4, 5, 6, 7], dtype='int64')
0.8252014925373135
```

Wrapper Feature Selection Methods

Recursive Feature Elimination

Recursive Feature Elimination

Given an external estimator that assigns weights to features, the goal of RFE is to select features by recursively considering smaller and smaller sets of features

Estimator is trained on initial set of features and importance of each feature is obtained

Then, the least important features are pruned from the current set of features

Repeat until desired number of features is reached

Recursive Feature Elimination

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

rfe_selector = RFE(estimator=LogisticRegression(),
                   n_features_to_select=num_feats,
                   step=10,
                   verbose=5
)

rfe_selector.fit(X_norm, y)
rfe_support = rfe_selector.get_support()
rfe_feature = X.loc[:,rfe_support].columns.tolist()
print(str(len(rfe_feature)), 'selected features')
```

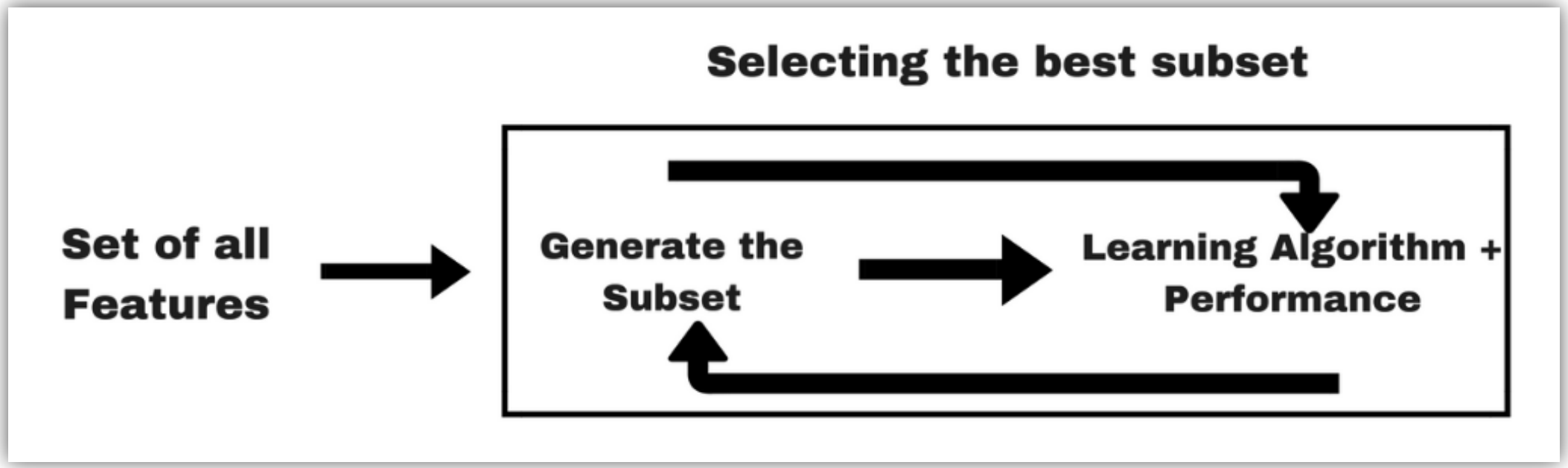
Case Study: Pima Indian Diabetes Dataset


```
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# load data
url = "https://raw.githubusercontent.com/subashgandyer/Datasets/pima-indians-diabetes.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(url, names=names)
array = dataframe.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_)
```

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

Embedded Feature Selection Methods

Embedder Methods



Combines the best worlds of both filter and wrapper methods

Lasso and Ridge Regularizations come under this method

Embedded Feature Selection Methods

Lasso (L1)

Lasso Regularization (L1)

Adding penalty to parameters of the model to reduce the freedom of the model thereby avoiding over-fitting

Adds penalty to absolute value of the magnitude of coefficients

Penalty is applied over coefficients that multiply each of the predictors

Ability to shrink some of the coefficients to zero, therefore that feature can be removed from the model

Lasso Regularization (L1)

```
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression

embeded_lr_selector = SelectFromModel(LogisticRegression(penalty="l1"), max_features=num_feats)
embeded_lr_selector.fit(X_norm, y)

embeded_lr_support = embeded_lr_selector.get_support()
embeded_lr_feature = X.loc[:,embeded_lr_support].columns.tolist()
print(str(len(embeded_lr_feature)), 'selected features')
```

Embedded Feature Selection Methods

Random Forest Importance

Embedded Feature Selection Methods

Tree-based: RandomForest

RandomForest: SelectFromModel

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier

embeded_rf_selector = SelectFromModel(RandomForestClassifier(n_estimators=100), max_features=num_feats)
embeded_rf_selector.fit(X, y)

embeded_rf_support = embeded_rf_selector.get_support()
embeded_rf_feature = X.loc[:,embeded_rf_support].columns.tolist()
print(str(len(embeded_rf_feature)), 'selected features')
```

Embedded Feature Selection Methods

Tree-based: LightGBM

LightGBM: SelectFromModel

```
from sklearn.feature_selection import SelectFromModel
from lightgbm import LGBMClassifier

lgbc=LGBMClassifier(n_estimators=500, learning_rate=0.05, num_leaves=32, colsample_bytree=0.2,
            reg_alpha=3, reg_lambda=1, min_split_gain=0.01, min_child_weight=40)

embeded_lgb_selector = SelectFromModel(lgbc, max_features=num_feats)
embeded_lgb_selector.fit(X, y)

embeded_lgb_support = embeded_lgb_selector.get_support()
embeded_lgb_feature = X.loc[:,embeded_lgb_support].columns.tolist()
print(str(len(embeded_lgb_feature)), 'selected features')
```

Task7: Create AutoFS tool

Create an AutoFS tool that will select the best features for a given dataset using a set of different feature selection methods on different algorithms

Exercise: Bringing it all together AutoFS

Use the previous code scripts in creating an AutoFS tool where you have to pass the dataset you want to analyze and get the best features from a set of algorithms you want to model