Scikit Learn

Machine Learning in Python

TEAM 1
Charul Bagla
Sonia Daryani
Utkarsh Jain
Boyang Tang
Jin Zhuo

Introduction

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

History

- Developed by David Cournapeau as a Google Summer of Code project in 2007
- Later used by Matthieu Brucher as part of his thesis work
- Publicly published for the first time in February 2010 by the French Institute for Research in Computer Science and Automation

Functions & Definitions

Regression

- Set of statistical models for modeling and analyzing relationships among variable

Classification

- Task of mapping each predictor variable to one of the predefined categories of a target variable

Types of Variables:

$$y = m * x + c$$

- Target
- Predictor
- Categorical
- Numerical

Preliminary Workings

Pandas & Numpy

- Data Loading
- Data Cleaning

Step 1: Data Loading

- Import relevant libraries
- Load the data from csv file using Pandas

```
import pandas as pd
import numpy as np
```

```
#Loading the text data file
loan_data = pd.read_csv("LoanStats_2018Q1.csv", skiprows= 1, low_memory = False)
loan_data = loan_data[:-2]

print(loan_data.head(2))
print(loan_data.tail(2))
```

Step 2: Data Cleaning

- Check missing values in columns
- Fill missing values using .fillna function
- Create separate data files for linear and logistic regression

```
# Fill missing values for the requires columns

loan_data['dti'] = loan_data['dti'].fillna(0)
loan_data['avg_cur_bal'] = loan_data['avg_cur_bal'].fillna(0)
```

```
#Selecting required columns for each regression

loan_data_lin = loan_data.filter(items = ["application_type", 'home_ownership',"funded_amnt","annual_inc", "delinq_2yrs",
                                          "loan_amnt", "max_bal_bc", "mort_acc", "num_accts_ever_120_pd", "num_actv_bc_tl",
                                          "num_actv_rev_tl", "num_bc_sats"])

loan_data_log = loan_data.filter(items = ['home_ownership',"acc_now_delinq" , "annual_inc", "avg_cur_bal", "delinq_amnt",
                                          "dti" , "mort_acc", "pub_rec_bankruptcies", "grade"])
```

Data Processing

Scikit Learn

- Data Encoding
- Dummy Creation

Data Encoding

One-Hot Encoding:

- Transforms categorical (discrete) features into a sparse matrix where each column corresponds to one possible value of a feature

Label Encoder

- Encodes labels with values between 0 and 1

Variable (Pre-Dummy Creation)
A
B
C

Variable (Post-Dummy Creation)	Variable _A	Variable _B	Variable _C
A	1	0	0
B	0	1	0
C	0	0	0

Data Encoding

```python
# Creating dummy variables for categorical predictors

from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder

def getdummies(res, ls):
    def encode(encode_df):
        encode_df = np.array(encode_df)
        enc = OneHotEncoder()
        le = LabelEncoder()
        le.fit(encode_df)
        res1 = le.transform(encode_df).reshape(-1, 1)
        enc.fit(res1)
        return pd.DataFrame(enc.transform(res1).toarray()), le, enc

    decoder = []
    outres = pd.DataFrame({'A' : []})

    for l in ls:
        cat, le, enc = encode(res[l])
        cat.columns = [l+str(x) for x in cat.columns]
        outres.reset_index(drop=True, inplace=True)
        outres = pd.concat([outres, cat], axis = 1)
        decoder.append([le,enc])

    return (outres, decoder)
```

Dummy Creation

- Call the function to create dummy variables

```
#Identifying categorical predictors to create dummy variables
categorical_lin = [ "application_type", 'home_ownership']

#Calling the dummy variable creation function
res = getdummies(loan_data_lin[categorical_lin],categorical_lin)
df = res[0]
decoder = res[1]
```

