# # Logistic Regression with Python

The objective of the Logistic Regression algorithm, is to find the best parameters ff, for $h\_\theta(x)= \sigma(\theta T X)$, in such a way that the model best predicts the class of each case.

## ## Let's first import required libraries:

In [48]:
```
import pandas as pd
import numpy as np
import pylab as pl
import scipy.optimize as opt
from sklearn import preprocessing
import matplotlib.pyplot as plt
%matplotlib inline
```

About the dataset
We will use a telecommunications dataset for predicting customer behaviour and segmentation. This is a historical
customer dataset where each row represents one customer. The data is relatively easy to understand,
and we may uncover insights we  can use immediately. This dataset appears to be a customer dataset where each row represents a customer and each column represents a different attribute or characteristic of that customer. The custcat column specifically categorizes customers into different segments, which could be used for targeted marketing, service customization, or customer retention strategies based on their respective needs and behaviors.

This data set provides information to help you predict what behavior will help you to retain
customers. We can analyze all relevant customer data and develop focused customer retention
programs.
The dataset includes information about:

region: Categorical variable indicating the region where the customer is located (e.g., 1 = Northeast, 2 = Midwest, etc.).
tenure: Number of months the customer has been with the service provider.
address: Number of years the customer has lived at their current address.
income: Annual income of the customer.
ed: Education level of the customer (e.g., 1 = High School, 2 = College, etc.).
employ: Number of years the customer has been employed.
retire: Binary variable indicating if the customer is retired (0 = No, 1 = Yes).
reside: Number of people residing with the customer.
custcat: Customer category or segment that each individual belongs to. This column categorizes customers into different segments based on various criteria such as their usage patterns, service preferences, or demographic characteristics.
• Demographic info about customers – gender, age range, and if they have partners and depen•dents

# # Load the Telco Churn data

In [49]:
```
telecust_df=pd.read_csv("C:\\Users\\prabh\\Desktop\\teleCust1000t.csv")
telecust_df.head()
```

Out[49]:

| | region | tenure | age | marital | address | income | ed | employ | retire | gender | reside | custcat |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2 | 13 | 44 | 1 | 9 | 64.0 | 4 | 5 | 0.0 | 0 | 2 | 1 |
| **1** | 3 | 11 | 33 | 1 | 7 | 136.0 | 5 | 5 | 0.0 | 0 | 6 | 4 |
| **2** | 3 | 68 | 52 | 1 | 24 | 116.0 | 1 | 29 | 0.0 | 1 | 2 | 3 |
| **3** | 2 | 33 | 33 | 0 | 12 | 33.0 | 2 | 0 | 0.0 | 1 | 1 | 1 |
| **4** | 2 | 23 | 30 | 1 | 9 | 30.0 | 1 | 2 | 0.0 | 0 | 4 | 3 |

We can use the Pandas method corr() to find the feature other than Custcat that is most correlated with Custcat.

```
In [50]: telecust_df.corr()["custcat"].sort_values()
```

```
Out[50]: region    -0.023771
         gender    -0.004966
         retire     0.008908
         age        0.056909
         address    0.067913
         reside     0.082022
         marital    0.083836
         employ     0.110011
         income     0.134525
         tenure     0.166691
         ed         0.193864
         custcat    1.000000
         Name: custcat, dtype: float64
```

**## Data pre-processing and selection**

Let’s select some features for the modeling. Also, we change the target data type to be an integer,as it is a requirement by the skitlearn algorithm

```
In [51]: telecust_df=telecust_df[['region','gender','retire','age','address','reside', 'marital','employ','income
         'tenure', 'ed','custcat' ]]
         telecust_df["custcat"]= telecust_df["custcat"].astype("int")
         telecust_df.head()
```

