Published in Towards Data Science

Ayushi Jain Follow

Jul 17, 2020 · 12 min read · · Listen

Save

Source: Pixabay

Learn how to do Feature Selection the Right Way

Is x really a predictor of y?

Struggling with finding the right features for your model? By right, I mean value-adding features. Consider you are working on a high dimensional data that's coming from IoT sensors or healthcare with hundreds to thousands of features, it is tough to figure out what subset of features will bring out a good sustaining model. This article is all about feature selection and implementation of its techniques using scikit-learn on the automobile dataset. I've tried my best to keep the tricky calculations aside, but some basic understanding of statistics would make your journey easier. 😃

music engineers often employ various techniques to tune their music such that there is no unwanted noise and the voice is crisp and clear. Similarly, even the datasets encounter noise and its crucial to remove them for better model optimization. That's where feature selection comes into the picture!

Now, keeping the model accuracy aside, theoretically, **feature selection**

- ***reduces overfitting*** ' The Curse of Dimensionality' — If your dataset has more features/columns than samples **(X)**, the model will be prone to overfitting. By removing irrelevant data/noise, the model gets to focus on essential features, leading to more generalization.
- ***simplifies models*** — Dimensionality adds many layers to a model, making it needlessly complicated. Overengineering is fun but they may not be better than their simpler counterparts. Simpler models are easier to interpret and debug.
- ***reduces training time*** — Lesser features/dimensions reduces the computation speed, speeding up model training.

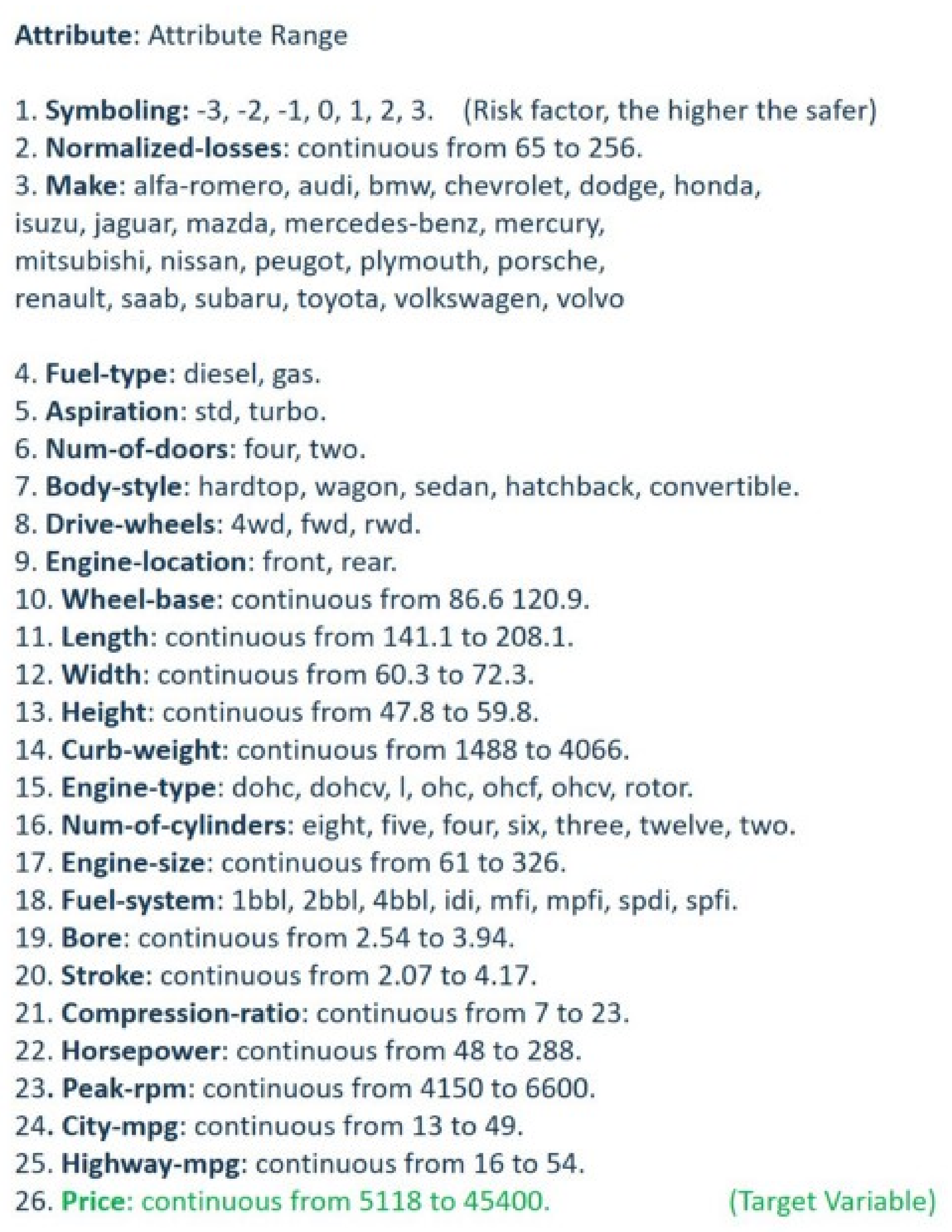
Keep in mind, all these benefits depend heavily on the problem. But for sure, it will result in a better model.