Linear Regression

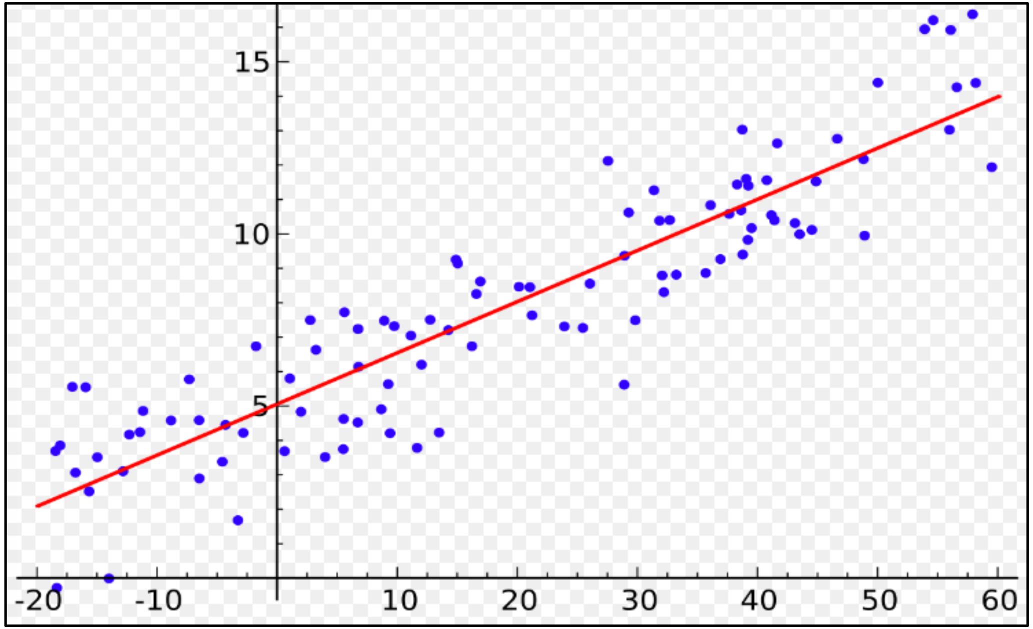
Scikit Learn

- Overview
- Data Selection
- Data Split – Train and Test
- Model Fitting
- Y-value Prediction & Model Accuracy
- Regression Line

Overview

- Basic predictive analysis model
- Statistical method to model the relationship between a scalar response and explanatory variable(s)

$$y_i = \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_i x_i + \varepsilon_i$$



Data Selection

- Separate target (lin_y) and predictor (lin_X) variables
- Drop unrelated variables

```
#Separating the predictors and target variable
lin_X = pd.concat([df,loan_data_lin],axis=1)
lin_X = lin_X.drop(columns = ['A',"application_type", 'home_ownership','funded_amnt'])

lin_y=loan_data_lin['funded_amnt']
```

Data Split – Train and Test

```python
#Split the data into test and training datasets
from sklearn.model_selection import train_test_split

lin_X_train, lin_X_test, lin_y_train, lin_y_test = train_test_split(lin_X, lin_y, test_size=0.25, random_state=42)
```

- Split the dataset into training/testing sets(test size is 25%)
- Random state is set to have uniform outputs across all runs

TRAINING DATA – Used to develop the model

TEST DATA – Used to predict target values using the model

Model Fitting

```
#Train linear regression model
from sklearn import linear_model

regr = linear_model.LinearRegression()
regr.fit(lin_X_train, lin_y_train)

print('Coefficients: \n', regr.coef_)
```

Outcome:

```
Coefficients:
 [-1.17290072e-10  1.17290106e-10 -9.19512074e-10  2.43348461e-10
  3.32493171e-10  3.43670794e-10 -3.33066907e-16 -3.19525338e-13
  1.00000000e+00  8.32667268e-17 -9.46316135e-14  5.83908769e-15
  2.68913879e-13 -3.35044079e-13  7.75917992e-14]
```

Y-value Prediction & Model Accuracy

```
#Predict values of test set based on trained model
lin_y_pred = regr.predict(lin_X_test)
```

```
#Assessing model accuracy. Getting summary statistics
from sklearn.metrics import mean_squared_error, r2_score

# The mean squared error
print("Mean squared error: %.2f"
      % mean_squared_error(lin_y_test, lin_y_pred))

# Explained variance score: 1 is perfect prediction
print('R-Squared value is: %.2f' % r2_score(lin_y_test, lin_y_pred))
```

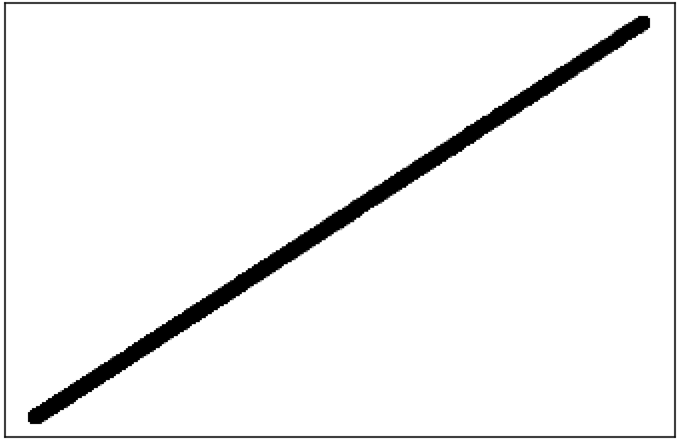
Outcome:

```
Mean squared error: 0.00
R-Squared value is: 1.00
```

