MACHINE LEARNING
FINAL ASSIGNMENT
COURSE 3

****The jupyter notebook with the code is at the end of the report***

I. INTRODUCTION

i. BACKGROUND

The data I want to analyze for this last assignment is telecommunication related. This data will be used to learn whether a costumer will Churn or not. This data is similar to the one used in the demo notebooks, however it is not the same. The dataframe I will describe in the next section is comprehensive of a little more then 7 thousands entries. The final objective of this little project is to be able to predict if a costumer is going to churn, hence we will focus prediction. The Churn will be our target variable. In order to solve this problem I will use K-Nearest-Neighbors model, Support-Vector-Machine, Decision-Tree and Random Forest.

II. THE DATA

i. DESCRIPTION OF THE DATASET

The dataset I will use is comprehensive of 7043 entries collected from a random selection anonymous users. The dataset is in a csv format and contains 21 features:

1. **CustomerID:** The costumer Identification code
2. **Gender:** Gender of the costumer
3. **SeniorCitizen:** Whether the customer is a senior citizen (0, 1)
4. **Partner:** Whether the customer has a partner (Yes, No)
5. **Dependents:** Whether the customer has dependents (Yes, No)
6. **Tenure:** Number of months the customer has stayed with the company
7. **PhoneService:** Whether the customer has a phone service (Yes, No)
8. **MultipleLines:** Whether the customer has multiple lines (Yes, No, No phone service)
9. **InternetService:** Customer's internet service provider (DSL, Fiber optic, No)
10. **OnlineSecurity:** Whether the customer has online security (Yes, No, No internet)
11. **OnlineBackup:** Whether the customer has online backup (Yes, No, No internet service)
12. **DeviceProtection:** Whether the customer has device protection (Yes, No, No internet)
13. **TechSupport:** Whether the customer has tech support (Yes, No, No internet service)
14. **StreamingTV:** Whether the customer has streaming TV (Yes, No, No internet service)
15. **StreamingMovies:** Whether the customer has streaming movies (Yes, No, No internet)
16. **Contract:** The contract term of the customer (Month-to-month, One year, Two year)
17. **PaperlessBilling:** Whether the customer has paperless billing (Yes, No)
18. **PaymentMethod:** The customer's payment method (Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic))
19. **MonthlyCharges:** The amount charged to the customer monthly
20. **TotalCharges:** The total amount charged to the customer
21. **Churn:** Whether the customer churned (Yes or No)

ii. DATA CLEANING & FEATURES ENGINEERING

The dataset is well presented. Reasonably, at first glance, all the features present can be important. However the ‘*costumerID*’ feature is too specific and completely filled with unique values which means that it is not going to help in the machine learning methods training. I will remove this feature from the dataset. Out of the 21 features, 19 are of type object, 2 of type int and 1 of type float. Running the dataset.info() showed that there are no missing values. However after a more thorough analysis, the feature ‘*TotalCharges*’, which is the total amount charged to the costumer, has 11 entries with value equal to an empty string. Moreover this feature should be of numerical value but instead is a string object. To fix this situation, firstly I found the indexes of the rows where the ‘*TotalCharges*’ value was missing and then I completely removed those 11 rows from the dataset. The consequent information loss should not be relevant considering the size of the data. Subsequently I proceed in transforming this feature from object type to float. There are multiple features where the outcome is ‘yes’ or ‘no’. All these features have been binary encoded as ‘1’ and ‘0’. The features that have more the 2 outcomes, such as ‘*StreamingTV*’, ‘*MultipleLines*’ *etc.* have been one-hot-encoded. The dataset, after all these operation is comprehensive of 41 features and 7031 rows. Our of all these features, only 2, ‘*MonthlyCharges*’ and ‘*TotalCharges*’ presents values different then 0 and 1. Therefore, I checked for skewness this features and found out that the ‘*TotalCharges*’ is skewed right. On this column I applied the log1p transformation to normalize the data.

iii. DATA EXPLORATION

In order to understand the relations between my target variable ‘*Churn*’ and all the other 40 features I proceed with calculating the Pearson correlation. We can see that the most negatively correlated feature is ‘*tenure*’. Tenure indicates the number of months the customer has stayed with the company. This results means that the more someone has been the costumer for the company the less likely they will churn, which is reasonable. We can

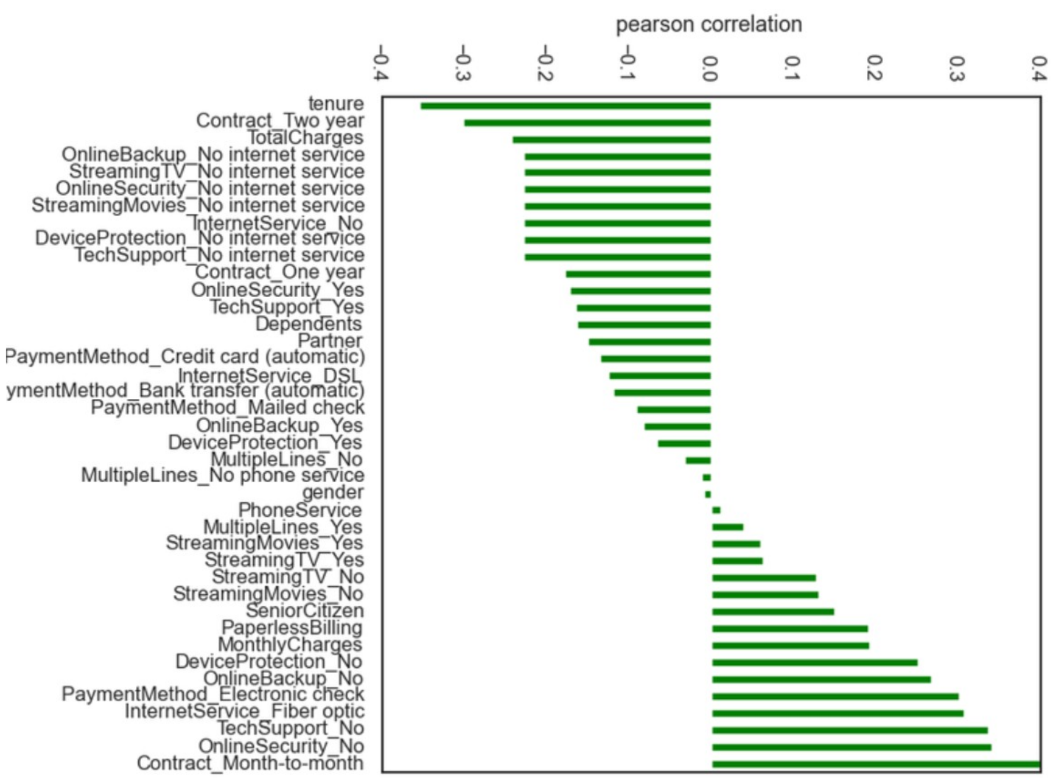


Figure 1. Pearson Correlation between the target variable and alll the other features

see that the 'TotalCharges' has a fairly negative correlation too. This is a logic finding since the tenure has the most negative correlation which is linked to the fact that a long time costumer has probably been charged more money then a casual costumer. Also it looks like that those that have a two year contract are less likely to churn and, on the other hand those that have a contract month-to-month are very likely to churn. This is also a reasonably finding since those that have a month-to-month contract are probably just trying the service and are not committed to the company. In fact this is the most positively correlated feature indicating that it has a major role in determine whether a costumer will churn or not. We can also see that those costumer that have a *'automatic'* payment method in place are those that are less likely to churn. We can also see that the '*gender*' as a negligible effect in determining if a costumer will churn or not. This latter finding is also reasonable. This exploratory analysis has brought to light the relations between our target variable and all the features. After a quick analysis we can say that these correlation looks reasonable.

III. MACHINE LEARNING METHODS

1. Train-Test Split

First of all I will create the train and test split which I will use throughout the whole machine learning methods I will use. Before proceeding with the train-test split I noticed that the data is not in a 50-50 proportion but it is rather a 30-70. Therefore I would like to keep the same proportion buy using the '*StratifiedShuffleSplit*'. Also the features have been scale using the Standard Scaler.

2. K-Nearest-Neighbors

The first machine learning method I will used to classify whether a costumer is going to churn or not is the k-nearest-neighbors. In order to find the right **k** I set its maximum to 200 and the I used a for loop to train and predict the outcome for every **k**. Subsequently I calculated the f1 scores and error rates for every single choice of **k** and I plotted the results.

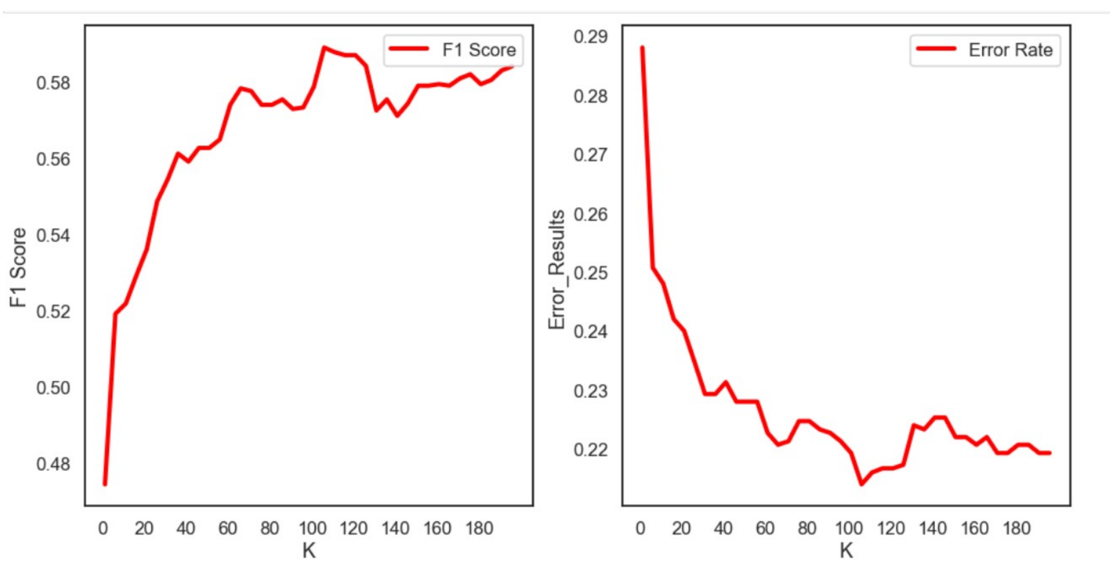


Figure 2. Pots of the f1-score and Error-Rates for the k-nearest-neighbors method

The maximum f1 score and the minimum error rate is when **k** is equal to 106. I will used k=106 as the hyper-parameter to build the best k-nearest-neighbors method for this particular random-state. This confusion matrix shows that the K-Nearest-Neighbors for

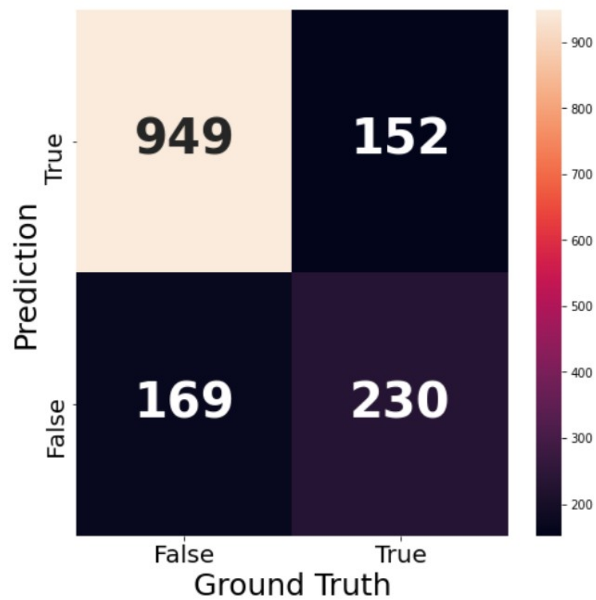


Figure 3. Confusion matrix of the best KNN method with k=106

k=106 shows that there is a pretty high error to the method. There are 169 entries that should have been classified as ‘did *not Churn*’ but where classified as ‘*did Churn*’ and 152 entries that should have been ‘*did Churn*’ but have been labeled ‘*did not Churn*’. Now lets see the overall precision/recall and f1-score of this method.

	precision	recall	f1-score	support
0	0.85	0.86	0.86	1101
1	0.60	0.58	0.59	399
accuracy			0.79	1500
macro avg	0.73	0.72	0.72	1500
weighted avg	0.78	0.79	0.78	1500

	train	test
accuracy	0.998915	0.786000
precision	0.999318	0.602094
recall	0.996599	0.576441
f1	0.997956	0.588988

