# Feature Importance — Everything you need to know

A machine learning model is only as good as the features that it is trained on. But how do we find the best features for the problem statement? This is where feature selection comes in.

Sandeep Ram Follow
Oct 25, 2020 · 5 min read

Photo by Anders Jildén on Unsplash

In this article, we will be exploring various feature selection techniques that we need to be familiar with, in order to get the best performance out of your model.

- SelectKBest
- Linear Regression
- Random Forest
- XGBoost
- Recursive Feature Elimination
- Boruta

## SelectKBest

SelectKbest is a method provided by sklearn to rank features of a dataset by their “importance ”with respect to the target variable. This “importance” is calculated using a score function which can be one of the following:

- **f_classif**: ANOVA F-value between label/feature for classification tasks
- **f_regression**: F-value between label/feature for regression tasks.
- **chi2**: Chi-squared stats of non-negative features for classification tasks.
- **mutaul_info_classif**: Mutual information for a discrete target.
- **SelectPercentile**: Select features based on the percentile of the highest scores.
- **SelectFpr**: Select features based on a false positive rate test.
- **SelectFdr**: Select features based on an estimated false discovery rate.
- **SelectFwe**: Select features based on the family-wise error rate.
- **GenericUnivariateSelect**: Univariate feature selector with configurable mode.

All of the above-mentioned scoring functions are based on statistics. For instance, the **f_regression** function arranges the **p_values** of each of the variables in increasing order and picks the best K columns with the least p_value. Features with a p_value of less than **0.05** are considered “significant” and only these features should be used in the predictive model.

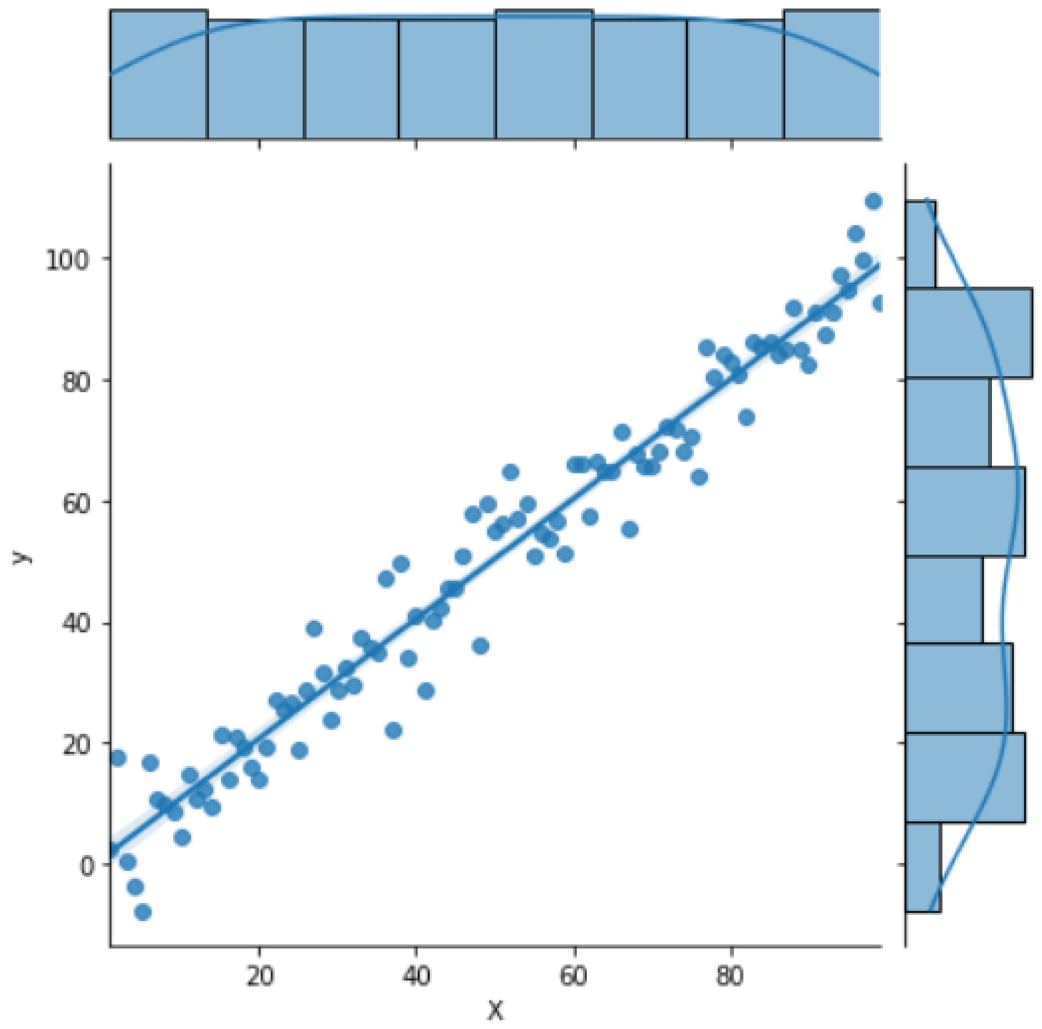
```
from sklearn.feature_selection import SelectKBest
```

```python
from sklearn.feature_selection import SelectKBest

df = SelectKBest(f_regression, k = 5).fit_transform(X_train, y_train)
print(df.head())
```

