

## Contents

<b>1.</b>	<b>Business Problem .....</b>	<b>2</b>
<b>2.</b>	<b>Data Analysis &amp; Data Preparation .....</b>	<b>2</b>
<b>3.</b>	<b>Model Training and Hyperparameters tuning.....</b>	<b>7</b>
<b>3.1</b>	<b>Random Forest Classifier.....</b>	<b>7</b>
<b>3.2</b>	<b>Support Vector Classification (SVC).....</b>	<b>10</b>
<b>4.</b>	<b>Conclusion and Result Analysis.....</b>	<b>12</b>



## 1. Business Problem

- In this case study we are planning to predict customer churn for a bank.
- 
- The bank wants us to build a machine-learning algorithm that helps them predict customer churn and in return, it would help them in retaining the same.
- When it comes to the banking domain there are a lot of data elements involved like the customer's age, his card membership type, his current holding in the bank, and his activeness in the account, well these are the few data set which would help us to build the model which would predict the customer churn.

## 2. Data Analysis & Data Preparation

- As per the given date we currently have 6237 customer entries including various attributes for the same.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	Attrition_C	Customer_Gender	Dependedents	Education	Marital_Status	Income_>\$10K - \$80K Blue	Total_Related_Months	Total_Related_Months	Contacts	Credit_Lin_	Total_Revolving_Balances	Total_Revolving_Balances	Total_Transaction_Count										
2	Existing_Cu	45 M	3	High Schol	Married	\$60K - \$80K Blue	39	5	1	3	12691	777	1144	42									
3	Existing_Cu	49 F	5	Graduate	Single	Less than \$1 Blue	44	6	1	2	8256	864	1291	33									
4	Existing_Cu	51 M	3	Graduate	Married	\$80K - \$12 Blue	36	4	1	0	3418	0	1887	20									
5	Existing_Cu	40 M	3	Uneducated	Married	\$60K - \$80 Blue	21	5	1		4716	0	816	28									
6	Existing_Cu	44 M	2	Graduate	Married	\$40K - \$60 Blue	36	3	1	2	4010	1247	1088	24									
7	Existing_Cu	37 M	3	Uneducated	Single	\$60K - \$80 Blue	36	5	2	0	22352	2517	1350	24									
8	Existing_Cu	48 M	2	Graduate	Single	\$80K - \$12 Blue	36	6	3	3	11656	1677	1441	32									
9	Existing_Cu	57 F	2	Graduate	Married	Less than \$1 Blue	48	5	2	2	2436	680	1570	29									
10	Existing_Cu	48 M	4	Post-Grad	Single	\$80K - \$12 Blue	36	6	2	3	30367	2362	1671	27									
11	Existing_Cu	61 M	1	High Schol	Married	\$40K - \$60 Blue	56	2	2	3	3193	2517	1336	30									
12	Existing_Cu	47 M	1	Doctorate	Divorced	\$60K - \$80 Blue	42	5	2	0	20929	1800	1178	27									
13	Attrited_Cu	62 F	0	Graduate	Married	Less than \$1 Blue	49	2	3	3	1438.3	0	692	16									
14	Existing_Cu	41 M	3	High Schol	Married	\$40K - \$60 Blue	33	4	2	1	4470	680	931	18									
15	Existing_Cu	41 F	3	Graduate	Single	Less than \$1 Blue	28	6	1	2	7768	1669	1051	22									
16	Existing_Cu	47 M	4	High Schol	Married	\$40K - \$60 Blue	42	6	0	0	4785	1362	1045	38									
17	Existing_Cu	53 M	2	Uneducated	Married	\$60K - \$80 Blue	48	2	5	1	2451	1690	1596	26									
18	Existing_Cu	41 M	4	Graduate	Married	\$60K - \$80 Blue	36	4	1	2	8923	2517	1589	24									
19	Existing_Cu	58 M	0	Graduate	Married	\$80K - \$12 Blue	49	6	2	2	12555	1696	1291	24									
20	Existing_Cu	55 F	3	Graduate	Married	Less than \$1 Blue	36	6	2	3	3035	2298	1877	37									
21	Existing_Cu	42 F	4	High Schol	Married	Less than \$1 Gold	36	2	3	3	15433	0	966	22									
22	Existing_Cu	45 M	3	Graduate	Single	\$80K - \$12 Blue	41	2	2	2	32426	578	1109	28									
23	Existing_Cu	49 M	3	High Schol	Married	\$60K - \$80 Blue	37	5	2	1	3006	0	1756	32									
24	Existing_Cu	49 M	4	Uneducated	Single	\$80K - \$12 Blue	30	3	2	3	34516	0	1444	28									
25	Existing_Cu	56 M	2	Doctorate	Married	\$60K - \$80 Blue	45	6	2	0	2283	1430	1741	37									
26	Existing_Cu	59 M	1	Doctorate	Married	\$40K - \$60 Blue	52	3	2	2	2548	2020	1719	27									
27	Existing_Cu	46 M	3	High Schol	Married	\$80K - \$12 Blue	40	4	3	3	19458	1435	1217	27									
28	Attrited_Cu	54 F	2	Graduate	Married	Less than \$1 Blue	40	2	3	1	1438.3	808	705	19									
29	Existing_Cu	66 F	0	High School	Married	Less than \$1 Blue	54	3	4	2	3171	2179	1945	38									