Figure 4. Classification report of the k-nearest-neighbors method with k=106. Using StratifiedSuffleSplit. On the right the comparison between the model on applied on the train and test

We see that the classification report of figure 4 is reflecting what we saw in the confusion matrix. The precision on the 0 (not churn) observation is of 85% however the precision on the 1 (did churn) observation is much lower around 60%. Same behavior for the recall and the f1 score. This method did a fairly good job in identifying correctly the costumer that did not churn but it did a poor job in predicting the costumer that churned. The overall accuracy of this method is of 79% and with an overall f1 score of 0.59. This difficulty in classifying the situation in which a costumer did churn can be due to the fact the there are less then half the points with that outcome compared to the outcome of not chun. On the right of figure 4 we can see that this model seems to overfit our data heavely. I also tried to run a K-Nearest-Neighbors scaling the features with MinMaxScaler instead of StandarScaler but the result where very similar. Also I tried to use the train and test splits generated with the standard TrainTestSplit instead of the stratified one. The results where a little better but comparable.

	precision	recall	f1-score	support
0	0.86	0.87	0.86	2079
1	0.61	0.59	0.60	734
accuracy			0.79	2813
macro avg	0.73	0.73	0.73	2813
weighted avg	0.79	0.79	0.79	2813

Figure 5. Classification report of the k-nearest-neighbors method with k=86. Using TrainTestSplit

3. Support Vector Machine Classifier

Next I will try a Support Vector Machine classification methods to compare it to the K-Nearest-Neighbors method. I will not use more complex Support Vector Machine methods with kernels, gammas or approximations since my dataset have a little more then 7k rows and just 41 features so it be considered pretty small. A LinearSVC should suffice. The LinearSVC method can be tuned by changing the hyper-parameter C which is the regularization term. Higher C mean a lower regularization while lower values means more regularization. I manually tested the C values so to come up with the range that would have the highest f1-score in it. Therefore I created a for loop where I run the LinearSVC 60 times with 60 different C ranging from $5e^{-4}$ and $5r^{-6}$.

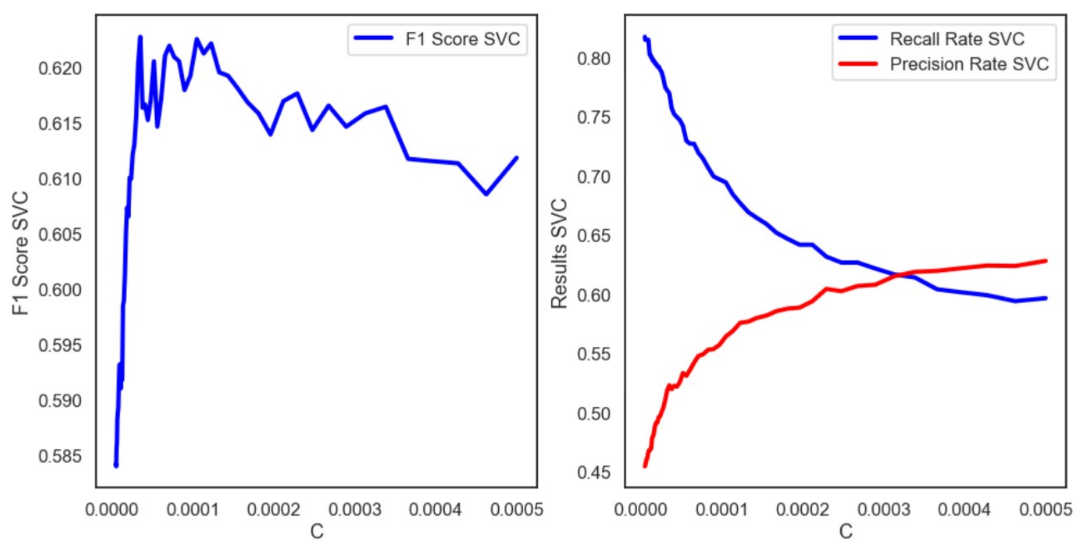


Figure 6. Left - Plots of the f1-score for different C values in a LinearSVC model. Right – Plot of both the Recall and Precision for this model

The highest f1-score is 0.6227 with C equal to: $3.52e^{-5}$ however since this is a binary (two-class) classification where the target variable can be either 0 or 1 there is another way to evaluate the model that is considered to be more fitted for this kinds of situations, the Matthews Correlation Coefficient. This measure takes into account both true and false positives and negatives. Moreover it is useful because it can be used even if the classes differ greatly in size.

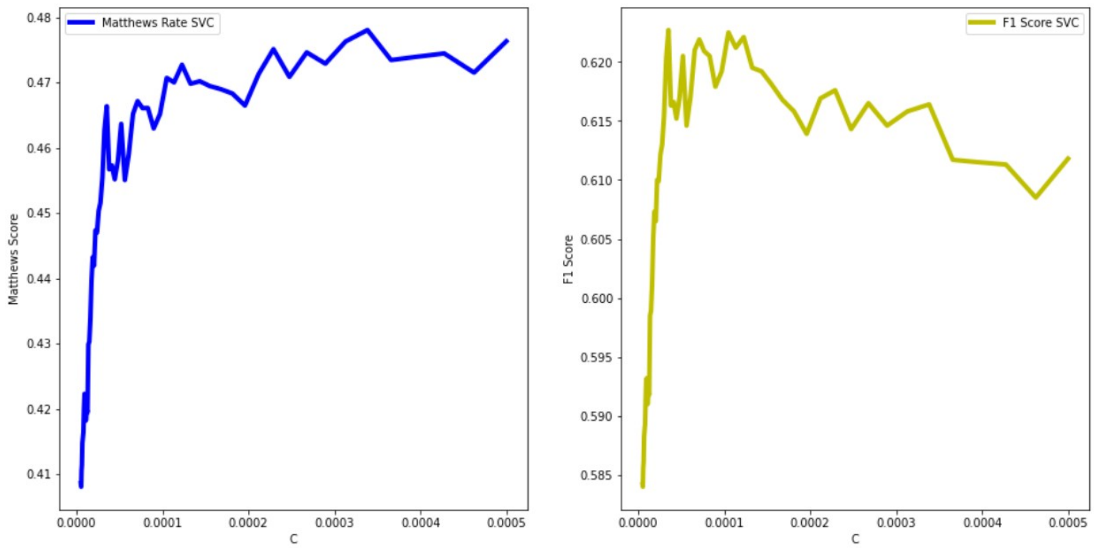


Figure 7. Left - Plots of the Matthews score for different C values in a LinearSVC model. Right – Plots of the f1-score for different C values in a LinearSVC model.

The f1-score and Matthews are on different scales. f1-score goes from 0 to 1 while the Matthews correlation coefficient goes from -1 to 1. However, if we compare these 2 plot we can see a similar behavior. The best Matthews value is for C = ~0.00034. I tried to use both the best C given by the f1-score and the one from the Matthews correlation coefficient. The latter resulted in a far better model with a ~10% increase in the precision of the *'did churn'* observations.

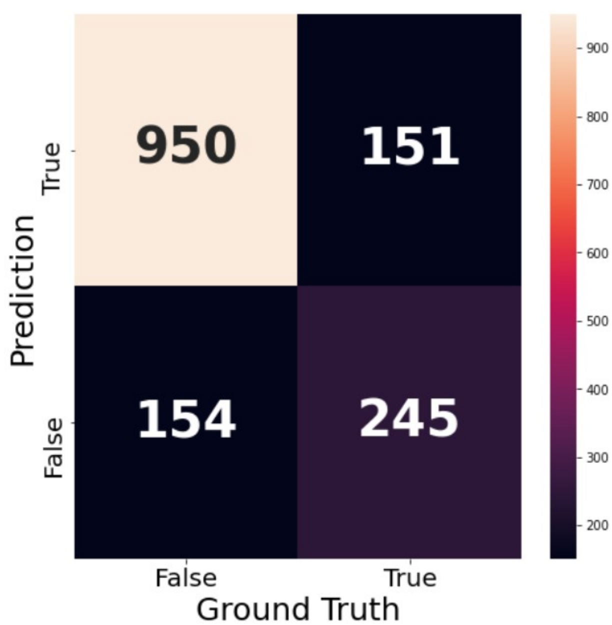


Figure 8. Confusion matrix of the best SVC method

	precision	recall	f1-score	support
0	0.86	0.86	0.86	1101
1	0.62	0.61	0.62	399
accuracy			0.80	1500
macro avg	0.74	0.74	0.74	1500
weighted avg	0.80	0.80	0.80	1500

	train	test
accuracy	0.806580	0.796667
precision	0.646628	0.618687
recall	0.600000	0.614035
f1	0.622442	0.616352

Figure 9. Classification report of the best SVC method. On the right the comparison between the method applied to the train and test set

Moreover, compared with the K-Nearest-Neighbors method it shows a slight improvement in the prediction of the *'did churn'* event. The precision/recall and f1-score of the *'not churn'* event remains practically the same. We can also see that compared to the K-Nearest-Neighbors model this method does overfit the data as much. The accuracy on the train set is ~20% lower but the accuracy on the test set is slightly higher.

4. Decision Tree Classifier

Before getting into the random forest classifier I want to take a look at a simple decision tree method. What I will do is to run this method on the train and test set previously created and analyze the results.

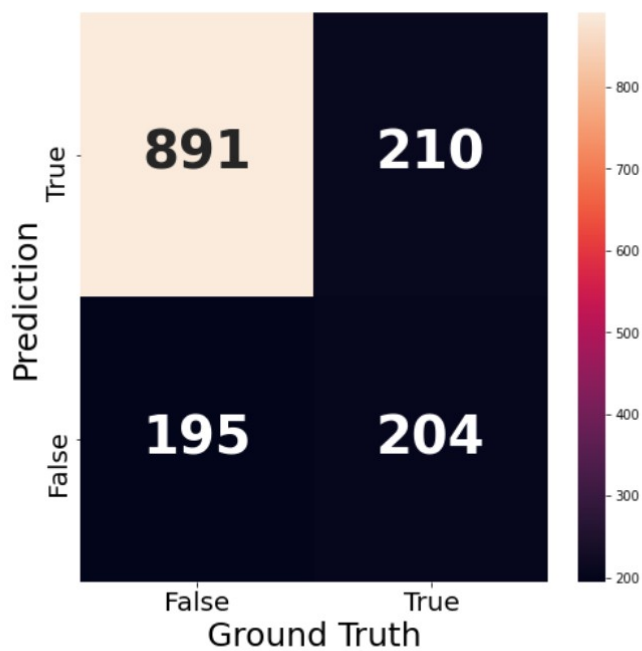


Figure 10. Confusion matrix of the Decision Tree Classifier

At first glance we can see that this method is not behaving nicely. The error on the True Negative looks pretty big, almost random. Lets take a look at the classification report for this method.

	precision	recall	f1-score	support
0	0.82	0.81	0.81	1101
1	0.49	0.51	0.50	399
accuracy			0.73	1500
macro avg	0.66	0.66	0.66	1500
weighted avg	0.73	0.73	0.73	1500

	train	test
accuracy	0.998735	0.730000
precision	0.999317	0.492754
recall	0.995918	0.511278
f1	0.997615	0.501845

Figure 11. Classification report for the Decision Tree Classifier. On the right the comparison between the method applied to the train and test set

As mentioned, the precision/recall and f1-score for the '1' (did churn) event are worse then both the previous models with a drop along all the values of ~8-10%. Also, the values for the other event '0' (did not churn) are worse but closer to the previous models values. From figure 11 right side we can see that, similarly to the k-nearest-neighbors model also this decision tree is vastly overfitting the data.

5. Best Decision Tree

Since the previous decision tree classifier did not show good results, I decided to used the GridSearchCV to find the optimal hyper-parameters. In this way I hope to find a better model for my dataset. What I did was probing the max depth of the tree and the max feature consider. I was able to come up with a better model.

	precision	recall	f1-score	support
0	0.84	0.88	0.86	1101
1	0.61	0.53	0.57	399
accuracy			0.78	1500
macro avg	0.72	0.70	0.71	1500
weighted avg	0.78	0.78	0.78	1500

	train	test
accuracy	0.799349	0.789333
precision	0.620321	0.602469
recall	0.631293	0.611529
f1	0.625759	0.606965

Figure 12. Classification report for the best Decision Tree Classifier. On the right the comparison between the method applied to the train and test set

We can see that the best improvement in the precision while the recall remains pretty low. Moreover we see that this model does not overfit the data as the model before.

