Linear Regression

Kannan Singaravelu, CQF

June 2023

1 Linear Regression

Linear regression is an approach we use to model the relationship between a scalar dependent variable y and one or more explanatory variables (aka independent variables) denoted by X. In machine learning parlance, these explanatory variables or predictors are called **features** and target or the dependent variable **labels**. Regression modesl are models which predict a continuous outcome.

Univariate Regression

The simplest possible relationship between two variables is a *univariate regression* when there is only one feature to be explained.

$$y \approx w_0 + w_1 x_1$$

Multivariate Regression

If there multiple features that are to be mapped to a target variable, then it is a *multivariate regression* problem.

$$y \approx w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots + w_n x_n$$

or more generally

$$y_i \approx \sum_j w_j X_{ij}$$

where i labels different **observations** and j labels different **features**.

If we can measure some (x_i, y_i) pairs, we could calculate our *model parameters* w. Then we could predict y in the future based on x, or even try to influence y in the future by controlling x. This is achieved by minimizing the mean squared error,

$$L = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - w_0 - \sum_{j=1}^{p} x_{ij} w_j \right)^2$$

2 The workflow

We'll employ the following linear regressor from `scikit-learn` for stock / equity index price prediction.

- Linear Regression
- Lasso Regression
- Ridge Regression
- Elastic Net Regression

Steps	Workflow	Remarks
Step 1	Ideation	Predict next trading day index price from the given dataset
Step 2	Data Collection	Load the dataset from my github page
Step 3	Exploratory Data Analysis	Study summary statistics
Step 4	Cleaning Dataset	Data already cleaned, no further imputation required
Step 5	Transformation	Perform feature scaling based on EDA
Step 6	Modeling	Building and training linear regressor
Step 7	Metrics	Validating the model performance using score method

Import Libraries

[ ]:
```
# Data Manipulation
import numpy as np
import pandas as pd

# Visualizaiton
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('seaborn')
sns.set(rc={'figure.figsize': (20, 8)})

# Data Preprocessing
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import (StandardScaler,
                                   MinMaxScaler)

from sklearn.pipeline import Pipeline

# Regressor
from sklearn.linear_model import (LinearRegression,
                                  Lasso,
                                  Ridge,
                                  ElasticNet)
```

```
# Metrics
from sklearn.metrics import mean_squared_error

# set display options
pd.options.display.float_format = "{:,.4f}".format
pd.set_option('display.max_columns', 100)

# Ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

2.1 Load Data

[ ]:
```
# load nifty index data
df = pd.read_csv('https://raw.githubusercontent.com/kannansingaravelu/datasets/
 ↪main/niftyindex.csv',
                 index_col=0,
                 dayfirst=True)
df
```

2.2 EDA of Original dataset

[ ]:
```
# descriptive statistics
df.describe()
```

2.3 Cleaning & Imputation

Data is already cleaned. No further processing or imputation required.

[ ]:
```
# check for missing values
df.isnull().sum()
```

2.4 Feature Engineering

Features are also known as an independent variable which are used to determine the value of the target variable. We will create features and target (label) from the raw dataset.

[ ]:
```
# create features
def create_features(frame):
    df = frame.copy()
    multiplier = 2

    # features
    df['OC'] = df['Close'] / df['Open'] - 1
    df['HC'] = df['High'] / df['Low'] - 1
    df['GAP'] = df['Open'] / df['Close'].shift(1) - 1
    df['RET'] = np.log(df['Close'] / df['Close'].shift(1))
```

```
    for i in [7, 14, 28]:
        df['PCHG' + str(i)] = df['Close'].pct_change(i)
        df['VCHG' + str(i)] = df['Volume'].pct_change(i)
        df['RET' + str(i)] = df['RET'].rolling(i).sum()
        df['MA' + str(i)] = df['Close'] / df['Close'].rolling(i).mean()
        df['VMA' + str(i)] = df['Volume'] / df['Volume'].rolling(i).mean()
        df['OC'+ str(i)] = df['OC'].rolling(i).mean()
        df['HC'+ str(i)] = df['HC'].rolling(i).mean()
        df['GAP'+ str(i)] = df['GAP'].rolling(i).mean()
        df['STD'+ str(i)] = df['RET'].rolling(i).std()
        df['UB'+str(i)] = df['Close'].rolling(i).mean() + df['Close'].rolling(i).
↪std() * multiplier
        df['LB'+str(i)] = df['Close'].rolling(i).mean() - df['Close'].rolling(i).
↪std() * multiplier

    # drop NaN values
    df['Label'] = df['Close'].shift(-1)
    df.drop(['Open', 'High', 'Low', 'Close', 'Volume'], axis=1, inplace=True)
    df.dropna(inplace=True)
    return df
```

2.4.1 Feature Specification

Features or Predictors are also known as an independent variable which are used to determine the value of the target variable. We will derive a features set from the original dataset.

[ ]:
```
# features
df1 = create_features(df)
display(df1.shape)