Fig 1.1 : Customer\_Data



	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	\
0	Existing Customer	45	M	3	High School	
1	Existing Customer	49	F	5	Graduate	
2	Existing Customer	51	M	3	Graduate	
3	Existing Customer	40	M	3	Uneducated	
4	Existing Customer	44	M	2	Graduate	
	Marital_Status	Income_Category	Card_Category	Months_on_book		\
0	Married	\$60K - \$80K	Blue	39		
1	Single	Less than \$40K	Blue	44		
2	Married	\$80K - \$120K	Blue	36		
3	Married	\$60K - \$80K	Blue	21		
4	Married	\$40K - \$60K	Blue	36		
	Total_Relationship_Count	Months_Inactive	Contacts_Count	Credit_Limit		\
0	5	1	3	12691.0		
1	6	1	2	8256.0		
2	4	1	0	3418.0		
3	5	1	0	4716.0		
4	3	1	2	4010.0		
	Total_Revolving_Bal	Total_Trans_Amt	Total_Trans_Ct			
0	777	1144	42			
1	864	1291	33			
2	0	1887	20			
3	0	816	28			
4	1247	1088	24			
(6237, 16)						

Fig 1.2 Data View

```
RangeIndex: 6237 entries, 0 to 6236
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Attrition_Flag    6237 non-null   object 
 1   Customer_Age      6237 non-null   int64  
 2   Gender            6237 non-null   object 
 3   Dependent_count   6237 non-null   int64  
 4   Education_Level  6237 non-null   object 
 5   Marital_Status    6237 non-null   object 
 6   Income_Category   6237 non-null   object 
 7   Card_Category     6237 non-null   object 
 8   Months_on_book   6237 non-null   int64  
 9   Total_Relationship_Count 6237 non-null   int64  
 10  Months_Inactive  6237 non-null   int64  
 11  Contacts_Count   6237 non-null   int64  
 12  Credit_Limit     6237 non-null   float64
 13  Total_Revolving_Bal 6237 non-null   int64  
 14  Total_Trans_Amt  6237 non-null   int64  
 15  Total_Trans_Ct   6237 non-null   int64  
dtypes: float64(1), int64(9), object(6)
memory usage: 779.8+ KB
```