6. Random Forest Classifier & Extra Trees

Lastly I will apply the random forest model. The hyper-parameter that I will change is the number of trees. In order to do so I created a list of 32 elements with the geomspace function. Subsequently I used a for loop to train the models with all the possible number of trees utilizing the warm_start signature. Moreover, I used the same procedure to train and predict the target variable with the extra tree classifier method for the sole purpose to compare the two. For every given number of trees I severd the out-of-bag error for both the aforementioned methods.

n_trees	RandomForest OOB	ExtraTrees OOB
25.0	0.221258	0.227223
50.0	0.214208	0.221981
75.0	0.214931	0.219270
100.0	0.211497	0.217462
125.0	0.208424	0.213666

Figure 13. Comparison between the out-of-bag error for the random forest and the extra trees models.

We can see that the RandomForest method is better suited for our purpose since the out-of-bag error is consistently lower compared to the other method. The extra randomness introduced by the Extra Trees method is not useful in this situation as the slightly higher error shows.

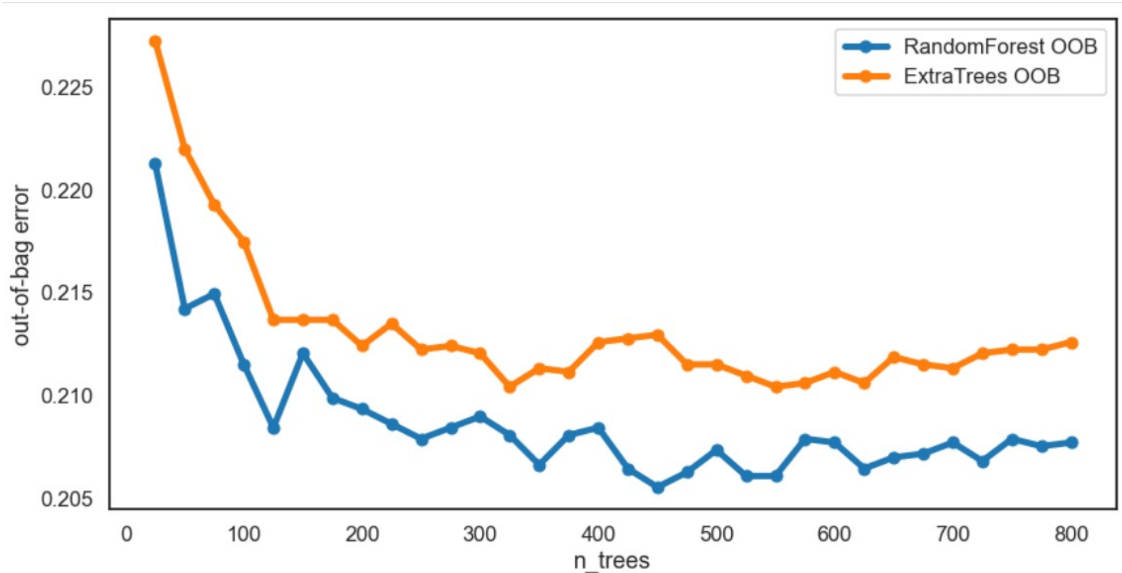


Figure 14. Comparison between the out-of-bag error for the random forest and the extra trees models.

The best number of trees is 450 with. I used this information to train a specific Random Forest model. Lets have a look at the classification report for such model.

	precision	recall	f1-score	support
0	0.82	0.89	0.85	1101
1	0.60	0.48	0.54	399
accuracy			0.78	1500
macro avg	0.71	0.68	0.69	1500
weighted avg	0.77	0.78	0.77	1500

Figure 13. Classification report for the best Random Forest Classifier.

IV. SUMMARY, KEY FINDINGS & RECOMMENDED METHODS

Lets have a look at the comparison between all the method we have used in this assignment. Firstly we used the K-Nearest-Neighbors which ended up greatly overfitting the data with an accuracy on the training set of ~99% but only ~79% on the test set. This method had a fairly high accuracy on the prediction of one of the target classes (0, costumer did not churn) but lacked accuracy for the other one. The reason behind this behavior can be the fact that the class 1 (costumer did churn) is less frequent. Subsequently we tried the Support Vector Machine Classifier which gave slightly better result in the prediction of the class 1 (costumer did churn). However, compared with the previous method, this model does not seem to greatly overfit the data having comparable accuracy when predicting the outcome on both train and test set. Next we tried out a simple decision tree classifier without any hyper-parameter variation. This method result in a vast overfit of the data and a decrease in performance across the board. The precision/recall and f1-score of both class 1 and 0 for the target variable is significantly worse compared to the previous two methods. Therefore since this version of the decision tree was not good enough I decided to try out a grid search for this method by probing both max depth of the tree and feature importance. As a result I was able to come up with a model that does not overfit the data and that have better scores then the previous decision tree. Interesting to see that the precision on the prediction of the class (0, costumer did not churn) is the highest we have obtained so far, however the precision on the other outcome is still very low. Last but not least I tried to use a random forest model and and extra trees classifier. The overall performance is better for the random forest as the out-of-bag error indicated. The extra randomness introduced by the extra tree model is not helping predicting the target variable more efficiently. The best number of trees for the random forest is 450 and the classification report of this model shows that there is no relevant improvement compared to the other models we have tested. In conclusion, the best overall method we used to classify and predict the target variable *'churn'* is the Support Vector Machine Classifier with have the best precision/recall/accuracy and f1-score then all the other methods and it does not overfit the data as some of the other models.

V. SUGGESTIONS

Finding the best classifier method for this project felt a little tangled. I was hoping that at least one of these models would give me very good results in term of precision/recall, accuracy and f1-score. However this was not the case. All the methods are similar in performance. One of the main reason for this behavior, I think, is due to the fact that one of those 2 classes of the target variable is present with a frequency less then half compared to the other class. It would be interesting to see if, after implementing a upsampling or downsampling the result would be better.

Final_course3

February 2, 2021

1 Machine Learning Final Exercise - 3rd Course

1.1 Working with Telco Churn Data

In this project I will analize a dataset comprehensive of the churn data for Telco. The final objective is to use machine learning to predict if a costumer is going to churn or not. Naturally i will try to use the classification methods used in this course like; logistic classification, k-nearest-neighbors, SVM and probaly also some ensamble methods.

Import the libraries

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from collections import defaultdict
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.model_selection import StratifiedShuffleSplit, train_test_split,␣
 ↪KFold, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import (confusion_matrix, accuracy_score,␣
 ↪matthews_corrcoef,
                             f1_score, recall_score, precision_score,␣
 ↪classification_report)
from sklearn.svm import LinearSVC
from sklearn.pipeline import Pipeline

%pylab inline
%matplotlib inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

Load the data

[2]:
```
filepath = 'WA_Fn-UseC_-Telco-Customer-Churn.csv'
data_main = pd.read_csv(filepath)
data_main.head()
```

[2]:
```
    customerID  gender  SeniorCitizen Partner Dependents  tenure PhoneService  \
0  7590-VHVEG  Female              0     Yes         No       1           No
1  5575-GNVDE    Male              0      No         No      34          Yes
2  3668-QPYBK    Male              0      No         No       2          Yes
3  7795-CFOCW    Male              0      No         No      45           No
4  9237-HQITU  Female              0      No         No       2          Yes

      MultipleLines InternetService OnlineSecurity  … DeviceProtection  \
0  No phone service             DSL             No  …               No
1                No             DSL            Yes  …              Yes
2                No             DSL            Yes  …               No
3  No phone service             DSL            Yes  …              Yes
4                No     Fiber optic             No  …               No

  TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
0          No          No              No  Month-to-month              Yes
1          No          No              No        One year               No
2          No          No              No  Month-to-month              Yes
3         Yes          No              No        One year               No
4          No          No              No  Month-to-month              Yes

               PaymentMethod MonthlyCharges  TotalCharges Churn
0           Electronic check          29.85         29.85    No
1               Mailed check          56.95        1889.5    No
2               Mailed check          53.85        108.15   Yes
3  Bank transfer (automatic)          42.30       1840.75    No
4           Electronic check          70.70        151.65   Yes

[5 rows x 21 columns]
```

[3]:
```python
data_main.shape
```

[3]:
```
(7043, 21)
```

[4]:
```python
data_main.dtypes
```

[4]:
```
customerID          object
gender              object
SeniorCitizen        int64
Partner             object
Dependents          object
tenure               int64
PhoneService        object
MultipleLines       object
InternetService     object
OnlineSecurity      object
OnlineBackup        object
```

```
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

Lets look at missing data

```
[5]: data_main.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

```
[6]: data_main.Churn.value_counts(normalize=True)
```

[6]:
```
No     0.73463
Yes    0.26537
Name: Churn, dtype: float64
```

[7]:
```python
data_main.Churn.value_counts()
```

[7]:
```
No     5174
Yes    1869
Name: Churn, dtype: int64
```

Almost all featrures are of type object, this means we well need to encode them so to have floats to feed the machine learning methods. Also the target variable is ~73% not churn and ~26% churn which means that in the train/test split we should be keeping the proportion by using the stratified train/test split method.

It shows there are no null values, however the string object TotalCharges has sometimes a empty string

1.2 Data Cleaning & Features Engineering

Fistly lets see if there are some columns that can be dropped. All features besides the costumerID seems to be useful. So lets remove the costumerID feature.

[8]:
```python
data_main=data_main.drop(['customerID'], axis=1)
```

[9]:
```python
data=data_main.copy()
```

[10]:
```python
data.TotalCharges.loc[488]
```

[10]:
```
' '
```

Lets drop all rows that have the TotalCharges feature empty. They are 11 so on this dataset of 7043 shoudlnt be a problem to drop em

[11]:
```python
for val, idx in zip(data.TotalCharges, data.index):
    if val==' ':
        print('empty', 'index=', idx)
        data=data.drop(idx)
```

```
empty index= 488
empty index= 753
empty index= 936
empty index= 1082
empty index= 1340
empty index= 3331
empty index= 3826
empty index= 4380
empty index= 5218
```

```
empty index= 6670
empty index= 6754
```

[12]:
```
data=data.reset_index(drop=True)
```

Lets transfrom the to numeric type

[13]:
```
data.TotalCharges=pd.to_numeric(data['TotalCharges'])
```

Next we will need to encode all the features that are of type object.

[14]:
```
print(data['MultipleLines'].unique())
print(data['InternetService'].unique())
print(data['PaymentMethod'].unique())
```

```
['No phone service' 'No' 'Yes']
['DSL' 'Fiber optic' 'No']
['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
```

All object features besides those with more then two values, such as; MultipleLines, InternetService and PaymentMethods, etc. will be binary encoded while these ones will be one-hot-encoded. Lets select all the columns to be binary encoded and that are of type object.

[15]:
```
bin_cols=data.loc[:,['gender', 'Partner', 'Dependents', 'PhoneService',␣
↪'PaperlessBilling', 'Churn']]
```

[16]:
```
#cols
```

[17]:
```
dle = defaultdict(LabelEncoder)
```

[18]:
```
bin_cols = bin_cols.apply(lambda x: dle[x.name].fit_transform(x))
```

[19]:
```
bin_cols.head()
```

[19]:

	gender	Partner	Dependents	PhoneService	PaperlessBilling	Churn
0	0	1	0	0	1	0
1	1	0	0	1	0	0
2	1	0	0	1	1	1
3	1	0	0	0	0	0
4	0	0	0	1	1	1

One hot encode the remaining object features

[20]:
```
bin_cols.columns
```

[20]:
```
Index(['gender', 'Partner', 'Dependents', 'PhoneService', 'PaperlessBilling',
       'Churn'],
      dtype='object')
```

Lets remove the just encoded columns

[21]:
```
ohe_cols=data.drop(bin_cols.columns, axis=1)
```

Lets remove those two columns that are of type integer and float

[22]:
```
ohe_cols=ohe_cols.drop(['tenure', 'SeniorCitizen', 'MonthlyCharges',␣
↪'TotalCharges'], axis=1)
```

[23]:
```
ohe_cols.shape
```

[23]:
```
(7032, 10)
```

[24]:
```
ohe_cols = pd.get_dummies(ohe_cols,drop_first=False)
```

[25]:
```
#ohe_cols
```

[26]:
```
left_out_cols=data.loc[:,['tenure', 'SeniorCitizen', 'MonthlyCharges',␣
↪'TotalCharges']]
```

[27]:
```
left_out_cols.dtypes
```

[27]:
```
tenure               int64
SeniorCitizen        int64
MonthlyCharges     float64
TotalCharges       float64
dtype: object
```

