Project Classification ML Applied to a TELCO Company

SEPT 2021

Index

- Main objective of the analysis
- Brief description of the data set.
- Brief summary of data exploration
- Summary of training at least three different classifier models,
- A paragraph explaining which of your classifier models you recommend
- Summary Key Findings and Insights,
- Suggestions for next steps in analyzing this data

- ## Main objective of the analysis

The company **Telco+** is concerned about the number of customers leaving their land-line business for cable competitors and want to analyze data from customers in order to predict **Churn,** and will use an initial **dataset (a)**, which has a target dependent variable with 2 possible values (Yes/No or 1/0).

They will apply two different algorithms for generate models of Classifications evaluating them to choose the one with best accuracy:

- **Logistic Regression**
- **SVM** (Support Vector Machines)

They will also use a complementary **dataset (b)** for a purpose of **segmentation** between 4 different types of customers. In this case the model will be generated using the algorithm **KNN.**

- ## Brief description of the data set.

Dataset "a" is a historical customer dataset where *each row represents one customer*.

The data is relatively easy to understand, and you may uncover insights you can use immediately. It is a reduced sample of 200 rows containing key information.

Typically, it is less expensive to keep customers than acquire new ones, so the focus of this analysis is to predict the customers who will stay with the company.

The dataset **ChurnData.csv** includes information about:

- Customers who left within the last month – the column is called **Churn**
- Services that each customer has signed up for – phone, multiple lines, internet, online security, online backup, device protection, tech support, and streaming TV and movies
- Customer account information – how long they had been a customer, contract, payment method, paperless billing, monthly charges, and total charges
- Demographic info about customers – **gender**, **age** range, and if they have partners and dependents

- ## Brief summary of data exploration

Once we download the file, we can read it to see the values and these are observations for first 5 customers… The file has **28 columns** with features.

Load Data From CSV File

```
churn_df = pd.read_csv("ChurnData.csv")
churn_df.head()
```

	tenure	age	address	income	ed	employ	equip	callcard	wireless	longmon	...	pager	internet	callwait	confer	ebill	loglong	logtoll	lninc	custcat	churn
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	4.40	...	1.0	0.0	1.0	1.0	0.0	1.482	3.033	4.913	4.0	1.0
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	9.45	...	0.0	0.0	0.0	0.0	0.0	2.246	3.240	3.497	1.0	1.0
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	6.30	...	0.0	0.0	0.0	1.0	0.0	1.841	3.240	3.401	3.0	0.0
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	6.05	...	1.0	1.0	1.0	1.0	1.0	1.800	3.807	4.331	4.0	0.0
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	7.10	...	0.0	0.0	1.0	1.0	0.0	1.960	3.091	4.382	3.0	0.0

5 rows × 28 columns

As **Data pre-processing and selection**,

- we'll first **select some features** for the modeling and the resulting dataset will have only **10 columns**, with most significant variables independent variables (9) and **Churn** as dependent variable with 2 classes (1, 0).

  Tenure, age, address, Income, ed, employ, equip, callcard, wireless

  will be the 9 selected features.

- And, we'll also change the target data type to be **integer**, as it is a requirement by the **skitlearn** algorithm:

```
churn_df = churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip', 'callcard', 'wireless','churn']]
churn_df['churn'] = churn_df['churn'].astype('int')
churn_df.head()
```

	tenure	age	address	income	ed	employ	equip	callcard	wireless	churn
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	1
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	1
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	0
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	0
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	0

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

*Finally, we will consider only **7 features for X** to simplify it.*

Lets define X, and y for our dataset:

```
X = np.asarray(churn_df[['tenure', 'age', 'address', 'income', 'ed', 'employ', 'equip']])
X[0:5]
```

```
array([[ 11.,  33.,   7., 136.,   5.,   5.,   0.],
       [ 33.,  33.,  12.,  33.,   2.,   0.,   0.],
       [ 23.,  30.,   9.,  30.,   1.,   2.,   0.],
       [ 38.,  35.,   5.,  76.,   2.,  10.,   1.],
       [  7.,  35.,  14.,  80.,   2.,  15.,   0.]])
```

```
y = np.asarray(churn_df['churn'])
y [0:5]
```

```
array([1, 1, 0, 0, 0])
```