Fig 1.3 Data Description



- As we can see in Fig 1.3 there are 16 non-null rows which is a good thing, if it had any null row we would have to cleanse it or drop it basis on its importance for training the model, but here we need not worry about that.
- We can also observe 6 categorial features which we would need to convert into numerical features as some of them might add some importance in model training. We are also considering the target variable as “**Attrition\_Flag**” which is one of the 6 categorial columns.

```

- RangeIndex: 6237 entries, 0 to 6236
Data columns (total 16 columns):
 #   Column           Non-Null Count Dtype  
--- 
 0   Attrition_Flag    6237 non-null   object  
 1   Customer_Age      6237 non-null   int64  
 2   Gender            6237 non-null   object  
 3   Dependent_count   6237 non-null   int64  
 4   Education_Level   6237 non-null   object  
 5   Marital_Status     6237 non-null   object  
 6   Income_Category    6237 non-null   object  
 7   Card_Category      6237 non-null   object  
 8   Months_on_book    6237 non-null   int64  
 9   Total_Relationship_Count 6237 non-null   int64  
 10  Months_Inactive   6237 non-null   int64  
 11  Contacts_Count    6237 non-null   int64  
 12  Credit_Limit       6237 non-null   float64 
 13  Total_Revolving_Bal 6237 non-null   int64  
 14  Total_Trans_Amt    6237 non-null   int64  
 15  Total_Trans_Ct     6237 non-null   int64  
dtypes: float64(1), int64(9), object(6)
memory usage: 779.8+ KB

```

- Out of the above 6 categorial features one categorial feature is “**Marital status**” we would be creating dummies for that because like other categorial features we cannot arrange the values in it in certain incremental order and if we do that it will affect the output for the model predication, as in marital status all the values “**Married**”, “**Single**” and “**Divorced**” customers hold an equal significance when it comes to predicting the churn.
- We would be dropping “**Attrition\_Flag**” as it is our target variable. Here we are also dividing the dataset into labels and features by doing this our model is aware of



dependent and independent variables, and the resulting operation gives us the data frame containing all the independent variables and series of a dependent variables, you can see the output in Fig 1.6 Data frame.

```
# Converting Categorical features into Numerical features
dataset['Gender'] = dataset['Gender'].map({'F': 1, 'M': 0})
dataset['Education_Level'] = dataset['Education_Level'].map({'Doctorate': 4, 'Graduate': 2, 'High School': 1, 'Post-Graduate': 3, 'Uneducated': 0})
dataset['Income_Category'] = dataset['Income_Category'].map({'Less than $40K': 0, '$40K - $60K': 1, '$60K - $80K': 2, '$80K - $120K': 3, '$120K +': 4})
dataset['Card_Category'] = dataset['Card_Category'].map({'Blue': 0, 'Gold': 2, 'Platinum': 3, 'Silver': 1})
dataset['Attrition_Flag'] = dataset['Attrition_Flag'].map({'Existing Customer': 0, 'Attrited Customer': 1})
print(dataset.info())

categorical_features = ['Marital_Status']
final_data = pd.get_dummies(dataset, columns=categorical_features)
print(final_data.info())
print(final_data.head(2))

# Dividing dataset into label and feature sets
X = final_data.drop('Attrition_Flag', axis=1) # Features
Y = final_data['Attrition_Flag'] # Labels
print(type(X))
print(type(Y))
print(X.shape)
print(Y.shape)
```

Fig 1.4 \_Converting to numerical

Data columns (total 18 columns):			
#	Column	Non-Null Count	Dtype
0	Attrition_Flag	6237 non-null	int64
1	Customer_Age	6237 non-null	int64
2	Gender	6237 non-null	int64
3	Dependent_count	6237 non-null	int64
4	Education_Level	6237 non-null	int64
5	Income_Category	6237 non-null	int64
6	Card_Category	6237 non-null	int64
7	Months_on_book	6237 non-null	int64
8	Total_Relationship_Count	6237 non-null	int64
9	Months_Inactive	6237 non-null	int64
10	Contacts_Count	6237 non-null	int64
11	Credit_Limit	6237 non-null	float64
12	Total_Revolving_Bal	6237 non-null	int64
13	Total_Trans_Amt	6237 non-null	int64
14	Total_Trans_Ct	6237 non-null	int64
15	Marital_Status_Divorced	6237 non-null	uint8
16	Marital_Status_Married	6237 non-null	uint8
17	Marital_Status_Single	6237 non-null	uint8
	dtypes:	float64(1), int64(14), uint8(3)	