Out[51]:

| | region | gender | retire | age | address | reside | marital | employ | income | tenure | ed | custcat |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 2 | 0 | 0.0 | 44 | 9 | 2 | 1 | 5 | 64.0 | 13 | 4 | 1 |
| **1** | 3 | 0 | 0.0 | 33 | 7 | 6 | 1 | 5 | 136.0 | 11 | 5 | 4 |
| **2** | 3 | 1 | 0.0 | 52 | 24 | 2 | 1 | 29 | 116.0 | 68 | 1 | 3 |
| **3** | 2 | 1 | 0.0 | 33 | 12 | 1 | 0 | 0 | 33.0 | 33 | 2 | 1 |
| **4** | 2 | 0 | 0.0 | 30 | 9 | 4 | 1 | 2 | 30.0 | 23 | 1 | 3 |

**## How many rows and columns are in this dataset in total? What are the names of columns?**

```
In [52]: telecust_df.shape
```

```
Out[52]: (1000, 12)
```

**# Let’s define X, and y for our dataset:**

```
In [53]: X=np.asarray(telecust_df[['region','gender','retire','age','address','reside', 'marital','employ','inco
         X[0:5]
```

```
Out[53]: array([[  2.,   0.,   0.,  44.,   9.,   2.,   1.,   5.,  64.],
                [  3.,   0.,   0.,  33.,   7.,   6.,   1.,   5., 136.],
                [  3.,   1.,   0.,  52.,  24.,   2.,   1.,  29., 116.],
                [  2.,   1.,   0.,  33.,  12.,   1.,   0.,   0.,  33.],
                [  2.,   0.,   0.,  30.,   9.,   4.,   1.,   2.,  30.]])
```

```
In [54]: y=np.asarray(telecust_df[["custcat"]])
         y[0:5]
```

```
Out[54]: array([[1],
                [4],
                [3],
                [1],
                [3]])
```

## Also, we normalize the dataset:

In [55]:
```
 from sklearn import preprocessing
X=preprocessing.StandardScaler().fit(X).transform(X)
X[0:5]
```

Out[55]:
```
array([[-0.03, -1.03, -0.22,  0.18, -0.25, -0.23,  1.01, -0.59, -0.13],
       [ 1.2 , -1.03, -0.22, -0.69, -0.45,  2.56,  1.01, -0.59,  0.55],
       [ 1.2 ,  0.97, -0.22,  0.82,  1.23, -0.23,  1.01,  1.79,  0.36],
       [-0.03,  0.97, -0.22, -0.69,  0.04, -0.93, -0.99, -1.09, -0.42],
       [-0.03, -1.03, -0.22, -0.93, -0.25,  1.16,  1.01, -0.89, -0.44]])
```

# Train/Test dataset

In [56]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X, y, test_size=0.2,random_state=4)
print("Train set :", X_train.shape, y_train.shape)
print("Test set :", X_test.shape, y_test.shape)
```