.py hosted with ♡ by GitHub view raw

## Significant Feature- P_value lesser than 0.05:

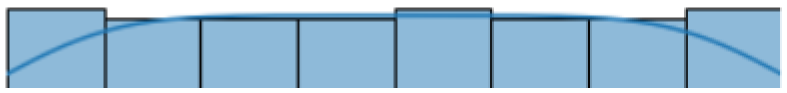
```
p_value =  4.434267837147035e-68
```

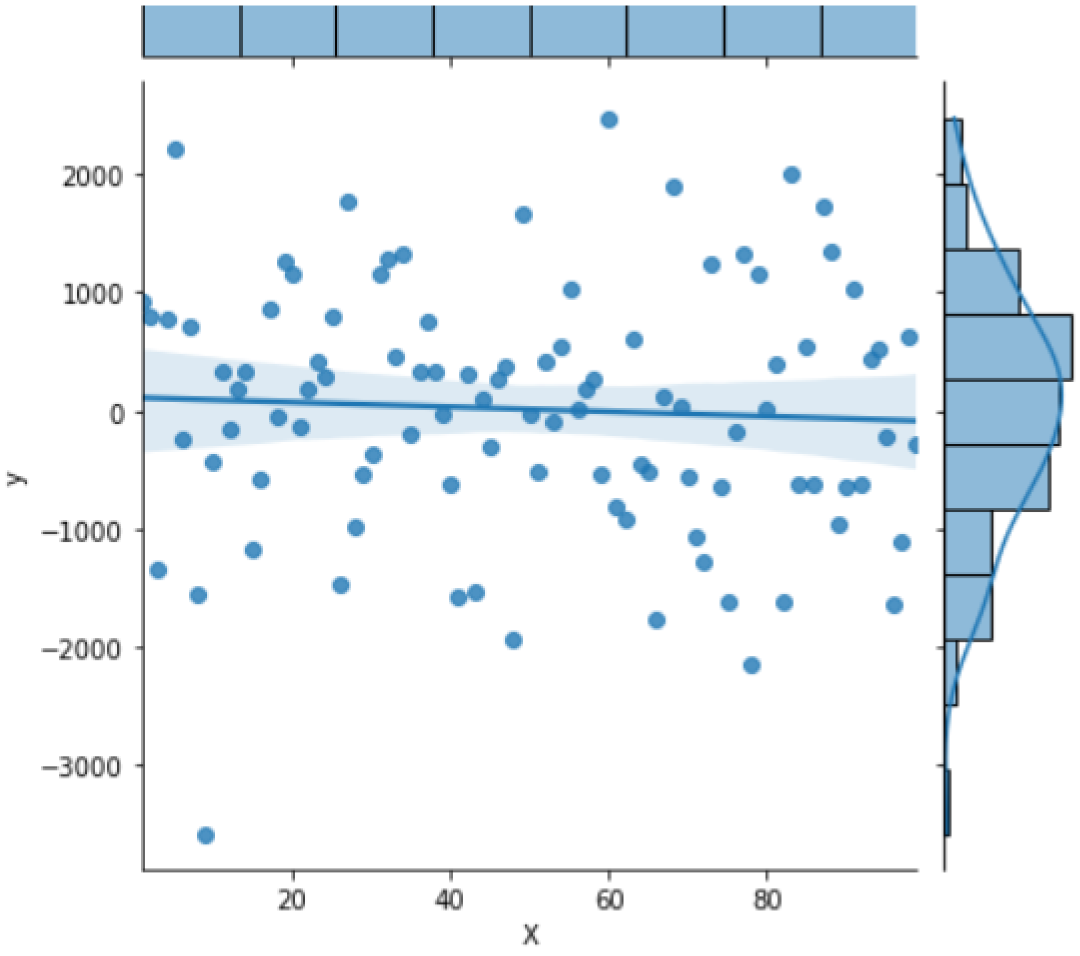


A Great Fit [Image by Author]

## Insignificant Features- P_value more than 0.05

```