Fig 1.5 Post\_Conversion

- After conversion we can see that there is an increase in the column count it is due to the presence of the dummy variable “Marital\_Status”

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
(6237, 17)
(6237,)
```

Fig 1.6 Data-Frame

- Normalizing numerical features as we can see in the image below all the variable ranges are quite close except one which is “**Total\_Revolving\_Balance**”, we would require some pre-processing to be done here to bring it into the standard range.

	Customer_Age	Dependent_count	Months_on_book	\
count	6237.000000	6237.000000	6237.000000	
mean	46.380952	2.331409	36.000962	
std	8.047893	1.297106	7.980412	
min	26.000000	0.000000	13.000000	
25%	41.000000	1.000000	31.000000	
50%	46.000000	2.000000	36.000000	
75%	52.000000	3.000000	40.000000	
max	73.000000	5.000000	56.000000	
	Total_Relationship_Count	Months_Inactive	Contacts_Count	\
count	6237.000000	6237.000000	6237.000000	
mean	3.827641	2.342312	2.462402	
std	1.546049	0.999853	1.109643	
min	1.000000	0.000000	0.000000	
25%	3.000000	2.000000	2.000000	
50%	4.000000	2.000000	2.000000	
75%	5.000000	3.000000	3.000000	
max	6.000000	6.000000	6.000000	
	Credit_Limit	Total_Revolving_Bal	Total_Trans_Amt	Total_Trans_Ct
count	6237.000000	6237.000000	6237.000000	6237.000000
mean	8474.209075	1170.792849	4412.084977	64.508578
std	9078.165850	810.392248	3499.842841	23.901158
min	1438.300000	0.000000	569.000000	10.000000
25%	2505.000000	495.000000	2075.000000	44.000000
50%	4307.000000	1285.000000	3825.000000	67.000000
75%	10741.000000	1781.000000	4748.000000	80.000000
max	34516.000000	2517.000000	17995.000000	134.000000

Fig 1.6 Data Range



- This can be achieved using Normalization, wherein it will standardize our data and bring all the values within the range of 1 and -1. This will bring the mean and variance of the data for a particular column to be 0 and 1 respectively.

```
# Normalizing numerical features so that each feature has mean 0 and variance 1
feature_scaler = StandardScaler()
X_scaled = feature_scaler.fit_transform(X)
```

- This sums up our data preparation part.

### 3. Model Training and Hyperparameters tuning.

- After all the data analysis and data preparation we will be starting with the model training phase, we will be using two models here that are **Random Forest Classifier** and a **Support Vector Classifier**.

#### 3.1 Random Forest Classifier

- As per the case our Target variable is “Attrition Flag” and we have mapped the values respectively “Attrited Customer” = 1 and “Existing Customer” = 0.
- We have also observed that there is an imbalance in both the class where in Class 1 value is 981 and in Class 0 is 5256.
- To avoid this we will be using SMOTE, which is a synthesis minority of the class wherein it will oversample the class with fewer values.
- We would be running two iterations on Random Forest Classifier with
  - Estimator1 : [10,20,30,40,50,100]
  - Estimator2: [150,200,250,300,350]
- For the scoring parameter we will be using “Recall” as we want to reduce the false negative because as per the business case bank wants to decrease the attrition rate of the customer and we don't want our model to predict a false churn.
- Here we are using Cross Validation CV = 10 because we want our model to get the train and tested in more folds and in smaller sets which will somehow give us good accuracy as per my analysis.



