Supervised Learning: Regression

DS 8015

Outline

1. Linear Regression Approaches

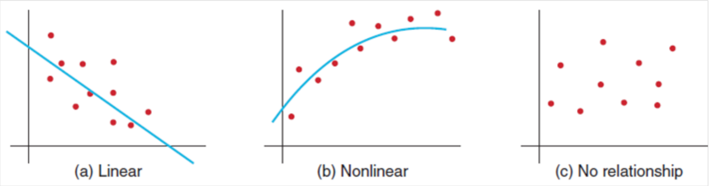
2. Python Implementations

Regression Analysis

- Regression analysis is a tool for building statistical models that characterize relationships among a dependent variable and one or more independent variables, all of which are numerical (continuous).

- Simple linear regression involves a single independent variable.

- Multiple regression involves two or more independent variables.



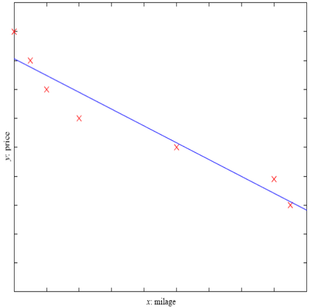
Linear Regression Approaches

SIMPLE LINEAR REGRESSION - 1

- Many statistical learning techniques can be seen as an extension of linear regression.
- Assume we only have one variable and one target. Then, linear regression is expressed as:

$$Y = \beta_0 + \beta_1 X$$

  - β_0: intercept, β_1: regression coefficient
  - We need to estimate β values to make predictions with our model.
  - How do we estimate β-values?



A training dataset of used cars
(x: mileage, y: car price)
The function fitted

$$y = \beta_0 + \beta_1 x$$

Simple Linear regression - 2

- To find the parameters, we need to minimize the **least squares** or the **sum of squared errors**.

  ⇒ The linear model will not predict all the data accurately, meaning that there is a difference between the actual value and the prediction.

- The error is calculated with:

  $$e_i = y_i - \hat{y}_i$$

- Why are the errors squared?
  - The prediction can be either above or below the true value, resulting in a negative or positive difference respectively.

SIMPLE LINEAR REGRESSION - 3

$$X = \{x_i, y_i\},\ y_i \in \mathbb{R}$$

$$y_i = g(x_i) + \epsilon$$

$$g(x) = \hat{y} = \beta_0 + \beta_1 x$$

$$E[g|X] = \tfrac{1}{n}\sum_{i=1}^{n}\left[y_i - g(x_i)\right]^2 \rightarrow \text{Expected error for regression function } g(x)$$

$$E[\beta_0, \beta_1|X] = \tfrac{1}{n}\sum_{i=1}^{n}\left[y_i - (\beta_0 + \beta_1 x_i)\right]^2 \rightarrow \text{Take derivative, set it to 0}$$

$$\tfrac{\partial E[\beta_0,\beta_1|X]}{\partial \beta_0} = \sum_{i=1}^{n} 2\big(y_i - (\beta_1 x_i + \beta_0)\big)\cdot(-1) = 0 \ldots \rightarrow \text{solve for } \beta_0$$

$$\tfrac{\partial E[\beta_0,\beta_1|X]}{\partial \beta_1} = \sum_{i=1}^{n} -2\big(y_i - (\beta_1 x_i + \beta_0)\big)\cdot x_i = 0 \ldots \rightarrow \text{solve for } \beta_1$$

- Using least squares estimator method, β-values are calculated as follows:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2}, \qquad \hat{\beta}_0 = \bar{y} - \beta_1 \bar{x}$$

$\Rightarrow$ $\bar{x}$ and $\bar{y}$ represent the mean.

SIMPLE LINEAR REGRESSION - 4

- Now that you have coefficients, how can you tell if they are relevant to predict your target?
- The best way is to find the *p*-value.

  ⇒ The *p*-value is used to quantify statistical significance; it allows to tell whether the null hypothesis is to be rejected or not.
- The null hypothesis?
  - For any modelling task, the hypothesis is that there is some correlation between the features and the target.
  - The null hypothesis is therefore the opposite: there is no correlation between the features and the target.
- Finding the *p*-value for each coefficient will tell if the variable is statistically significant to predict the target.

  ⇒ If the *p*-value is less than 0.05, there is a strong relationship between the variable and the target.

SIMPLE LINEAR REGRESSION - 5

- How do you know if your linear model is any good?
- To assess that, we usually use the RSE (residual standard error) and the R^2 statistic.