[28]:
```
data_enc=pd.concat([bin_cols, ohe_cols, left_out_cols], axis=1)
```

[29]:
```
data_enc.shape
```

[29]:
```
(7032, 41)
```

[30]:
```
data_enc.head()
```

[30]:
```
   gender  Partner  Dependents  PhoneService  PaperlessBilling  Churn  \
0       0        1           0             0                 1      0
1       1        0           0             1                 0      0
2       1        0           0             1                 1      1
3       1        0           0             0                 0      0
4       0        0           0             1                 1      1

   MultipleLines_No  MultipleLines_No phone service  MultipleLines_Yes  \
0                 0                               1                  0
1                 1                               0                  0
2                 1                               0                  0
3                 0                               1                  0
```

```
4                 1                               0                   0

   InternetService_DSL  …  Contract_One year  Contract_Two year  \
0                    1  …                  0                  0
1                    1  …                  1                  0
2                    1  …                  0                  0
3                    1  …                  1                  0
4                    0  …                  0                  0

   PaymentMethod_Bank transfer (automatic)  \
0                                        0
1                                        0
2                                        0
3                                        1
4                                        0

   PaymentMethod_Credit card (automatic)  PaymentMethod_Electronic check  \
0                                      0                               1
1                                      0                               0
2                                      0                               0
3                                      0                               0
4                                      0                               1

   PaymentMethod_Mailed check  tenure  SeniorCitizen  MonthlyCharges  \
0                           0       1              0           29.85
1                           1      34              0           56.95
2                           1       2              0           53.85
3                           0      45              0           42.30
4                           0       2              0           70.70

   TotalCharges
0         29.85
1       1889.50
2        108.15
3       1840.75
4        151.65

[5 rows x 41 columns]
```

[31]:
```
data_enc.describe().T
```

[31]:
```
                                          count      mean       std  \
gender                                   7032.0  0.504693  0.500014
Partner                                  7032.0  0.482509  0.499729
Dependents                               7032.0  0.298493  0.457629
PhoneService                             7032.0  0.903299  0.295571
PaperlessBilling                         7032.0  0.592719  0.491363
```

Churn	7032.0	0.265785	0.441782
MultipleLines_No	7032.0	0.481371	0.499688
MultipleLines_No phone service	7032.0	0.096701	0.295571
MultipleLines_Yes	7032.0	0.421928	0.493902
InternetService_DSL	7032.0	0.343572	0.474934
InternetService_Fiber optic	7032.0	0.440273	0.496455
InternetService_No	7032.0	0.216155	0.411650
OnlineSecurity_No	7032.0	0.497298	0.500028
OnlineSecurity_No internet service	7032.0	0.216155	0.411650
OnlineSecurity_Yes	7032.0	0.286547	0.452180
OnlineBackup_No	7032.0	0.438993	0.496300
OnlineBackup_No internet service	7032.0	0.216155	0.411650
OnlineBackup_Yes	7032.0	0.344852	0.475354
DeviceProtection_No	7032.0	0.439989	0.496421
DeviceProtection_No internet service	7032.0	0.216155	0.411650
DeviceProtection_Yes	7032.0	0.343857	0.475028
TechSupport_No	7032.0	0.493743	0.499996
TechSupport_No internet service	7032.0	0.216155	0.411650
TechSupport_Yes	7032.0	0.290102	0.453842
StreamingTV_No	7032.0	0.399460	0.489822
StreamingTV_No internet service	7032.0	0.216155	0.411650
StreamingTV_Yes	7032.0	0.384386	0.486484
StreamingMovies_No	7032.0	0.395478	0.488988
StreamingMovies_No internet service	7032.0	0.216155	0.411650
StreamingMovies_Yes	7032.0	0.388367	0.487414
Contract_Month-to-month	7032.0	0.551052	0.497422
Contract_One year	7032.0	0.209329	0.406858
Contract_Two year	7032.0	0.239619	0.426881
PaymentMethod_Bank transfer (automatic)	7032.0	0.219283	0.413790
PaymentMethod_Credit card (automatic)	7032.0	0.216297	0.411748
PaymentMethod_Electronic check	7032.0	0.336320	0.472483
PaymentMethod_Mailed check	7032.0	0.228100	0.419637
tenure	7032.0	32.421786	24.545260
SeniorCitizen	7032.0	0.162400	0.368844
MonthlyCharges	7032.0	64.798208	30.085974
TotalCharges	7032.0	2283.300441	2266.771362

	min	25%	50%	75% \
gender	0.00	0.0000	1.000	1.0000
Partner	0.00	0.0000	0.000	1.0000
Dependents	0.00	0.0000	0.000	1.0000
PhoneService	0.00	1.0000	1.000	1.0000
PaperlessBilling	0.00	0.0000	1.000	1.0000
Churn	0.00	0.0000	0.000	1.0000
MultipleLines_No	0.00	0.0000	0.000	1.0000
MultipleLines_No phone service	0.00	0.0000	0.000	0.0000
MultipleLines_Yes	0.00	0.0000	0.000	1.0000

InternetService_DSL	0.00	0.0000	0.000	1.0000
InternetService_Fiber optic	0.00	0.0000	0.000	1.0000
InternetService_No	0.00	0.0000	0.000	0.0000
OnlineSecurity_No	0.00	0.0000	0.000	1.0000
OnlineSecurity_No internet service	0.00	0.0000	0.000	0.0000
OnlineSecurity_Yes	0.00	0.0000	0.000	1.0000
OnlineBackup_No	0.00	0.0000	0.000	1.0000
OnlineBackup_No internet service	0.00	0.0000	0.000	0.0000
OnlineBackup_Yes	0.00	0.0000	0.000	1.0000
DeviceProtection_No	0.00	0.0000	0.000	1.0000
DeviceProtection_No internet service	0.00	0.0000	0.000	0.0000
DeviceProtection_Yes	0.00	0.0000	0.000	1.0000
TechSupport_No	0.00	0.0000	0.000	1.0000
TechSupport_No internet service	0.00	0.0000	0.000	0.0000
TechSupport_Yes	0.00	0.0000	0.000	1.0000
StreamingTV_No	0.00	0.0000	0.000	1.0000
StreamingTV_No internet service	0.00	0.0000	0.000	0.0000
StreamingTV_Yes	0.00	0.0000	0.000	1.0000
StreamingMovies_No	0.00	0.0000	0.000	1.0000
StreamingMovies_No internet service	0.00	0.0000	0.000	0.0000
StreamingMovies_Yes	0.00	0.0000	0.000	1.0000
Contract_Month-to-month	0.00	0.0000	1.000	1.0000
Contract_One year	0.00	0.0000	0.000	0.0000
Contract_Two year	0.00	0.0000	0.000	0.0000
PaymentMethod_Bank transfer (automatic)	0.00	0.0000	0.000	0.0000
PaymentMethod_Credit card (automatic)	0.00	0.0000	0.000	0.0000
PaymentMethod_Electronic check	0.00	0.0000	0.000	1.0000
PaymentMethod_Mailed check	0.00	0.0000	0.000	0.0000
tenure	1.00	9.0000	29.000	55.0000
SeniorCitizen	0.00	0.0000	0.000	0.0000
MonthlyCharges	18.25	35.5875	70.350	89.8625
TotalCharges	18.80	401.4500	1397.475	3794.7375

	max
gender	1.00
Partner	1.00
Dependents	1.00
PhoneService	1.00
PaperlessBilling	1.00
Churn	1.00
MultipleLines_No	1.00
MultipleLines_No phone service	1.00
MultipleLines_Yes	1.00
InternetService_DSL	1.00
InternetService_Fiber optic	1.00
InternetService_No	1.00
OnlineSecurity_No	1.00

```
OnlineSecurity_No internet service          1.00
OnlineSecurity_Yes                          1.00
OnlineBackup_No                             1.00
OnlineBackup_No internet service            1.00
OnlineBackup_Yes                            1.00
DeviceProtection_No                         1.00
DeviceProtection_No internet service        1.00
DeviceProtection_Yes                        1.00
TechSupport_No                              1.00
TechSupport_No internet service             1.00
TechSupport_Yes                             1.00
StreamingTV_No                              1.00
StreamingTV_No internet service             1.00
StreamingTV_Yes                             1.00
StreamingMovies_No                          1.00
StreamingMovies_No internet service         1.00
StreamingMovies_Yes                         1.00
Contract_Month-to-month                     1.00
Contract_One year                           1.00
Contract_Two year                           1.00
PaymentMethod_Bank transfer (automatic)     1.00
PaymentMethod_Credit card (automatic)       1.00
PaymentMethod_Electronic check              1.00
PaymentMethod_Mailed check                  1.00
tenure                                     72.00
SeniorCitizen                               1.00
MonthlyCharges                            118.75
TotalCharges                             8684.80
```

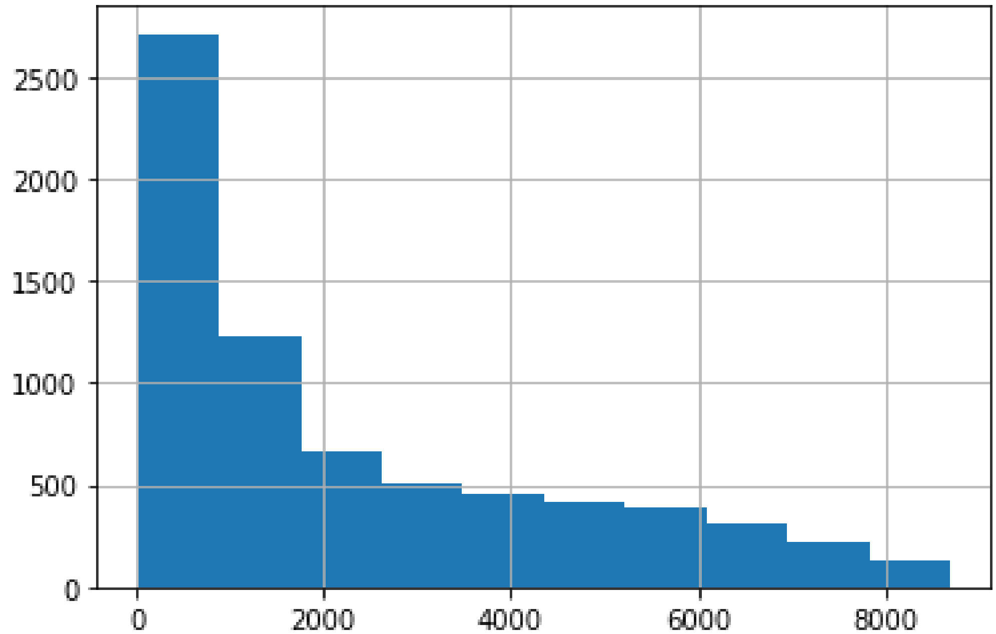
The only two features where makes sense to check for skewness are the MonthlyCharges and TotalCharges features.

[32]:
```
skew_limit=0.75
skew_vals = data_enc.loc[:,['MonthlyCharges', 'TotalCharges']].skew()

skew_cols = (skew_vals
             .sort_values(ascending=False)
             .to_frame()
             .rename(columns={0:'Skew'})
             .query('abs(Skew) > {0}'.format(skew_limit)))

skew_cols
```

[32]:
```
                  Skew
TotalCharges  0.961642
```

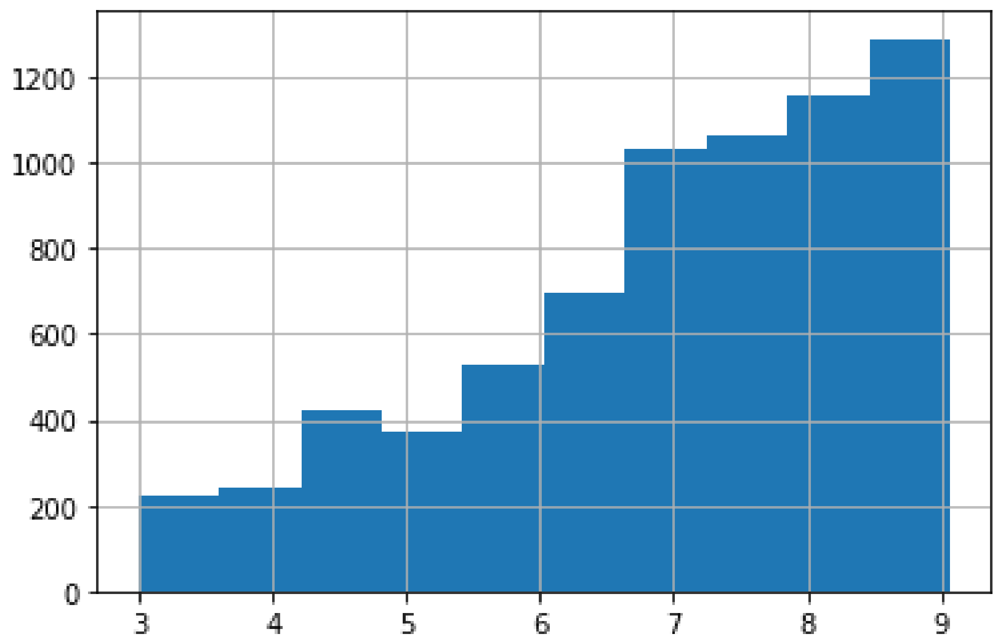
[33]:
```
data_enc.TotalCharges.hist()
```

