index

March 3, 2022

1 Feature Selection Methods

1.1 Introduction

In a previous section, you learned about many different ways to create features to model more complex relationships. However, you also saw that this can be problematic at times. For example, if you include interaction terms between all of your existing features and higher order polynomial functions, you will no doubt have an issue with overfitting to noise. In this lesson, you'll learn about the different techniques you can use to only use features that are most relevant to your model.

1.1.1 Objectives

You will be able to:

- Use feature selection to obtain the optimal subset of features in a dataset
- Identify when it is appropriate to use certain methods of feature selection

1.2 Feature selection

Feature selection is the process by which you select a subset of features relevant for model construction. Feature saelection comes with several benefits, the most obvious being the improvement in performance of a machine learning algorithm. Other benefits include:

- Decrease in computational complexity: As the number of features is reduced in a model, the easier it will be to compute the parameters of your model. It will also mean a decrease in the amount of data storage required to maintain the features of your model
- Understanding your data: In the process of feature selection, you will gain more understanding of how features relate to one another

Now, let's look at the different types of feature selection approaches and their advantages/disadvantages:

1.2.1 Types of feature selection

Like many things in data science, there is no clear and easy answer for deciding which features to include in a model. There are, however, different strategies you can use to process features in an efficient way:

- Domain knowledge
- Wrapper methods
- Filter methods
- Embedded methods

Domain knowledge One of the most important aspects when determining important features is the knowledge of the specific domain related to your dataset. This might mean reading past research papers that have explored similar topics or asking key stakeholders to determine what they believe the most important factors are for predicting the target variable.

Wrapper methods Wrapper methods determine the optimal subset of features using different combinations of features to train models and then calculating performance. Every subset is used to train models and then evaluated on a test set. As you might imagine, wrapper methods can end up being very computationally intensive, however, they are highly effective in determining the optimal subset. Because wrapper methods are so time-consuming, it becomes challenging to use them with large feature sets.

An example of a wrapper method in linear regression is **Recursive Feature Elimination**, which starts with all features included in a model and removes them one by one. After the model has had a feature removed, whichever subset of features resulted in the least statistically significant deterioration of the model fit will indicate which omitted feature is the least useful for prediction. The opposite of this process is **Forward Selection**, which undergoes the same process in reverse. It begins with a single feature and continues to add the one feature at a time that improves model performance the most.

Filter methods Filter methods are feature selection methods carried out as a pre-processing step before even running a model. Filter methods work by observing characteristics of how variables are related to one another. Depending on the model that is being used, different metrics are used to determine which features will get eliminated and which will remain. Typically, filter methods will return a "feature ranking" that will tell you how features are ordered in relation to one another. They will remove the variables that are considered redundant. It's up to the Data Scientist to determine the cut-off point at which they will keep the top n features, and this $$ is usually determined through cross-validation.

In the linear regression context, a common filter method is to eliminate features that are highly correlated with one another. Another method is to use a variance threshold. This sets some threshold of required variance among features in order to include them in a model. The thought process behind this is that if variables do not have a high variance, they will not change much and will therefore not have much impact on our dependent variable.

Embedded methods Embedded methods are feature selection methods that are included within the actual formulation of your machine learning algorithm. The most common kind of embedded method is regularization, in particular Lasso, because it has the capability of reducing your set of features automatically.

1.3 Feature selection in action

Now, we're going to review the process behind performing feature selection with a dataset pertaining to diabetes. The dataset contains the independent variables age, sex, body mass index, blood pressure, and 6 different blood serum measurements. The target variable represents a quantitative measurement progression of diabetes from one year after a baseline observation. With feature selection, our goal is to find a model that is able to maintain high accuracy while not overfitting to noise.

1.3.1 Process the data

To begin with, we are going to load the neccesary libraries and functions, import the data, and create a dummy variable for the variable `'SEX'`. The target variable is in the column `'Y'`.