Is this Dimensionality Reduction?

Not exactly!

Often, feature selection and dimensionality reduction are used interchangeably, credits to their similar goals of reducing the number of features in a dataset. However, there is an important difference between them. Feature selection yields a subset of features from the original set of features, which are the best representatives of the data. While dimensionality reduction is the introduction of a new feature space where the original features are represented. It basically transforms the feature space to a lower dimension, keeping the original features intact. This is done by either combining or excluding a few features. To sum up, you can consider feature selection as a part of dimensionality

We will be using the automobile dataset from the UCI Machine Learning repository. The dataset contains information on car specifications, its insurance risk rating and its normalized losses in use as compared to other cars. The goal of the model would be to predict the 'price'. A regression problem, it comprises a good mix of continuous and categorical variables, as shown below:



Attribute: Attribute Range

1. **Symboling:** -3, -2, -1, 0, 1, 2, 3. (Risk factor, the higher the safer)
2. **Normalized-losses**: continuous from 65 to 256.
3. **Make**: alfa-romero, audi, bmw, chevrolet, dodge, honda,
isuzu, jaguar, mazda, mercedes-benz, mercury,
mitsubishi, nissan, peugot, plymouth, porsche,
renault, saab, subaru, toyota, volkswagen, volvo

4. **Fuel-type**: diesel, gas.
5. **Aspiration**: std, turbo.
6. **Num-of-doors**: four, two.
7. **Body-style**: hardtop, wagon, sedan, hatchback, convertible.
8. **Drive-wheels**: 4wd, fwd, rwd.
9. **Engine-location**: front, rear.
10. **Wheel-base**: continuous from 86.6 120.9.
11. **Length**: continuous from 141.1 to 208.1.
12. **Width**: continuous from 60.3 to 72.3.
13. **Height**: continuous from 47.8 to 59.8.
14. **Curb-weight**: continuous from 1488 to 4066.
15. **Engine-type**: dohc, dohcv, l, ohc, ohcf, ohcv, rotor.
16. **Num-of-cylinders**: eight, five, four, six, three, twelve, two.
17. **Engine-size**: continuous from 61 to 326.
18. **Fuel-system**: 1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi.
19. **Bore**: continuous from 2.54 to 3.94.
20. **Stroke**: continuous from 2.07 to 4.17.
21. **Compression-ratio**: continuous from 7 to 23.
22. **Horsepower**: continuous from 48 to 288.
23. **Peak-rpm**: continuous from 4150 to 6600.
24. **City-mpg**: continuous from 13 to 49.
25. **Highway-mpg**: continuous from 16 to 54.
26. **Price**: continuous from 5118 to 45400. (Target Variable)

selection techniques, I won't go deep into the modeling process. For the complete regression code, check this jupyter notebook on the Github link below.

Ayushijain09/Feature-Selection-Techniques

Check this link for the complete Multiple Linear Regression code along with the feature techniques for the Automobile dataset.

github.com

Now, let's try to improve the model by feature selection!

Techniques

Concisely, feature selection methods can be divided into three major buckets, filter, wrapper & embedded.