```

feature_scaler = StandardScaler()
X_scaled = feature_scaler.fit_transform(X)

# Implementing Random Forest Classifier
# Tuning the random forest parameter 'n_estimators' and implementing cross-validation using Grid Search
model = Pipeline([
    ('balancing', SMOTE(random_state=101)),
    ('classification', RandomForestClassifier(criterion='entropy', max_features='auto', random_state=1))
])
grid_param = {'classification__n_estimators': [10, 20, 30, 40, 50, 100]}

gd_sr = GridSearchCV(estimator=model, param_grid=grid_param, scoring='recall', cv=10)

```

Fig 1.7 RFC with Estimators 1

```

feature_scaler = StandardScaler()
X_scaled_ = feature_scaler.fit_transform(X)

#Tuning the random forest parameter 'n_estimators' and implementing cross-validation using Grid Search
model = Pipeline([
    ('balancing', SMOTE(random_state=101)),
    ('classification', RandomForestClassifier(criterion='entropy', max_features='auto', random_state=1))
])
grid_param = {'classification__n_estimators': [150, 200, 250, 300, 350]}

gd_sr = GridSearchCV(estimator=model, param_grid=grid_param, scoring='recall', cv=10)

```

Fig 1.7 RFC with Estimators 2

- After Running the first iteration of RFC with Estimators [10,20,30,50,100], we get the below output:

```

{'classification__n_estimators': 100}
0.8072974644403216

```

- After Running the second iteration of RFC with Estimators [150,200,250,300,350]

```

{'classification__n_estimators': 200}
0.8134096062667491

```

- After completing both the iteration we found out that 100 and 200 are the optimal values and we can also see that there is not much difference in the feature importance list of both, here we will go for the more accurate estimator value i.e E = 200



Total_Trans_Ct	0.243079
Total_Trans_Amt	0.188568
Total_Revolving_Bal	0.152209
Total_Relationship_Count	0.081553
Months_Inactive	0.070021
Contacts_Count	0.054605
Credit_Limit	0.044319
Customer_Age	0.041606
Months_on_book	0.029025
Dependent_count	0.027385
Income_Category	0.022592
Education_Level	0.019013
Gender	0.009298
Marital_Status_Married	0.006339
Marital_Status_Single	0.005063
Card_Category	0.003563
Marital_Status_Divorced	0.001760

Feature list of E\_100

Total_Trans_Ct	0.234732
Total_Trans_Amt	0.197933
Total_Revolving_Bal	0.152163
Total_Relationship_Count	0.082218
Months_Inactive	0.067136
Contacts_Count	0.055742
Credit_Limit	0.044377
Customer_Age	0.041629
Months_on_book	0.028814
Dependent_count	0.025901
Income_Category	0.022345
Education_Level	0.020336
Gender	0.010111
Marital_Status_Married	0.006247
Marital_Status_Single	0.005129
Card_Category	0.003566
Marital_Status_Divorced	0.001622
dtype: float64	

Feature list of E\_200

- Now basis on the feature significance we will train the model by passing the top 10 most significant values.
- After passing the top 10 values in the which are: ['Total\_Trans\_Ct', 'Total\_Trans\_Amt', 'Total\_Revolving\_Bal', 'Total\_Relationship\_Count', 'Months\_Inactive', 'Contacts\_Count', 'Credit\_Limit', 'Customer\_Age', 'Months\_on\_book', 'Dependent\_count'], we get the below result.