[1]:
```python
# Importing necessary libraries
import pandas as pd
import numpy as np
from sklearn import datasets, linear_model
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LinearRegression

# Import the data
df = pd.read_csv('diabetes.tab.txt', sep='\t')
df.head()
```

[1]:

	AGE	SEX	BMI	BP	S1	S2	S3	S4	S5	S6	Y
0	59	2	32.1	101.0	157	93.2	38.0	4.0	4.8598	87	151
1	48	1	21.6	87.0	183	103.2	70.0	3.0	3.8918	69	75
2	72	2	30.5	93.0	156	93.6	41.0	4.0	4.6728	85	141
3	24	1	25.3	84.0	198	131.4	40.0	5.0	4.8903	89	206
4	50	1	23.0	101.0	192	125.4	52.0	4.0	4.2905	80	135

[2]:
```python
# Obtain the target and features from the DataFrame
target = df['Y']
features = df.drop(columns='Y')
```

[3]:
```python
# Create dummy variable for sex
features['female'] = pd.get_dummies(features['SEX'], drop_first=True)
features.drop(columns=['SEX'], inplace=True)
features.head()
```

[3]:

	AGE	BMI	BP	S1	S2	S3	S4	S5	S6	female
0	59	32.1	101.0	157	93.2	38.0	4.0	4.8598	87	1
1	48	21.6	87.0	183	103.2	70.0	3.0	3.8918	69	0
2	72	30.5	93.0	156	93.6	41.0	4.0	4.6728	85	1
3	24	25.3	84.0	198	131.4	40.0	5.0	4.8903	89	0
4	50	23.0	101.0	192	125.4	52.0	4.0	4.2905	80	0

For both regularization (an embedded method) and various filters, it is important to standardize the data. This next cell is fitting a `StandardScaler` from `sklearn` to the data.

[4]:
```python
from sklearn.preprocessing import StandardScaler

# Split the data
X_train, X_test, y_train, y_test = train_test_split(features, target,␣
↪random_state=20, test_size=0.2)
```

```
# Initialize the scaler
scaler = StandardScaler()

# Scale every feature except the binary column - female
transformed_training_features = scaler.fit_transform(X_train.iloc[:,:-1])
transformed_testing_features = scaler.transform(X_test.iloc[:,:-1])

# Convert the scaled features into a DataFrame
X_train_transformed = pd.DataFrame(scaler.transform(X_train.iloc[:,:-1]),
                                   columns=X_train.columns[:-1],
                                   index=X_train.index)
X_train_transformed['female'] = X_train['female']

X_test_transformed = pd.DataFrame(scaler.transform(X_test.iloc[:,:-1]),
                                  columns=X_train.columns[:-1],
                                  index=X_test.index)
X_test_transformed['female'] = X_test['female']
```

Before we perform feature selection, we should see how well the baseline model performs. Because we are going to be running many different models here, we have created a function to ensure that we are following the D.R.Y. principle.

[5]:
```
def run_model(model, X_train, X_test, y_train, y_test):

    print('Training R^2 :', model.score(X_train, y_train))
    y_pred_train = model.predict(X_train)
    print('Training Root Mean Square Error', np.sqrt(metrics.
 ↪mean_squared_error(y_train, y_pred_train)))
    print('\n----------------\n')
    print('Testing R^2 :', model.score(X_test, y_test))
    y_pred_test = model.predict(X_test)
    print('Testing Root Mean Square Error', np.sqrt(metrics.
 ↪mean_squared_error(y_test, y_pred_test)))
```

[6]:
```
lm = LinearRegression()
lm.fit(X_train_transformed, y_train)
run_model(lm, X_train_transformed, X_test_transformed, y_train, y_test)
```

```
Training R^2 : 0.5371947100976313
Training Root Mean Square Error 52.21977472848369

----------------


Testing R^2 : 0.41797754631986483
Testing Root Mean Square Error 58.83589708889503
```

The model has not performed exceptionally well here, so we can try adding some additional features.

Let's go ahead and add a polynomial degree of up to 3.

[7]:
```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, interaction_only=False, include_bias=False)
X_poly_train = pd.DataFrame(poly.fit_transform(X_train_transformed),␣
 ↪columns=poly.get_feature_names(features.columns))
X_poly_test = pd.DataFrame(poly.transform(X_test_transformed), columns=poly.
 ↪get_feature_names(features.columns))
X_poly_train.head()
```