[33]: <AxesSubplot:>



[34]:
```
data_enc.TotalCharges.apply(np.log1p).hist()
```

[34]: <AxesSubplot:>



[35]:
```python
data_enc.TotalCharges=data_enc.TotalCharges.apply(np.log1p)
```

[36]:
```python
data_enc.TotalCharges
```

[36]:
```
0       3.429137
1       7.544597
2       4.692723
3       7.518471
4       5.028148
          …
7027    7.596643
7028    8.904345
7029    5.850621
7030    5.728800
7031    8.831347
Name: TotalCharges, Length: 7032, dtype: float64
```

1.3 Exploratory Data Analysis

Lets see the correlation between our target variable 'Churn' and all the features

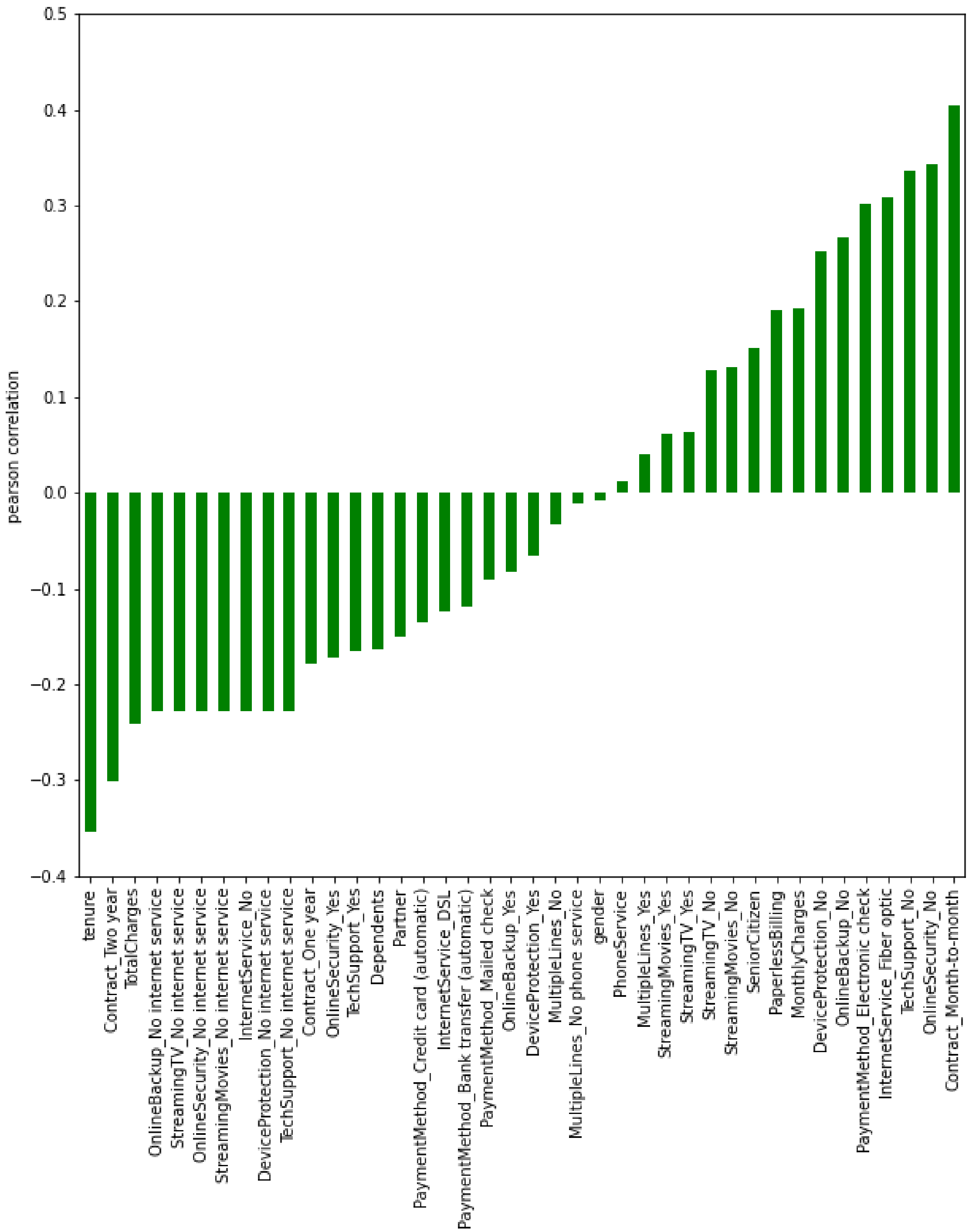
[37]:
```python
y = (data_enc['Churn'])
features = list(data_enc.loc[:, data_enc.columns != 'Churn'])  # everything␣
↪except "color"
correlations = data_enc[features].corrwith(y)
correlations.sort_values(inplace=True)
correlations
```

[37]:
```
tenure                                    -0.354049
Contract_Two year                         -0.301552
TotalCharges                              -0.241908
OnlineBackup_No internet service          -0.227578
StreamingTV_No internet service           -0.227578
OnlineSecurity_No internet service        -0.227578
StreamingMovies_No internet service       -0.227578
InternetService_No                        -0.227578
DeviceProtection_No internet service      -0.227578
TechSupport_No internet service           -0.227578
Contract_One year                         -0.178225
OnlineSecurity_Yes                        -0.171270
TechSupport_Yes                           -0.164716
Dependents                                -0.163128
Partner                                   -0.149982
PaymentMethod_Credit card (automatic)     -0.134687
InternetService_DSL                       -0.124141
PaymentMethod_Bank transfer (automatic)   -0.118136
```

```
PaymentMethod_Mailed check                -0.090773
OnlineBackup_Yes                          -0.082307
DeviceProtection_Yes                      -0.066193
MultipleLines_No                          -0.032654
MultipleLines_No phone service            -0.011691
gender                                    -0.008545
PhoneService                               0.011691
MultipleLines_Yes                          0.040033
StreamingMovies_Yes                        0.060860
StreamingTV_Yes                            0.063254
StreamingTV_No                             0.128435
StreamingMovies_No                         0.130920
SeniorCitizen                              0.150541
PaperlessBilling                           0.191454
MonthlyCharges                             0.192858
DeviceProtection_No                        0.252056
OnlineBackup_No                            0.267595
PaymentMethod_Electronic check             0.301455
InternetService_Fiber optic                0.307463
TechSupport_No                             0.336877
OnlineSecurity_No                          0.342235
Contract_Month-to-month                    0.404565
dtype: float64
```

[38]:
```python
ax = correlations.plot(kind='bar', color='g', figsize=(10,10))
ax.set(ylim=[-.4, .5], ylabel='pearson correlation')
```

[38]:
```
[(-0.4, 0.5), Text(0, 0.5, 'pearson correlation')]
```



1.4 Machine Learning

First lets split train and test sets.

[39]:
```
feature_cols = [x for x in data_enc.columns if x != 'Churn']
```

[40]:
```
strat_shuff_split = StratifiedShuffleSplit(n_splits=1, test_size=1500,␣
↪random_state=42)

train_idx, test_idx = next(strat_shuff_split.split(data_enc[feature_cols],␣
↪data_enc['Churn']))

X_train = data_enc.loc[train_idx, feature_cols]
y_train = data_enc.loc[train_idx, 'Churn']

X_test = data_enc.loc[test_idx, feature_cols]
y_test = data_enc.loc[test_idx, 'Churn']
```

[41]:
```
y_train.value_counts(normalize=True)
```

[41]:
```
0    0.734273
1    0.265727
Name: Churn, dtype: float64
```

Lets scale it even though I dont think its necessary but it never hurts do to it.

[42]:
```
ss = StandardScaler()
X_train_s = ss.fit_transform(X_train)
X_test_s = ss.transform(X_test)
```

[43]:
```
#mm=MinMaxScaler()
#X_train_m = mm.fit_transform(X_train)
#X_test_m = mm.transform(X_test)
```

Lets define a function to copare the prediction on the model between training and test sets

[44]:
```
def measure_error(y_true, y_pred, label):
    return pd.Series({'accuracy':accuracy_score(y_true, y_pred),
                      'precision': precision_score(y_true, y_pred),
                      'recall': recall_score(y_true, y_pred),
                      'f1': f1_score(y_true, y_pred)},
                      name=label)
```

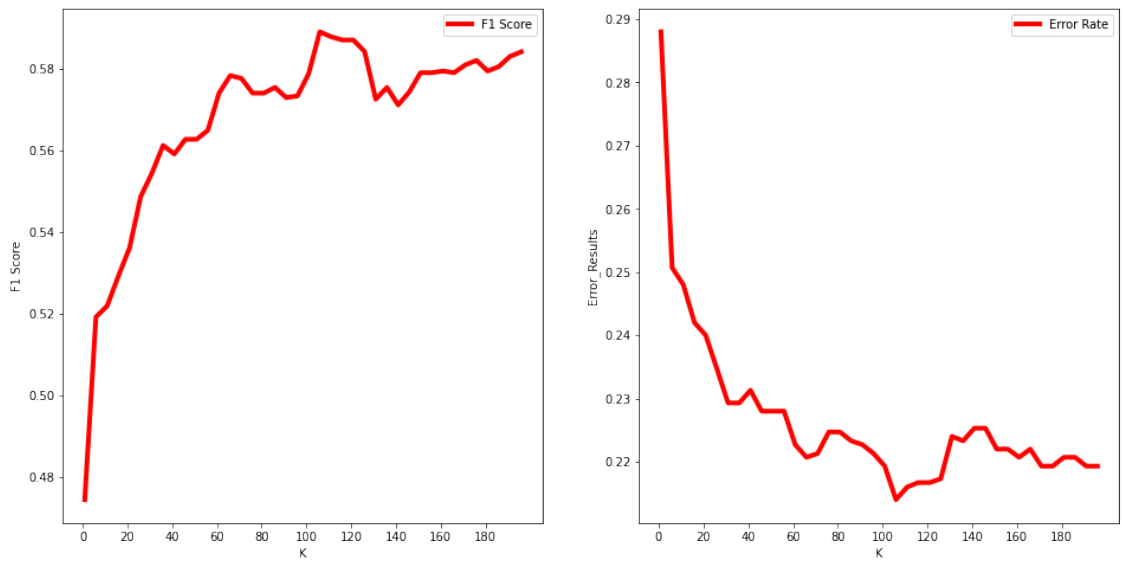
1.4.1 K-Nearest-Neighbors

Lets try to classify the costumer churn or not churn with a k-nearest-neighbors method.

