✳ Logistic Regression

The most common use of logistic regression models is in binary classification problems.

Examples of classification

1) finance company wants to know whether a customer is default or not

2) Predicting an email is spam or not.

3) Categorizing email into promotional, personal and official.

A binary classification problem is a type of supervised learning problem where the goal is to classify input data into one of two possible categories or classes

Key features

① Two classes : the target variable (output) has exactly two unique values often represented as : → 0 and 1 (eg Yes/No, Positive/Negative)
→ True or false
→ Any two distinct labels (Spam not spam)

② Objective : to learn a decision boundary or function that can accurately distinguish btw two classes based on features in a dataset.

Simple Boundary Decision Method in LR

It is the process of using a decision boundary to separate data points into two classes based on their predicted probabilities.

Logistic regression models the probability of a binary outcome Y (eg 0 or 1) as a function of the input features X.

Steps:

① Probability Prediction:

→ Logistic regression uses the sigmoid func to predict the probability $P(y=1/x)$ which lies b/w 0 and 1:

$$P(y=1/x) = \frac{1}{1+e^{-(B_0+B_1x)}}$$

Here B_0 is the intercept

B_1 is the coefficient for the feature x.

② Decision Threshold:

→ A Threshold T (commonly $T=0.5$) is chosen to classify the outcome

if $P(y=1/x) \geq T$, classify as 1

if $P(y=1/x) < T$, classify as 0

3. Decision Boundary

→ It is the value of X where $P(y = 1/x) = T$

for $T = 0.5$, this corresponds to solving

$$B_0 + B_1 X = 0$$

In higher dimensions this becomes a hyperplane

$$B_0 + B_1 X_1 + B_2 X_2 + \ldots + B_n X_n = 0$$

Sigmoid Curve

The sigmoid curve is the graphical representation of the sigmoid func, which maps any real number Z to a value b/w 0 and 1, It is used in the logistic regression to predict the probabilities

⇒ Sigmoid func = $\sigma(z) = \frac{1}{1+e^{-z}}$

Where

$Z = B_0 + B_1 X$ (a linear combination of input features and model parameters)

* Properties of Sigmoid Curve:

① S Shape

↳ Sigmoid curve is S shaped.

It asymptotically approaches 1 as $z \to +\infty$ and 0 as $z \to -\infty$

② Range → O/P of the sigmoid func always b/w 0 and 1

3. Interpretation

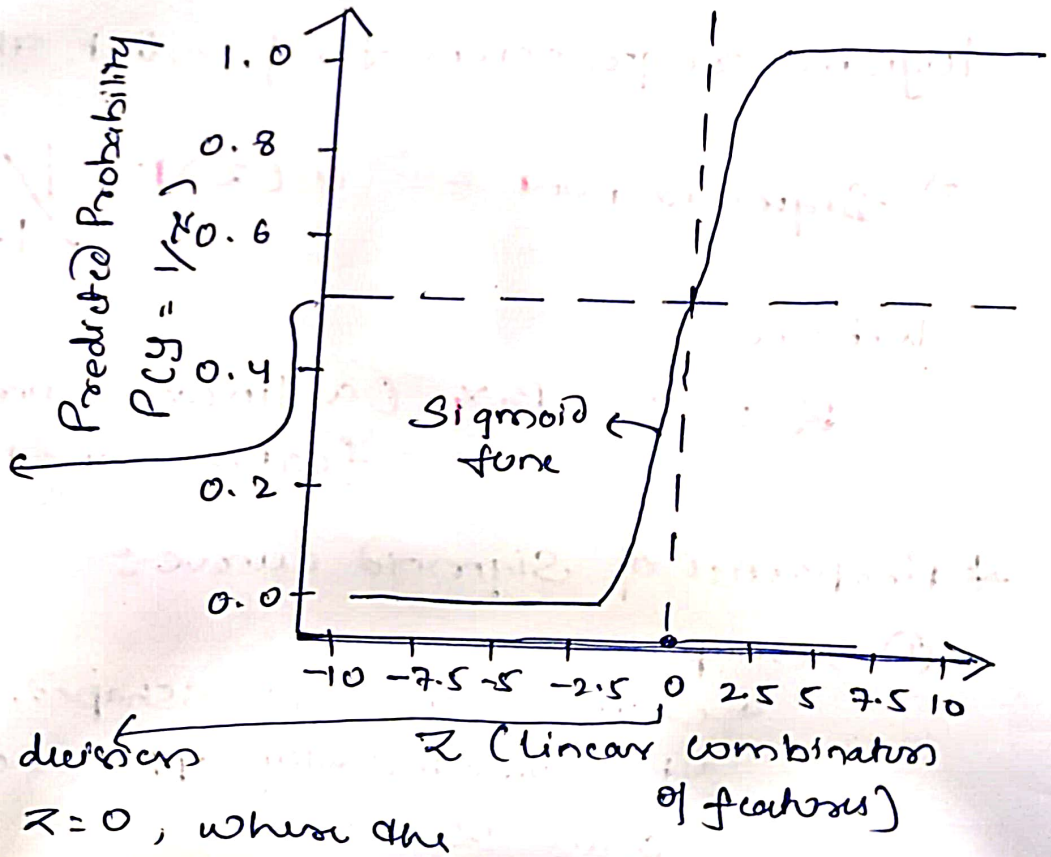
↳ The value $\sigma(z)$ represents the predicted prob of the positive class

the point where $\sigma(z) = 0.5$ corresponds to the decision boundary in logistic regression.

* Connection b/w Decision Boundary and Sigmoid Curve

→ the sigmoid curve defines the probability of the Positive class across the feature space.

→ the decision boundary corresponds to the point on the sigmoid curve, where the probability is equal to the threshold $T = 0.5$.