I. Filter Methods

With filter methods, we primarily apply a statistical measure that suits our data to assign **each feature column** a calculated score. Based on that score, it will be decided whether that feature will be kept or removed from our predictive model. These methods are computationally inexpensive and are best for eliminating redundant irrelevant features. However, one downside is that they don't take feature correlations into consideration since they work independently on each feature.

Moreover, we have ***Univariate filter methods*** that work on ranking a single feature and ***Multivariate filter methods*** that evaluate the entire feature space. Let's explore the most notable filter methods of feature selection:

1.) Missing Values Ratio

Data columns with too many missing values won't be of much value. Theoretically, 25–30% is the acceptable threshold of missing values, beyond which we should drop those features from the analysis. If you have the domain knowledge, it's always better to make cumbersome, so I've added the output side by side using Github gist considering the same automobile dataset.

```
print(df_train.isnull().sum()/len(df_train)*100).nlargest())
#output => Returns the 5 largest values from the series. No missing values in automobile dataset, s
"""symboling     0.0
   doornumber    0.0
   wheelbase     0.0
   carlength     0.0
   carwidth      0.0
   dtype: float64"""
```

Missing_Values.py hosted with ❤ by GitHub view raw

nlargest() returns default 5 features with most missing values

2.) Variance Threshold

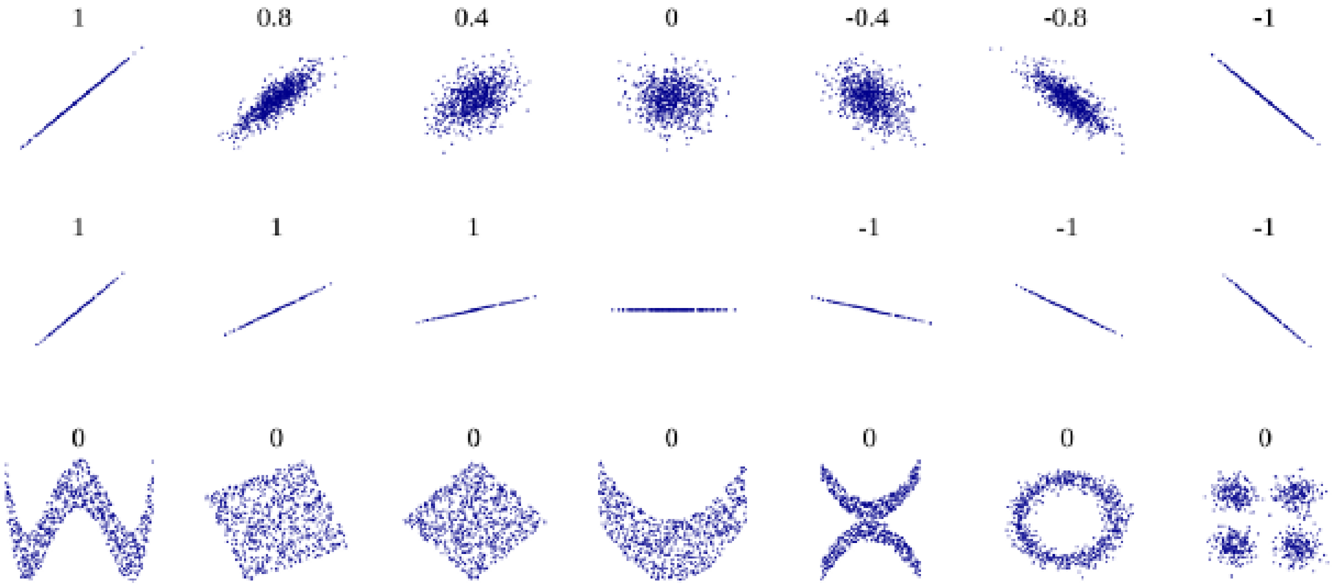
Features in which identical value occupies the majority of the samples are said to have zero variance. Such features carrying little information will not affect the target variable and can be dropped. You can adjust the threshold value, default is 0, i.e remove the features that have the same value in all samples. For quasi-constant features, that have the same value for a very large subset, use threshold as 0.01. In other words, drop the column where 99% of the values are similar.

```
from sklearn.feature_selection import VarianceThreshold
print(df_train.shape)     #output (143, 59)
var_filter = VarianceThreshold(threshold = 0.0)
train = var_filter.fit_transform(df_train)
#to get the count of features that are not constant
print(train.shape())    # output (143, 56)
#or
print(len(df_train.columns[var_filter.get_support()]))  #output 56
```