[45]:
```
max_k = 200
f1_scores = list()
error_rates = list()

for k in range(1, max_k, 5):

    knn = KNeighborsClassifier(n_neighbors=k, weights='distance')
```

```
    knn = knn.fit(X_train_s, y_train)

    y_pred = knn.predict(X_test_s)
    f1_scores.append((k, round(f1_score(y_test, y_pred), 4)))
    error = 1-round(accuracy_score(y_test, y_pred), 4)
    error_rates.append((k, error))

f1_results = pd.DataFrame(f1_scores, columns=['K', 'F1 Score'])
error_results = pd.DataFrame(error_rates, columns=['K', 'Error Rate'])
```

[46]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16,8))
plt.figure(dpi=300)
f1_results.set_index('K').plot(color='r', linewidth=4, ax=ax1)
ax1.set(xlabel='K', ylabel='F1 Score')
ax1.set_xticks(range(0, max_k, 20))
error_results.set_index('K').plot(color='r', linewidth=4, ax=ax2)
ax2.set(xlabel='K', ylabel='Error_Results');
ax2.set_xticks(range(0, max_k, 20));
```




```
<Figure size 1800x1200 with 0 Axes>
```

[47]:
```
print('The maximum F1 Score is:', f1_results['F1 Score'].max(), 'with K equal␣
 ↪to:',
      f1_results['K'].loc[f1_results['F1 Score'].idxmax()])
print('The minimum error rate is:', round(error_results['Error Rate'].min(),␣
 ↪5), 'with K equal to:',
      error_results['K'].loc[error_results['Error Rate'].idxmin()])
```

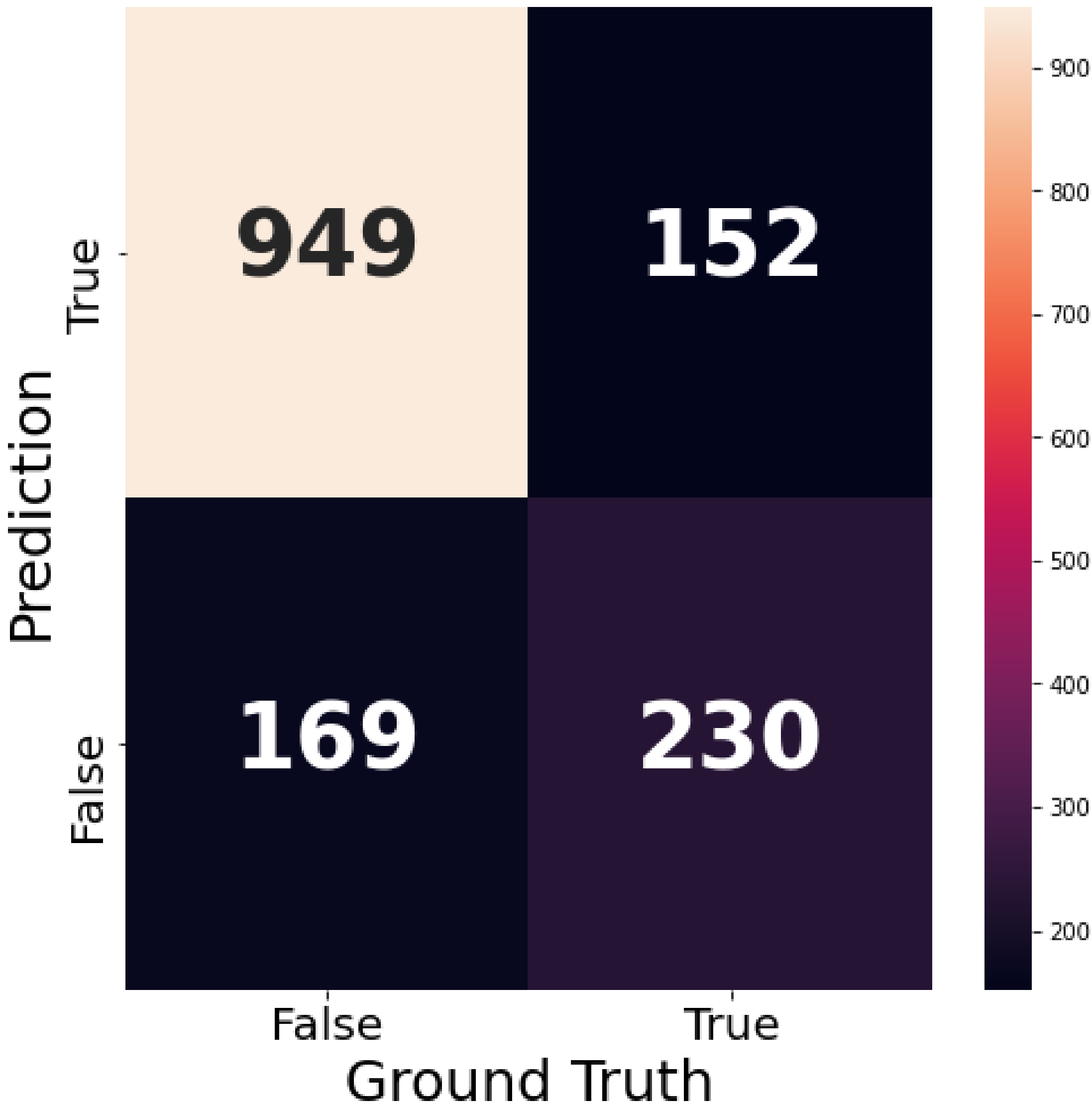
```
The maximum F1 Score is: 0.589 with K equal to: 106
```

The minimum error rate is: 0.214 with K equal to: 106

[48]:
```
knn = KNeighborsClassifier(n_neighbors=106, weights='distance')
knn = knn.fit(X_train_s, y_train)

y_pred_best_knn = knn.predict(X_test_s)
y_train_pred = knn.predict(X_train_s)
```

[49]:
```
cm = confusion_matrix(y_test, y_pred_best_knn)
_, ax = plt.subplots(figsize=(8,8))
ax = sns.heatmap(cm, annot=True, fmt='d', annot_kws={"size": 40, "weight":␣
 ↪"bold"})

labels = ['False', 'True']
ax.set_xticklabels(labels, fontsize=20)
ax.set_yticklabels(labels[::-1], fontsize=20)
ax.set_ylabel('Prediction', fontsize=25)
ax.set_xlabel('Ground Truth', fontsize=25)
```

[49]: Text(0.5, 51.0, 'Ground Truth')



[50]:
```
print(classification_report(y_test, y_pred_best_knn))
```

```
              precision    recall  f1-score   support

           0       0.85      0.86      0.86      1101
           1       0.60      0.58      0.59       399

    accuracy                           0.79      1500
   macro avg       0.73      0.72      0.72      1500
weighted avg       0.78      0.79      0.78      1500
```

[51]:
```
train_test_full_error = pd.concat([measure_error(y_train, y_train_pred,␣
↪'train'),
                              measure_error(y_test, y_pred_best_knn, 'test')],
                              axis=1)

train_test_full_error
```

[51]:

	train	test
accuracy	0.998915	0.786000
precision	0.999318	0.602094
recall	0.996599	0.576441
f1	0.997956	0.588988

1.4.2 Knn with train-test-split instead of the stratified one

[52]:
```
Xknn=data_enc.drop(['Churn'], axis=1)
yknn=data_enc['Churn']
```

[53]:
```
X_train_x, X_test_x, y_train_y, y_test_y = train_test_split(Xknn, yknn,␣
↪test_size=0.4, random_state=42)
```

[54]:
```
max_k = 200
f1_scores = list()
error_rates = list()

X_train_x = ss.fit_transform(X_train_x)
X_test_x = ss.transform(X_test_x)

for k in range(1, max_k, 5):

    knn = KNeighborsClassifier(n_neighbors=k, weights='distance')
    knn = knn.fit(X_train_x, y_train_y)

    y_pred_y = knn.predict(X_test_x)
    f1_scores.append((k, round(f1_score(y_test_y, y_pred_y), 4)))
    error = 1-round(accuracy_score(y_test_y, y_pred_y), 4)
    error_rates.append((k, error))

f1_results = pd.DataFrame(f1_scores, columns=['K', 'F1 Score'])
error_results = pd.DataFrame(error_rates, columns=['K', 'Error Rate'])
```

[55]:
```
print('The maximum F1 Score is:', f1_results['F1 Score'].max(), 'with K equal␣
↪to:',
      f1_results['K'].loc[f1_results['F1 Score'].idxmax()])
print('The minimum error rate is:', round(error_results['Error Rate'].min(),␣
↪5), 'with K equal to:',
```

```
    error_results['K'].loc[error_results['Error Rate'].idxmin()])
```

```
The maximum F1 Score is: 0.5974 with K equal to: 81
The minimum error rate is: 0.2065 with K equal to: 81
```

[56]:
```
knn = KNeighborsClassifier(n_neighbors=81, weights='distance')
knn = knn.fit(X_train_x, y_train_y)

y_pred_best_knn_y = knn.predict(X_test_x)
y_train_pred_y = knn.predict(X_train_x)
```

[57]:
```
print(classification_report(y_test_y, y_pred_best_knn_y))
```

```
              precision    recall  f1-score   support

           0       0.86      0.87      0.86      2079
           1       0.61      0.59      0.60       734

    accuracy                           0.79      2813
   macro avg       0.73      0.73      0.73      2813
weighted avg       0.79      0.79      0.79      2813
```

1.4.3 Linear Support Vector Machine

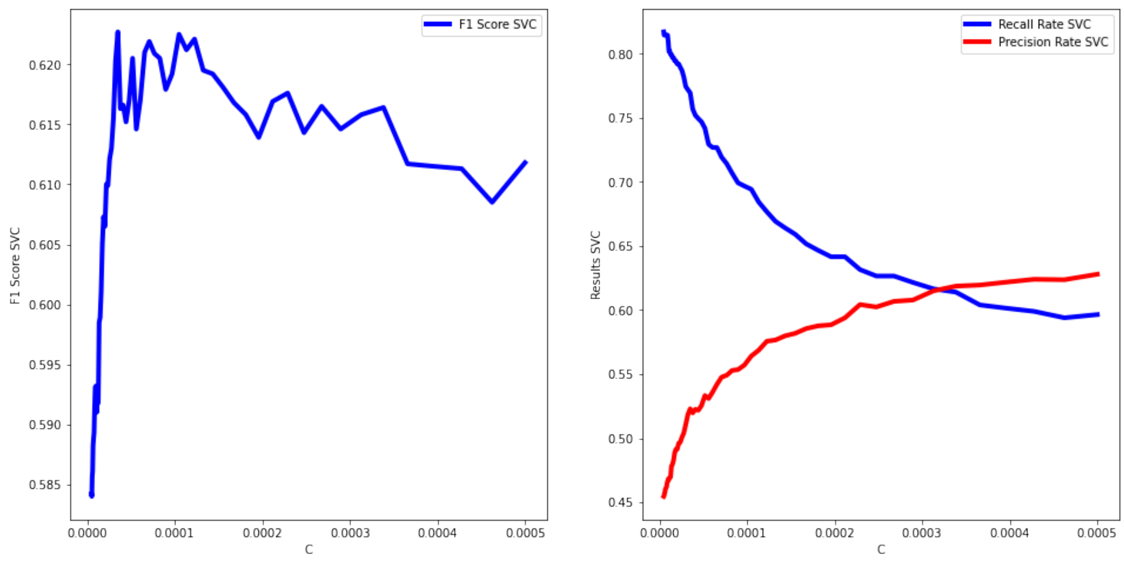
Next I will try a Support Vector Machine classification methods to compare it to the KNN method. I will not use more complex SVM methods with kernels, gammas or approximations since my dataset have a little more then 7k rows and just 31 features so it be considered pretty small. A LinearSVC shoud suffice

[58]:
```
C_list = np.geomspace(5e-4, 5e-6, 60)
f1_scores_SVC = []
error_rates_SVC = []
recall_rates_SVC = []
precision_rates_SVC = []
accuracy_rates_SVC = []
matthews_rates_SVC = []

for val in C_list:
    LSVC = LinearSVC(C=val)
    LSVC.fit(X_train_s, y_train)
    y_pred_SVC = LSVC.predict(X_test_s)
    accuracy_SVC = accuracy_score(y_test, y_pred_SVC)
    error_SVC = 1-accuracy_SVC
    f1_SVC = f1_score(y_test, y_pred_SVC)
    recall_SVC = recall_score(y_test, y_pred_SVC)
    precision_SVC =precision_score(y_test, y_pred_SVC)
    matthews_SVC = matthews_corrcoef(y_test, y_pred_SVC)
```

```
    accuracy_rates_SVC.append((val, accuracy_SVC))
    f1_scores_SVC.append((val, round(f1_SVC, 4)))
    error_rates_SVC.append((val, error_SVC))
    recall_rates_SVC.append((val, recall_SVC))
    precision_rates_SVC.append((val, precision_SVC))
    matthews_rates_SVC.append((val, matthews_SVC))

acc_results_SVC = pd.DataFrame(accuracy_rates_SVC, columns=['C', 'Accuracy␣
 ↪SVC'])
f1_results_SVC = pd.DataFrame(f1_scores_SVC, columns=['C', 'F1 Score SVC'])
error_results_SVC = pd.DataFrame(error_rates_SVC, columns=['C', 'Error Rate␣
 ↪SVC'])
recall_results_SVC = pd.DataFrame(recall_rates_SVC, columns=['C', 'Recall Rate␣
 ↪SVC'])
precision_results_SVC = pd.DataFrame(precision_rates_SVC, columns=['C',␣
 ↪'Precision Rate SVC'])
matthews_results_SVC = pd.DataFrame(matthews_rates_SVC, columns=['C', 'Matthews␣
 ↪Rate SVC'])