Indicates the threshold

Marks the decision boundary at $z = 0$, where the Predicted Prob is 0.5

The sigmoid curve equn

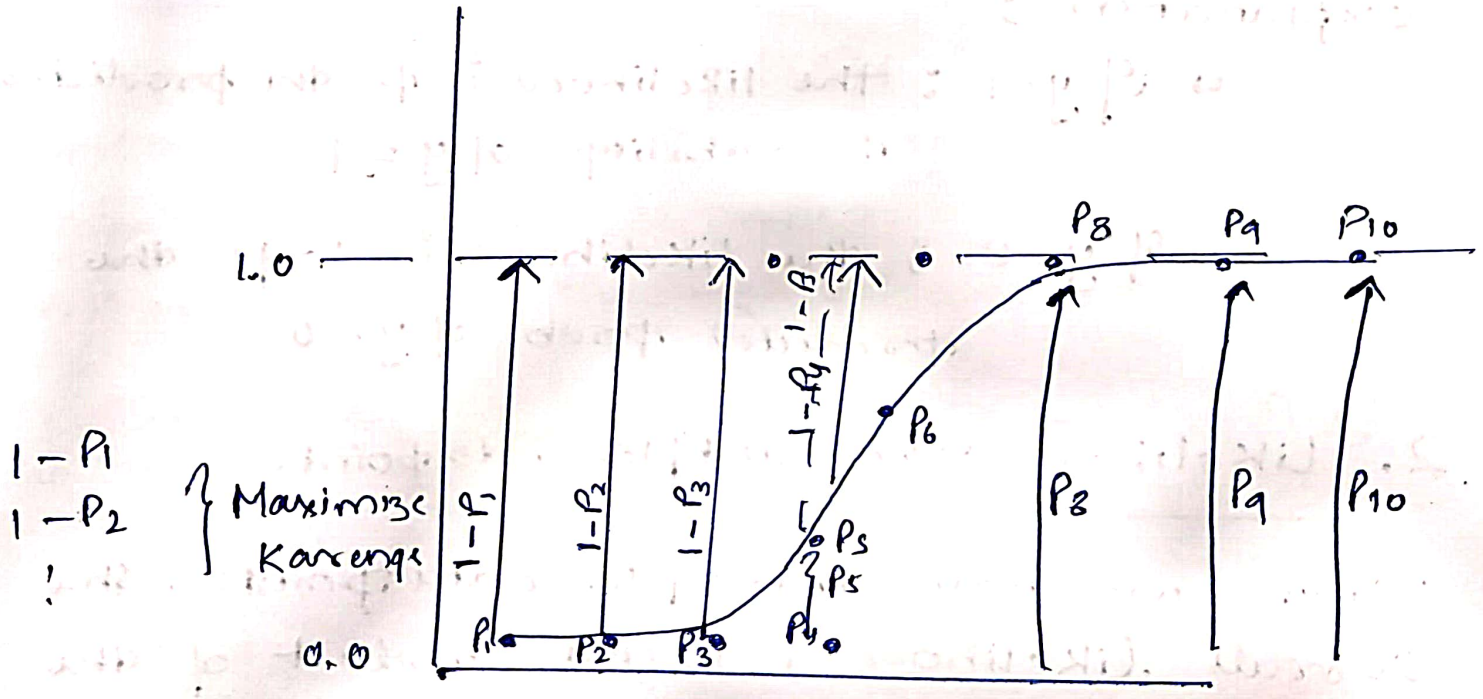
$$y = P(\text{Diabetes}) = \frac{1}{1 + e^{-(B_0 + B_1 x)}}$$ let's say

You take $B_0 = -15$ and $B_1 = 0.065$ now what will be the prob of diabetes for a patient with sugar level 220 ⇒ $1/1 + e^{-(-15 + 0.065 \times 220)}$

$= 0.33$

✱ By varying the values of B_0 and B_1, we get diff sigmoid curves, Now based on some function that we have to minimize or maximize, we get the best sigmoid curve.

⇒ Finding the best fit sigmoid Curve



For Best B_0, B_1

Product = $(1-P_1)(1-P_2)(1-P_3)(1-P_4)$
$(1-P_6)(P_5)(P_7)(P_8)(P_9)(P_{10})$
Has to be maximize

What is Likelihood?

Likelihood measures how well the predicted probabilities of a model align with the actual outcomes in the data. It's a core concept in statistical modelling and is particularly useful for evaluating models like logistic regression.

1. Likelihood for a single datapoint

↳ for a binary outcome y, the likelihood depends on whether $y=1$ or $y=0$

formula : $L = p^y \cdot (1-p)^{1-y}$

Explanation :

↳ if $y=1$: the likelihood is p, the predicted probability of $y=1$

if $y=0$: the likelihood is $1-p$, the predicted prob of $y=0$

2. Likelihood for multiple datapoints

When we have multiple datapoints, the overall likelihood L is the product of the likelihoods for all individual data points:

$$L = \prod_{i=1}^{n} \left(P_i^{y_i} \cdot (1 - P_i)^{1 - y_i} \right)$$

Example table :

Datapoint	Predicted Prob (P)	Actual outcome y	Likelihood Contribution
1	0.3	0	1 − 0.3
2	0.4	0	1 − 0.4
3	0.7	1	0.7
4	0.9	1	0.9

$$L = (1 - 0.3)(1 - 0.4)(0.7)(0.9) = 0.2268$$

3. Log-Likelihood

Computing likelihood directly for many datapoints can result in very small numbers (due to multiplying probabilities), leading to numerical instability, to simplify we use the natural algorithm:

$$l = \sum_{i=1}^{n} \left(y_i \cdot \log(P_i) + (1 - y_i) \cdot \log(1 - P_i) \right)$$

Log likelihood converts the product into sum making it easier to optimize.

Why is likelihood important?

① Model Evaluation : Likelihood Quantifies how well the model's predicted probabilities align with observed data.

② Parameter evaluation : In logistic regression we maximize the likelihood (or equivalently the log-likelihood) to find the best fitting model parameters.

What are B_0 and B_1?

These are the coefficients in logistic regression :

→ B_0 is the intercept (shifts the curve up or down)

→ B_1 is the slope (controls how steep the curve is)

We want to find the best values for B_0 and B_1, so the curve fits the data perfectly, to do this we maximize a likelihood func.

The likelihood func measures how well the logistic curve predicts the actual data, It looks like this

$$L = \prod_{i=1}^{n} P_i^{y_i} (1 - P_i)^{1-y_i}$$

Here P_i is the predicted Probability.

y_i is the actual outcome (1 or 0)

Logistic Regression Equation

$$P(\text{diabetes}) = \frac{1}{1+e^{-(B_0+B_1X)}}$$

where
$B_0 = -13.5$
$B_1 = 0.06$

Now we can plugin x and compute the Prob of a person having diabetes.

Linearising the Sigmoid Equation

$$P = \frac{1}{1+e^{-(B_0+B_1X)}} \quad \& \quad 1-P = \frac{e^{-(B_0+B_1X)}}{1+e^{-(B_0+B_1X)}}$$

$$P/_{1-P} = \frac{1}{1+e^{-(B_0+B_1X)}} \div \frac{e^{-(B_0+B_1X)}}{1+e^{-(B_0+B_1X)}}$$

$$\Rightarrow P/_{1-P} = e^{(B_0+B_1X)} \quad \text{\# taking log on both sides}$$

$$\Rightarrow \ln(P/_{1-P}) = B_0 + B_1X$$

Here $P/_{1-P}$ is called as odds

$\ln(P/_{1-P})$ = Log odds

If $\frac{P}{1-P}$ is 4 $\Rightarrow$ P(Diabetes) = 4 × P(No Diabetes)

$\therefore$ P → Prob of diabetes

Relationship b/w ODDS and Probability

Suppose $P = 0.5 \Rightarrow 1-P = 0.5 \Rightarrow \text{odds} = \frac{P}{1-P} = \frac{0.5}{0.5} = 1$

If $\text{odds} = 2 = \frac{P}{1-P} = 2 \Rightarrow P = 2 - 2P \Rightarrow 3P = 2$

$\Rightarrow P = 2/3 = 0.67$

In logistic regression, odds and log odds are used to model probabilities in a way that ensures the predicted values remains b/w 0 and 1.

