Regression

- Regression is the process of predicting a continuous value.

- x : Independent variable
  y : Dependent variable → should be continuous

- Types of regression model:

  1. Simple Regression
     → Simple linear Regression
     → Simple Non-linear Regression.

  2. Multiple Regression
     → Multiple linear Regression.
     → Multiple Non-linear Regression.

- Applications of Regression

  1. Sales forecasting
  2. Satisfaction analysis
  3. Price estimation
  4. Employment income

- **Regression Algorithms**

1. ordinal regression
2. Poisson regression
3. Fast forest quantile regression
4. Linear, Polynomial, Lasso, Stepwise, Ridge regression.
5. Bayesian linear Regression.
6. Neural Network Regression.
7. Decision forest Regression.
8. KNN (K-nearest neighbors).

Simple Linear Regression

- We can plot scatter plot in b/w Dependent & Independent variable to find the correlation b/w them.

- **Fitting line**

$$\hat{y} = \theta_0 + \theta_1 x_1$$

response or target variable

x_1 → dependent variable or a single predictor

θ_0 & θ_1 → Coefficient of equations.

- The objective of linear regression is to minimize the MSE (Mean Squared Error).

- How to calculate θ_0 & θ_1 mathematically?

$$\hat{y} = \theta_0 + \theta_1 x_1$$

$$\theta_1 = \frac{\sum_{i=1}^{s} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{s} (x_i - \bar{x})^2}$$

$$\theta_0 = \bar{y} - \theta_1 \bar{x}$$

- Pros of Linear Regression

1. very fast
2. No Parameter Tuning
3. Easy to understand, and highly interpretable.

- Model Evaluation Approaches

1. Train and Test on the same dataset
2. Train / Test split.

- What is an Error of the model?

Error: measure of how far the data is from the fitted regression line.

↳ Mean Absolute Error

$$MAE = \frac{1}{n} \sum_{j=1}^{n} |y_j - \hat{y}_j|$$

↳ Mean Squared Error

$$MSE = \frac{1}{n} \sum_{j=1}^{n} (y_i - \hat{y}_i)^2$$

↳ Root Mean Squared Error

$$RMSE = \sqrt{\frac{1}{n} \sum_{j=1}^{n} (y_j - \hat{y}_j)^2}$$

↳ Relative Absolute Error

$$RAE = \frac{\sum_{j=1}^{n} |y_j - \hat{y}_j|}{\sum_{j=1}^{n} |y_j - \bar{y}|}$$

↳ Relative Square Error

$$RSE = \frac{\sum_{j=1}^{n} (y_j - \hat{y}_j)^2}{\sum_{j=1}^{n} (y_j - \bar{y})^2}$$

$$\boxed{R^2 = 1 - RSE}$$

2. Fuel consumption

- Reading the data

```
df = pd.read_csv("FuelConsumption.csv")
```

- Data Exploration

```
# summarize the data
df.describe()

# let's select some features to explore more
cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPT
          ION_COMB','CO2EMISSIONS']]

# we can plot each of these features:
viz = cdf[[ ____ ]]
viz.hist()
plt.show()

# Now, let's plot each of these features vs
  the EMISSION, to see how linear is their
  relation:
plt.scatter(cdf.FUELCONSUMPTION_COMB,
            cdf.CO2EMISSIONS, color='blue')
plt.xlabel("FUELCONSUMPTION_COMB")
plt.ylabel("Emission")
plt.show()
```

Let's split our dataset into train and test sets, 80% of the entire data for training, and the 20% for testing. We ~~can~~ create a mask to select random rows using np.random.rand() function:

```
msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]
```

modeling
Using sklearn package to model data.

```
from sklearn import linear_model
regr = linear_model.LinearRegression()
train_x = np.asanarray(train[['ENGINESIZE']])
train_y = np.asanarray(train[['CO2EMISSIONS']])
regr.fit(train_x, train_y)

#The coefficients
print("Coefficients:", regr.coef_)
print("Intercept:", regr.intercept_)
```

Plot outputs

We can plot the fit line over the data:

```
plt.scatter (train.ENGINESIZE , train.CO2EMISSIONS,
                color = 'blue')
plt.plot(train_x, regr.coef_[0][0] * train_x
        + regr.intercept_[0], '-r')
plt.xlabel("Engine Size")
plt.ylabel ("Emission")
```