#print(f1_results_SVC.C.loc[f1_results_SVC['F1 Score SVC'].idxmax()])
```

[59]:
```
fig, (ax1, ax3) = plt.subplots(1, 2, figsize=(16,8))
plt.figure(dpi=300)
f1_results_SVC.set_index('C').plot(color='b', linewidth=4, ax=ax1)
ax1.set(xlabel='C', ylabel='F1 Score SVC')
#error_results_SVC.set_index('C').plot(color='b', linewidth=4, ax=ax2)
#ax2.set(xlabel='C', ylabel=' Results SVC')
recall_results_SVC.set_index('C').plot(color='b', linewidth=4, ax=ax3)
ax3.set(xlabel='C', ylabel='Results SVC')
precision_results_SVC.set_index('C').plot(color='r', linewidth=4, ax=ax3)
ax3.set(xlabel='C', ylabel='Results SVC')
#acc_results_SVC.set_index('C').plot(color='g', linewidth=4, ax=ax3)
#ax3.set(xlabel='C', ylabel='Results SVC')
```

[59]:
```
[Text(0.5, 0, 'C'), Text(0, 0.5, 'Results SVC')]
```



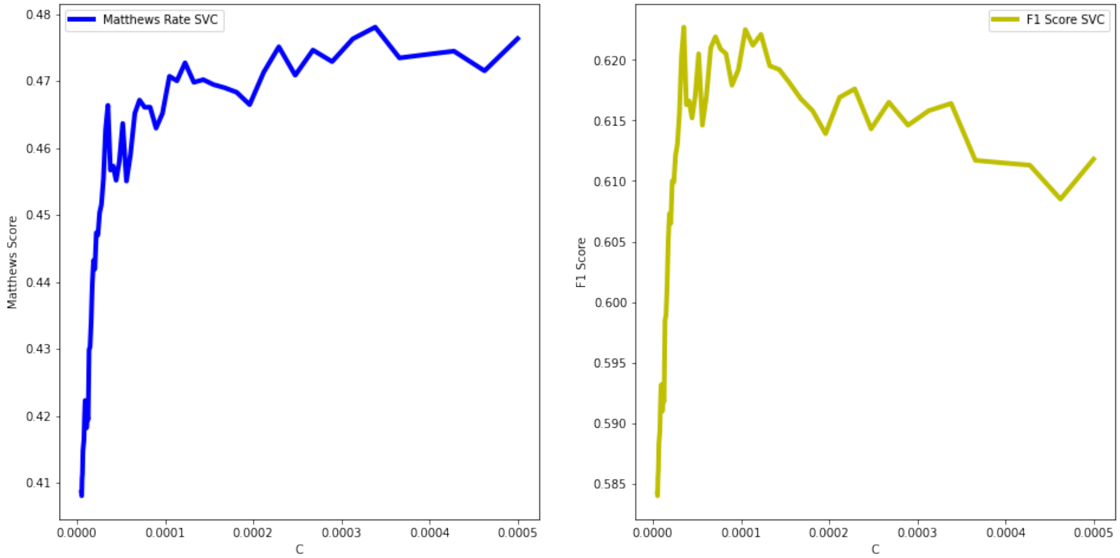

```
<Figure size 1800x1200 with 0 Axes>
```

[60]:
```
print('The maximum F1 Score is:', f1_results_SVC['F1 Score SVC'].max(), 'with C␣
↪equal to:',
      f1_results_SVC['C'].loc[f1_results_SVC['F1 Score SVC'].idxmax()])
```

```
The maximum F1 Score is: 0.6227 with C equal to: 3.5190677774657735e-05
```

[61]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16,8))
plt.figure(dpi=300)
matthews_results_SVC.set_index('C').plot(color='b', linewidth=4, ax=ax1)
ax1.set(xlabel='C', ylabel='Matthews Score')
f1_results_SVC.set_index('C').plot(color='y', linewidth=4, ax=ax2)
ax2.set(xlabel='C', ylabel='F1 Score')
```

[61]:
```
[Text(0.5, 0, 'C'), Text(0, 0.5, 'F1 Score')]
```




```
<Figure size 1800x1200 with 0 Axes>
```

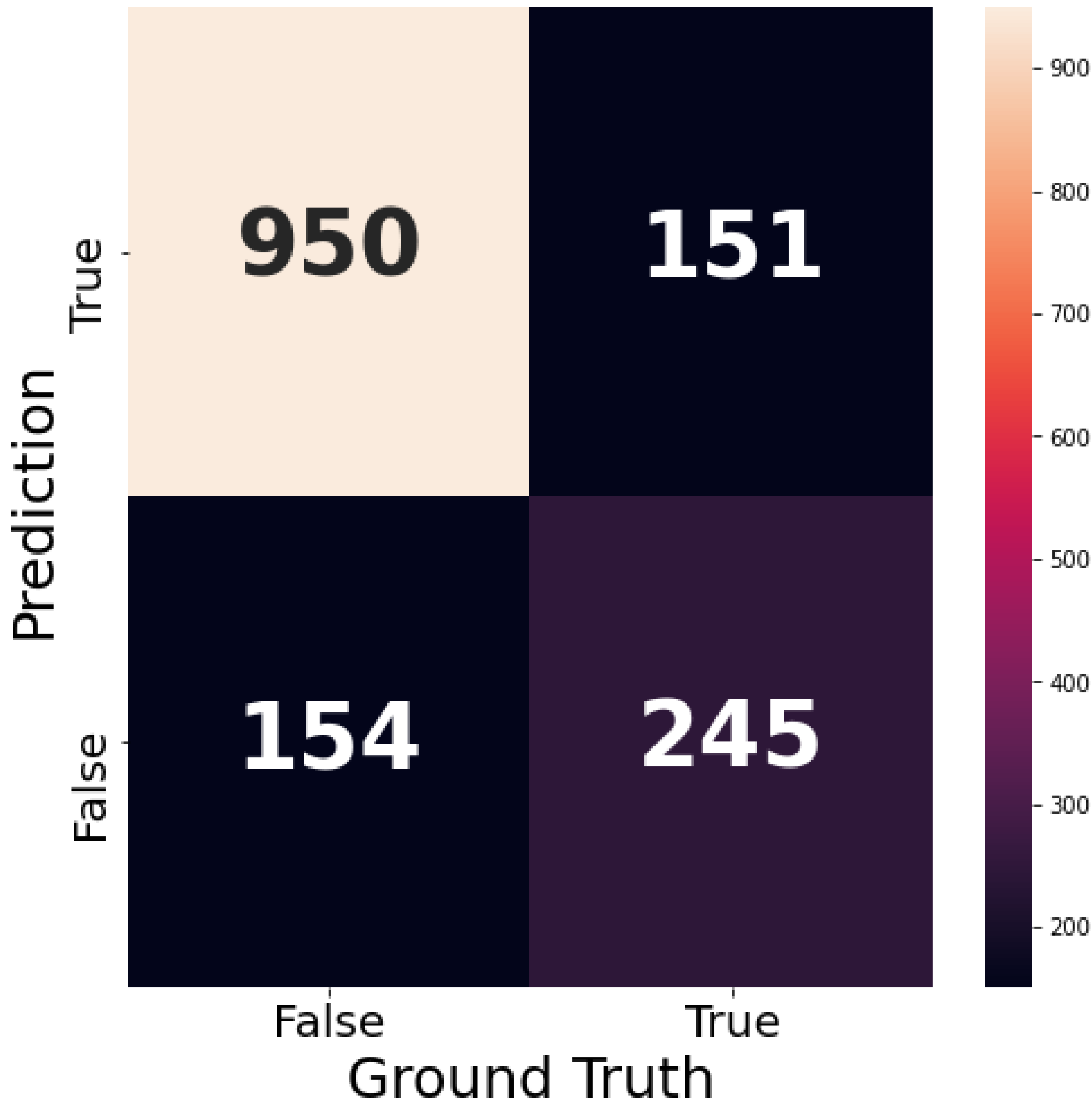
[62]:
```
print('The maximum Matthews Score is:', matthews_results_SVC['Matthews Rate SVC'].max(), 'with C equal to:',
      matthews_results_SVC['C'].loc[matthews_results_SVC['Matthews Rate SVC'].idxmax()])
```

```
The maximum Matthews Score is: 0.47803918600121775 with C equal to:
0.0003384375004729267
```

[63]:
```
best_linear_SVC = LinearSVC(C=0.0003384375004729267)

best_linear_SVC.fit(X_train_s, y_train)
y_pred_best_SVC = best_linear_SVC.predict(X_test_s)
y_train_pred = best_linear_SVC.predict(X_train_s)
```

[64]:
```
cm = confusion_matrix(y_test, y_pred_best_SVC)
_, ax = plt.subplots(figsize=(8,8))
ax = sns.heatmap(cm, annot=True, fmt='d', annot_kws={"size": 40, "weight": "bold"})

labels = ['False', 'True']
ax.set_xticklabels(labels, fontsize=20)
ax.set_yticklabels(labels[::-1], fontsize=20)
ax.set_ylabel('Prediction', fontsize=25)
ax.set_xlabel('Ground Truth', fontsize=25)
```

[64]:
```
Text(0.5, 51.0, 'Ground Truth')
```



[65]:
```python
print(classification_report(y_test, y_pred_best_SVC))
```

```
              precision    recall  f1-score   support

           0       0.86      0.86      0.86      1101
           1       0.62      0.61      0.62       399

    accuracy                           0.80      1500
   macro avg       0.74      0.74      0.74      1500
weighted avg       0.80      0.80      0.80      1500
```

[66]:
```
train_test_full_error = pd.concat([measure_error(y_train, y_train_pred,␣
↪'train'),
                                   measure_error(y_test, y_pred_best_SVC, 'test')],
                                   axis=1)

train_test_full_error
```

[66]:
```
              train      test
accuracy   0.806580  0.796667
precision  0.646628  0.618687
recall     0.600000  0.614035
f1         0.622442  0.616352
```

1.4.4 Decision Tree & Random Forest Classifier

Lets now try some random forest classifier

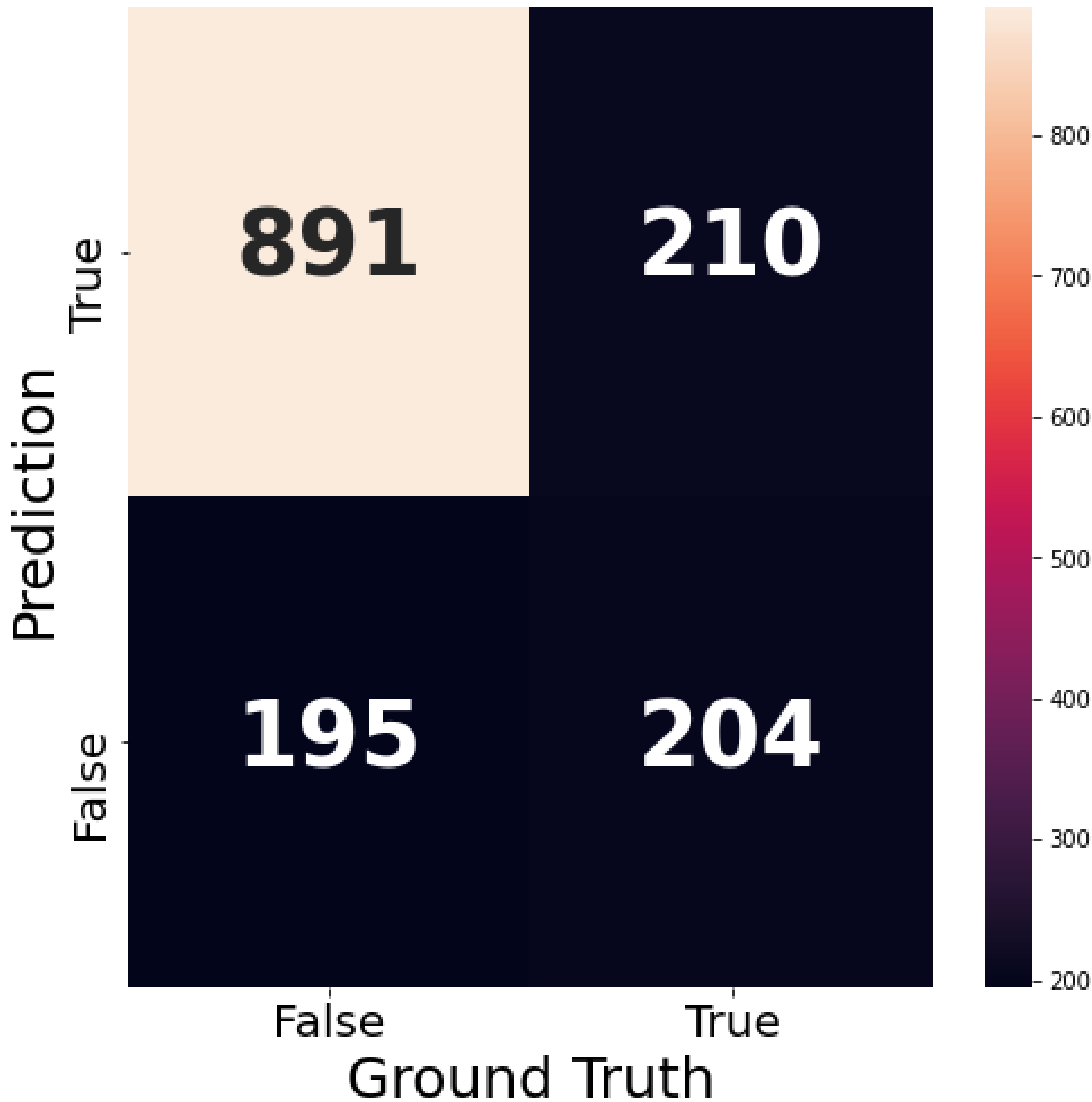
[67]:
```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state=42)
dt = dt.fit(X_train_s, y_train)
dt.tree_.node_count, dt.tree_.max_depth
```