The odds measure the likelihood of an event happening compared to it not happening.

Mathematically ODDS are calculated as

$$\text{ODDS} = \frac{P}{1-P}$$

where
P ⇒ Prob of event happening
1-P ⇒ Prob of event not happening.

eg if $P = 0.75$ then odds are $\Rightarrow \frac{0.75}{1-0.75} = 3$ meaning the event is 3 times as likely to happen as not

Log-odds (Logit)

↳ Log odds is the natural logarithm of the odds

$$\text{Log-odds (Logit)} = \ln\left(\frac{P}{1-P}\right)$$

the log-odds transforms the probability range $[0,1]$ into a range of $(-\infty, \infty)$, this transformation is important because logistic regression models a linear relationship b/w the input variables and the log-odds

Why to use odds and Log-odds in Logistic Regression

① Handling Non-Linearity: Logistic regression cannot directly model probabilities because they are bounded b/w 0 and 1, by modelling the log odds the regression becomes a linear problem

⇒ Probabilities are bounded b/w 0 and 1, if we try to directly model probabilities P as a linear func of the input X.

$$P = B_0 + B_1 X$$

this approach has two major issues:

① Bound violation: the linear equation can produce values outside the range $[0,1]$, which are not valid probabilities

2. Non linear relationship : The change in Probability for a one unit increase in x is not consistent across the range of P

for eg increasing x might lead to a large change in P when P is near 0.5 but only a small change when P is near 0 or 1

The Role of Log-odds

To overcome these issues, logistic regression models the log-odds instead of directly modeling probabilities.

$$\text{Log-odds} = \ln\left(P/1-P\right)$$

this transformation maps the probability range [0, 1] to the log-odds range $(-\infty, \infty)$

unlike probabilities log-odds are not bounded and can take any real value.

Modelling the log-odds as a linear func of x

$$\ln\left(P/1-P\right) = B_0 + B_1X$$

this eqn represents a linear relationship b/w log odds and X, which can be solved using standard regression techniques.

the coefficients B_0 and B_1 can now be interpreted as changes in log-odds per unit change in X.

Mapping Back to Probabilities

to predict the probabilities from the linear model of log-odds, we use the inverse transformation (the sigmoid func)

$$P = \frac{1}{1 + e^{-(B_0 + B_1 X)}}$$

the sigmoid func ensures the predicted values are always b/w 0 and 1

this effectively captures the non linear relationship b/w X and P.

⊛ Multivariate Logistic Regression (Model Building)

Eqn used in case of univariate logistic regression

$$P = 1/1 + e^{-(B_0 + B_1 X)}$$ ⇒ the above eqn has only one feature variable X, for which the coefficient is B_1

If we have multiple features say n, you can simply extend the above eqn with n features variables

$$P = \frac{1}{1 + e^{-(B_0 + B_1 X_1 + B_2 X_2 + B_3 X_3 + \ldots + B_n X_n)}}$$

Multivariate Logistic Regression Telecom churn example:

Problem statement

↳ We will be looking at the telecom churn prediction example → we will be using features to predict whether a particular customer will switch to another telecom provider or not (ie churn or not)

→ Here 'Churn' is a binary variable: 1 means the customer has churned and 0 means the customer has not churned.

Datasets that we have: ① churn dataset
② Internet_data dataset
③ Customer_data dataset

* Dataframe1: churn_data columns

cust_id, tenure, phoneservice, contract, Paperlessbilling, Monthly charges, Total Charges, churn column.

* Dataframe 2: customer data

cust_id, gender, senior citizen, partner, dependents

* Dataframe 3: Internet_data

cust_id, multiple lines, internet service, online security, online Backup, Device Protection, tech support, streaming TV, streaming Movies

the cust_ids are same in all the dataframes, here we can merge the data into single dataframe.

Let's see the Data cleaning & Preparation

```
# suppress warnings
> import warnings
> warnings.filterwarnings('ignore')
# importing pandas and numpy
> import pandas as pd, numpy as np
# Importing all datasets
> churn_data = Pd.read_csv("churn_data.csv")
> customers_data = Pd.read_csv("customers_data.csv")
> internet_data = Pd.read_csv("Internet_data.csv")
# Combining all data files into one consolidated df
> df_1 = Pd.merge(churn_data, customers_data,
                  how='inner', on='customerID')
# final dataframe with all the predictor vars
> telecom = pd.merge(df_1, internet_data, how='inner'
                     on='customerID')
> telecom.head()
> telecom.shape
    (7043, 21)
```

Statistical Info

> telecom.describe()

let's see the type of each col

> telecom.info()

↳ No Null values

Step 3 : Data Preparation

Converting some binary variables (Yes/No) to (1/0)

List of variables to map

> varlist = ['PhoneServic', 'PaperlessBilling', 'Churn',
'Partner', 'Dependents']

Defining the map func

> def binary_map(x):
return x.map({'Yes': 1, 'No': 0})

> telecom[varlist] = telecom[varlist].apply(binary_map)

Creating a dummy variable for some of the categorical variables and dropping the first one

> dummy1 = Pd.get_dummies(telecom[['contract',
'PaymentMethod', 'gender', 'InternetServic']
], drop_first = True)

```
# Adding the results to the master dataframe
telecom = pd.concat([telecom, dummy1], axis=1)
# Now we got the telecom data.
# Now doing the same for the remaining categorical variables
# creating dummy variables for the variable 'Multiple lines'
> ml = pd.get_dummies(telecom['MultipleLines'], prefix='MultipleLines')
# Dropping MultipleLines_No phone service column
> ml1 = ml.drop(['MultipleLines_No Phone service'], 1)
#Adding the results to master dataframe
> telecom = pd.concat([telecom, ml1], axis=1)
# creating ... for "Onlinesecurity" col
> os = pd.get_dummies(telecom['Onlinesecurity'], Prefix='Onlinesecurity')
> os1 = os.drop(['Online Security_No internet service'], 1)
> telecom = pd.concat([telecom, os1], axis=1)
# creating dummy for the variable 'Online Backup'
> ob = pd.get_dummies(telecom['Online Backup'], Prefix='Online Backup')
> ob1 = ob.drop(['Online Backup_No Internet service'],1)
> telecom = pd.concat([telecom, ob1], axis=1)
```