Regression Line

```
plt.scatter(df_Y_pred, df_Y_test,  color='black')

plt.xticks(())
plt.yticks(())

plt.show()
```

Outcome:



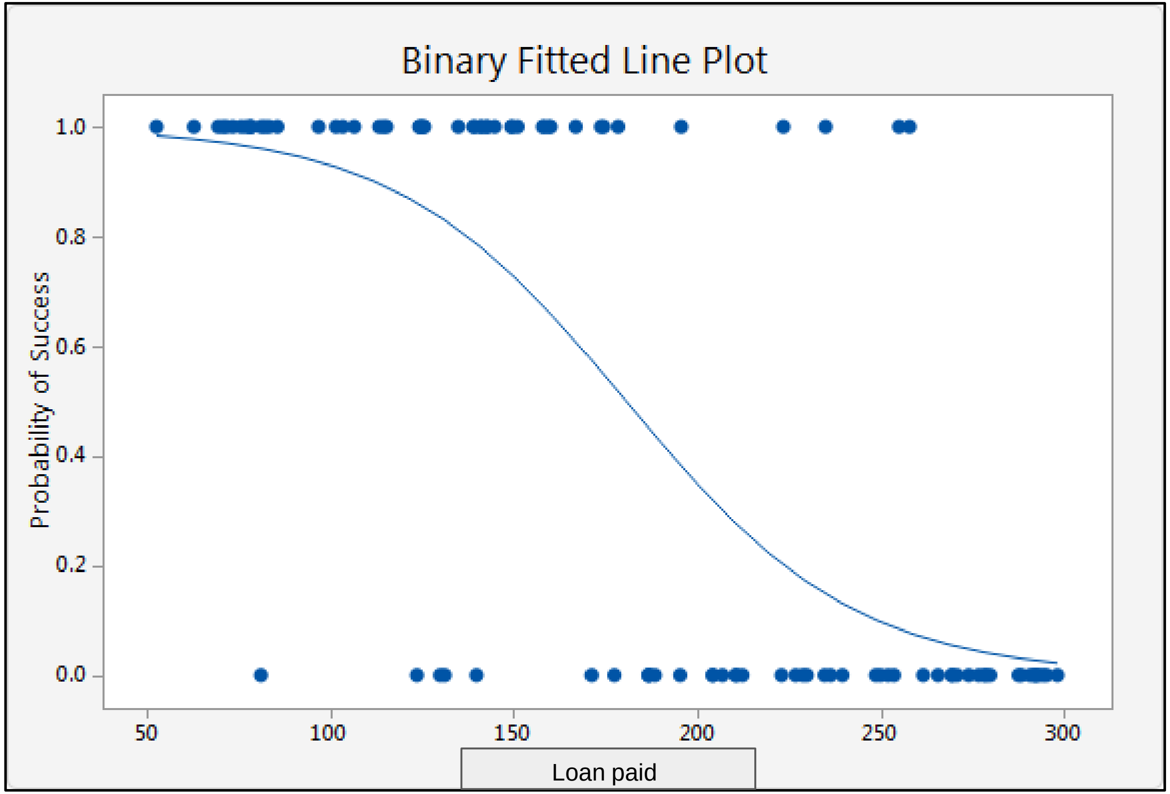
Logistic Regression

Scikit Learn

- Overview
- Data Preparation – Target Variable Balancing
- Data Selection
- Data Split – Train and Test
- Model Fitting
- Prediction - Values & Probabilities
- Model Accuracy

Logistic Regression

Find a logistic approach to model the relationship between one ***dependent categorical variable*** and one or more independent predictor variables.



Data Preparation –Target Variable Balancing

```
#Balancing target variable counts

def conv_grade(x):
    if(x == 'A'):
        r = 'A'
    elif(x == 'B'):
        r = 'B'
    elif(x == 'C'):
        r = 'C'
    elif(x == 'D'):
        r = 'D'
    else:
        r = 'D'
    return r
print("Before:")
print(loan_data_log.grade.value_counts())
loan_data_log['grade'] = loan_data_log['grade'].apply(conv_grade)
print("After:")
print(loan_data_log.grade.value_counts())
```

```
Before:
B      32482
C      28747
A      26769
D      15377
E       3691
F        682
G        116
```

```
After:
B      32482
C      28747
A      26769
D      19866
```

Data Selection

- Separate target (log_y) and predictor (log_X) variables
- Drop unrelated variables

```
#Separating the predictors and target variable
log_X = pd.concat([df,loan_data_log],axis=1)
log_X = log_X.drop(columns = ['A', 'home_ownership','grade'])

log_y=loan_data_log['grade']
```