Also, we normalize the dataset:

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

```
array([[-1.13518441, -0.62595491, -0.4588971 ,  0.4751423 ,  1.6961288 ,
        -0.58477841, -0.85972695],
       [-0.11604313, -0.62595491,  0.03454064, -0.32886061, -0.6433592 ,
        -1.14437497, -0.85972695],
       [-0.57928917, -0.85594447, -0.261522  , -0.35227817, -1.42318853,
        -0.92053635, -0.85972695],
       [ 0.11557989, -0.47262854, -0.65627219,  0.00679109, -0.6433592 ,
        -0.02518185,  1.16316   ],
       [-1.32048283, -0.47262854,  0.23191574,  0.03801451, -0.6433592 ,
         0.53441472, -0.85972695]])
```

- **Summary of training & a paragraph explaining which of your classifier models you recommend**

To split the dataset into Train and Test we apply the following code

```
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)
```

where we consider a **size of 20% for Test Set**.

So, we'll have a Training set of 160 rows and a Test set of 40 rows.

We'll consider the same split both for

a. **Logistic Regression**
b. **SVM. Support Vector Machine**

The third case will be using a different subset and will use a model based in KNN

c. **KNN applied for Customer Segmentation**

a. And, let's build our model using **LogisticRegression** from Scikit-learn package

- The version of Logistic Regression in Scikit-learn, support regularization. **C** parameter indicates **inverse of regularization strength** which must be a positive float. Smaller values specify stronger regularization. Now let's fit the model with train set:

```
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
```

```
LogisticRegression(C=0.01, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False)
```

Now we can predict using our test set:

```
yhat = LR.predict(X_test)
yhat
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0])
```

predict_proba returns estimates for all classes, ordered by the label of classes.
So, the first column is the probability of class 1, and second column is probability of class 0:

```
yhat_prob = LR.predict_proba(X_test)
yhat_prob
```

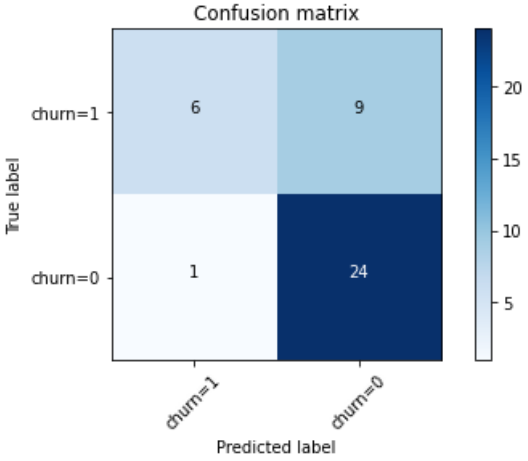
```
array([[0.54, 0.46],
       [0.61, 0.39],
       [0.56, 0.44],
       [0.63, 0.37],
       [0.56, 0.44],
       [0.55, 0.45],
       [0.52, 0.48],
       [0.61, 0.39],
       [0.41, 0.59],
       [0.63, 0.37],
       [0.58, 0.42],
       [0.63, 0.37],
       [0.48, 0.52],
       [0.43, 0.57],
       [0.66, 0.34],
       [0.55, 0.45],
       [0.52, 0.48],
       [0.49, 0.51],
       [0.49, 0.51],
       [0.52, 0.48],
       [0.62, 0.38],
       [0.53, 0.47],
       [0.64, 0.36],
       [0.52, 0.48],
       [0.51, 0.49],
       [0.71, 0.29],
       [0.55, 0.45],
       [0.52, 0.48],
       [0.52, 0.48],
       [0.71, 0.29],
       [0.68, 0.32],
       [0.51, 0.49],
       [0.42, 0.58],
       [0.71, 0.29],
       [0.6 , 0.4 ],
       [0.64, 0.36],
       [0.4 , 0.6 ],
       [0.52, 0.48],
       [0.66, 0.34],
       [0.51, 0.49]])
```

For **Evaluation,** we can use the **Confusion Matrix**

We obtain the following results

```
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=['churn=1','churn=0'],normalize= False,  title='Confusion matrix')
```

```
Confusion matrix, without normalization
[[ 6  9]
 [ 1 24]]
```



In the first row, for 6 customers, the actual churn value was 1 in test set, and classifier also correctly predicted those as 1. However, while the actual label of 9 customers were 1, the classifier predicted those as 0, which is not very good.

