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 (Account User ID 4810027).

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 hyperparamters, or fitting a different class of model, like a K-nearest neighbors model.

```python
# runall.py
from bin.library import *
from sklearn.preprocessing import OneHotEncoder, RobustScaler
from sklearn.pipeline import make_pipeline, FeatureUnion
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.impute import SimpleImputer
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.metrics import mean_squared_error
from scipy import stats

import pandas as pd
import numpy as np

# Load training data
data_file = get_dataset_file_path('2020-04-13', 'train.csv')
train = pd.read_csv(data_file)

# Remove label
X_train = train.drop(columns='SalePrice')
y_train = train['SalePrice'].copy()

# Variables to use in the model
x_cols = ['MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street', 'LotShape', 'LandContour', 'Utilities',
          'LotConfig', 'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'OverallQual',
          'OverallCond', 'YearBuilt', 'YearRemodAdd', 'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd',
          'MasVnrType', 'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual', 'BsmtCond', 'BsmtExposure',
          'BsmtFinType1', 'BsmtFinSF1', 'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
          'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea', 'BsmtFullBath',
          'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual', 'TotRmsAbvGrd',
          'Functional', 'Fireplaces', 'GarageType', 'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea',
          'GarageQual', 'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF', 'EnclosedPorch', '3SsnPorch',
          'ScreenPorch', 'PoolArea', 'MiscVal', 'MoSold', 'YrSold', 'SaleType', 'SaleCondition']

# Build preprocessing pipeline
preprocess_pipeline = make_pipeline(
    ColumnSelector(columns=x_cols),
    CategoricalEncoder(),
    FeatureUnion(transformer_list=[
        ("numeric_features", make_pipeline(
            TypeSelector(np.number),
            SimpleImputer(missing_values=np.nan, strategy='median'),
            RobustScaler()
        )),
        ("categorical_features", make_pipeline(
            TypeSelector("category"),
            SimpleImputer(strategy="most_frequent"),
            OneHotEncoder()
        ))
    ])
)

# Preprocess data
X_train = preprocess_pipeline.fit_transform(X_train)

# Instantiate a simple Ridge Regression model and assess its accuracy using a 10-fold cross validation
ridge_reg = Ridge()
scores = cross_val_score(ridge_reg, X_train, y_train, scoring="neg_mean_squared_error", cv=10)
ridge_reg_rmse_scores = np.sqrt(-scores)
display_scores(ridge_reg_rmse_scores)

# Instantiate a simple Lasso Regression model and assess its accuracy using a 10-fold cross validation
lasso_reg = Lasso()
scores = cross_val_score(lasso_reg, X_train, y_train, scoring="neg_mean_squared_error", cv=10)
lasso_reg_rmse_scores = np.sqrt(-scores)
display_scores(lasso_reg_rmse_scores)

# Instantiate a simple Elastic Net Regression model and assess its accuracy using a 10-fold cross validation
elastic_net = ElasticNet()
scores = cross_val_score(elastic_net, X_train, y_train, scoring="neg_mean_squared_error", cv=10)
elastic_net_reg_rmse_scores = np.sqrt(-scores)
display_scores(elastic_net_reg_rmse_scores)

# Fine tune the Ridge Regression model using a randomized search cross validation
param_grid = [
    {"alpha": [0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0],
     "fit_intercept": [False],
     "normalize": [True, False],
     "max_iter": [10, 100, 1000, 1000],
     "tol": [0.01, 0.001, 0.001],
     "solver": ["cholesky"]}
```

```
]

grid_search = GridSearchCV(Ridge(), param_grid, cv=10, scoring="neg_mean_squared_error", n_jobs=4,
                           return_train_score=True)
grid_search.fit(X_train, y_train)
print(grid_search.best_estimator_)

final_model = grid_search.best_estimator_

# Evaluate final model on training data
scores = cross_val_score(final_model, X_train, y_train, scoring="neg_mean_squared_error", cv=10)
ridge_reg_rmse_scores = np.sqrt(-scores)
display_scores(ridge_reg_rmse_scores)

# Evaluate final model on the test set

# Load test data
data_file = get_dataset_file_path('2020-04-13', 'test.csv')
X_test = pd.read_csv(data_file)
output = X_test["Id"]

# Preprocess data
X_test = preprocess_pipeline.transform(X_test)

# Make final predictions
final_predictions = final_model.predict(X_test)

# Output predictions
final_predictions = pd.Series(final_predictions, name="SalePrice")
output = pd.concat([output, final_predictions], axis=1)
output.to_csv("predictions.csv", index=False)
```

```python
# libary.py
import os
from sklearn.base import BaseEstimator, TransformerMixin
import pandas as pd


def get_dataset_file_path(date: object, filename: object) -> object:
    """Produces a filepath for the dataset.

    :parameter date (string): The date folder name.  Ex: "2020-02-05"
    :parameter filename (string): The csv filename.
    :returns filepath (string): The filepath for the dataset.

    Example:

    project_root
    ├── README.md
    ├── data
    │   └── 2020-04-13
    │       ├── README.md
    │       ├── data_description.txt
    │       ├── test.csv
    │       └── train.csv
    ├── docs
    ├── requirements.yml
    └── results
        └── 2020-04-13
            └── runall.py

    The function is called from the 'runall.py' file.
    >> get_data_file_path('2020-04-13', 'train.csv')
    '~/project_root/data/2020-04-13/train.csv'
    """

    basepath = os.path.abspath('')
    filepath = os.path.abspath(os.path.join(basepath, "..", "..")) + "/data/" + date + "/" + filename
    return filepath

def convert_object_to_categorical(df):
    """Converts columns in a pandas dataframe of dtype 'object' to dtype 'categorical.'  This is a destructive method

    :parameter df (pandas dataframe): A pandas dataframe
    """
    assert isinstance(df, pd.DataFrame)

    object_columns = df.select_dtypes(include='object').columns.tolist()
    for obj_col in object_columns:
        df[obj_col] = df[obj_col].astype('category')

def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())


class ColumnSelector(BaseEstimator, TransformerMixin):
    def __init__(self, columns):
        self.columns = columns

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        assert isinstance(X, pd.DataFrame)

        try:
            return X[self.columns]
        except KeyError:
            cols_error = list(set(self.columns) - set(X.columns))
            raise KeyError("The DataFrame does not include the columns: %s" % cols_error)


class TypeSelector(BaseEstimator, TransformerMixin):
    """Returns a dataframe that only includes columns of the specified datatype.

    :parameter dtype (string):  The datatype to filter on.
    """
    def __init__(self, dtype):
```

```python
        self.dtype = dtype

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        assert isinstance(X, pd.DataFrame)
        return X.select_dtypes(include=self.dtype)


class CategoricalEncoder(BaseEstimator, TransformerMixin):
    """Converts columns in a pandas dataframe of dtype 'object' to dtype 'categorical.'
    """
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        assert isinstance(X, pd.DataFrame)

        object_columns = X.select_dtypes(include='object').columns.tolist()
        for obj_col in object_columns:
            X[obj_col] = X[obj_col].astype('category')
        return X


class DropNaNs(BaseEstimator, TransformerMixin):
    """Drops all NaNs in a pandas dataframe.
    """
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        assert isinstance(X, pd.DataFrame)

        return X.dropna(inplace=True)
```