p_value =  0.5954976628383999
```





A Bad Fit [Image by Author]

This is one of the simplest methods as it is very computationally efficient and takes just a few lines of code to execute.

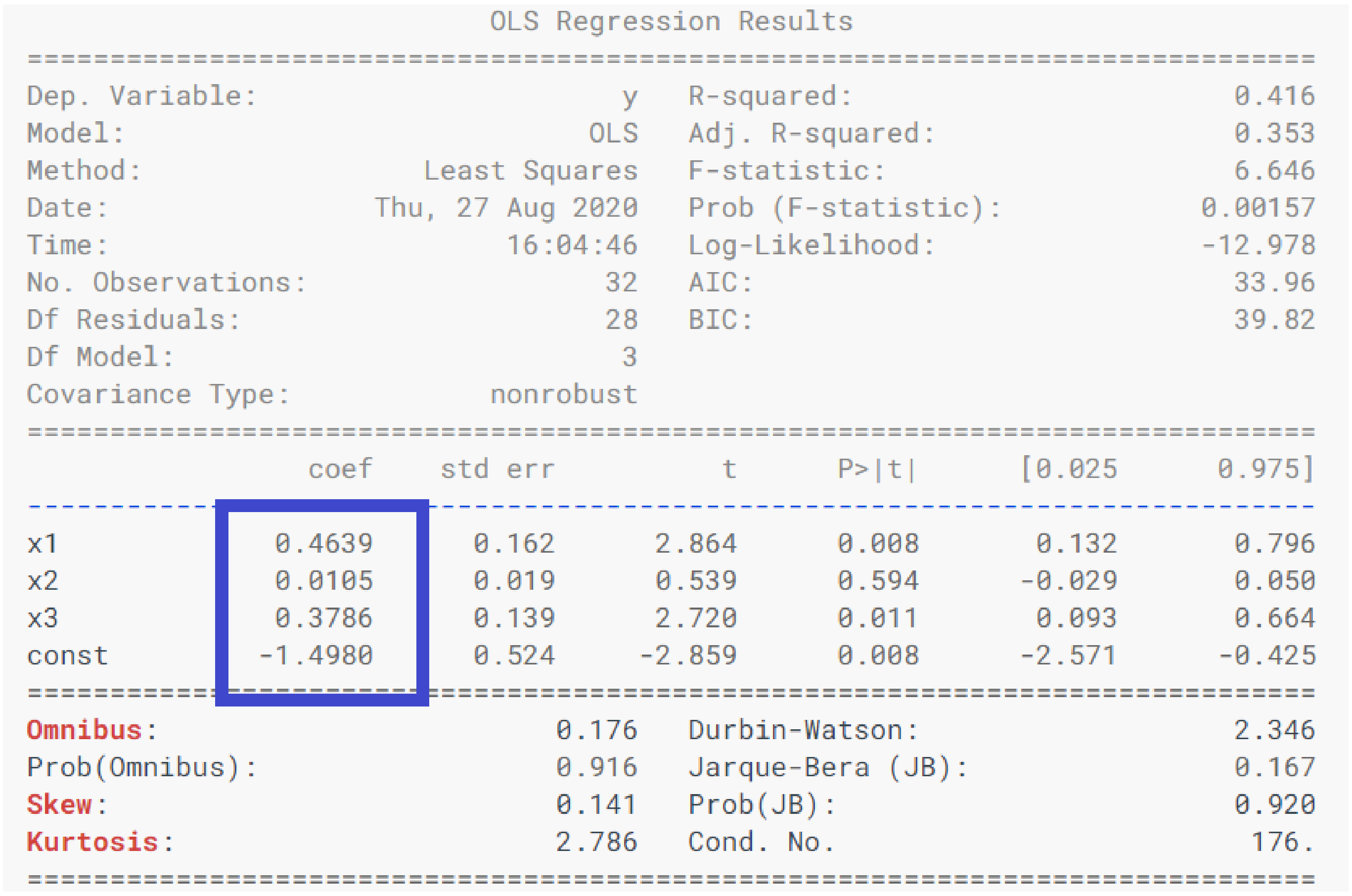
**Why P_value is not the perfect feature selection technique?**