```
Train set : (800, 9) (800, 1)
Test set : (200, 9) (200, 1)
```

## Modeling (Logistic Regression with Scikit-learn)

Let's build our model using LogisticRegression from the Scikit-learn package. This function im•plements logistic regression and can use different numerical optimizers to find parameters, including 'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga' solvers. You can find extensive information about thepros and cons of these optimizers if you search it in the internet.

The version of Logistic Regression in Scikit-learn, support regularization. Regularization is a tech•nique used to solve the overfitting problem of machine learning models. C parameter indicatesinverse of regularization strength which must be a positive float. Smaller values specifystronger regularization. Now let's fit our model with train set:

In [57]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
LR=LogisticRegression(C=0.01, solver="liblinear").fit(X_train, y_train)
LR
```

```
C:\Users\prabh\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1184: DataConversionWarning: A c
olumn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[57]:
```
▾               LogisticRegression
LogisticRegression(C=0.01, solver='liblinear')
```

## Now we can predict using our test set:

In [58]:
```
yhat=LR.predict(X_test)
yhat
```

Out[58]:
```
array([4, 1, 4, 1, 4, 3, 3, 1, 3, 3, 4, 1, 3, 1, 3, 1, 3, 4, 3, 1, 3, 1,
       1, 1, 1, 3, 1, 1, 1, 1, 3, 1, 4, 3, 1, 3, 3, 1, 3, 3, 3, 1, 1, 1,
       4, 1, 1, 1, 1, 2, 3, 1, 1, 3, 3, 1, 3, 1, 1, 1, 1, 3, 3, 3, 3, 3,
       3, 3, 4, 1, 1, 3, 3, 3, 1, 1, 1, 2, 1, 3, 1, 1, 3, 1, 1, 3, 3, 1,
       4, 1, 1, 3, 1, 4, 4, 3, 1, 3, 1, 1, 3, 1, 1, 1, 1, 1, 3, 3, 3, 1,
       4, 3, 3, 1, 1, 3, 1, 3, 4, 1, 1, 1, 3, 3, 1, 1, 1, 3, 4, 3, 3, 1,
       4, 3, 3, 3, 1, 1, 1, 3, 1, 3, 1, 1, 1, 3, 4, 3, 1, 1, 1, 3, 3, 3,
       3, 3, 3, 3, 3, 4, 3, 3, 1, 1, 1, 1, 1, 1, 3, 1, 3, 1, 3, 3, 1, 3,
       3, 3, 4, 1, 3, 3, 1, 1, 3, 3, 3, 1, 3, 1, 3, 1, 1, 1, 3, 4, 3, 1,
       1, 1])
```

## ## predict_proba returns estimates for all classes, ordered by the label of classes.

This method computes the predicted probabilities of the classes for each sample in X_test. It returns an array of shape (n_samples, n_classes) where each row corresponds to a sample and each column corresponds to a class, with probabilities summing to 1 across each row.

In [44]:
```
yhat_proba=LR.predict_proba(X_test)
yhat_proba
```

```
       [0.3 , 0.24, 0.2 , 0.26],
       [0.27, 0.25, 0.26, 0.23],
       [0.37, 0.22, 0.18, 0.23],
       [0.12, 0.23, 0.46, 0.19],
       [0.32, 0.22, 0.22, 0.24],
       [0.21, 0.21, 0.31, 0.27],
       [0.29, 0.22, 0.25, 0.24],
       [0.36, 0.22, 0.2 , 0.22],
       [0.35, 0.17, 0.28, 0.2 ],
       [0.25, 0.24, 0.25, 0.25],
       [0.24, 0.25, 0.26, 0.26],
       [0.24, 0.24, 0.27, 0.25],
       [0.32, 0.23, 0.23, 0.22],
       [0.3 , 0.22, 0.26, 0.23],
       [0.32, 0.23, 0.21, 0.24],
       [0.19, 0.24, 0.32, 0.25],
       [0.2 , 0.25, 0.31, 0.24],
       [0.18, 0.26, 0.32, 0.24],
       [0.2 , 0.25, 0.33, 0.22],
       [0.17, 0.24, 0.39, 0.2 ],
```

## ## Evaluation

## # jaccard index

Let’s try the jaccard index for accuracy evaluation. we can define jaccard as the size of theintersection divided by the size of the union of the two label sets. If the entire set of predictedlabels for a sample strictly matches with the true set of labels, then the subset accuracy is 1.0;otherwise it is 0.0.

In [59]:
```
from sklearn.metrics import jaccard_score

# Assuming y_test and yhat are your actual and predicted labels respectively

# If your data is multiclass:
jaccard = jaccard_score(y_test, yhat, average='weighted')  # Choose 'micro', 'macro', or 'weighted' base

# If your data is binary (two classes):
# jaccard = jaccard_score(y_test, yhat, average='binary')  # Use this if your data is binary