- Evaluation

```
from sklearn.metrics import r2_score
test_x = np.asanarray (test[['ENGINESIZE']])
test_y = np.asanarray (test[['CO2EMISSIONS']])
test_y_hat = regr.predict (test_x)

print("Mean absolute error: %.2f" %
      np.mean(np.absolute (test_y_hat - test_y)))
print("Residual sum of squares (MSE): %.2f"
      %. np.mean ((test_y_hat - test_y) ** 2))
print("R2-score : %.2f" %. r2_score (test_y_hat,
                                     test_y))
```

05/06/2020

Introduction to Regression

[DATACAMP]

Simple Linear Regression

```
#Import numpy and pandas
import numpy as np
import pandas as pd

# Read the csv file into a dataframe : df
df = pd.read_csv("gapminder.csv")

# create arrays for features and target variable
y = df['life'].values
X = df['fertility'].values

# print the dimension of X & y before reshaping.

print("Dimension of y before reshaping: {}".
                    format(y.shape))

print("Dimension of x before reshaping: {}".
                    format(X.shape))

# Reshape X & y
y = y.reshape(-1,1)
X = X.reshape(-1,1)
# Print the dimension of X & y after reshaping.
```

Exploring Gapminder data

→ df.info()

→ df.describe()

→ df.head()

Seaborn heatmap function

sns.heatmap(df.corr(), square=True, cmap='RdYlGn')

06/06/2020

Regression Mechanics

[DATACAMP]

- $y = ax + b$
  - y = target
  - x = single feature
  - a, b = parameters of model
- How do we choose a & b?
- Define an error function for any given line
  - choose the line that minimizes the error function.

The loss function

- ordinary least squares (OLS):
  Minimize sum of squares of residuals.

Linear Regression in higher dimension

$$y = a_1 x_1 + a_2 x_2 + b$$

- To fit a linear regression model here:
  - Need to specify 3 variables.
    a_1, a_2 & b

- In higher dimensions:
  - Must specify coefficient for each feature and the variable b.

$$y = a_1x_1 + a_2x_2 + a_3x_3 + \cdots + a_nx_n + b$$

- Scikit-learn API works exactly the same way:
  - Pass 2 arrays: features and target

Linear Regression on all features

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

X_train, X_test, y_train, y_test = train_test_split(X,y,
                    test_size = 0.3, random_state = 42)

reg_all = LinearRegression()
reg_all.fit(X_train, y_train)
y_pred = reg_all.predict(X_test)
reg_all.score(X_test, y_test)
```

Fit & Predict for Regression [working on gapminder dataset]

```
# Import linear Regression
from sklearn.linear_model import LinearRegression

# Create the regressor: reg
reg = LinearRegression()

# Create the prediction space
prediction_space = np.linspace(min(X_fertility),
                   max(X_fertility)).reshape(-1,1)

# Fit the model to the data
reg.fit(X_fertility, y)

# Compute predictions over the prediction space:
y_pred

y_pred = reg.predict(prediction_space)

# Print R^2
print(reg.score(X_fertility, y))

# Plot regression line
plt.plot(prediction_space, y_pred,
         color='black', linewidth=3)
plt.show()
```

train/test split for Regression

```
# Import necessary modules
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import
        train_test_split

# Create training & test sets.

X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=0.3,
                     random_state=42)

# Create the regressor: reg_all

reg_all = LinearRegression()

# Fit the regressor to the training data
reg_all.fit(X_train, y_train)

# Predict on the test data: y_pred
y_pred = reg_all.predict(X_test)

# Compute & print R^2 & RMSE
print("R^2: {}".format(reg_all.score(X_test, y_test)))
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error: {}".format(rmse))
```

Cross-validation motivation

- Model performance is dependent on way the data is split.
- Not representative of the model's ability to generalize.
- Solution: Cross-validation!

Cross-validation basics:

Split 1	**Fold 1**	Fold 2	Fold 3	Fold 4	Fold 5	metric 1
Split 2	1	**Fold 2**	3	4	5	metric 2
Split 3	1	2	**Fold 3**	4	5	3
Split 4	1	2	3	**Fold 4**	5	4
Split 5	1	2	3	4	**Fold 5**	5

Training data | Test data

- 5 folds = 5-fold CV
  10 folds = 10-fold CV
  K folds = K-fold CV

- More folds = More computationally expensive.

Cross-validation in Scikit-learn

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
cv_results = cross_val_score(reg, X, y, cv=5)
print(cv_results)
```