[7]:
```
        AGE       BMI        BP        S1        S2        S3        S4  \
0 -0.433522 -0.967597 -2.067847 -1.623215 -1.280312 -0.347527 -0.852832
1  1.117754 -0.516691  1.142458 -0.168101 -0.129601 -0.424950 -0.083651
2  1.350445  1.850570  1.427819  0.413945  0.764667 -1.044334  1.454710
3 -0.511086 -1.373413 -1.711146 -0.837453 -1.148802  1.278358 -1.622013
4 -0.743778  0.114579 -0.141664 -1.565010 -1.339491 -0.115257 -0.852832

         S5        S6  female  …      S4^2     S4 S5     S4 S6  S4 female  \
0 -1.095555 -1.006077     0.0  …  0.727322  0.934324  0.858015  -0.000000
1  0.543382 -0.831901     1.0  …  0.006998 -0.045455  0.069589  -0.083651
2  0.597504  1.519478     1.0  …  2.116182  0.869195  2.210400   1.454710
3 -0.796071 -0.918989     0.0  …  2.630925  1.291237  1.490612  -0.000000
4 -0.970101  0.648597     1.0  …  0.727322  0.827333 -0.553144  -0.852832

       S5^2     S5 S6  S5 female      S6^2  S6 female  female^2
0  1.200240  1.102213  -0.000000  1.012192  -0.000000       0.0
1  0.295264 -0.452040   0.543382  0.692060  -0.831901       1.0
2  0.357011  0.907894   0.597504  2.308813   1.519478       1.0
3  0.633729  0.731581  -0.000000  0.844541  -0.000000       0.0
4  0.941095 -0.629204  -0.970101  0.420678   0.648597       1.0

[5 rows x 65 columns]
```

As you can see, we now have 65 total columns! You can imagine that this model will greatly overfit to the data. Let's try it out with our training and test set.

[8]:
```
lr_poly = LinearRegression()
lr_poly.fit(X_poly_train, y_train)

run_model(lr_poly, X_poly_train, X_poly_test, y_train, y_test)
```

```
Training R^2 : 0.6237424326763868
Training Root Mean Square Error 47.084553723292274

----------------

Testing R^2 : 0.3688694444251669
```

```
Testing Root Mean Square Error 61.26777562691565
```

Clearly, the model has fit very well to the training data, but it has fit to a lot of noise. The testing R^2 is worse than the simple model we fit previously! It's time to get rid of some features to see if this improves the model.

1.4 Filter methods

Let's begin by trying out some filter methods for feature selection. The benefit of filter methods is that they can provide us with some useful visualizations for helping us gain an understanding about the characteristics of our data. To begin with, let's use a simple variance threshold to eliminate the features with low variance.

[9]:
```
from sklearn.feature_selection import VarianceThreshold

threshold_ranges = np.linspace(0, 2, num=6)

for thresh in threshold_ranges:
    print(thresh)
    selector = VarianceThreshold(thresh)
    reduced_feature_train = selector.fit_transform(X_poly_train)
    reduced_feature_test = selector.transform(X_poly_test)
    lr = LinearRegression()
    lr.fit(reduced_feature_train, y_train)
    run_model(lr, reduced_feature_train, reduced_feature_test, y_train, y_test)
    print('--------------------------------------------------------------------')
```

```
0.0
Training R^2 : 0.6237424326763871
Training Root Mean Square Error 47.08455372329226

----------------

Testing R^2 : 0.368869444425172
Testing Root Mean Square Error 61.2677756269154
--------------------------------------------------------------------
0.4
Training R^2 : 0.6035018897144957
Training Root Mean Square Error 48.33440733222434

----------------

Testing R^2 : 0.358080196285809
Testing Root Mean Square Error 61.789246194094176
--------------------------------------------------------------------
0.8
Training R^2 : 0.5894227238666293
Training Root Mean Square Error 49.18506972762005
```

```
----------------

Testing R^2 : 0.3640169603035095
Testing Root Mean Square Error 61.502855071460274
--------------------------------------------------------------------
1.2000000000000002
Training R^2 : 0.1991536009695497
Training Root Mean Square Error 68.69267841228015

----------------

Testing R^2 : 0.03625491787410018
Testing Root Mean Square Error 75.71006122152008
--------------------------------------------------------------------
1.6
Training R^2 : 0.1719075561672746
Training Root Mean Square Error 69.8514213627012

----------------

Testing R^2 : 0.09270595287961192
Testing Root Mean Square Error 73.45925856313265
--------------------------------------------------------------------
2.0
Training R^2 : 0.06445844090783526
Training Root Mean Square Error 74.24502865009542

----------------

Testing R^2 : 0.0420041242549255
Testing Root Mean Square Error 75.48389982682222
--------------------------------------------------------------------
```

Well, that did not seem to eliminate the features very well. It only does a little better than the base polynomial.