VarianceThreshold.py hosted with ❤ by GitHub view raw

Drops features with zero variance (customizable threshold) features to be fed to the model, if one can suffice. It centrally takes into consideration the fitted line, slope of the fitted line and the quality of the fit. There are various approaches for calculating correlation coefficients and if a pair of columns cross a certain threshold, the one that shows a high correlation with the target variable (y) will be kept and the other one will be dropped.

Pearson correlation *(for continuous data)* is a parametric statistical test that measures the similarity between two variables. Got confused by the parametric term? It means that this test assumes that the observed data follows some distribution pattern(e.g. normal, gaussian). Its coefficient value '**r**' ranges between **-1(**negative correlation) to **1(**positive correlation) indicating how well the data fits the model. It also returns '**p-value**' to determine whether the correlation between variables is significant by comparing it to a significance level 'alpha' (α). If the p-value is less than α, it means that the sample contains sufficient evidence to reject the null hypothesis and conclude that the correlation coefficient does not equal zero.



Source: Several sets of (x, y) points, with the correlation coefficient of x and y for each set. Note that the correlation reflects the strength and direction of a linear relationship (top row), but not the slope of that relationship (middle), nor many aspects of nonlinear relationships (bottom).

conducting Pearson's correlation. For newbies, ordinal data is categorical data but with a slight nuance of ranking/ordering (e.g low, medium and high). An important assumption to be noted here is that there should be a monotonic relationship between the variables, i.e. variables increase in value together or if one increases, the other one decreases.

Kendall correlation coefficient *(for discrete/ordinal data)* - Similar to Spearman correlation, this coefficient compares the number of concordant and discordant pairs of data.

> Let's say we have a pair of observations (x_i, y_i), (x_j, y_j), with $i < j$, they are:
> * *concordant if either* $(x_i > x_j \text{ and } y_i > y_j)$ *or* $(x_i < x_j \text{ and } y_i < y_j)$
> * *discordant if either* $(x_i < x_j \text{ and } y_i > y_j)$ *or* $(x_i > x_j \text{ and } y_i < y_j)$
> * *neither if there's a tie in* $\mathbf{x}$ $(x_i = x_j)$ *or a tie in* $\mathbf{y}$ $(y_i = y_j)$

Denoted with the Greek letter tau (**τ**), this coefficient varies between -1 to 1 and is based on the difference in the counts of concordant and discordant pairs relative to the number of x-y pairs.

```python
#USING SCIPY
from scipy.stats import spearmanr
from scipy.stats import pearsonr
from scipy.stats import kendalltau

coef, p = pearsonr(x, y)    #Pearson's r
coef, p = spearmanr(x, y)   # Spearman's rho
coef, p = kendalltau(x, y)  # Kendall's tau

#USING PANDAS
x.corr(y)                     #Pearson's r
x.corr(y, method='spearman')  # Spearman's rho
x.corr(y, method='kendall')   # Kendall's tau
```

Pearson_Spearman_Kendall_Correlation.py hosted with ❤ by GitHub view raw

Pearson, Spearman, Kendall's Correlation Coefficient using Scipy & Pandas

4.) Chi-Square Test of Independence(for categorical data)

Before diving into chi-square, let's understand an important concept: hypothesis testing! Imagine XYZ makes a claim, a commonly accepted fact, you call it a *Null Hypothesis*. Now you come up with an alternate hypothesis, one that you think explains that phenomenon better and then work towards rejecting the null hypothesis.
In our case:
Null Hypothesis: The two variables are independent.
Alternative Hypothesis: The two variables are dependent.

So, Chi-Square tests come in two variations - one that evaluates the **goodness-of-fit** and the other one where we will be focusing on is the **test of independence**. Primarily, it compares the observed data to a model that distributes the data according to the **expectation** that the variables are independent. Then, you basically need to check where the observed data doesn't fit the model. If there are too many data points/outliers, there is a huge possibility that the variables are dependent, proving that the null hypothesis is incorrect!