Data Split – Train and Test

```python
#Split the data into test and training datasets

from sklearn.model_selection import train_test_split

log_X_train, log_X_test, log_y_train, log_y_test = train_test_split(log_X, log_y, test_size=0.25, random_state = 55)
```

- Split the dataset into training/testing sets(test size is 25%)
- Random state is set to have uniform outputs across all runs

TRAINING DATA – Used to develop the model

TEST DATA – Used to predict target values using the model

Model Fitting

Create an Regression Object

```
from sklearn.linear_model import LogisticRegression

#create logisitic regression object and fitting the data with the classifier
logitR = LogisticRegression(dual=False, tol=0.0001, solver="lbfgs", multi_class='ovr',random_state=55)
logitR.fit(log_X_train, log_y_train)
```

Fit classifier with training set

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=55, solver='lbfgs', tol=0.0001,
          verbose=0, warm_start=False)
```

- **tol :** Tolerance for stopping criteria
- **solver** : {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}, default: liblinea
  - For small datasets, 'liblinear' is a good choice, whereas 'sag' and
  - For multiclass problems, only 'newton-cg', 'sag', 'saga' and 'lbfgs'
- **multi_class :** Selecting multi class solver {'ovr' , 'multinomial'}, default: 'ovr'
- **random_state :** Set random seed to have uniform output across runs

Prediction: Values and Probabilities

```
#predicting the values of the test dataset
log_pred_y = logitR.predict(log_X_test)
log_pred_p = logitR.predict_proba(log_X_test)
```

Predict class labels for samples in test data

Predict probability of a sample belonging to a particular class

A	B	C	D

```
array([[0.25364693, 0.29747168, 0.25585807, 0.19302332],
       [0.23720441, 0.28636047, 0.26535917, 0.21107595],
       [0.27113636, 0.31089446, 0.26077439, 0.15719478],
       ...,
       [0.38815288, 0.32091457, 0.2069485 , 0.08398405],
       [0.27245647, 0.27074187, 0.25197846, 0.2048232 ],
       [0.22737908, 0.27268112, 0.265912  , 0.2340278 ]])
```

Probability estimates for all classes are ordered by the label of classes

Model Accuracy - Confusion Matrix

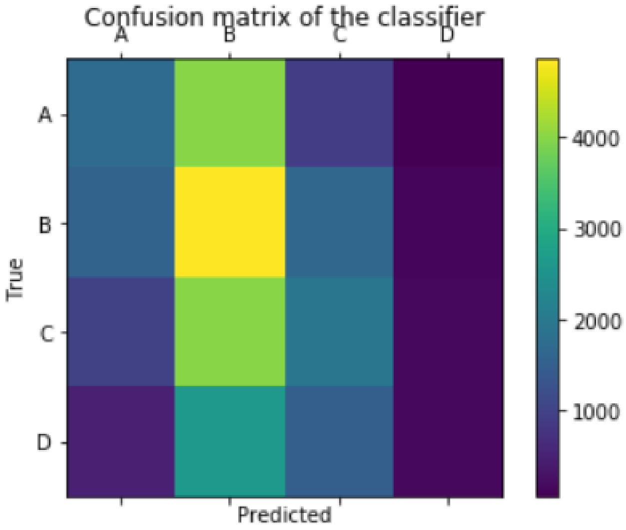
```
#Assessing model accuracy. Getting summary statistics
from sklearn.metrics import confusion_matrix, precision_score, classification_report

#Loading the confusion matrix module
labels = ["A","B","C","D"]
conf_mat= confusion_matrix(log_y_test, log_pred_y,labels = ["A","B","C","D"])
print("Confusion Matrix :")
print(conf_mat)

fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(conf_mat)
plt.title('Confusion matrix of the classifier')
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

Creating a Confusion Matrix

Plotting the confusion matrix with the labels of all classes

Model Accuracy - Confusion Matrix

```
Confusion Matrix :
[[1746 3998  938   67]
 [1595 4853 1639  144]
 [1004 4005 1948  165]
 [ 515 2660 1523  166]]
```



Model Accuracy – Precision and Classification Report

```
#Calculating the precision score
print("\nOverall Precision Score of Model : ",precision_score(log_y_test, log_pred_y, average = 'macro'))

#Genrating classification report
print("\nClassification Report :")
print(classification_report(log_y_test,log_pred_y))
```

Precision Score Function used to calculate the precision of the model

```
Overall Precision Score of Model :  0.32509904497039666

Classification Report :
             precision    recall  f1-score   support

          A       0.36      0.26      0.30      6749
          B       0.31      0.59      0.41      8231
          C       0.32      0.27      0.30      7122
          D       0.31      0.03      0.06      4864

avg / total       0.33      0.32      0.29     26966
```

Precision score and the Classification report of the model

THANKS

TEAM 1