P_value is an analysis of how each dependent variable is individually related to the target variable. However, this is not always the case. Let's take an example to illustrate this

Consider a predictive regression model that tried to predict the price of a plot given the length and breadth of a plot. The p_value of each of these variables might actually be very large since neither of these features is directly related to the price. However, a combination of these 2 variables, specifically their product, gives the land area of the plot. This product has a very strong relationship with the price. Thus both length and breadth are significant features that are overlooked during p_value feature selection.

## Linear Regression- Comparing Coefficients

By comparing the coefficients of linear models, we can make an inference about which features are more important than others. This method does not work well when your linear model itself isn't a good fit for the dataset given. This method can be used if your model's accuracy is around 95%



```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.416
Model:                            OLS   Adj. R-squared:                  0.353
Method:                 Least Squares   F-statistic:                     6.646
Date:                Thu, 27 Aug 2020   Prob (F-statistic):            0.00157
Time:                        16:04:46   Log-Likelihood:                -12.978
No. Observations:                  32   AIC:                             33.96
Df Residuals:                      28   BIC:                             39.82
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1             0.4639      0.162      2.864      0.008       0.132       0.796
x2             0.0105      0.019      0.539      0.594      -0.029       0.050
x3             0.3786      0.139      2.720      0.011       0.093       0.664
const         -1.4980      0.524     -2.859      0.008      -2.571      -0.425
==============================================================================
Omnibus:                        0.176   Durbin-Watson:                   2.346
Prob(Omnibus):                  0.916   Jarque-Bera (JB):                0.167
Skew:                           0.141   Prob(JB):                        0.920
Kurtosis:                       2.786   Cond. No.                         176.
==============================================================================
```

[Image from Source]

In the case of the above example, the coefficient of x1 and x3 are much higher than x2, so dropping x2 might seem like a good idea here. This approach is valid in this example as this model is a very good fit for the given data.

## Random Forest

The Random Forest is a very elegant algorithm that usually gives highly accurate predictions, even with minimal hyperparameter tuning. However, this is not where its usefulness ends!

