Regularization with SciKit-Learn

Previously we created a new polynomial feature set and then applied our standard linear regression on it, but we can be smarter about model choice and utilize regularization.

Regularization attempts to minimize the RSS (residual sum of squares) *and* a penalty factor. This penalty factor will penalize models that have coefficients that are too large. Some methods of regularization will actually cause non useful features to have a coefficient of zero, in which case the model does not consider the feature.

Let's explore two methods of regularization, Ridge Regression and Lasso. We'll combine these with the polynomial feature set (it wouldn't be as effective to perform regularization of a model on such a small original feature set of the original X).

Imports

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Data and Setup

```
df = pd.read_csv("D:\\Study\\Programming\\python\\Python course from udemy\\Udemy - 2022 Python for Machine Learning & Data Science Masterclass\\01 - Introduction to Course\\1UNZIP-FOR-NOTEBOOKS-FINAL\\08-Linear-Regression-Models\\Advertising.csv")
df.head()
```

	TV	radio	newspaper	sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9

```
# Divide data in tranning and test
X = df.drop('sales',axis=1)
Y = df['sales']
```

Polynomial Conversion

```
from sklearn.preprocessing import PolynomialFeatures

polynomial_converter = PolynomialFeatures(degree = 3 , include_bias=False)

poly_features=polynomial_converter.fit_transform(X)

poly_features.shape
```

```
(200, 19)
```

```
from sklearn.model_selection import train_test_split
```

Train | Test Split

```
X_train, X_test, Y_train, Y_test =
train_test_split(poly_features,Y,test_size=.30, random_state=101)
```

```
X_train.shape
```

```
(140, 19)
```

Scaling the Data

While our particular data set has all the values in the same order of magnitude ($1000s of dollars spent), typically that won't be the case on a dataset, and since the mathematics behind regularized models will sum coefficients together, its important to standardize the features. Review the theory videos for more info, as well as a discussion on why we only **fit** to the training data, and **transform** on both sets separately.

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
# we are fitting on the train set only so that no information got leak
aboout test set
scaler.fit(X_train)
```

```
StandardScaler()
```

```
scaler_X_train = scaler.transform(X_train)
```

```
scaler_X_test = scaler.transform(X_test)
```

```
# Here values after scaled
scaler_X_train[0]
```

```
array([ 0.49300171, -0.33994238,  1.61586707,  0.28407363, -
0.02568776,
        1.49677566, -0.59023161,  0.41659155,  1.6137853 ,
0.08057172,
       -0.05392229,  1.01524393, -0.36986163,  0.52457967,
1.48737034,
       -0.66096022, -0.16360242,  0.54694754,  1.37075536])
```

```
# Here are the value before scaled
poly_features[0]
```

```
array([2.30100000e+02, 3.78000000e+01, 6.92000000e+01, 5.29460100e+04,
       8.69778000e+03, 1.59229200e+04, 1.42884000e+03, 2.61576000e+03,
       4.78864000e+03, 1.21828769e+07, 2.00135918e+06, 3.66386389e+06,
```

```
       3.28776084e+05, 6.01886376e+05, 1.10186606e+06, 5.40101520e+04,
       9.88757280e+04, 1.81010592e+05, 3.31373888e+05])
```

Ridge Regression

Make sure to view video lectures for full explanation of Ridge Regression and choosing an alpha.

```
from sklearn.linear_model import Ridge

# Here we choose alpha = 10 for that we have do cross validation
# Here we pick 10 as arbitrary
ridge_model = Ridge(alpha=10)

ridge_model.fit(scaler_X_train,Y_train)

Ridge(alpha=10)

test_prediction = ridge_model.predict(scaler_X_test)

from sklearn.metrics import mean_absolute_error, mean_squared_error

MAE = mean_absolute_error(Y_test,test_prediction)
MAE

0.5774404204714183

RMSE = np.sqrt(mean_squared_error(Y_test, test_prediction))
RMSE

0.8946386461319685
```

How did it perform on the training set? (This will be used later on for comparison)

```
# Traning Set performance
train_prediction = ridge_model.predict(scaler_X_train)
MAE = mean_absolute_error(Y_train,train_prediction)
MAE

0.5288348183025332
```

Choosing an alpha value with Cross-Validation

Review the theory video for full details.