# Print or use the jaccard score
print("Jaccard Score:", jaccard)
```

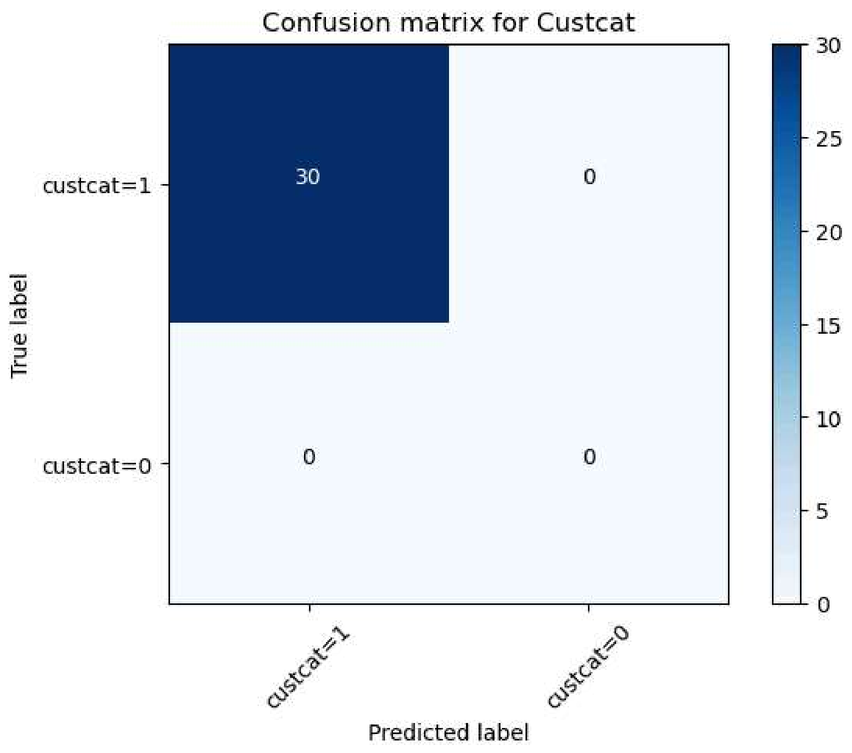
```
Jaccard Score: 0.15447792802529414
```

A Jaccard score of approximately 0.154 means that there is a relatively low similarity or overlap between the predicted custcat labels and the true custcat labels. This suggests that the model's predictions do not align well with the actual classes in the dataset.

In [65]:

```python
from sklearn.metrics import classification_report, confusion_matrix
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Compute confusion matrix
# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['custcat=1', 'custcat=0'],
                      title='Confusion matrix for Custcat')
plt.show()

# Optionally, print classification report for more detailed metrics
print("Classification Report:")
print(classification_report(y_test, yhat))
```

```
Confusion matrix, without normalization
[[30  0]
 [ 0  0]]
```



```
Classification Report:
              precision    recall  f1-score   support

           1       0.32      0.59      0.41        51
           2       0.50      0.02      0.04        44
           3       0.30      0.46      0.36        54
           4       0.32      0.12      0.17        51

    accuracy                           0.31       200
   macro avg       0.36      0.30      0.25       200
weighted avg       0.35      0.31      0.26       200
```

In [36]:
```
print(classification_report(y_test, yhat))
```

```
              precision    recall  f1-score   support

           1       0.32      0.59      0.41        51
           2       0.50      0.02      0.04        44
           3       0.30      0.46      0.36        54
           4       0.32      0.12      0.17        51

    accuracy                           0.31       200
   macro avg       0.36      0.30      0.25       200
weighted avg       0.35      0.31      0.26       200
```

**## Based on the count of each section, we can calculate precision and recall of each label:**

• Precision is a measure of the accuracy provided that a class label has been predicted. It isdefined by: precision = TP / (TP + FP)
• Recall is the true positive rate. It is defined as: Recall = TP / (TP + FN)

**## So, we can calculate the precision and recall of each class.**

**## F1 score:**

Now we are in the position to calculate the F1 scores for each label based on the precisionand recall of that label.The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its bestvalue at 1 (perfect precision and recall) and worst at 0. It is a good way to show that a classiferhas a good value for both recall and precision.

## Finally, we can tell the average accuracy for this classifier is the average of the F1-score for bothlabels, which is 0.26 in our case.

# log loss

Now, let’s try log loss for evaluation. In logistic regression, the output can be the probabilityof customer churn is yes (or equals to 1). This probability is a value between 0 and 1. Logloss( Logarithmic loss) measures the performance of a classifier where the predicted output is aprobability value between 0 and 1.

In [41]:
```python
from sklearn.metrics import log_loss
import numpy as np