Creating dummy variable for the variable 'Device Protection'.

```
> dp = pd.get_dummies (telecom ['DeviceProtection'],
                       Prefix = 'DeviceProtection')
> dp1 = dp.drop ([['DeviceProtection_No internet
                    service'], 1)
> telecom = pd.concat ([[telecom, dp1], axis=1)
```

Creating dummy variable for the variable 'Tech Support'

```
> ts = Pd.get_dummies (telecom ['TechSupport'],
                       Prefix = 'TechSupport')
> ts1 = ts.drop ([['TechSupport_No internet service'],
                  1)
> telecom = Pd.concat ([telecom, ts1], axis = 1)
```

Creating dummy variables for 'StreamingTV'

```
> st = Pd.get_dummies (telecom ['StreamingTV']
                       Prefix = 'StreamingTV')
> st1 = st.drop ([['StreamingTV_No internet service'],1)
> telecom = Pd.concat ([[telecom, st1], axis=1)
```

Creating dummy variables for the variable StreamingMovies

```
> sm = pd.get_dummies (telecom ['StreamingMovies'],
                       Prefix = 'StreamingMovies')
> sm1 = sm.drop ([['StreamingMovies_No Internet
                   service'], 1)
> telecom = Pd.concat ([[telecom, sm1]], axis= 1)
```

Dropping the repeated variables

> telecom = telecom.drop([['Contract', 'PaymentMethod
....], 1)

the variable was imported as string we need to convert it to float

> telecom['TotalCharges'] = telecom['TotalCharges'].
convert_objects(convert_numeric
= True)

So the process of dummy variable creation was quiet familiar, except this time, you manual dropped one of the columns for many dummy variables for eg the column 'MultipleLines' we dropped the level 'MultipleLines_NoPhoneService' manually, instead of simply using the drop_first = True, the reason we did this is that if you check the variable multiple lines

telecom['MultipleLines'].astype('category').value_counts

No	3390
Yes	2971
No Phone Service	682

Now out of these, it is best to drop 'No Phone Service' since it isn't of any use because it is anyway being indicated by the variable 'PhoneService' already present in dataframe

Checking for outliers in continious variables

> num_telecom = telecom [['tenure', 'MonthlyCharges', 'SeniorCitizen', 'TotalCharges']]

checking outliers at 25, 50, 75, 90, 95, 99

> num_telecom.describe (Percentiles = [.25, .5, .75, .90, .95, .99])

↳ From the distribution shown above, you can see that there are no outliers in the Data, the nos are gradually increasing.

Checking the missing values and imputing them

Adding up the missing values (colwise)

> telecom.isnull().sum()

↳ total charges ⇒ 11 values are missing

rest all the values are present in the dataset

It means that 11 out of 7043 records have missing values ie 0.1%. → It is best to remove these observations from analysis.

Checking the % age of Missing values

> round (100 * (telecom.isnull().sum() / len (telecom.index)),2)

removing NAN total charges rows

> telecom = telecom [~ np.isnan (telecom ['TotalCharges'])]

checking the % of missing values after removing the missing values.

> round(100 * (telecom.isnull().sum()/len(telecom.index)), 2)

#now we don't have missing values

Step 4 : Test-Train Split

> from sklearn.model_selection import train_test_split

Putting feature variables to X

> X = telecom.drop(['churn', 'customerID'], axis=1)

Putting response variable to y

> y = telecom['churn']

Splitting the data into train and test

> X_train, X_test, y_train, y_test = train_test_split(X, y, train_size= 0.7, test_size = 0.3, random_state = 32)

Step 5 : feature Scaling

> from sklearn.preprocessing import StandardScaler

> scaler = StandardScaler()

> X_train [['tenure', 'MonthlyCharges', 'TotalCharges']] = scaler.fit_transform(X_train[['tenure', 'MonthlyCharges', 'TotalCharges']])

Step6 : Looking at correlations

> import matplotlib.pyplot as plt
> import seaborn as sns
% matplotlib inline

let's see the correlation matrix

> Plt.figure (figsize = (20,10))
> sns.heatmap (telecom.corr(), annot = True)
> Plt.show()

Dropping highly correlated dummy variables
(Here in this part little, business context also helps)

> X_test = X_test.drop (['MultipleLines_NO', 'OnlineSecurity
-NO', 'OnlineBackup_NO', 'DeviceProtection_NO',
'StreamingTV_NO', 'StreamingMovies_NO'], 1)

> X_train = X_train.drop (['MultipleLines_NO', 'Online
Security_NO', 'OnlineBackup_NO',
'DeviceProtection_NO',
'StreamingMovies_NO'], 1)

After dropping highly correlated variables, we will check the correlation

> Plt.figure (figsize = (20, 10))
> sns.heatmap (X_train.corr(), annot = True)
> Plt.show()

Running the first training Model

> import statsmodels.api as sm

Logistic Regression Model

> logm1 = sm.GLM(y_train, (sm.add_constant(X_train)), family = sm.families.Binomial())

this is for the whole binomial thing

> logm1.fit().summary()

↳ we can see the Pvalues

the lower the Pvalue, the higher is its significance.

✳ Feature elimination using RFE

We built the first model in the previous segment, based on the summary statistics we concluded that many of the features might be insignificant hence we need to do some feature elimination.

step 8 : feature selection using RFE

> from sklearn.linear_model import LogisticRegression

> logreg = LogisticRegression()

```
> from sklearn.feature_selection import RFE
> rfe = RFE(logreg, 15)
                    ↳ 15 top features to select
> rfe = rfe.fit(X_train, y_train)
> rfe.support_
        ↳ It will return true/false if feature
          is there in the top 15.
> list(zip(X_train.columns, rfe.support_,
                              rfe.ranking_))
    ↳
      Colnames    support    rank
# let's see the included cols
> col = X_train.columns[rfe.support_]
> Print(col)
# let's see the excluded cols
                    Negation operator
> X_train.columns[~rfe.support_]
        ↳ Cols which are excluded

# Assessing the models with StatsModels
X_train_sm = sm.add_constant(X_train[col])
                  generalized linear model
logm2 = sm.GLM(y_train, X_train_sm,
                family=sm.families.Binomial())
res = logm2.fit()
res.summary()
```

All of them have a good p value.