Random Forest, when imported from the sklearn library, provides a method where you can get the feature importance of each of the variables. This is a good method to gauge the feature importance on datasets where Random Forest fits the data with high accuracy.

```python
from sklearn.ensemble import RandomForestClassifier as RClf

model = RClf(n_estimators = 100)
model.fit(X, y)
importances = model.feature_importances_
std = np.std([tree.feautre_importances_ for tree in model.estimators_], axis = 0)

indices = np.argsort(importances)[::-1]

print('Feature Ranking:')

for f in range(X.shape[1]):
        print('%d. features %d (%f)'% (f+1, indices[f], importances[indices[f]]))
```

.py hosted with ♡ by GitHub view raw

## XGBoost

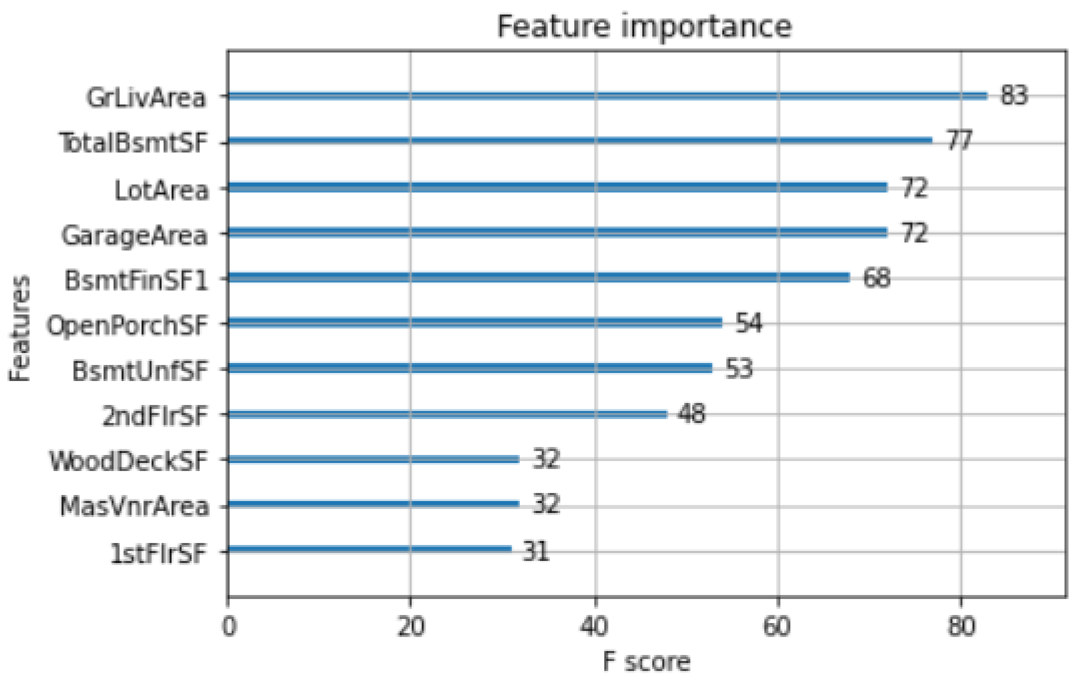
Just like random forests, XGBoost models also have an inbuilt method to directly get the feature importance. XGBoost feature accuracy is much better than the methods that are mentioned above since:

- Faster than Random Forests by far!
- It is way more reliable than Linear Models, thus the feature importance is usually much more accurate
- P_value test does not consider the relationship between two variables, thus the features with p_value > 0.05 might actually be important and vice versa. XGBoost usually does a good job of capturing the relationship between multiple variables while calculating feature importance

```python
from xgboost import XGBClassifier
from xgboost import plot_importance

# fit model to training data
xgb_model = XGBClassifier(random_state = 0 )
xgb_model.fit(X_train, y_train)

```

```python
print("Feature Importances : ", xgb_model.feature_importances_)

# plot feature importance
plot_importance(xgb_model)
plt.show()
```

.py hosted with ♡ by GitHub view raw



[Image by Author]

