

Brian Elinsky

Professor Lawrence Fulton, Ph.D.

2020SP_MSDS_422-DL_SEC55

18 April 2020

Assignment 2: Evaluating Regression Models

PROBLEM DESCRIPTION

Given a set of explanatory variables for a house, my business objective is to predict its market value. In this fictional scenario, I'm advising a real estate brokerage firm. The firm currently uses conventional methods to price houses. This model will complement those methods. The model will be used to help ensure that the company buys houses below market value, and sells them above market value, ensuring a profit.

RESEARCH DESIGN AND MODELING METHODS

My plan was to start by fitting a basic linear regression model using a 10-fold cross validation methodology. Then fit Lasso Regression, Ridge Regression, and Elastic Net Regression models all using the default hyperparameters and a 10-fold cross validation. Then I would select the most promising model for fine tuning. Fine tuning would involve a grid search over the hyperparameter space. With the final model selected, I would then test it on my test dataset.

DATA PREPARATION

I dropped categorical variables that were missing over 10% of the data. This included: 'Alley', 'FireplaceQu', 'PoolQC', 'Fence', 'MiscFeature'. For numeric features, I imputed missing values with medians. For categorical features, I imputed missing values with the

most frequent value. I encoded categorical features to dummy arrays. Numeric variables were scaled using a robust scaler.

RESULTS AND MODEL EVALUATION

The average root mean square error of my basic linear model is \$32,274.13. This is an OK model, but definitely not great. This is a good baseline to use to compare future models.

The ridge regression did perform slightly better than the standard linear regression, with a mean RMSE of \$30,746.70. The Lasso model had an average RMSE of \$31,465.81. The Elastic Net had an average RMSE of \$33,778.98. I decided to move forward with the Ridge Regression because it had the lowest RMSE.

I did a grid search over the ridge regression, varying the following parameters: alpha, normalize, max_iter, and tol. Testing the best model on the test dataset resulted in a RMSE of \$29,823.75. The Kaggle Log RMSE was 0.15529 and I ranked in the 61st percentile.

MANAGEMENT RECOMMENDATIONS

I would recommend deploying the final version of the Ridge Regression model to production. I would advise management to not replace the conventional pricing methods with this model, rather to use the model to supplement the existing methods. With a mean housing price of ~\$180k, a model with an RMSE of ~\$25k will give you a rough estimate of the housing price, but not a hyper-accurate one. Additional work could be done to improve the model. That could include treating the pre-processing steps as hyperparameters, or fitting a different class of model, like a K-nearest neighbors model.

```

1  # runall.py
2  from bin.library import *
3  from sklearn.preprocessing import OneHotEncoder, RobustScaler
4  from sklearn.pipeline import make_pipeline, FeatureUnion
5  from sklearn.linear_model import Ridge, Lasso, ElasticNet
6  from sklearn.impute import SimpleImputer
7  from sklearn.model_selection import cross_val_score, GridSearchCV
8  from sklearn.metrics import mean_squared_error
9  from scipy import stats
10
11 import pandas as pd
12 import numpy as np
13
14 # Load training data
15 data_file = get_dataset_file_path('2020-04-13', 'train.csv')
16 train = pd.read_csv(data_file)
17
18 # Remove label
19 X_train = train.drop(columns='SalePrice')
20 y_train = train['SalePrice'].copy()
21
22 # Variables to use in the model
23 x_cols = ['MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street', 'LotShape', 'LandContour', 'Utilities',
24           'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'OverallQual',
25           'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd',
26           'MasVnrType', 'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure',
27           'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
28           'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
29           'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual', 'TotRmsAbvGrd',
30           'Functional', 'Fireplaces', 'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea',
31           'GarageQual', 'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
32           'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'SaleType', 'SaleCondition']
33
34 # Build preprocessing pipeline
35 preprocess_pipeline = make_pipeline(
36     ColumnSelector(columns=x_cols),
37     CategoricalEncoder(),
38     FeatureUnion(transformer_list=[
39         ("numeric_features", make_pipeline(
40             TypeSelector(np.number),
41             SimpleImputer(missing_values=np.nan, strategy='median'),
42             RobustScaler()
43         )),
44         ("categorical_features", make_pipeline(
45             TypeSelector("category"),
46             SimpleImputer(strategy="most_frequent"),
47             OneHotEncoder()
48         ))
49     ])
50
51 # Preprocess data
52 X_train = preprocess_pipeline.fit_transform(X_train)
53
54 # Instantiate a simple Ridge Regression model and assess its accuracy using a 10-fold cross validation
55 ridge_reg = Ridge()
56 scores = cross_val_score(ridge_reg, X_train, y_train, scoring="neg_mean_squared_error", cv=10)
57 ridge_reg_rmse_scores = np.sqrt(-scores)
58 display_scores(ridge_reg_rmse_scores)
59
60 # Instantiate a simple Lasso Regression model and assess its accuracy using a 10-fold cross validation
61 lasso_reg = Lasso()
62 scores = cross_val_score(lasso_reg, X_train, y_train, scoring="neg_mean_squared_error", cv=10)
63 lasso_reg_rmse_scores = np.sqrt(-scores)
64 display_scores(lasso_reg_rmse_scores)
65
66 # Instantiate a simple Elastic Net Regression model and assess its accuracy using a 10-fold cross validation
67 elastic_net = ElasticNet()
68 scores = cross_val_score(elastic_net, X_train, y_train, scoring="neg_mean_squared_error", cv=10)
69 elastic_net_rmse_scores = np.sqrt(-scores)
70 display_scores(elastic_net_rmse_scores)
71
72 # Fine tune the Ridge Regression model using a randomized search cross validation
73 param_grid = [
74     {"alpha": [0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0],
75      "fit_intercept": [False],
76      "normalize": [True, False],
77      "max_iter": [10, 100, 1000, 1000],
78      "tol": [0.01, 0.001, 0.001],
79      "solver": ["cholesky"]}
80 ]