We can consider it as error of the model for first row.

Let's look at the second row. It looks like there were 25 customers whom their churn value was 0. The classifier correctly predicted 24 of them as 0, and one of them wrongly as 1.

So, it has done a good job in predicting the customers with churn value 0.

And, we can obtain the following **report**, allowing to calculate **F1,** with an average value of 0,72.

```
print (classification_report(y_test, yhat))
```

	precision	recall	f1-score	support
0	0.73	0.96	0.83	25
1	0.86	0.40	0.55	15
micro avg	0.75	0.75	0.75	40
macro avg	0.79	0.68	0.69	40
weighted avg	0.78	0.75	0.72	40

b. **Applying SVM model, Support Vector Machines,** we'll use the same split for Train & Test, and afterward I'll apply the algorithm for **svm.**

Train/Test dataset

```
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: (160, 7) (160,)
Test set: (40, 7) (40,)
```

```
from sklearn import svm
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, y_train)
```

```
SVC()
```

```
yhat = clf.predict(X_test)
yhat [0:5]
```

```
array([0, 0, 0, 0, 0])
```

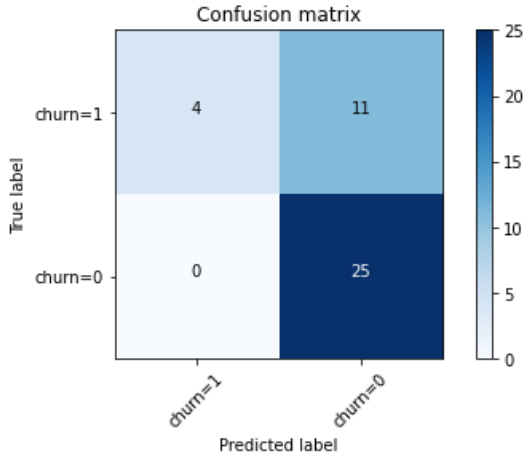
Then, we can calculate the **Confusion Matrix** and obtain a similar **report of Classification**

```
              precision    recall  f1-score   support

           0       0.69      1.00      0.82        25
           1       1.00      0.27      0.42        15

    accuracy                           0.73        40
   macro avg       0.85      0.63      0.62        40
weighted avg       0.81      0.72      0.67        40

Confusion matrix, without normalization
[[ 4 11]
 [ 0 25]]
```



And we see that for Churn=1 this SVM model is working worse than Logistic Regression although for Churn=0 is predicting OK.

In any case average F1 for SVM is 0,67, smaller than previous one, so **I'll recommend Logistic Regression,** but considering a regularization value of C=0.01.

(*Curiously, if C=0.05 ffor Logistic Regression the results in Confusion Matrix and F1 are similar than for SVM.)*

c. The third different model is used for another purpose related to **customer segmentation** and the dataset is a complementary one. In this case the algorithm applied is **KNN, K-Nearest Neighbors**

Telco+ has segmented its customer base by service usage patterns, categorizing the customers into four groups. If demographic data can be used to predict group membership, the company can customize offers for individual prospective customers.

That is, given the **dataset b** containing *1000 rows*, with predefined labels, we need to build a model to be used to predict class of a new or unknown case. This case focuses on using demographic data, such as region, age, and marital, to predict usage patterns.