```
X_ = final_data[['Total_Trans_Ct','Total_Trans_Amt','Total_Revolving_Bal','Total_Relationship_Count','Months_Inactive','Contacts_Count','Credit_Limit','Customer_Age','Months_on_book','Dependent_count']]

# Normalizing numerical features so that each feature has mean 0 and variance 1
feature_scaler = StandardScaler()
X_scaled = feature_scaler.fit_transform(X_)

## Implementing Random Forest Classifier
## Tuning the random forest parameter 'n_estimators' and implementing cross-validation using Grid Search
model = Pipeline([
    ('balancing', SMOTE(random_state=101)),
    ('classification', RandomForestClassifier(criterion='entropy', max_features='auto', random_state=1))
])
grid_param = {'classification__n_estimators': [10,20,30,40,100]}
```

Fig 1.8 E1\_with Top 10 variable



```
X_ = final_data[['Total_Trans_Ct','Total_Trans_Amt','Total_Revolving_Bal','Total_Relationship_Count','Months_Inactive','Contacts_Count','Credit_Limit','Customer_Age','Month_on_book','Dependent_count']]

X_ = final_data[['Total_Trans_Ct','Total_Trans_Amt','Total_Revolving_Bal']]
feature_scaler = StandardScaler()
X_scaled_ = feature_scaler.fit_transform(X_)

#Tuning the random forest parameter 'n_estimators' and implementing cross-validation using Grid Search
model = Pipeline([
    ('balancing', SMOTE(random_state=101)),
    ('classification', RandomForestClassifier(criterion='entropy', max_features='auto', random_state=1))
])
grid_param = {'classification__n_estimators': [150,200,250,300,350]}

gd_sr = GridSearchCV(estimator=model, param_grid=grid_param, scoring='recall', cv=10)
```

Fig 1.9 E2\_with top 10 variable

```
{'classification__n_estimators': 40}
0.8317769532055246
{'classification__n_estimators': 150}
0.8327973613687899
```

- So from the above observation we can conclude that according to the feature importance list top 10 features with the most significant values are the ones from which we can get the most optimal accuracy.
- We can also consider that there is no overfitting here as we have considered the most significant variables here which give us the best and almost the same accuracy for the different estimators mentioned above
- Hence we can conclude that after all the optimization above we are able to achieve the accuracy of **83.27 percent** in Random Forest Classification.

### 3.2 Support Vector Classification (SVC)

- In SVC we will run the classifier using the below parameters, we will be running the classifier for all four kernels.
- We will also be passing the same top 10 variables as per the significance that we got in the RFC classification.  
`['Total_Trans_Ct','Total_Trans_Amt','Total_Revolving_Bal','Total_Relationship_Count','Months_Inactive','Contacts_Count','Credit_Limit','Customer_Age','Month_on_book','Dependent_count']`
- Our kernels would be [Linear, Poly, RBF, Sigmoid] with the following classification\_C  
`[.001,.01,.1,1,10,100]`



- For the scoring parameter we will be using “Recall” as we want to reduce the false negative because as per the business case bank wants to decrease the attrition rate of the customer and we don’t want our model to predict a false churn.
- Cross Validation CV = 10 because we want our model to get the train and tested in more folds and in smaller sets assuming that it will give us good required accuracy as per my analysis.

```
X_ = final_data[['Total_Trans_Ct','Total_Trans_Amt','Total_Revolving_Bal','Total_Relationship_Count','Months_Inactive','Contacts_Count','Credit_Limit','Customer_Age','Month']]
feature_scaler = StandardScaler()
X_scaled = feature_scaler.fit_transform(X_)
print("-----Linear-----")
#####
# Implementing Support Vector Classifier
# Tuning the kernel parameter and implementing cross-validation using Grid Search
model = Pipeline([
    ('balancing', SMOTE(random_state=101)),
    ('classification', SVC(random_state=1))
])
grid_param = {'classification__kernel': ['linear'], 'classification__C': [.001,.01,.1,.10,.100]}
```

Fig 2.0 Linear

```
X_ = final_data[['Total_Trans_Ct','Total_Trans_Amt','Total_Revolving_Bal','Total_Relationship_Count','Months_Inactive','Contacts_Count','Credit_Limit','Customer_Age','Month']]
feature_scaler = StandardScaler()
X_scaled = feature_scaler.fit_transform(X_)