It primarily returns a test statistic **"p-value"** to help us decide! On a high level, if the p-value is less than some critical value- '**level of significance**'(usually 0.05), we reject the null hypothesis and believe that the variables are dependent!

Chi-square would not work with the automobile dataset since it needs categorical variables and non-negative values! For that reason, we can use Mutual Information & ANOVA.

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
X_train = X_train.astype(int)
chi2_features = SelectKBest(chi2 , k=12)
X_kbest_features = chi2_features.fit_transform(X_train, y_train)
```

Chi2.py hosted with ❤ by GitHub view raw

5.) Mutual Information(for both regression & classification)

is deterministic of Y. MI is primarily the entropy of X, which measures or quantifies the amount of information obtained about one random variable, through the other random variable.

The best thing about MI is that it allows one to detect non-linear relationships and works for both regression and classification. Cool! isn't it 😃

```
from sklearn.feature_selection import mutual_info_regression
from sklearn.feature_selection import SelectKBest
selector = SelectKBest(mutual_info_regression, k=10)
X_train_new = selector.fit_transform(X_train, y_train)  #Applying transformation to the training s
#to get names of the selected features
mask = selector.get_support()     # Output   array([False, False,  True,  True,  True, False ....]

print(selector.scores_)     #Output array([0.16978127, 0.01829886, 0.45461366, 0.55126343, 0.66081

new_features = X_train.columns[mask]
print(new_features)   #Output Index(['wheelbase', 'carlength', 'carwidth', 'curbweight', 'enginesi
print(train.shape)  #Output (143, 10)
```

mutual_info_regression.py hosted with ❤ by GitHub view raw

6.) Analysis of Variance (ANOVA)

Okay honestly, this is a bit tricky but let's understand it step by step. Firstly, here instead of features we deal with groups/ levels. Groups are different groups within the same independent(categorical) variable. ANOVA is primarily an **extension of a t-test**. With a t-test, you can study only two groups but with ANOVA you need at least three groups to see if there's a difference in means and determine if they came from the same population.

> *It assumes Hypothesis as*
> *H0: Means of all groups are equal.*
> *H1: At least one mean of the groups are different.*

Let's say from our automobile dataset, we use a feature 'fuel-type' that has 2 groups/levels

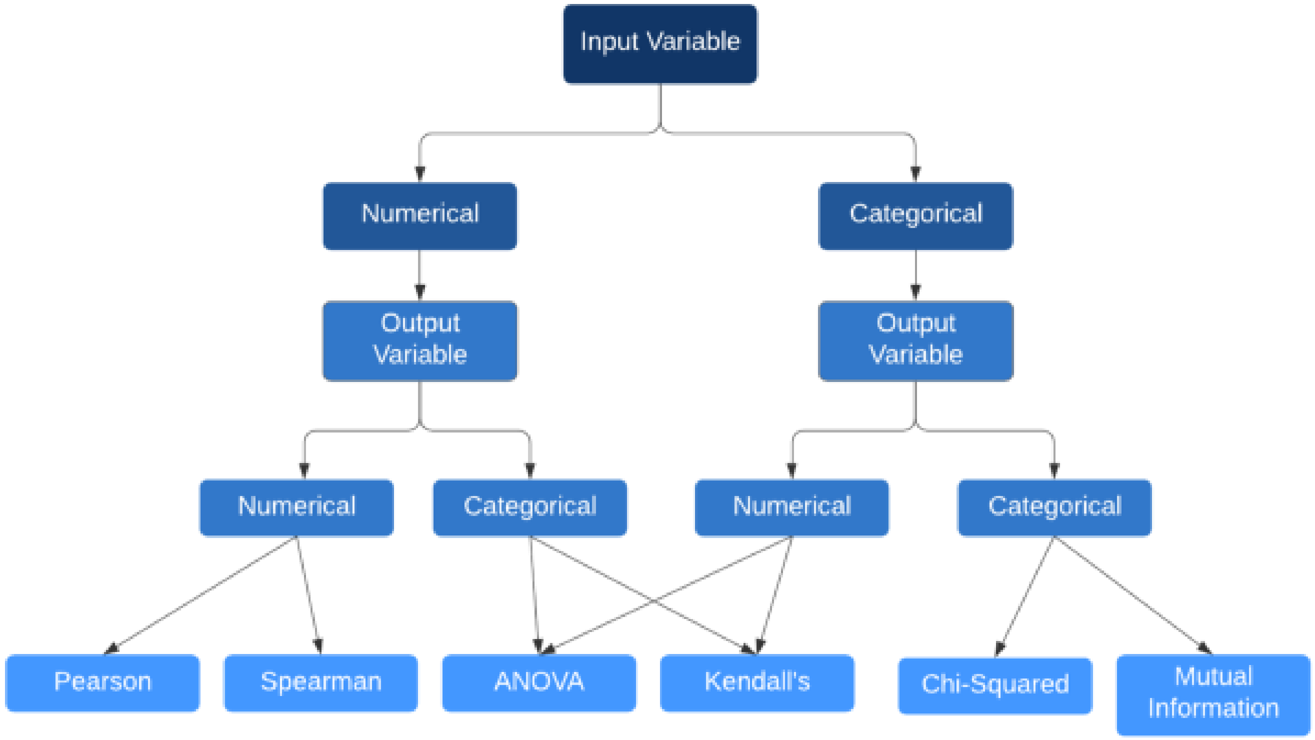
groups and the larger this number is, the more likely it is that the means of the groups really *are* different, and that you should reject the null hypothesis.