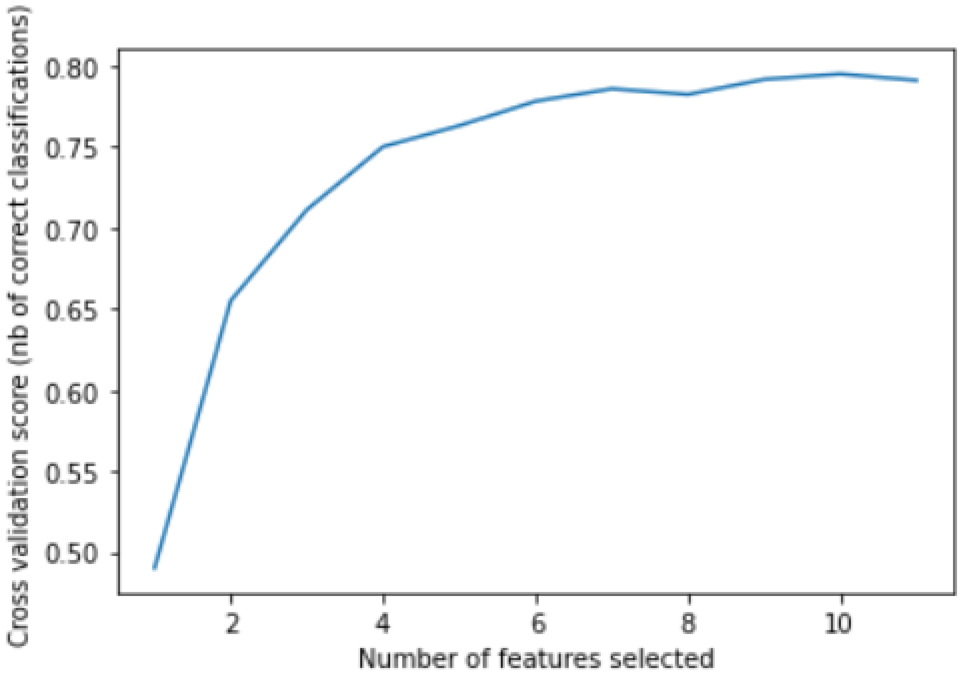
## RFE- Recursive Feature Elimination

This algorithm recursively calculates the feature importances and then drops the least important feature. It starts off by calculating the feature importance for each of the columns. It then drops the column with the least importance score and proceeds to repeat the same.

NOTE: This algorithm assumes that none of the features are correlated. It is not advisable to use a feature if it has a Pearson correlation coefficient of more than 0.8 with any other feature.

```python
# Plot number of features VS. cross-validation scores
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score (nb of correct classifications)")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
```

.py hosted with ♡ by GitHub view raw



[Image By Author]

## Boruta Feature Selection Algorithm

Unlike the previously mentioned algorithms, Boruta is an all-relevant feature selection method while most algorithms are minimal optimal. What this means is that Boruta tries to find all features carrying useful information rather than a compact subset of features that give a minimal error.

### Steps to Build a Boruta Selector:

- Install Bortua

```
!pip install boruta
```

- Make the necessary imports:

- Establish Base Score to build upon

- Train Boruta Feature Selector

- Calculate scores on the shortlisted features and compare them!

I personally use this method in most of my work. More often than not, using Boruta significantly reduces the dimension while also providing a minor boost to accuracy.

When trained on Housing Price Regression Dataset, Boruta reduced the dimensions from 80+ features to just 16 while it also provided an accuracy boost of 0.003%!

**Summary:**

- If the dataset is not too large, use Boruta for feature selection.
- If XGboost or RandomForest gives more than 90% accuracy on the dataset, we can directly use their inbuilt method ".feature_importance_"
- If you just want the relationship between any 2 variables and not the whole dataset itself, it's ideal to go for p_value score or person correlation.

## Conclusion

I hope you found this article informative. This article gives a surface-level understanding of many of the feature selection techniques. However, this is not an exhaustive list. Leave a comment if you feel any important feature selection technique is missing.

Sign up for Top 10 Stories

By The Startup

Get smarter at building your thing. Subscribe to receive The Startup's top 10 most read stories — delivered straight into your inbox, twice a month. Take a look.

Your email

Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

Data Science Machine Learning Artificial Intelligence Big Data

About Write Help Legal

Get the Medium app

Download on the App Store

GET IT ON Google Play