2. 5-fold cross validation

```
# Import the necessary modules
from sklearn.model_selection import cross_val_score

from sklearn.linear_model import
        LinearRegression

# Create a linear Regression object : reg
reg = LinearRegression()

# Compute 5-fold cross-validation scores:
    cv_scores
cv_scores = cross_val_score(reg, X, y, cv=5)

# Print the 5-fold cross validation scores
print(cv_scores)

print("Average 5-fold CV scores: {}".format
        (np.mean(cv_scores)))
```

To see how much time a single cross validation score takes or K-CV takes.

```
%. timeit cross_val_score(reg, X, y, cv=
                          K)
```

↓

Here K_0 → can be
1, 2, 3 ... K.

Multiple Linear Regression [Term]

Examples of Multiple Linear Regression

- Independent variable effectiveness on prediction
  - Does revision time, test anxiety, lecture attendance and gender have any effect on the exam performance of students?
- Predicting Impact of changes
  - How much does blood pressure go up & down for every unit increase or decrease in the BMI of a patient?

Estimating multiple linear Regression Parameters

- How to estimate θ?
  - → Ordinary Least Squares
    - Linear Algebra operations
    - Takes a long time for large datasets (10K+ rows)
  - → An optimization algorithm
    - Gradient Descent
    - Proper approach if you have a very large dataset.

7. What is Polynomial Regression?

- Some curvy data can be modelled by a polynomial regression.
- for eg.

$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

08/06/2020

Regularized Regression [DATACAMP]

why Regularize:

- Recall: Linear Regression minimizes a loss function.
- It chooses a coefficient for each feature variable.
- large coefficients can lead to overfitting.
- Penalize large coefficients: Regularization.

[1] Ridge Regression

- loss function = OLS loss function + $\alpha * \sum_{i=1}^{n} a_i^2$
- Alpha: Parameter we need to choose
- Picking alpha here is similar to picking k in KNN.
- Alpha controls model complexity
  - Alpha = 0: we get back OLS (can lead to overfitting).
  - very high alpha: can lead to underfitting.

Ridge Regression in scikit-learn

```
from sklearn.linear_model import Ridge
X_train, X_test, y_train, y_test = train_test_split
        (X, y, test_size = 0.3, random_state=42)
ridge = Ridge (alpha=0.1, normalize = True)
ridge.fit (X_train, y_train)
ridge_pred = ridge.predict (X_test)
ridge.score (X_test, y_test)
```

↳ output = 0.69969...