# Assuming y_test and yhat_proba are defined
y_test = np.array([0, 1, 1, 0])
yhat_proba = np.array([[0.2, 0.8], [0.6, 0.4], [0.3, 0.7], [0.9, 0.1]])

# Compute log loss
loss = log_loss(y_test, yhat_proba)

print("Log Loss:", loss)
```

```
Log Loss: 0.7469410259762035
```

Log Loss value of approximately 0.747 suggests that, on average, the model's predicted probabilities are somewhat confident but not perfectly adjusted with the true probabilities. This indicates that there is room for improvement in refining the predicted probabilities to better match the actual probabilities observed in the test data.

Jaccard Score (0.154): Indicates low similarity between predicted and true labels.
Log Loss (0.747): Indicates some uncertainty or mismatch between predicted probabilities and actual outcomes.

## Try to build Logistic Regression model again for the same dataset, but this time, use differentsolver and regularization values? What is new logLoss value?

In [38]:
```python
LR2 = LogisticRegression(C=0.01, solver='sag').fit(X_train,y_train)
yhat_prob2 = LR2.predict_proba(X_test)
print ("LogLoss: : %.2f" % log_loss(y_test, yhat_prob2))
```

```
LogLoss: : 1.36

C:\Users\prabh\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1184: DataConversionWarning: A c
olumn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ),
for example using ravel().
  y = column_or_1d(y, warn=True)
```

# Result

## From the classification report , we can summarize the following:

## Precision:
The average precision across all classes (1, 2, 3, 4) is 0.36, indicating that, on average, 36% of the predictions for each class were correct.

## Recall:
The average recall across all classes is 0.30, meaning that, on average, the model correctly identified 30% of the samples belonging to each class.

## F1-score:
The average F1-score across all classes is 0.25, which is the harmonic mean of precision and recall. It provides a balanced measure of model performance considering both precision and recall.

## Support:
This shows the number of samples for each class in the test set. For example, class 1 has a support of 51, class 2 has 44, class 3 has 54, and class 4 has 51.

## Accuracy:
Accuracy: The overall accuracy of the model is 31%. This means that 31% of all predictions made by the model were correct across all classes (1, 2, 3, 4).

## Precision:
The precision scores for individual classes (1, 2, 3, 4) vary. For instance, class 2 has the highest precision of 0.50, indicating that when the model predicts class 2, it is correct 50% of the time. On the other hand, class 3 has a precision of 0.30, suggesting that predictions for class 3 are correct 30% of the time.

## Recall:
The recall scores also vary across classes. Class 1 has the highest recall of 0.59, meaning that the model correctly identifies 59% of the actual samples belonging to class 1. Class 2, however, has a very low recall of 0.02, indicating that the model struggles to correctly identify class 2 samples.

## F1-score:
The F1-scores provide a balance between precision and recall. Class 1 has an F1-score of 0.41, which is the highest among the classes. This indicates a relatively better balance between precision and recall for class 1 compared to other classes.

## Support:
The support values show how many samples belong to each class in the test set. This helps in understanding the distribution of classes and their impact on model evaluation metrics.

# Conclusion:
Based on these metrics, your model shows varying performance across different classes (1, 2, 3, 4). The overall accuracy of 31% indicates that the model's predictions are correct only about a third of the time. This suggests that there is room for improvement in the model's performance, potentially through feature engineering, selecting different algorithms, or tuning hyperparameters.

## Analyzing these metrics helps in understanding where the model performs well and where it needs improvement, guiding further steps in refining the model to achieve better predictive performance for the custcat column in your dataset.