```python
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_regression

fvalue_selector = SelectKBest(f_regression, k=20)  #select features with 20 best ANOVA F-Values
X_train_new = fvalue_selector.fit_transform(X_train, y_train)
print(X_train.shape, X_train_new.shape)     #output (143, 59) (143, 20)
```

ANOVA.py hosted with ❤ by GitHub view raw

Anova Feature Selection for Regression task



Choosing a Feature selection Method (Image by Author)

II. Wrapper Methods

In Wrapper methods, we primarily choose a subset of features and train them using a

The downside is, it becomes computationally expensive as the features increase, but on the good side, it takes care of the interactions between the features, ultimately finding the optimal subset of features for your model with the lowest possible error.

1.) Sequential Feature Selection

A greedy search algorithm, this comes in two variants- **Sequential Forward Selection** (SFS) and **Sequential Backward Selection** (SBS). It basically starts with a null set of features and then looks for a feature that **minimizes the cost function**. Once the feature is found, it gets added to the feature subset and in the same way one by one, it finds the right set of features to build an optimal model. That's how SFS works. With Sequential Backward Feature Selection, it takes a totally opposite route. It starts with all the features and iteratively removes one by one feature depending on the performance. Both algorithms have the same goal of attaining the lowest cost model.

> *The main limitation of SFS is that it is* unable to remove features *that become non-useful after the addition of other features. The main limitation of SBS is its* inability to reevaluate *the usefulness of a feature after it has been discarded.*

```
from mlxtend.feature_selection import SequentialFeatureSelector
sfs = SequentialFeatureSelector(LinearRegression(),         # cv = k-fold cross validation, floati
           k_features=10,
           forward=True,
           floating=False,
           scoring='accuracy',
           cv=2)
sfs = sfs.fit(X_train, y_train)

selected_features = x_train.columns[list(sfs.k_feature_idx_)]
print(selected_features)

# print the selected features.
selected_features = x_train.columns[list(sfs.k_feature_idx_)]
print(selected_features)

# final prediction score.
```

Sequential_Forward_Selection.py hosted with ❤ by GitHub view raw

SFS (for Sequential Backward Selection, change the forward parameter to False)

2.) Recursive Feature Elimination (RFE)

Considering, you have an initial set of features, what this greedy algorithm does is it repeatedly performs model building by considering smaller subsets of features each time. How does it do that? After an estimator is trained on the features, it returns a rank value based on the model's *coef_* or *feature_importances_* attribute conveying the importance of each feature. For the next step, the least important features are pruned from the current set of features. This process is recursively repeated until the specified number of features are attained.