[2] Lasso Regression

- Loss function = OLS loss function + $\alpha * \sum_{i=1}^{n} |a_i|$

Lasso Regression for scikit learn

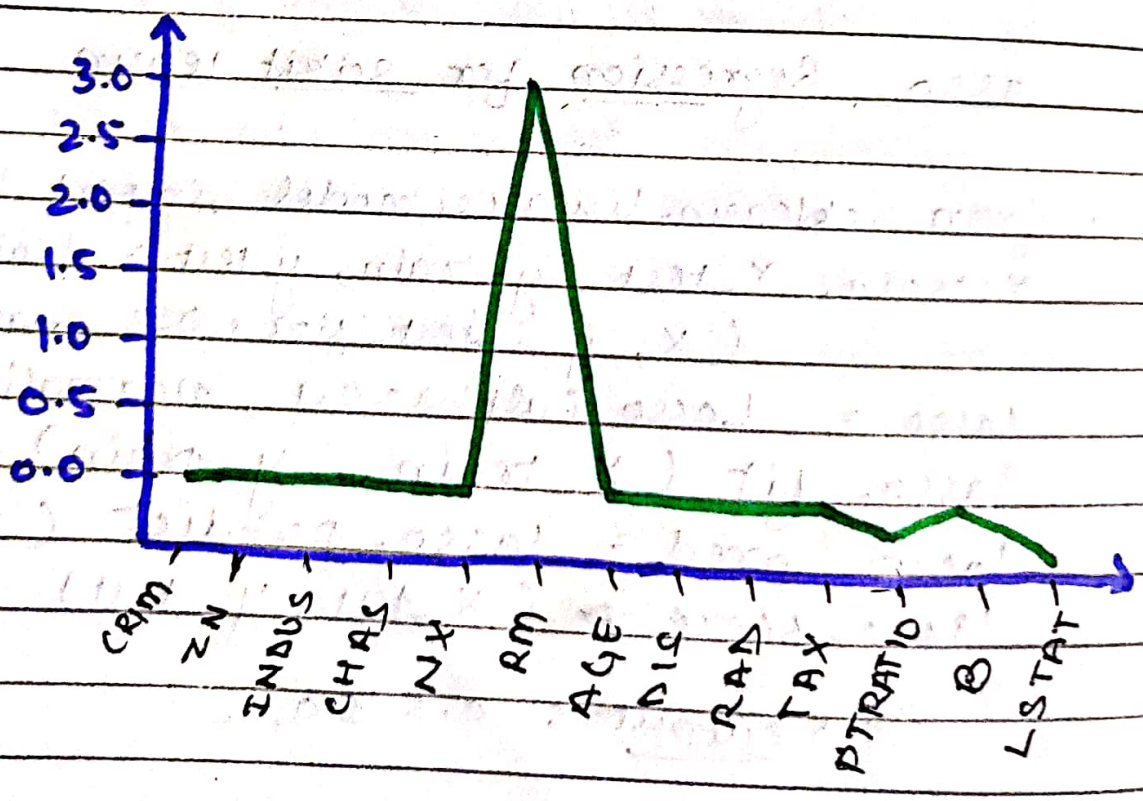
```
from sklearn.linear_model import Lasso
X_train, X_test, y_train, y_test = train_test_split
        (X, y, test_size = 0.3, random_state=42)
lasso = Lasso (alpha=0.1, normalize=True)
lasso.fit (X_train, y_train)
lasso_pred = lasso.predict (X_test)
lasso.score (X_test, y_test)
```

↳ output : 0.5980...

Lasso Regression for feature selection

- Can be used to select important features of a dataset.
- shrinks the coefficients of less important features to exactly 0.

```
from sklearn.linear_model import Lasso
names = boston.drop('MEDV', axis=1).columns
lasso = Lasso(alpha=0.1)
lasso_coef = lasso.fit(X, y).coef_
_ = plt.plot(range(len(names)), lasso_coef)
_ = plt.xticks(range(len(names)), names, rotation=60)
_ = plt.ylabel('coefficients')
plt.show()
```



DATACAMP

SUPERVISED LEARNING with scikit-learn

CHAPTER-3

FINE-TUNING YOUR MODEL

HOW GOOD IS YOUR MODEL?

1. Classification metrics

- Measuring model performance with accuracy:
  - → Fraction of correctly classified samples.
  - → Not always a useful metric.

class imbalanced examples: Emails

- Spam classification
  - → 99% of emails are real, 1% of emails are spam.
- Could build a classifier that predicts ALL emails as real
  - → 99% accurate!
  - → But horrible at actually classifying spam.
  - → Fails at its original purpose.
- Need more nuanced metrics.

Diagnosing classification predictions

- Confusion matrix

	Predicted: Spam Email	Predicted: Real Email
Actual: Spam Email	TRUE POSITIVE	FALSE NEGATIVE
Actual: Real Email	FALSE POSITIVE	TRUE NEGATIVE

- Accuracy: $\frac{TP + TN}{TP + TN + FP + FN}$

Metrics from the confusion matrix

- Precision: $\frac{TP}{TP + FP}$
- Recall: $\frac{TP}{TP + FN}$
- F1 score: $2 \times \frac{Precision \times Recall}{Precision + Recall}$

- High Precision : Not many real emails predicted as spam.
- High Recall : Predicted most spam emails correctly

Confusion matrix in scikit learn

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

knn = kNeighborsClassifier(n_neighbors = 8)

X_train, X_test, y_train, y_test = train_test_split
        (X, y, test_size=0.4, random_state=42)

knn.fit(X_train, y_train)
y_pred = knn.pred(X_test)

print( confusion_matrix(y_test, y_pred))
```

↳ output

```
[[152   7]
 [ 3   112]]
```

```
print(classification_report(y_test, y_pred))
```

↳ output

	Precision	Recall	f1-score	Support
0	0.95	0.88	0.91	59
1	0.94	0.97	0.96	115
avg/total	0.94	0.94	0.94	174