```
import numpy as np
import matplotlib.pyplot as plt
import math
```

```
### logistic regression in python
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
def sigmoid(x):
    return 1/(1+np.e**-x)

x = np.linspace(-10, 10, 20)
z = sigmoid(x)
plt.plot(x, z)
plt.xlabel("x")
plt.ylabel("Sigmoid(X)")
plt.grid()
plt.show()
```

```
# log loss
y = 1
yhat=0.9

print(-y*math.log(yhat)-(1-y)*math.log(1-yhat))
```

```
    0.10536051565782628
```

```
y = 1
yhat=0.99

print(-y*math.log(yhat)-(1-y)*math.log(1-yhat))
```

```
    0.01005033585350145
```

```
y = 1
yhat=0.99999

print(-y*math.log(yhat)-(1-y)*math.log(1-yhat))
```

```
1.0000050000287824e-05
```

```
y=1
yhat=0.1

print(-y*math.log(yhat)-(1-y)*math.log(1-yhat))
```

```
2.3025850929940455
```

```
y=1
yhat=0.01

print(-y*math.log(yhat)-(1-y)*math.log(1-yhat))
```

```
4.605170185988091
```

```
y=0
yhat=0.1

print(-y*math.log(yhat)-(1-y)*math.log(1-yhat))
```

```
0.10536051565782628
```

```
y=0
yhat=0.01

print(-y*math.log(yhat)-(1-y)*math.log(1-yhat))
```

```
0.01005033585350145
```

```
y=0
yhat=0.9

print(-y*math.log(yhat)-(1-y)*math.log(1-yhat))
```

```
2.302585092994046
```

```
# Churn prediction in telecom.
import numpy as np
import matplotlib.pyplot as plt
```

```
#https://drive.google.com/file/d/1Hryt6VSnHklyw3xxBG3nlhymzNAQhG-_/view?usp=sharing
```

```
id = "1Hryt6VSnHklyw3xxBG3nlhymzNAQhG-_"
print("https://drive.google.com/uc?export=download&id=" + id)
```

```
https://drive.google.com/uc?export=download&id=1Hryt6VSnHklyw3xxBG3nlhymzNAQhG
```

```
!wget "https://drive.google.com/uc?export=download&id=1Hryt6VSnHklyw3xxBG3nlhymzNAQ
```

```
--2022-04-29 16:30:45--  https://drive.google.com/uc?export=download&id=1Hryt6
Resolving drive.google.com (drive.google.com)... 74.125.195.138, 74.125.195.10
Connecting to drive.google.com (drive.google.com)|74.125.195.138|:443... conne
HTTP request sent, awaiting response... 303 See Other
Location: https://doc-0g-ag-docs.googleusercontent.com/docs/securesc/ha0ro937g
Warning: wildcards not supported in HTTP.
--2022-04-29 16:30:45--  https://doc-0g-ag-docs.googleusercontent.com/docs/sec
Resolving doc-0g-ag-docs.googleusercontent.com (doc-0g-ag-docs.googleuserconte
Connecting to doc-0g-ag-docs.googleusercontent.com (doc-0g-ag-docs.googleuserc
HTTP request sent, awaiting response... 200 OK
Length: 289296 (283K) [text/csv]
Saving to: ‘Churn.csv’

Churn.csv           100%[===================>] 282.52K  --.-KB/s    in 0.002s

2022-04-29 16:30:45 (120 MB/s) - ‘Churn.csv’ saved [289296/289296]
```

```
import pandas as pd

churn = pd.read_csv("Churn.csv")
churn.head()
```

```
churn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Account Length  3333 non-null   int64  
 1   VMail Message   3333 non-null   int64  
 2   Day Mins        3333 non-null   float64
 3   Eve Mins        3333 non-null   float64
 4   Night Mins      3333 non-null   float64
 5   Intl Mins       3333 non-null   float64
 6   CustServ Calls  3333 non-null   int64  
 7   Churn           3333 non-null   int64  
 8   Intl Plan       3333 non-null   int64  
 9   VMail Plan      3333 non-null   int64  
 10  Day Calls       3333 non-null   int64  
 11  Day Charge      3333 non-null   float64
 12  Eve Calls       3333 non-null   int64  
 13  Eve Charge      3333 non-null   float64
 14  Night Calls     3333 non-null   int64  
 15  Night Charge    3333 non-null   float64
 16  Intl Calls      3333 non-null   int64  
 17  Intl Charge     3333 non-null   float64
 18  State           3333 non-null   object 
 19  Area Code       3333 non-null   int64  
 20  Phone           3333 non-null   object 
dtypes: float64(8), int64(11), object(2)
memory usage: 546.9+ KB
```

```
churn["Churn"].value_counts()
```

```
0    2850
1     483
Name: Churn, dtype: int64
```

```
churn.columns
```

```
Index(['Account Length', 'VMail Message', 'Day Mins', 'Eve Mins', 'Night Mins'
       'Intl Mins', 'CustServ Calls', 'Churn', 'Intl Plan', 'VMail Plan',
       'Day Calls', 'Day Charge', 'Eve Calls', 'Eve Charge', 'Night Calls',
       'Night Charge', 'Intl Calls', 'Intl Charge', 'State', 'Area Code',
       'Phone'],
      dtype='object')
```

```
# Basic EDA, not comprehensive
import seaborn as sns

sns.boxplot(x='Churn', y='Day Mins', data = churn)
```

```
# simple correlation, not full collienarity

sns.pairplot(data=churn, y_vars=["Day Mins"], x_vars=['Account Length', 'VMail Mess
       'Intl Mins', 'CustServ Calls', 'Churn', 'Intl Plan', 'VMail Plan',
       'Day Calls', 'Day Charge', 'Eve Calls', 'Eve Charge', 'Night Calls',
       'Night Charge', 'Intl Calls', 'Intl Charge'], height=1.5, aspect=1)
plt.show()
# Day charge vs Day Mins
```

```
sns.boxplot(x = 'Churn', y= 'Account Length', data = churn)
```

```
# Skipping rest of the EDA :
# Exercise for students complete the rest of the EDA to find out which variables ha
```

```
# using a few features as an example. We can drop all useless features. Not a perfe
cols = ['Account Length', 'VMail Message', 'Day Mins', 'Eve Mins', 'Night Mins',
       'Intl Mins', 'CustServ Calls', 'Intl Plan', 'VMail Plan', 'Day Calls',
       'Day Charge', 'Eve Calls', 'Eve Charge', 'Night Calls', 'Night Charge',
       'Intl Calls', 'Intl Charge']
```

```
y = churn["Churn"]
```

```
X = churn[cols]
```

```
X.shape
```

```
(3333, 17)
```

```
# Train, CV, test split
from sklearn.model_selection import train_test_split
#0.6, 0.2, 0.2 split

X_tr_cv, X_test, y_tr_cv, y_test = train_test_split(X, y, test_size=0.2, random_sta

X_train, X_val, y_train, y_val = train_test_split(X_tr_cv, y_tr_cv, test_size=0.25,
```

```
X_train.shape
```

```
(1999, 17)
```

```
#scaling the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
```

```
StandardScaler()
```

```
from sklearn.linear_model import LogisticRegression
#https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticReg
model = LogisticRegression()
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818:
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
LogisticRegression()
```

```
model.coef_
```

```
array([[-1.15863972e-03, -3.62927654e-02,  7.85591837e-03,
         1.07852828e-03, -1.84885574e-03,  8.59866407e-03,
         4.23609435e-01,  1.76836501e-01, -1.15844819e-02,
        -1.12177721e-02,  1.29476481e-03, -1.17254607e-02,
        -1.04683713e-04, -8.69404735e-03, -1.31109534e-04,
        -1.19589837e-01,  2.39253414e-03]])
```

```
model.intercept_
```

```
array([-0.04111358])
```

```
# Hyper-pram tuning
from sklearn.pipeline import make_pipeline
train_scores = []
val_scores = []
scaler = StandardScaler()

for la in np.arange(0.01, 100.0, 5):
  scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la))
  scaled_lr.fit(X_train, y_train)
  train_score = scaled_lr.score(X_train, y_train)
  val_score = scaled_lr.score(X_val, y_val)
  train_scores.append(train_score)
  val_scores.append(val_score)
```

```
len(val_scores)
```

```
20
```

```
plt.figure()
plt.plot(list(np.arange(0.01, 100.0, 5)), train_scores, label="train")
plt.plot(list(np.arange(0.01, 100.0, 5)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("lambda")
plt.ylabel("Score")
plt.grid()
plt.show()
```

```
np.argmax(val_scores)
```

```
14
```

```
val_scores[0]
```

```
0.856071964017991
```

```
l_best = 0.01+5*14
```

```
# Model with lambda=
scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/l_best))
scaled_lr.fit(X_train, y_train)
```

```
    Pipeline(steps=[('standardscaler', StandardScaler()),
                    ('logisticregression',
                     LogisticRegression(C=0.0142836737608913))])
```

```
test_score = scaled_lr.score(X_test, y_test)
print(test_score)

y_pred = scaled_lr.predict(X_test)
```

```
    0.856071964017991
```

```
from sklearn.metrics import accuracy_score

print(f"Accuracy : {accuracy_score(y_test, y_pred)*100}%")
```

```
    Accuracy : 85.6071964017991%
```