Getting the predicted values on the train set

> y_train_pred = res.predict(X_train_sm)

> y_train_pred = y_train_pred.values.reshape(-1)
y reshaping it

Creating a dataframe with the actual churn flag and predicted probabilities

> y_train_pred_final = Pd.Dataframe({'Churn': y_train.values,
'Churn_prob': y_train_pred})

> y_train_pred_final['cust_id'] = y_train.index

> y_train_pred_final.head()

#	Churn	Churn Prob	Cust_id
	0	0.2251	879
	0	0.2743	5790
	1	0.6921	6498

creating a new column 'Predicted' with 1 if the churn_prob > 0.5 else 0

> y_train_pred_final['predicted'] = y_train_pred_final.Churn_Prob.map(lambda x: 1 if x > 0.5 else 0)

> y_train_pred_final.head()

Confusion Matrix and Accuracy

We chose a cutoff of 0.5 in order to classify the customers into 'churn' and 'Not churn', Now since we are classifying the customers into two classes, we will obviously have some errors.

The classes of errors that would be there are:

→ Churn customers being (incorrectly) classified as Non-churn

→ Non churn customers being (incorrectly) classified as 'churn'

⇒ To capture these errors and to evaluate how well the model is → we will use confusion matrix

A Confusion Matrix would look like

Actual	Predicted	
	No (Non churn)	Yes (churn)
No (Non-churn)	1406	143
Yes (churn)	263	296

The table shows the comparison of predicted and actual labels, the actual labels are along the vertical axis while the predicted labels are along the horizontal axis

Thus the second row and first column (263) is the nos of customers who have actually churned but the model has predicted them as non churn

Similarly the cell at the second row, second col (298) is the number of customers who are actually churned and also predicted as churned.

The simplest model evaluation metric for classification model is "accuracy" → It is the percentage of correctly predicted labels

So what would be the corrected predicted labels be

↳ Churn customers being actually identified as churn

↳ Non churn customers being actually identified as non churn.

$$\text{Accuracy} = \frac{\text{Correctly predicted labels}}{\text{Total Number of labels}}$$

$$\Rightarrow \frac{1406 + 298}{1406 + 143 + 263 + 298} = 80.75$$

> y_train_pred_final.head()

↳

Churn	Churn_Prob	CustID	predicted
0	0.225	879	0
0	0.274	5790	0
1	0.692	6498	1
1	0.504	880	1

> from sklearn import metrics

Confusion Matrix

> Confusion = metrics.confusion_matrix (y_train_pred_final.churn, y_train_pred_final.predicted)

> Print (confusion)

↳ [[3270 365]
[579 708]]

> Print(metrics.accuracy_score (y_train_pred_final.churn, y_train_pred_final.predicted))

Manual Feature Elimination

→ We used RFE to select 15 features, but still few features having higher Correlation, so for better feature elimination we use VIF

$$VIF = \frac{1}{1-R_i^2}$$

where i refers to the i^{th} variable which is being represented as a combination of the rest of the Independent variable.

Here R^2 means, how well the other features predict this feature.

Checking the VIF values for the feature variables

> from statsmodels.stats.outliers_influence import variance_inflation_factor

creating a VIF dataframe

> vif = pd.Dataframe()

> vif['features'] = X_train[col].columns

> vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in range(X_train.[col].shape(1))]

> vif['VIF'] = round(vif['VIF'], 2)

> vif = vif.sort_values(by='VIF', ascending=false)

> vif

↳ Phoneservice → 8.86 VIF (Hence drop it)

> col = col.drop('Phoneservice', 1)

> col

Let's rerun the model using selected variables

```
> X_train_sm = sm.add_constant(X_train[col])
> logm3 = sm.GLM(y_train, X_train_sm,
                 family = sm.families.Binomial())
> res = logm3.fit()
> res.summary()
> y_train_pred = res.predict(X_train_sm).values.
                 reshape(-1)
> y_train_pred_final['Churn_prob'] = y_train_pred
```

Creating new col 'Predicted' with 1 if churn_prob > 0.5 else 0

```
> y_train_pred_final['Predicted'] = y_train_pred_final.
      Churn_Prob.map(lambda x:
      1 if x > 0.5 else 0)
> y_train_pred_final.head()
```

↳

Churn	Churn_prob	CustID	Predicted
0	0.254	879	0
0	0.224	5790	0
1	0.693	6498	1
1	0.510	880	1

let's check the overall accuracy

Point(metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.predicted)

↳ 0.805 → 80.5%.

Again check for VIF

↳ total charges → VIF 7

Now we can drop those columns based on business model or business use case.

Better is VIF < 5 for all features.

Q You are building a logistic regression model for a media company to predict whether a person will like a new show "Sacred Games" variable value of 1 indicates that the person liked the show, while 0 means they didn't, the coefficient for the 5 shows are as follows:

variable_name	coefficient val
True_detective_liked	0.47
Modern_family_liked	-0.45
Mindhunters_liked	0.39
Friends_liked	-0.23
Narcos_liked	0.55

Now we have the data of 3 consumers Reetesh, Kshitij, Shruti for these 5 shows indicating whether or not they liked these shows

Consumer	True-Detective	Modernfamily	Mindhunter	Friends	Narcos
Reetesh	1	0	0	0	1
Kshitij	1	1	1	0	1
Shruti	0	1	0	1	1

to find the person who is most likely to like the show we can use log odds

$$\ln\left(\frac{P}{1-P}\right) = B_0 + B_1X_1 + B_2X_2 + \dots B_nX_n$$

$$\Rightarrow 0.47X_1 - 0.45X_2 + 0.39X_3 - 0.23X_4 + 0.55X_5$$

For Reetesh: $0.47\times1 - 0.45\times0 + 0.39\times0 - 0.23\times0 + 0.55\times1 = 1.02$

For Kshitij: $0.47\times1 - 0.45\times1 + 0.39\times1 - 0.23\times0 + 0.55\times1 = 0.96$

for Shruti $\Rightarrow -0.13$

$\Rightarrow$ Reetesh liking the show is the highest

Metrics Beyond Accuracy : Sensitivity & Specificity

Ques is : Is accuracy enough to assess the goodness of the model? ⇒ Big NO!

to understand why accuracy is often not the best Metric (Example)

→ Increasing churn is the most serious issue in the telecom company, company desperately wants to retain customers, Marketing Head decides to roll out discounts/ offers to the customers likely to churn

Its imp that the model identifies almost all the churn customers correctly. It is fine if it incorrectly predicts some of the non churn customers as churn

Since in that case → the most that will happen is that company will offer discounts to them.

Confusion matrix we got for our model again - the actual labels are along the column while the predicted labels are along the rows.