[10]:

```python
from sklearn.feature_selection import f_regression, mutual_info_regression,
 ↪SelectKBest
selector = SelectKBest(score_func=f_regression)
X_k_best_train = selector.fit_transform(X_poly_train, y_train)
X_k_best_test= selector.transform(X_poly_test)
lr = LinearRegression()
lr.fit(X_k_best_train ,y_train)
run_model(lr,X_k_best_train,X_k_best_test,y_train,y_test)
```

```
Training R^2 : 0.5229185029521006
Training Root Mean Square Error 53.01907218972858
```

```
----------------

Testing R^2 : 0.42499888052723567
Testing Root Mean Square Error 58.4799314703427
```

[11]:
```
selector = SelectKBest(score_func=mutual_info_regression)
X_k_best_train = selector.fit_transform(X_poly_train, y_train)
X_k_best_test= selector.transform(X_poly_test)
lr = LinearRegression()
lr.fit(X_k_best_train ,y_train)
run_model(lr,X_k_best_train,X_k_best_test,y_train,y_test)
```

```
Training R^2 : 0.4947424871473227
Training Root Mean Square Error 54.56224442488476

----------------

Testing R^2 : 0.41157350315844654
Testing Root Mean Square Error 59.15869978194653
```

1.5 Wrapper methods

Now let's use Recursive Feature elimination (RFE) to try out a wrapper method. You'll notice that scikit-learn has a built in `RFECV()` function, which automatically determines the optimal number of features to keep when it is run based off the estimator that is passed into it. Here it is in action:

[12]:
```
from sklearn.feature_selection import RFE, RFECV
from sklearn.linear_model import LinearRegression

rfe = RFECV(LinearRegression(),cv=5)
X_rfe_train = rfe.fit_transform(X_poly_train, y_train)
X_rfe_test = rfe.transform(X_poly_test)
lm = LinearRegression().fit(X_rfe_train, y_train)
run_model(lm, X_rfe_train, X_rfe_test, y_train, y_test)
print ('The optimal number of features is: ', rfe.n_features_)
```

```
Training R^2 : 0.37930327292162724
Training Root Mean Square Error 60.474956480110215

----------------

Testing R^2 : 0.37417761375281067
Testing Root Mean Square Error 61.009583057744294
The optimal number of features is:  10
```

With Recursive Feature Elimination, we went from an R^2 score of 0.368 to 0.374 (a tiny bit better). Let's see if we can improve upon these results even more by trying embedded methods.

1.6 Embedded methods

To compare to our other methods, we will use Lasso as the embedded method of feature selection. Luckily for us, sklearn has a built-in method to help us find the optimal features! It performs cross validation to determine the correct regularization parameter (how much to penalize our function).

[13]:
```
from sklearn.linear_model import LassoCV
lasso = LassoCV(max_iter=100000, cv=5)
lasso.fit(X_train_transformed, y_train)
run_model(lasso,X_train_transformed, X_test_transformed, y_train, y_test)
print('The optimal alpha for the Lasso Regression is: ', lasso.alpha_)
```

```
Training R^2 : 0.535154465585244
Training Root Mean Square Error 52.33475174961373

----------------

Testing R^2 : 0.4267044232634858
Testing Root Mean Square Error 58.39313677555575
The optimal alpha for the Lasso Regression is:  0.23254844944953376
```

Let's compare this to a model with all of the polynomial features included.

[14]:
```
lasso2 = LassoCV(max_iter=100000, cv=5)

lasso2.fit(X_poly_train, y_train)
run_model(lasso2, X_poly_train, X_poly_test, y_train, y_test)
print('The optimal alpha for the Lasso Regression is: ', lasso2.alpha_)
```

```
Training R^2 : 0.5635320505309954
Training Root Mean Square Error 50.71214913665365

----------------

Testing R^2 : 0.43065954589620015
Testing Root Mean Square Error 58.191363260874226
The optimal alpha for the Lasso Regression is:  1.7591437388826368
```

As we can see, the regularization had minimal effect on the performance of the model, but it did improve the RMSE for the test set ever so slightly! There are no set steps someone should take in order to determine the optimal feature set. In fact, now there are automated machine learning pipelines that will determine the optimal subset of features for a given problem. One of the most important and often overlooked methods of feature selection is using domain knowledge about a given area to either eliminate features or create new ones.

Additional Resources:

- Feature Selection
- An Introduction to Variable and Feature Selection

1.7 Summary

This lesson formalized the different types of feature selection methods and introduced some new techniques to you. You learned about filter methods, wrapper methods, and embedded methods as well as advantages and disadvantages to both.