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, y_pred)
```

```
    array([[560,   6],
           [ 90,  11]])
```

```
# many class-1 points classified as class-0 => useless model
```

```
# how to fix the model?
```

```
# Hyper-pram tuning
from sklearn.pipeline import make_pipeline
train_scores = []
val_scores = []
scaler = StandardScaler()

for la in np.arange(0.01, 100.0, 5):
  scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/la , class_weight={ 0:0
  scaled_lr.fit(X_train, y_train)
  train_score = scaled_lr.score(X_train, y_train)
  val_score = scaled_lr.score(X_val, y_val)
  train_scores.append(train_score)
  val_scores.append(val_score)

plt.figure()
plt.plot(list(np.arange(0.01, 100.0, 5)), train_scores, label="train")
```

```
plt.plot(list(np.arange(0.01, 100.0, 5)), val_scores, label="val")
plt.legend(loc='lower right')
plt.xlabel("lambda")
plt.ylabel("Score")
plt.grid()
plt.show()
```

```
# Model with lambda= 0.01
l_best = 0.01
scaled_lr = make_pipeline( scaler, LogisticRegression(C=1/l_best, class_weight={ 0:
scaled_lr.fit(X_train, y_train)

test_score = scaled_lr.score(X_test, y_test)
print(test_score)

y_pred = scaled_lr.predict(X_test)
```

```
0.704647676161919
```

```
from sklearn.metrics import accuracy_score

print(f"Accuracy : {accuracy_score(y_test, y_pred)*100}%")
```

```
Accuracy : 70.4647676161919%
```

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, y_pred)
# Better for class-1, but worse for class-0
# Any ideas on how to solve?
```

```
array([[383, 183],
       [ 14,  87]])
```

✓ 0s completed at 22:01