Actual / Predicted	Not churn	Churn
Not churn	3869	366
Churn	595	692

From the table we can see that there are

595 + 692 = 1287 actual churn customers

So ideally model should predict all of them as churn but out of these 1287, the current model only predicts 692 as churn ⇒ only 692 out of 1284 or only about 53% of churn customers will be predicted by the model as churn ⇒ It's very risky.

So even if the accuracy is 80% the model only predicts 53% of churn cases correctly.

Here what's happening is that we care more about one class (class = 'churn') than the other, It's a very common situation is classification problem → we may also care more about one class than the other.

On the other hand the accuracy tells you the model performance on both classes combined → which is fine but not the most imp metric.

Example 2 : Suppose we are building a logistic regression model for cancer patients, based on certain features we need to predict whether the patient has cancer or not, Here if we incorrectly predict many diseased patients as not having cancer, It can be very risky.

In such cases it is better that instead of looking at the overall accuracy, you care about predicting the 1s (the diseased) correctly.

Similarly, if we are building a model to determine wether you should block (where blocking is 1 and not blocking is 0) a customer's transactions or not based on the past transaction behaviour in order to identify frauds → Here we would care more about getting the 0s right, Because we might not want to wrongly block a good customers transactions as it might lead to a very bad customer experience.

* Hence it's crucial that we consider the overall business problem you are trying to solve to decide the metric you want to maximize or minimize.

This brings us to two of the most commonly used Metrics to evaluate a classification model.

① Sensitivity

② Specificity

Sensitivity is defined as :

$$\text{Sensitivity} = \frac{\text{Nos of actual Yeses correctly predicted}}{\text{Total Nos of Actual Yeses}}$$

Here 'yes' means 'churn' and 'no' means 'non-churn'

Actual / Predicted	Not churn	churn
Not churn	3869	366
churn	595	692

the different elements in the matrix can be labelled as follows

Actual / Predicted	Not churn	churn
Not churn	True negatives	False Positive
churn	False Negatives	True Positives

⇒ the first cell contains the actual Not churns being predicted as not churns hence labelled as True Negatives. (Negatives imply that class is 0 here, Not-churn)

⇒ the second cell contains actual not churns being Predicted as churn → Hence it is labelled as false tve

⇒ Similarly the third cell contains the actual churns being predicted as not churn → false -ve.

⇒ Fourth cell contains actual churns which are Predicted as churns hence true tve.

Sensity $= \frac{692}{595+692}$ → Actual Yes → total Yes $\approx 53.76\%$

So the sensitivity is quite low.

Similarly we have specificity:

$$\text{Specificity} = \frac{\text{Nos of actual No correctly predicted}}{\text{Total Nos of actual No}}$$

$$= \frac{3269}{3269+366}$$

When evaluating the performance of the classification model, accuracy (the ratio of correctly predicted observations to the total observations) is often insufficient, especially when dealing with imbalanced datasets, so to better assess model performance we use metrics like sensitivity and specificity, these are derived from Confusion Matrix.

Confusion Matrix is a must to understand for both the sensitivity and specificity.

Confusion Matrix

It is a table that summarizes the performance of a classification model with two classes (binary classification)

	Predicted Positive	Predicted Negative
Actual Positive	True Positive	False Negative
Actual Negative	False Positive	True Negative

True Positive : correctly predicted positives (TP)

True Negative (TN): correctly predicted Negatives

false Positives (FP) : Incorrectly predicted as +ve (Type I error)

false Negatives (FN) : Incorrectly predicted as -ve. (Type II error)

① Sensitivity (Recall or True Positive Rate)

→ Sensitivity measures the proportion of actual +ves that are correctly identified by the model.

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

Key points :

Answers the ques : How good is the model at identifying +ve cases?

High sensitivity means fewer false negatives which is critical in applications like disease diagnosis (eg detecting cancer)

eg : In a cancer test, sensitivity ensures that most cancer patients are correctly identified as +ve.

2. Specificity (True negative Rate)

It measures the proportion of actual -ves that are correctly identified as -ve by the model

$$\text{Specificity} = \frac{TN}{TN + FP}$$

Keypoints :

→ Answers the ques: How good is the model at identifying -ve cases

→ Higher specificity means fewer false +ve, which is critical in applications like fraud detection

eg In a fraud detection system, it ensures that the legitimate transactions are not flagged as fraudlent.

On our model

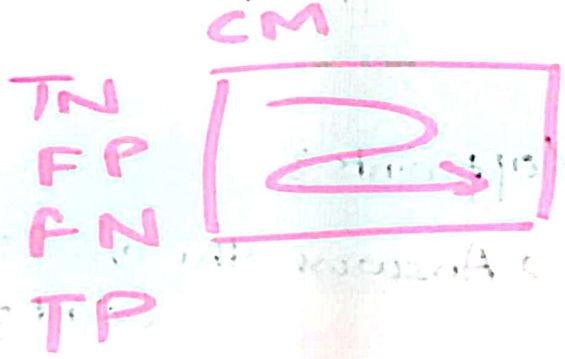
> confusion = metrics.confusion_matrix(
y_train_pred_final.churn, y_train_pred_
final.predicted)

> confusion

↳ [[3269, 366],
[595, 692]]

> metrics.accuracy_score(y_train_pred_final.
churn,
y_train_pred_final.predicted)

↳ 80.75



> TP = confusion[1,1]
> TN = confusion[0,0]
> FP = confusion[0,1]
> FN = confusion[1,0]

> Sens = TP / float(TP+FN)
> spec = TN / TN+FP

Calculate false +ve rate → Predicting churn when customer does not have churned

> FP/TN+FP.

Notes:

A confusion Matrix is a tool often used in the classification problems to evaluate the performance of the model ⇒ It is particularly usefull for the binary and multi class classification problem.

let's see in detail

Confusion Matrix Overview

for a binary classification problem, the confusion matrix is structured as follows:

	Predicted Positive	Predicted Negative
Actual Positive	True Positive	False Negative
Actual Negative	False Positive	True Negative

जब Predicted +ve class है और Actual भी Positive class ⇒ True Positive (TP)

जब Predicted +ve class है और Actual में -ve class है तो It is False Negative (FN)

जब Predicted -ve class है और Actual में -ve class है तो It is (TN)

जब Predicted +ve class है और Actual में -ve class है तो It is (FP)

→ True Positive (TP): the Model actually predicts a positive case, ie correctly Predicts a positive case.

→ False Positive (FP): Model Incorrectly predicts a Positive case, when its actually Negative.

→ True Negative (TN): Model correctly predicts a -ve case

→ False Negative (FN): The Model Incorrectly predicts a -ve case, when its actually +ve.