```
from sklearn.feature_selection import RFE
lm = LinearRegression()
rfe1 = RFE(lm, 20)   # RFE with 20 features

# Fit on train and test data with 20 features
X_train_new = rfe1.fit_transform(X_train, y_train)
X_test_new = rfe1.transform(X_test)

# Print the boolean results
print(rfe1.support_)       # Output [False False False False  True False False False  True False F
print(rfe1.ranking_)       # Output [36 34 23 26  1 21 12 27  1 13 28  1 18 19 32 25  1 11  9  7

lm.fit(X_train_new, y_train)
predictions_rfe = lm.predict(X_test_new)
RMSE = np.sqrt(mean_squared_error(y_test, predictions_rfe))
R2 = r2_score(y_test, predictions)
print('R2:',R2,'RMSE:',RMSE)     #Output R2: 0.88 RMSE: 0.33
```

RFE_Regression.py hosted with ❤ by GitHub view raw

RFE for Regression

III. Embedded Methods

they are called embedded! The computational speed is as good as of filter methods and of course better accuracy, making it a win-win model!

1.) L1 (LASSO) Regularization

Before diving into L1, let's understand a bit about regularization. Primarily, it is a technique used to reduce overfitting to highly complex models. We add a penalty term to the cost function so that as the model complexity increases the cost function increases by a huge value. Coming back to LASSO (Least Absolute Shrinkage and Selection Operator) Regularization, what you need to understand here is that it comes with a parameter, **'alpha'** and the higher the alpha is, the more feature coefficients of least important features are shrunk to zero. Eventually, we get a much simple model with the same or better accuracy!

However, in cases where a certain feature is important, you can try Ridge regularization (L2) or Elastic Net (a combination of L1 & L2), wherein instead of dropping it completely, it reduces the feature weightage.

```python
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV

lasso= Lasso()
parameters = {'alpha':[1e-15, 1e-10, 1e-8, 1e-4,1e-3,1e-2,1,3,5]}
lasso_model = GridSearchCV(lasso, parameters, scoring = 'r2',cv=5)
lasso_model.fit(X_train,y_train)
pred = lasso_model.predict(X_test)
print(lasso_model.best_params_)   #output {'alpha': 0.001}
print(lasso_model.best_score_)    #output 0.8630550401365724
```

Lasso_Regression_Model.py hosted with ❤ by GitHub view raw

Lasso Regression

2.) Tree Model (for Regression & Classification)

One of the most popular and accurate machine learning algorithms, random forests are an ensemble of randomized **decision** trees. An individual tree won't contain all the root, the function used to create the tree tries all possible splits by making conditional comparisons at each step and chooses the one that splits the data into the most homogenous groups (most pure). The importance of each feature is derived by how "pure" each of the sets is.

Using **Gini impurity** for classification and variance for regression, we can identify the features that would lead to an optimal model. The same concept can be applied to CART (Classification and Regression Trees) and boosting tree algorithms as well.

```python
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor()
# fit the model
model.fit(X_train, y_train)
# get importance
importance = model.feature_importances_
# summarize feature importance
impList = zip(X_train.columns, importance)
for feature in sorted(impList, key = lambda t: t[1], reverse=True):
    print(feature)

#Output - Important features
"""   ('enginesize', 0.6348884035234398)
      ('curbweight', 0.2389770360203148)
      ('horsepower', 0.03458620700119025)
      ('carwidth', 0.027170640676336785)
      ('stroke', 0.012516866412495744)
      ('peakrpm', 0.011750282673996262)
      ('carCompany_bmw', 0.009801675326218959)
      ('carlength', 0.008737911775028553) ..... """
```

tree_regression.py hosted with ❤ by GitHub view raw

Feature Importances using Decision tree

selection algorithm doesn't guarantee better accuracy always, but will surely lead to a simpler model than before!

If you have any questions/thoughts, feel free to leave your feedback in the comment section below or you can reach me on Linkedin.

Stay tuned for more! 😃

References

NumPy, SciPy, and Pandas: Correlation With Python - Real Python

In this tutorial, you'll learn what correlation is and how you can calculate it with Python. You'll use SciPy, NumPy...

realpython.com

Hands-on with Feature Selection Techniques: Embedded Methods

Part 4: Regularization and tree-based embedded methods

heartbeat.fritz.ai

Sign up for The Variable

our Privacy Policy for more information about our privacy practices.