```
from sklearn.linear_model import RidgeCV

# Choosing a scoring:
https://scikit-learn.org/stable/modules/model_evaluation.html
# Negative RMSE so all metrics follow convention "Higher is better"

# See all options: sklearn.metrics.SCORERS.keys()

# Here we are providing multiple alpha values to check which one is
```

```
better
# There is term cv= here by default its none means leave one out else
we can set desire number too
ridge_cv_model =
RidgeCV(alphas=(0.1,1.0,10.0),scoring='neg_mean_absolute_error')

# The more alpha options you pass, the longer this will take.
# Fortunately our data set is still pretty small
ridge_cv_model.fit(scaler_X_train,Y_train)

RidgeCV(scoring='neg_mean_absolute_error')

# This is alpha that perform best
ridge_cv_model.alpha_

0.1

from sklearn.metrics import SCORERS

# SCORERS is dictionary which have list of errors, we can call them by
keys as we do in ditionary
SCORERS.keys()
# Here we have neg_mean_absolute_error that is negative of mean_ab...
we will use it in RidgeCV above as scoring

dict_keys(['explained_variance', 'r2', 'max_error',
'matthews_corrcoef', 'neg_median_absolute_error',
'neg_mean_absolute_error', 'neg_mean_absolute_percentage_error',
'neg_mean_squared_error', 'neg_mean_squared_log_error',
'neg_root_mean_squared_error', 'neg_mean_poisson_deviance',
'neg_mean_gamma_deviance', 'accuracy', 'top_k_accuracy', 'roc_auc',
'roc_auc_ovr', 'roc_auc_ovo', 'roc_auc_ovr_weighted',
'roc_auc_ovo_weighted', 'balanced_accuracy', 'average_precision',
'neg_log_loss', 'neg_brier_score', 'positive_likelihood_ratio',
'neg_negative_likelihood_ratio', 'adjusted_rand_score', 'rand_score',
'homogeneity_score', 'completeness_score', 'v_measure_score',
'mutual_info_score', 'adjusted_mutual_info_score',
'normalized_mutual_info_score', 'fowlkes_mallows_score', 'precision',
'precision_macro', 'precision_micro', 'precision_samples',
'precision_weighted', 'recall', 'recall_macro', 'recall_micro',
'recall_samples', 'recall_weighted', 'f1', 'f1_macro', 'f1_micro',
'f1_samples', 'f1_weighted', 'jaccard', 'jaccard_macro',
'jaccard_micro', 'jaccard_samples', 'jaccard_weighted'])

test_prediction = ridge_cv_model.predict(scaler_X_test)

MAE = mean_absolute_error(Y_test,test_prediction)
MAE

0.42737748843352086

RMSE = np.sqrt(mean_squared_error(Y_test,test_prediction))
RMSE
```

```
0.6180719926924644
```

```
# Here we see there is not coeficent that zero or close to zero
ridge_cv_model.coef_
```

```
array([ 5.40769392,  0.5885865 ,  0.40390395, -6.18263924,
4.59607939,
       -1.18789654, -1.15200458,  0.57837796, -0.1261586 ,
2.5569777 ,
       -1.38900471,  0.86059434,  0.72219553, -0.26129256,
0.17870787,
        0.44353612, -0.21362436, -0.04622473, -0.06441449])
```

Lasso Regression

Lasso try to make some coeficent to be zero

LASSO : Least Absolute Shrinkage and Selection Operator

```
from sklearn.linear_model import LassoCV

#
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model
.LassoCV.html
# eps means alpha_min / alpha_max = 1e-3 that is 0.001 it will take
time to perform as we will put 0.1 or
                               # we can put max_iter = 10000 here we
limit our ilterations
# Number of alphas along the regularization path.
# If we increase n_alphas then it will take more time to compilation
# cv by default is none ,cv default value if None changed from 3-fold
to 5-fold.
lasso_cv_model = LassoCV(eps=0.1,n_alphas=100,cv=5 )
```

```
lasso_cv_model.fit(scaler_X_train,Y_train)
```

```
LassoCV(cv=5, eps=0.1)
```

```
lasso_cv_model.alpha_
```

```
0.4943070909225828
```

```
test_prediction = lasso_cv_model.predict(scaler_X_test)
```

```
MAE = mean_absolute_error(Y_test,test_prediction)
MAE
```

```
0.6541723161252854
```

```
RMSE = np.sqrt(mean_squared_error(Y_test,test_prediction))
RMSE
```

```
1.130800102276253
```

Elastic Net

Elastic Net combines the penalties of ridge regression and lasso in an attempt to get the best of both worlds!

```
from sklearn.linear_model import ElasticNetCV

# Here we have taken eps = 0.001 along we limit max_iter to 1 million
# Here l1_ratio is mostly .1,.5,.7,.9,.95,.99,1
elastic_model =
ElasticNetCV(l1_ratio=[.1,.5,.7,.9,.95,.99,1],eps=0.001,n_alphas=100,m
ax_iter=1000000)

elastic_model.fit(scaler_X_train,Y_train)

ElasticNetCV(l1_ratio=[0.1, 0.5, 0.7, 0.9, 0.95, 0.99, 1],
max_iter=1000000)

# To see all l1_ratio which we have taken
elastic_model.l1_ratio

[0.1, 0.5, 0.7, 0.9, 0.95, 0.99, 1]

# Best l1_ratio
elastic_model.l1_ratio_

1.0

elastic_model.alpha_

0.004943070909225827

test_prediction = elastic_model.predict(scaler_X_test)

MAE = mean_absolute_error(Y_test,test_prediction)
MAE

0.43350346185900673

RMSE = np.sqrt(mean_squared_error(Y_test,test_prediction))
RMSE

0.6063140748984039
```