# verify the output
df1.head(2)
```

2.4.2 Target or Label Definition

Label or the target variable is also known as the dependent variable. Here, the target variable is the closing price of the index the next trading day.

[ ]:
```
# label
y = np.array(df1['Label'])
y
```

2.4.3 Feature Selection

Reduce the number of input variables that are believed to be most useful to a model. We use correlation measures which is one the filter methods to address multicollinearity among features.

[ ]:
```
# drop label from dataframe
df1.drop('Label', axis=1, inplace=True)

# remove features that are highly correlated
sns.heatmap(df1.corr()>0.9,
            annot=True,
            annot_kws={"size": 8},
            fmt=".2f",
            linewidth=.5,
            cmap="coolwarm",
            cbar=True); #cmap="crest", virids, magma

plt.title('Features Set Correlations');
```

[ ]:
```
# remove the first feature that is correlated with any other feature
def correlated_features(data, threshold=0.9):
    col_corr = set()
    corr_matrix = df1.corr()
    for i in range(len(corr_matrix.columns)):
            for j in range(i):
                if abs(corr_matrix.iloc[i, j]) > threshold:
                    colname = corr_matrix.columns[i]
                    col_corr.add(colname)
    return col_corr
```

[ ]:
```
# total correlated features
drop_correlated_features = correlated_features(df1)

# drop the highly correlated features
X = df1.drop(drop_correlated_features, axis=1)

# record feature names
feature_names = X.columns
```

[ ]:
```
# display the new features set
X
```

2.4.4 EDA for Feature Scaling

The choice of scaling techniques to be used should come from the Exploratory Data Analysis of features set.

[ ]:
```
# decide which scaling to use
X.describe()
```

[ ]:
```
# study the distribution
# fig, ax = plt.subplots(figsize=(14,8))
sns.boxplot(x='variable', y='value', data=pd.melt(X))
```

```
plt.xlabel(' ')
plt.title('Boxplot of Features');
```

3 Regression Class

We'll now create a Regression class for linear estimators : Linear, Lasso, Ridge and ElasticNet

[ ]:
```
class Regression:
    def __init__(self, X, y, testsize=0.20):

        self.X = X
        self.y = y
        self.testsize = testsize

        # split training and testing dataset
        self.X_train, self.X_test, self.y_train, self.y_test =␣
↪train_test_split(self.X,
                                                                    ␣
↪self.y,
                                                                    ␣
↪test_size=self.testsize,
                                                                    ␣
↪random_state=0,
                                                                    ␣
↪shuffle=False)

    # fit and predict
    def fit_predict(self, estimator, transformer, alpha=None, l1_ratio=None):

        try:
            # subsume estimators and transformer into a pipeline
            model = Pipeline([
            ('scaler', transformer),
            ('regressor', estimator)
            ])

            # fit/train model
            model.fit(self.X_train, self.y_train)

            # predict lables
            y_pred = model.predict(self.X_test)

        except Exception as e:
            print(str(e))
```

```
        return model, model['regressor'].intercept_, model['regressor'].coef_,␣
↪y_pred

    # evaluate metrics
    def eval_metrics(self, model, y_pred):

        # evaluate metrics
        mse = mean_squared_error(self.y_test, y_pred, squared=True)
        rmse = mean_squared_error(self.y_test, y_pred, squared=False)
        r2train = model.score(self.X_train, self.y_train)
        r2test = model.score(self.X_test, self.y_test)

        return mse, rmse, r2train, r2test

    # plot coefficients as a function of lambda
    def plot_coeff(self, modelname):

        coef = []
        model = Regression(self.X, self.y, 0.20)
        alpha_range = np.logspace(2, -2, 200)

        if modelname == 'Lasso':
            for i in alpha_range:
                coef.append(model.fit_predict(Lasso(alpha=i, random_state=0),␣
↪MinMaxScaler())[2])

        elif modelname == 'Ridge':
            alpha_range = np.logspace(6, -2, 200)
            for i in alpha_range:
                coef.append(model.fit_predict(Ridge(alpha=i, random_state=0),␣
↪MinMaxScaler())[2])

        elif modelname == 'ElasticNet':
            for i in alpha_range:
                coef.append(model.fit_predict(ElasticNet(alpha=i,␣
↪random_state=0), MinMaxScaler())[2])

        # Plot Coefficients
        fig = plt.figure(figsize=(20,8))
        ax = plt.axes()

        ax.plot(alpha_range, coef)
        ax.set_xscale('log')
        ax.legend(feature_names, loc=0)
        # ax.set_xlim(ax.get_xlim()[::-1])  # reverse axis
        ax.set_title(f'{modelname} coefficients as a function of the␣
↪regularization')
```

```
        ax.set_xlabel('$\lambda$')
        ax.set_ylabel('$\mathbf{w}$')

        return plt.show()
```

3.1 Linear Regression

In linear regression, the model coefficients are selected by minimizing a loss function. First, we instantiate the class object and pass the estimator and transformer to fit and predict the model.

[ ]:
```
# instantiate
lr = Regression(X, y)