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \qquad TSS = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

$$RSE = \sqrt{\frac{1}{n-2} RSS}, \qquad R^2 = 1 - \frac{RSS}{TSS}$$

- The lower the residual errors, the better the model fits the data (in this case, the closer the data is to a linear relationship).
- As for the R^2 metric, it measures the proportion of variability in the target that can be explained using a feature X.
  - Therefore, assuming a linear relationship, if feature X can explain (predict) the target, then the proportion is high and the R^2 value will be close to 1.
  - If the opposite is true, the R^2 value is then closer to 0.

Simple Linear regression - Example

- The data regarding the production of wheat in tons (X) and the price of the kilo of flour in dollars (Y) in the decade of the 80's in US were:

Wheat Production	30	28	32	25	25	25	22	24	35	40
Flour price	25	30	27	40	42	40	50	45	30	25

Fit the regression line using the method of least squares. What is the $RSE = \sqrt{\frac{1}{n-2}\left(\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\right)}$ of the fitted regression line?

- Solution: $$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2} = \frac{9734 - 10 \times 28.6 \times 35.4}{8468 - 10 \times 28.6^2} = -1.35$$

$$\hat{\beta}_0 = \bar{y} - \beta_1 \bar{x} = 35.4 + 1.35 \times 28.6 = 74.12$$

The regression line is: $\hat{y} = 74.12 - 1.35x$

MULTIPLE LINEAR REGRESSION - 1

- In real life, there will never be a single feature to predict a target.

  $\Rightarrow$ Solution is to perform multiple linear regression.

- Multiple linear regression equation (with m features):

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_m X_m$$

- How to assess the relevancy of a predictor:
  - The F-statistic (n is # of data pts, m is # of predictors):

$$F = \frac{\frac{TSS-RSS}{m}}{\frac{RSS}{n-m-1}}$$

  - Here, the F-statistic is calculated for the overall model, whereas the p-value is specific to each predictor.
    - ✓ If there is a strong relationship, F will be much larger than 1.
    - ✓ Otherwise, it will be approximately equal to 1.

MULTIPLE LINEAR REGRESSION - 2

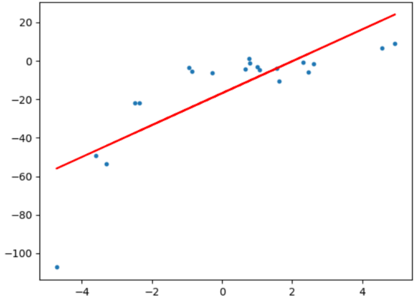
- How to assess the accuracy of the model:
  - R^2 can be used.

    $\Rightarrow$ Adding more predictors will always increase the R^2 value
- Adding interaction:
  - Having multiple predictors in a linear model means that some predictors may have an influence on other predictors.
  - For example, you want to predict the salary of a person, knowing her age and number of years spent in school. Of course, the older the person, the more time that person could have spent in school.
  - Consider this very simple example with 2 predictors:

    $$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2$$
  - As a general rule, if we include the interaction model, we should include the individual effect of a feature, even if its p-value is not significant (hierarchical principle).

POLYNOMIAL REGRESSION - 1

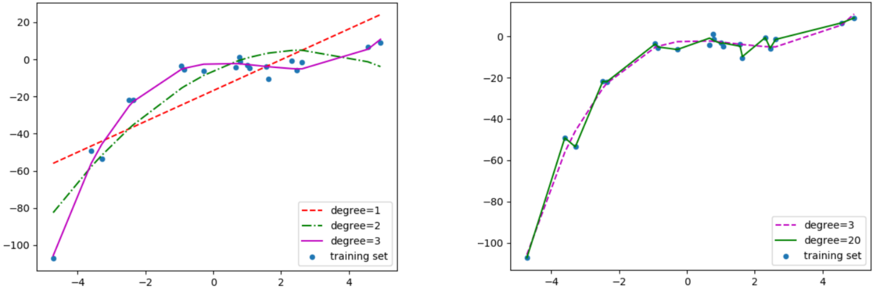
- A simple linear regression would not be the best fit for many data sets (underfitting).



- To overcome under-fitting, we need to increase the complexity of the model.
- To generate a higher order equation we can add powers of the original features as new features. The linear model, $y = \beta_0 + \beta_1 x$ can be transformed to $y = \beta_0 + \beta_1 x + \beta_2 x^2$

  $\Rightarrow$ This is still considered to be linear model as the coefficients/weights associated with the features are still linear. x^2 is only a feature. However the curve that we are fitting is quadratic in nature.