* Key Metrics Derived from Confusion Matrix

① Accuracy: Measures the overall correctness of the model

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

eg If TP = 50, TN = 30, FP = 10, FN = 10

$$\text{Accuracy} = \frac{50+30}{50+30+10+10} = 80/100 = 0.8$$

② Sensitivity (Recall or True Positive Rate)

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

Measures the proportion of actual +ve correctly identified

By using the same numbers

$$\text{Sensitivity} = \frac{50}{50+10} = 0.83$$

3. Specificity (True -ve Rate)

$$\text{Specificity} = \frac{TN}{TN+FP}$$

Measures the proportion of actual -ves correctly identified.

4. False Positive Rate (FPR)

$$FPR = \frac{FP}{FP+TN}$$

Measures the proportion of actual negatives, incorrectly classified as positive.

$$FPR = 1 - \text{Specificity}$$

$$FPR = \frac{10}{10+30} = \frac{10}{40} = 0.25$$

5. Precision (Positive Predicted Value)

$$\text{Precision} = \frac{TP}{TP+FP}$$

Measures the proportion of predicted values which are actually +ve.

Q Given the confusion matrix, to calculate the True Positive Rate and False Positive Rate

$$\text{TPositive Rate} = \frac{TP}{TP + FN}$$

$$\text{F Positive Rate} = \frac{FP}{FP + TN}$$

CM	Churn	Not
Churn	300	200
Not	100	400

$$\Rightarrow TPR = \frac{300}{300 + 200} = \frac{60}{500} \times 100 = 60$$

$$FPR = \frac{100}{100 + 400} = \frac{100}{500} \times 100 = 20$$

Q You have the following table showcasing the actual churn labels and predicted probabilities for 5 customers

Customers	Churn	Predicted Churn Prob
A1	1	0.52
A2	0	0.56
A3	1	0.78
A4	0	0.45
A5	0	0.22

To calculate true +ve rate and false +ve rate for cutoffs of 0.4 and 0.5, which of these cutoff gives a better model?

Note : the good model is the one in which TPR is high and FPR is low

Ans : to determine which cutoff (0.4 or 0.5) gives a better model we calculate TPR and FPR for both cutoff values.

→ True +ve (TP) : Predicted "churn" when actual churn is 1.

→ False +ve (FP) : Predicted churn when actual churn is 0.

→ false -ve (FN) : Predicted Non churn when actual churn is 1.

→ True -ve (TN) : Predicted Non churn when actual churn is 0.

$$\text{True Positive Rate (TPR}_{+ve}\text{)} = \frac{TP}{TP + FN}$$

$$\text{False Positive Rate (FPR)} = \frac{FP}{FP + TN}$$

1. Calculation for cutoff = 0.4 ⇒ if Predicted churn Prob ≥ 0.4 Predict churn (1) else predict Non-churn (0)

customer	Actual churn	Pred Prob	Predicted churn
A1	1	0.52	1
A2	0	0.56	1
A3	1	0.98	1
A4	0	0.45	1
A5	0	0.22	0

Step 2: Confusion Matrix

→ TP: 2 (A1, A3)
→ RP: 2 (A2, A4)
→ FN: 0
→ TN: 1 (A5)

$TPR = \frac{2}{2+0} = 1$

$FPR = \frac{2}{2+1} = 0.67$

2. Calculations for cutoff = 0.5

↳ if Predicted churn Prob ≥ 0.5 → Predict churn 1
otherwise predict non churn (0)

Customer	Actual Churn	Predicted Prob	Predicted churn
A1	1	0.52	1
A2	0	0.56	1
A3	1	0.78	1
A4	0	0.45	0
A5	0	0.22	0

TP = 2 (A1, A3)
RP = 1 (A2)
FN = 0
TN = 2 (A4, A5)

$TPR = \frac{2}{2+0} = 1$

$FPR = \frac{1}{1+2} = \frac{1}{3} = 0.33$

↳ Better

ROC Curve : Receiver Operating Characteristic Curve

The ROC curve is a graphical representation of the trade off b/w the True +ve Rate (TPR) also called as sensitivity and the False Positive Rate (FPR) at various classification thresholds.

Why do we need ROC curve?

When building a binary classification model (eg predicting churn vs non-churn), the classificat decision depends on a threshold applied to predicted probabilities (commonly 0.5 by default), However this threshold may not be optimal.

→ Sensitivity (TPR) : How well the model identifies actual positives.

→ Specificity : How well the model identifies actual -ve s.

The ROC curve helps us:

① Visualize the trade-off b/w TPR and FPR for different thresholds

② Choose an optimal threshold for classificaten balancing TPR and FPR

③ Evaluate overall performance of the model using AUC (Area Under curve)

Key Metrics

① True Positive Rate (TPR) : $\frac{TP}{TP+FN}$

↳ Also called as sensitivity or recall
It measures the proportion of Actual +ve correctly identified.

② False Positive Rate (FPR) : $\frac{FP}{FP+TN}$

↳ It measures the proportion of actual -ves incorrectly classified as +ves

How is ROC curve constructed

① Calculate TPR and FPR for various thresholds
eg (0.1, 0.2 ... 0.9)

② Plot TPR (y axis) vs FPR (X axis)

⇒ threshold = 1 : All predictions are -ve
(TPR = 0, FPR = 0)

⇒ threshold = 0 : All Predictions are +ve
(TRR = 1, FPR = 1)

As you vary the threshold the curve moves from (0, 0) to (1, 1)

Interpreting the ROC curve

① Closer to the top-left corner
↳ Indicates better performance (high TPR and low FPR)

② Diagonal line :
↳ Represents a model with no discriminatory pwr.
(AUC = 0.5)

③ AUC (Area under the curve)
AUC = 1 (Perfect Model)
AUC = 0.5 (Random guessing)

Higher AUC value indicates better model performance

Choosing the optimal threshold

① Maximize TPR while minimizing FPR
→ Identify the point on the curve closest to top left corner.
→ this often corresponds to a balance b/w sensitivity and specificity.

② Business context :
→ Optimal threshold depends on the problem.
→ for eg
In medical diagnosis, prioritize sensitivity (reduce false negatives)
→ In fraud detection → prioritize specificity (reduce false +ve)

Eg Let's assume the following predicted prob for 5 customers:

Customer	True Label	Predicted Prob
A	1	0.9
B	0	0.8
C	1	0.7
D	0	0.4
E	0	0.2

① Set diff thresholds (eg 0.5, 0.6 etc)
② Classify each customer as +ve or -ve
③ Calculate TPR and FPR for each threshold.
④ Plot TPR vs FPR for each threshold to get ROC curve.