```

```

81 ]
82
83 grid_search = GridSearchCV(Ridge(), param_grid, cv=10, scoring="neg_mean_squared_error", n_jobs=4,
84                             return_train_score=True)
85 grid_search.fit(X_train, y_train)
86 print(grid_search.best_estimator_)
87
88 final_model = grid_search.best_estimator_
89
90 # Evaluate final model on training data
91 scores = cross_val_score(final_model, X_train, y_train, scoring="neg_mean_squared_error", cv=10)
92 ridge_reg_rmse_scores = np.sqrt(-scores)
93 display_scores(ridge_reg_rmse_scores)
94
95 # Evaluate final model on the test set
96
97 # Load test data
98 data_file = get_dataset_file_path('2020-04-13', 'test.csv')
99 X_test = pd.read_csv(data_file)
100 output = X_test["Id"]
101
102 # Preprocess data
103 X_test = preprocess_pipeline.transform(X_test)
104
105 # Make final predictions
106 final_predictions = final_model.predict(X_test)
107
108 # Output predictions
109 final_predictions = pd.Series(final_predictions, name="SalePrice")
110 output = pd.concat([output, final_predictions], axis=1)
111 output.to_csv("predictions.csv", index=False)
112
113

```

```

1  # library.py
2  import os
3  from sklearn.base import BaseEstimator, TransformerMixin
4  import pandas as pd
5
6
7  def get_dataset_file_path(date: object, filename: object) -> object:
8      """Produces a filepath for the dataset.
9
10     :parameter date (string): The date folder name. Ex: "2020-02-05"
11     :parameter filename (string): The csv filename.
12     :returns filepath (string): The filepath for the dataset.
13
14     Example:
15
16     project_root
17     |--- README.md
18     |--- data
19     |   |--- 2020-04-13
20     |   |   |--- README.md
21     |   |   |--- data_description.txt
22     |   |   |--- test.csv
23     |   |   |--- train.csv
24     |--- docs
25     |--- requirements.yml
26     |--- results
27     |   |--- 2020-04-13
28     |   |   |--- runall.py
29
30     The function is called from the 'runall.py' file.
31     >> get_data_file_path('2020-04-13', 'train.csv')
32     '-/project_root/data/2020-04-13/train.csv'
33     """
34
35     basepath = os.path.abspath('')
36     filepath = os.path.abspath(os.path.join(basepath, "..", "..")) + "/data/" + date + "/" + filename
37     return filepath
38
39  def convert_object_to_categorical(df):
40      """Converts columns in a pandas dataframe of dtype 'object' to dtype 'categorical.' This is a destructive method
41
42     :parameter df (pandas dataframe): A pandas dataframe
43     """
44     assert isinstance(df, pd.DataFrame)
45
46     object_columns = df.select_dtypes(include='object').columns.tolist()
47     for obj_col in object_columns:
48         df[obj_col] = df[obj_col].astype('category')
49
50  def display_scores(scores):
51     print("Scores:", scores)
52     print("Mean:", scores.mean())
53     print("Standard deviation:", scores.std())
54
55
56  class ColumnSelector(BaseEstimator, TransformerMixin):
57     def __init__(self, columns):
58         self.columns = columns
59
60     def fit(self, X, y=None):
61         return self
62
63     def transform(self, X):
64         assert isinstance(X, pd.DataFrame)
65
66         try:
67             return X[self.columns]
68         except KeyError:
69             cols_error = list(set(self.columns) - set(X.columns))
70             raise KeyError("The DataFrame does not include the columns: %s" % cols_error)
71
72
73  class TypeSelector(BaseEstimator, TransformerMixin):
74     """Returns a dataframe that only includes columns of the specified datatype.
75
76     :parameter dtype (string): The datatype to filter on.
77     """
78     def __init__(self, dtype):

```

```

79         self.dtype = dtype
80
81     def fit(self, X, y=None):
82         return self
83
84     def transform(self, X):
85         assert isinstance(X, pd.DataFrame)
86         return X.select_dtypes(include=self.dtype)
87
88
89 class CategoricalEncoder(BaseEstimator, TransformerMixin):
90     """Converts columns in a pandas dataframe of dtype 'object' to dtype 'categorical.'
91     """
92     def fit(self, X, y=None):
93         return self
94
95     def transform(self, X):
96         assert isinstance(X, pd.DataFrame)
97
98         object_columns = X.select_dtypes(include='object').columns.tolist()
99         for obj_col in object_columns:
100             X[obj_col] = X[obj_col].astype('category')
101         return X
102
103
104 class DropNaNs(BaseEstimator, TransformerMixin):
105     """Drops all NaNs in a pandas dataframe.
106     """
107     def fit(self, X, y=None):
108         return self
109
110     def transform(self, X):
111         assert isinstance(X, pd.DataFrame)
112
113         return X.dropna(inplace=True)
114

```