# fit Linear Regression
lr_model , lr_intercept, lr_coef, lr_y_pred = lr.fit_predict(LinearRegression(),␣
 ↪MinMaxScaler())
print(f"\n Model: {lr_model} \n Intercept: {lr_intercept} \n Coefficients: \n␣
 ↪{lr_coef}")
```

4 Regularized Regression

If the coeficients are too large in linear regression, it can lead to overfitting the model. We do regularization to over come such issues that penalises large coefficients. Lasso and Ridge regression are penalty regression that prevent over-fitting from the simple linear regression.

4.1 LASSO

The Least Absolute Shrinkage and Selection Operator (LASSO) is a variation of linear regression. In Lasso, the loss function is minimized by limiting the sum of absolute values of the model coefficients where the L1 penalty term is added to the Mean Square Error (MSE). The L1 penalty term not only shrinks the coefficents, but shrinks some of them to zero and that is very useful for feature selection.

$$L = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - w_0 - \sum_{j=1}^{p} x_{ij}w_j\right)^2 + \lambda\sum_{j=1}^{p} \mid w_j \mid$$

where, λ is the regularization penalty.

[ ]:
```
# instantiate
lasso = Regression(X,y)

# fit Lasso
lasso_model , lasso_intercept, lasso_coef, lasso_y_pred = lasso.
 ↪fit_predict(Lasso(alpha=0.3, random_state=0), MinMaxScaler())
print(f"\n Model: {lasso_model} \n Intercept: {lasso_intercept} \n Coefficients:␣
 ↪\n {lasso_coef}")
```

[ ]:
```
# plot coefficients
lasso.plot_coeff('Lasso')
```

4.2 Ridge

In ridge regression, the cost function is altered by adding a L2 penality equivalent to square of the magnitude of the coefficients. The Ridge regression shrinks the coefficients and helps to reduce the multi-collinearity. As seen from the above graph, when $\lambda \to 0$, the cost function becomes similar to the linear regression cost function.

$$L = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - w_0 - \sum_{j=1}^{p} x_{ij}w_j\right)^2 + \lambda\sum_{j=1}^{p} w_j^2$$

[ ]:
```
# instantiate
ridge = Regression(X,y)

# fit Ridge
ridge_model , ridge_intercept, ridge_coef, ridge_y_pred = ridge.
 ↪fit_predict(Ridge(alpha=1, random_state=0), MinMaxScaler())
print(f"\n Model: {ridge_model} \n Intercept: {ridge_intercept} \n Coefficients:␣
 ↪\n {ridge_coef}")
```

[ ]:
```
# plot coefficients
ridge.plot_coeff('Ridge')
```

4.3 ElasticNet Regression

ElasticNet combines the properties of both Lasso and Ridge regression. It penalizes the model using both the L1 and L2 norm.

$$L = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - w_0 - \sum_{j=1}^{p} x_{ij}w_j\right)^2 + \lambda\left(\frac{1-\alpha}{2}\sum_{j=1}^{p} w_j^2 + \alpha\sum_{j=1}^{p} \mid w_j \mid\right)$$

where $0 \leq \alpha \leq 1$. Elastic net is same as lasso when $\alpha = 1$ and ridge when $\alpha = 0$. For other values of α, the penalty term interpolate between L1 and L2 norm of the coefficient.

[ ]:
```
# instantiate
elasticnet = Regression(X,y)

# fit ElasticNet
elasticnet_model , elasticnet_intercept, elasticnet_coef, elasticnet_y_pred =␣
 ↪elasticnet.fit_predict(ElasticNet(alpha=0.1, l1_ratio=1e-10, random_state=0),␣
 ↪MinMaxScaler())
print(f"\n Model: {elasticnet_model} \n Intercept: {elasticnet_intercept} \n␣
 ↪Coefficients: \n {elasticnet_coef}")
```

[ ]:
```
# plot coefficients
elasticnet.plot_coeff('ElasticNet')
```

5 Model Comparision

[ ]:
```
# compare model coefficients
coef_df = pd.DataFrame({
    'LR': lr_coef,
    'Lasso': lasso_coef,
    'Ridge': ridge_coef,
    'ElasticNet': elasticnet_coef
}, index = feature_names)

coef_df
```

[ ]:
```
# compare evaluation metrics
eval_df = pd.DataFrame({
    'LR': lr.eval_metrics(lr_model, lr_y_pred),
    'Lasso': lasso.eval_metrics(lasso_model, lasso_y_pred),
    'Ridge': ridge.eval_metrics(ridge_model, ridge_y_pred),
    'ElasticNet': elasticnet.eval_metrics(elasticnet_model, elasticnet_y_pred)
}, index = ['MSE', 'RMSE', 'R2_train', 'R2_test'])

eval_df
```

6 References

- Scikit-learn Documentation
- Scikit-learn API Reference
- Python Resources

June 2023, Certificate in Quantitative Finance.