Polynomial Regression - 2



- It is clear from the plots that the quadratic curve is able to fit the data better than the linear line.
- Higher order polynomials seems to be fitting even better (overfitting!)

Python Implementations

Performance metrics for Regression

- Root mean squared error (RMSE):

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(|y_i - f(x_i)|)^2}.$$

- Median Absolute Error:

$$MAE = median_{i=1,\ldots,N}\{|y_i - f(x_i)|\}.$$

- R2 score:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - f(x_i))^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2} = 1 - \frac{SS_{res}}{SS_{tot}}$$

where SS_{res} is residual sum of squares and SS_{tot} is total sum of squares.

Methods for regression

- Linear models:
  - LinearRegression
  - Ridge
  - Lasso
  - ElasticNet
  - SGDRegressor (good for $> 100k$ rows)
- KernelRidge
- KNeighborsRegressor
- GaussianProcessRegressor
- DecisionTreeRegressor
- RandomForestRegressor
- AdaBoostRegressor
- GradientBoostingRegressor
- BaggingRegressor
- VotingRegressor
- SVR
- MLPRegressor
- ...

See scikit-learn documentation here!

Regression Implementation - 1

```
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score

my_df = pd.read_csv('creditApproval.csv')

# Labels are the values we want to predict
labels = np.array(my_df['Credit Score'])
my_df = my_df.drop('Credit Score', axis=1)

train_instances, test_instances, train_labels, test_labels
   = train_test_split(my_df, labels, test_size = 0.20)

def evaluate_results(gTestLabel, gTestPredictions):
   mse =  round(mean_squared_error(gTestLabel, gTestPredictions),
   rmse = givenDec(sqrt(mse))
   var_score = round(r2_score(gTestLabel, gTestPredictions), 2)
   return mse, rmse, var_score
```

Regression Implementation - 2

```
# linear regression
regr = linear_model.LinearRegression()
model = regr.fit(train_instances, train_labels)
test_predictions = model.predict(test_instances)
# get mse, rmse, var_score
evaluate_results(test_labels, test_predictions)

# print regression model
print('Intercept: \n', regr.intercept_)
print('Coefficients: \n', regr.coef_)

# KNN regression
regr = KNeighborsRegressor(n_neighbors=2)
model = regr.fit(train_instances, train_labels)
test_predictions = model.predict(test_instances)
# get mse, rmse, var_score
evaluate_results(test_labels, test_predictions)
```

Regression Implementation - 3

```
# POLYNOMIAL REGRESSION EXAMPLE
# generate data
import numpy as np
import matplotlib.pyplot as plt
n_points = 50
b = 6 # intercept
m = 2.25 # slope
noise_mean = 0.0
noise_var = 1.21

X = []
r = []
for i in range(n_points):
    #rnd_num = random.random()
    rnd_num = np.random.uniform(-2, 2)
    X.append(rnd_num)
    r.append(b + m*rnd_num +
      np.random.normal(loc=noise_mean, scale=noise_var))

plt.scatter(X, r, marker='o');
```

Regression Implementation - 4

```python
# a function to calculate RMSE
def getRMSE(gX, gr, gprVals):
    # report Root Mean Squared Error (RMSE) as output
    sum_squared_error=0
    N = len(X)
    for index, x in enumerate(gX):
        error = gprVals[index] - gr[index]
        squared_error = pow(error, 2)
        sum_squared_error += squared_error

    mean_squared_error = sum_squared_error/N
    tmp_RMSE = pow(mean_squared_error, 0.5)

    return tmp_RMSE
```

Regression Implementation - 5

```
# data reformatting
xx = np.array(X)
rr = np.array(r)
# transforming the data to include another axis
xx = xx[:, np.newaxis]
rr = rr[:, np.newaxis]

import operator
from sklearn.preprocessing import PolynomialFeatures

polynomial_features= PolynomialFeatures(degree=3)
x_poly = polynomial_features.fit_transform(xx)

model = LinearRegression()
model.fit(x_poly, rr)
r_poly_pred = model.predict(x_poly)

# The coefficients + Intercept
print('Intercept: \n', model.intercept_)
print('Coefficients: \n', model.coef_)
```

Regression Implementation - 6

```
RMSE = getRMSE(xx, rr, r_poly_pred)
print('RMSE',RMSE)

plt.scatter(xx, rr, s=10)

# sort the values of x before line plot
sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(xx,r_poly_pred), key=sort_axis)
xx, r_poly_pred = zip(*sorted_zip)
plt.plot(xx, r_poly_pred, color='m')
plt.show()

print(model.intercept_, model.coef_)
```