⊛ ROC Curve in Python

Step1 ; import necessary libraries

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# True Labels
y_true = np.array([1, 0, 1, 0, 0])
# Predicted Prob from you model
y_scores = np.array([0.9, 0.8, 0.7, 0.4, 0.2])
```

Step3 : Compute TPR, FPR, thresholds

> fpr, tpr, thresholds = roc_curve (y_true, y_scores)

Step4 : Calculate AUC

> roc_auc = auc (fpr, tpr)

Step5 : Plot the ROC curve

```
> Plt.figure (figsize = (8,6))
> Plt.plot (fpr, tpr, color = "blue")
> Plt.plot ([0,1], [0,1], linestyle = "--", color = 'gray')
> Plt.xlabel ("False +ve rate")
> Plt.ylabel ("true +ve rate")
> Plt.show()
```

Diagonal line on ROC → Serves as a baseline

✳ Finding the optimal threshold for classification Model.

To find the optimal threshold for a classification model, we balance specificity, sensitivity and accuracy.

1. Understanding Metrics

→ Sensitivity (Recall / True Positive Rate):

↳ Proportion of actual positives correctly identified

$$\text{Sensitivity} = \frac{TP}{TP + FN}$$

→ Specificity (True Negative Rate):

↳ Proportion of actual negatives correctly identified.

$$\text{Specificity} = \frac{TN}{TN + FP}$$

→ Accuracy: Proportion of all correct predictions (both +ve and −ve)

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

2. Generate Predictions Across Thresholds

→ For classification models, probabilities are often output instead of binary labels, Vary the threshold from 0 to 1 to compute the corresponding confusion matrix and metrics at each threshold.

3. Compute Metrics for each threshold

→ At each threshold

① Classify data points as positive or negative based on threshold.

② Compute TP, TN, FP, FN from the confusion matrix

③ Calculate Specificity, sensitivity and accuracy

4. Choose an Optimization Criterion

Decide what you want to optimize

→ Maximize Accuracy: Choose the threshold where accuracy is highest.

→ Balanced Sensitivity and Specificity:

Choose the threshold where sensity ≈ specificity

Eg Python Implementation

```
import numpy as np
from sklearn.metrics import confusion_matrix
                            roc_curve, auc

y_true = np.array([0, 1, 0, 1, 0, 1, 1, 0, 0, 1])
y_probs = np.array([0.1, 0.4, 0.35, 0.8, 0.2, 0.9,
                    0.7, 0.15, 0.05, 0.85])
# Generate metrics for the thresholds
thresholds = np.arange(0, 1, 0.01)
sensitivities, specificities, accuracies = [], [], []
for threshold in thresholds:
    y_pred = (y_probs >= threshold).astype(int)
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred
                                      .ravel()
    sensitivity = tp/(tp+fn)
    specifity = tn/(tn+fp)
    accuracy = (tp+tn)/(tp+tn+fp+fn)
    sensitivities.append(sensitivity)
    specificities.append(specifity)
    accuracies.append(accuracy)
```

ravel() → flatten of 2D to 1D

find optimal threshold by maximizing accuracy

```
optimal_idx = np.argmax(accuracies)
optimal_threshold = thresholds[optimal_idx]
```

Plotting Metrics

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
plt.plot(thresholds, sensitivities, label='Sensitivity')
plt.plot(thresholds, specificities, label='specifity')
plt.plot(thresholds, accuracies, label='Accuracy')
plt.xlabel('Threshold')
plt.ylabel('Metric value')
plt.title('Sensitivity, specificity, Accuracy')
plt.show()
```

⊛ Precision & Recall

Confusion Matrix

True / Predicted	NO	Yes
NO	TN	FP
Yes	FN	TP

A prob that a Predicted yes is actually yes.

Actual +ve

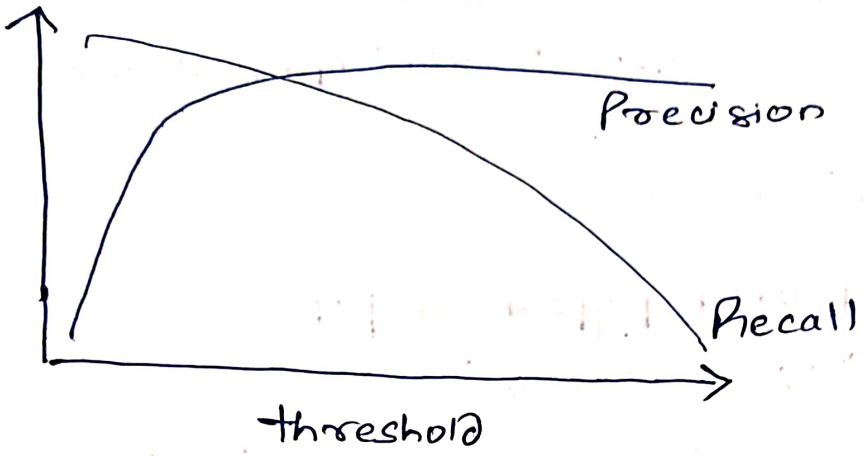
$$\text{Precision} = \frac{TP}{TP + FP}$$

total +ves

$$\text{Recall} : \frac{TP}{TP + FN}$$

→ Prob that a yes case is predicted as such

Precision and Recall Trade Off's



As threshold ↑ → Precision ↑

As threshold ↓ → Recall ↑

⊛ Precision: Probability that a predicted yes is actually a yes.

True/Predicted	NO	Yes
NO	TN	FP
yes	FN	TP

the formula for precision can be given as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

remember that Precision is the same as the Positive Predicted value.

* Recall : Probability that an actual yes is predicted correctly.

True / Predicted	No	Yes
No	TN	FP
Yes	fN	TP

$$Recall = \frac{TP}{TP + fN}$$ (Same as Sensitivity)

we can use Sklearn library

```
> from sklearn.metrics import precision_score,
                              recall_score
> Precision_score (y_train_pred_final.churn,
                   y_train_pred_final.predicted)

> recall_score (y_train_pred_final.churn,
                y_train_pred_final.predicted)
```

Q We have the confusion matrix as

Actual / Predicted	not churn	churn
not churn	1294	234
churn	223	359

What will be the approximate value of accuracy?

⇒ $\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Observations}}$

TN ⇒ 1294 (Not churn predicted as not churn)

TP = 359 (churn predicted as churn)

FN ⇒ 223 (wrongly predicted as not churn)

FP ⇒ 234 (wrongly predicted as churn)

total observations = TN + TP + FN + FP = 1294 + 359 + 223 + 234 = 2110

TN + TP = 1294 + 359 = 1653

⇒ $1653/2110 = 0.78$

$\text{Recall} = \frac{TP}{TP + FN} \Rightarrow \frac{359}{359 + 223} = 0.61$