The target field is called **custcat**, and has **four possible values** that correspond to the four customer groups, as follows:

1- Basic Service 2- E-Service 3- Plus Service 4- Total Service

This is a sample of first rows:

Load Data From CSV File

```
df = pd.read_csv('teleCust1000t.csv')
df.head()
```

	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

Data Visualization and Analysis

Let's see how many of each class is in our data set ¶

```
df['custcat'].value_counts()
```

```
3    281
1    266
4    236
2    217
Name: custcat, dtype: int64
```

281 Plus Service, 266 Basic-service, 236 Total Service, and 217 E-Service customers

Then, we continue defining X, and normalizing Data. *In this dataset there are **11 independent variables**, some of them different from the one used for Churn (as **reside***

X = df[['region', 'tenure','age', 'marital', 'address', 'income', 'ed', 'employ','retire', 'gender', 'reside']]

X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))

And, for **Train Test Split** we proceed in a similar way than in previous cases

```
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)
```

where we are also considering **20% size**, obtaining the following result:

*Train set: (**800**, 11) (800,)*
*Test set: (**200**, 11) (200,)*

Now I import the **K-Nearest Neighbors** algorithm

```
from sklearn.neighbors import KNeighborsClassifier
```

Let's start the algorithm with **k=4**, and we can use the model to predict the test set:

Training

Lets start the algorithm with k=4 for now:

```
k = 4
#Train Model and Predict
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
neigh
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=4, p=2,
           weights='uniform')
```

Predicting

we can use the model to predict the test set:

```
yhat = neigh.predict(X_test)
yhat[0:5]
```

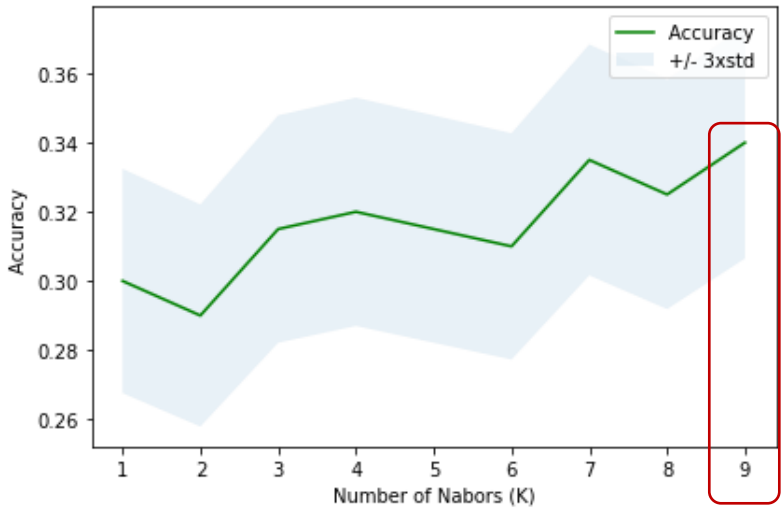
```
array([1, 1, 3, 2, 4])
```

In multilabel classification, **accuracy classification score** is a function that computes subset accuracy. We'll use it for this case obtaining:

```
from sklearn import metrics
print("Train set Accuracy: ", metrics.accuracy_score(y_train, neigh.predict(X_train)))
print("Test set Accuracy: ", metrics.accuracy_score(y_test, yhat))
```

```
Train set Accuracy:  0.5475
Test set Accuracy:  0.32
```

We can see the evolution of accuracy for **different values of ks**, and we obtain the following graph



The best accuracy was with 0.34 with k= 9

- **Summary Key Findings and Insights & Suggestions for next steps**

As summary, we've seen that

a. for the **Churn prediction**, having only 2 classes, the model using **Logistic Regression** with a strong **regularization** of parameter **C** (**C=0.01**) we could obtain better results (F1) than for using **SVM.**

b. **And for segmentation purposes,** with the target of **CustCast**, having 4 possible classes, we have generated the model applying the algorithm **KNN,** and seeing the evolution of accuracy depending of the value of K.

*For this case and dataset, with 1000 observations, we have seen that the best value was obtained with **k=9**.*

As **Suggestions for next steps**, we can recommend to

a) **Using a bigger dataset for predicting Churn,** as the sample in this case has been small. For example, trying to have at least 500 customers.

b) compare with **other values of regularization** for Logistic Regression changing the parameter C, ***for example for a value of C=0.05*** *where the result is equivalent in F1 to the model of SVM*

c) See if **Decision Trees** can be a good model to be applied to Customer Segmentation, and **compare it with the accuracy of KNN**

d) And we could also check if we could obtain better results with a different configuration of **Train Test Split**, for example using **0.3** ratio instead **0.2,** and applying **Cross Validation.**