#####
# Implementing Support Vector Classifier
# Tuning the kernel parameter and implementing cross-validation using Grid Search
model = Pipeline([
    ('balancing', SMOTE(random_state=101)),
    ('classification', SVC(random_state=1))
])
grid_param = {'classification__kernel': ['poly'], 'classification__C': [.001,.01,.1,.10,.100]}

gd_sr = GridSearchCV(estimator=model, param_grid=grid_param, scoring='recall', cv=10)
```

Fig 2.1 Poly

```
X_ = final_data[['Total_Trans_Ct','Total_Trans_Amt','Total_Revolving_Bal','Total_Relationship_Count','Months_Inactive','Contacts_Count','Credit_Limit','Customer_Age','Month']]
feature_scaler = StandardScaler()
X_scaled = feature_scaler.fit_transform(X_)

#####
# Implementing Support Vector Classifier
# Tuning the kernel parameter and implementing cross-validation using Grid Search
model = Pipeline([
    ('balancing', SMOTE(random_state=101)),
    ('classification', SVC(random_state=1))
])
grid_param = {'classification__kernel': ['rbf'], 'classification__C': [.001,.01,.1,.10,.100]}

gd_sr = GridSearchCV(estimator=model, param_grid=grid_param, scoring='recall', cv=10)
```

Fig 2.2 RBF

```
X_ = final_data[['Total_Trans_Ct','Total_Trans_Amt','Total_Revolving_Bal','Total_Relationship_Count','Months_Inactive','Contacts_Count','Credit_Limit','Customer_Age','Month']]
feature_scaler = StandardScaler()
X_scaled = feature_scaler.fit_transform(X_)

#####
# Implementing Support Vector Classifier
# Tuning the kernel parameter and implementing cross-validation using Grid Search
model = Pipeline([
    ('balancing', SMOTE(random_state=101)),
    ('classification', SVC(random_state=1))
])
grid_param = {'classification__kernel': ['sigmoid'], 'classification__C': [.001,.01,.1,.10,.100]}

gd_sr = GridSearchCV(estimator=model, param_grid=grid_param, scoring='recall', cv=10)
```

Fig 2.3 Sigmoid



- After running the above result, we get the following result:

Kernel	Classification	Accuracy
Linear	0.1	0.8338899196042053
Poly	0.001	0.8816326530612244
RBF	1	0.8368893011750155
Sigmoid	0.001	0.7920325706039992

```

-----Linear-----
{'classification__C': 0.1, 'classification__kernel': 'linear'}
0.8338899196042053

-----Poly-----
{'classification__C': 0.001, 'classification__kernel': 'poly'}
0.8816326530612244

-----RBF-----
{'classification__C': 1, 'classification__kernel': 'rbf'}
0.8368893011750155

-----Sigmoid-----
{'classification__C': 0.001, 'classification__kernel': 'sigmoid'}
0.7920325706039992
  
```

- From the above result we can clearly see that kernel “Poly” is putting out higher accuracy which is of **88.163** percent with recall.

#### 4. Conclusion and Result Analysis

- From the above analysis of both models we get the below result:

Model	Classification	Accuracy
RFC	Variables: Top 10,n = 150	<b>83.27 percent</b>
SVC	Variables Top 10, Kernel = Poly, C = 0.001	<b>88.163 percent</b>

- With the above comparison we can clearly state that SVC is giving us more accuracy than that of RFC.
- From the above analysis I can also conclude that both the models are not underfitting because we passed the optimal and significant features in both of them and we are getting an accuracy of above 80 percent.
- The only reason we will go with SVC is that it is giving us more accuracy than that RFC



- So our final model for real-world deployment will be SVC with CV = 10 , the parameter will be "Poly" and "C" : 0.001 with variables:  
['Total\_Trans\_Ct','Total\_Trans\_Amt','Total\_Revolving\_Bal','Total\_Relationship\_Count','Months\_Inactive','Contacts\_Count','Credit\_Limit','Customer\_Age','Months\_on\_book','Dependent\_count']