[67]:
```
(2157, 23)
```

[68]:
```
y_train_pred = dt.predict(X_train_s)
y_pred_dt = dt.predict(X_test_s)
```

[69]:
```
cm = confusion_matrix(y_test, y_pred_dt)
_, ax = plt.subplots(figsize=(8,8))
ax = sns.heatmap(cm, annot=True, fmt='d', annot_kws={"size": 40, "weight":␣
↪"bold"})

labels = ['False', 'True']
ax.set_xticklabels(labels, fontsize=20)
ax.set_yticklabels(labels[::-1], fontsize=20)
ax.set_ylabel('Prediction', fontsize=25)
ax.set_xlabel('Ground Truth', fontsize=25)
```

[69]:
```
Text(0.5, 51.0, 'Ground Truth')
```



[70]:
```
print(classification_report(y_test, y_pred_dt))
```

```
              precision    recall  f1-score   support

           0       0.82      0.81      0.81      1101
           1       0.49      0.51      0.50       399

    accuracy                           0.73      1500
   macro avg       0.66      0.66      0.66      1500
weighted avg       0.73      0.73      0.73      1500
```

[71]:
```
train_test_full_error = pd.concat([measure_error(y_train, y_train_pred,␣
↪'train'),
                                   measure_error(y_test, y_pred_dt, 'test')],
                                   axis=1)

train_test_full_error
```

[71]:
```
              train      test
accuracy   0.998735  0.730000
precision  0.999317  0.492754
recall     0.995918  0.511278
f1         0.997615  0.501845
```

Lets search for the best decision tree

[72]:
```
#from sklearn.model_selection import GridSearchCV

param_grid = {'max_depth':range(1, dt.tree_.max_depth+1, 2),
              'max_features': range(1, len(dt.feature_importances_)+1)}

GR = GridSearchCV(DecisionTreeClassifier(random_state=42),
                  param_grid=param_grid,
                  scoring='accuracy',
                  n_jobs=-1)

GR = GR.fit(X_train_s, y_train)
```

[73]:
```
GR.best_estimator_
```

[73]:
```
DecisionTreeClassifier(max_depth=5, max_features=11, random_state=42)
```

[74]:
```
GR.best_estimator_.tree_.node_count, GR.best_estimator_.tree_.max_depth
```

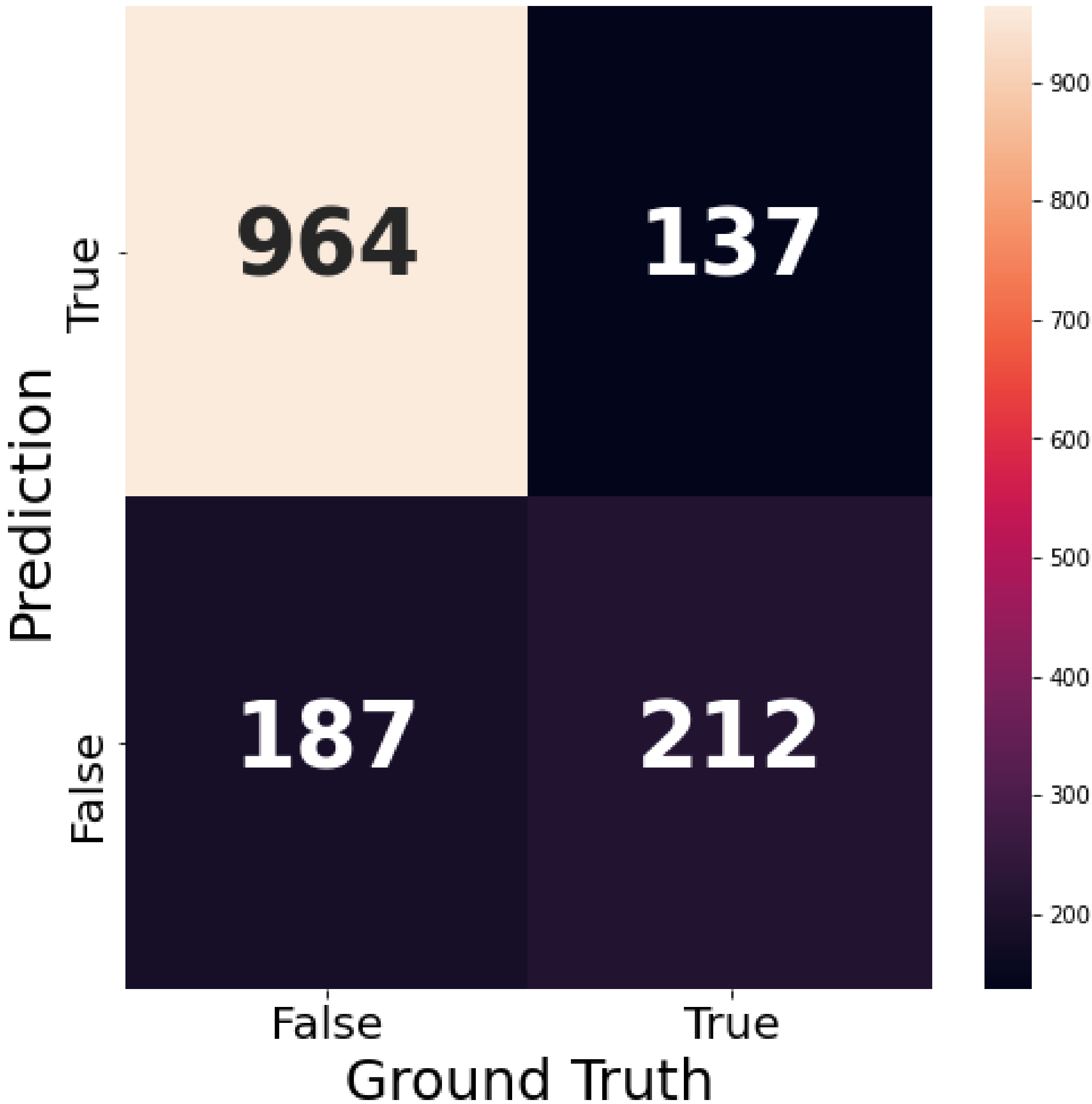
[74]:
```
(63, 5)
```

[75]:
```
y_train_pred_gr = GR.predict(X_train_s)
y_test_pred_gr = GR.predict(X_test_s)
```

[76]:
```
cm = confusion_matrix(y_test, y_test_pred_gr)
_, ax = plt.subplots(figsize=(8,8))
ax = sns.heatmap(cm, annot=True, fmt='d', annot_kws={"size": 40, "weight":␣
↪"bold"})

labels = ['False', 'True']
ax.set_xticklabels(labels, fontsize=20)
ax.set_yticklabels(labels[::-1], fontsize=20)
ax.set_ylabel('Prediction', fontsize=25)
```

```python
ax.set_xlabel('Ground Truth', fontsize=25)
```

[76]: Text(0.5, 51.0, 'Ground Truth')



[77]:
```python
print(classification_report(y_test, y_test_pred_gr))
```

```
              precision    recall  f1-score   support

           0       0.84      0.88      0.86      1101
           1       0.61      0.53      0.57       399

    accuracy                           0.78      1500
   macro avg       0.72      0.70      0.71      1500
```

```
weighted avg       0.78      0.78      0.78      1500
```

[78]:
```
train_test_gr_error = pd.concat([measure_error(y_train, y_train_pred_gr,␣
 ↪'train'),
                                  measure_error(y_test, y_test_pred_gr, 'test')],
                                 axis=1)
train_test_gr_error
```

[78]:
```
              train      test
accuracy   0.799168  0.784000
precision  0.641005  0.607450
recall     0.555102  0.531328
f1         0.594969  0.566845
```

Finally using Random Forset

[79]:
```
tree_list=np.linspace(25, 800, 32, dtype=int)
tree_list
```

[79]:
```
array([ 25,  50,  75, 100, 125, 150, 175, 200, 225, 250, 275, 300, 325,
       350, 375, 400, 425, 450, 475, 500, 525, 550, 575, 600, 625, 650,
       675, 700, 725, 750, 775, 800])
```

[80]:
```
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier

RF = RandomForestClassifier(oob_score=True,
                            random_state=43210,
                            warm_start=True,
                            n_jobs=-1)

ET = ExtraTreesClassifier(oob_score=True,
                          random_state=43210,
                          warm_start=True,
                          bootstrap=True,
                          n_jobs=-1)

oob_list_rf = list()
oob_list_et = list()

for n_trees in tree_list:
    #RandomForest
    RF.set_params(n_estimators=n_trees)
    RF.fit(X_train_s, y_train)
    oob_error_rf = 1 - RF.oob_score_
    oob_list_rf.append(pd.Series({'n_trees': n_trees, 'RandomForest OOB':␣
 ↪oob_error_rf}))
```

```
    #ExtraTrees
    ET.set_params(n_estimators=n_trees)
    ET.fit(X_train_s, y_train)
    oob_error_et = 1 - ET.oob_score_
    oob_list_et.append(pd.Series({'n_trees': n_trees, 'ExtraTrees OOB':␣
 ↪oob_error_et}))


rf_oob_df = pd.concat(oob_list_rf, axis=1).T.set_index('n_trees')
et_oob_df = pd.concat(oob_list_et, axis=1).T.set_index('n_trees')
```

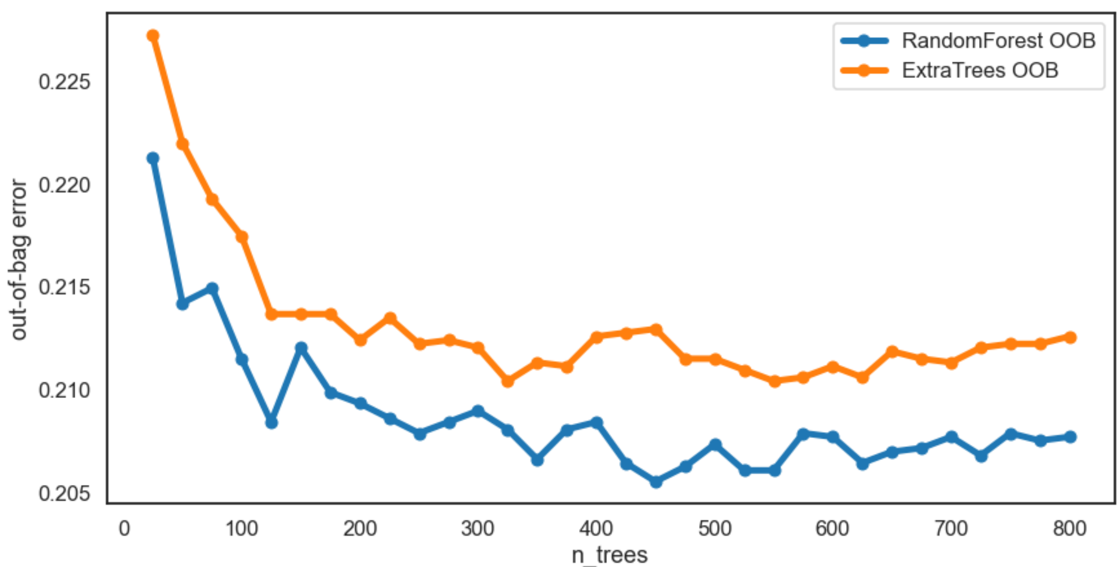
[81]:
```
oob_df = pd.concat([rf_oob_df, et_oob_df], axis=1)
```

[82]:
```
oob_df.head()
```

[82]:

n_trees	RandomForest OOB	ExtraTrees OOB
25.0	0.221258	0.227223
50.0	0.214208	0.221981
75.0	0.214931	0.219270
100.0	0.211497	0.217462
125.0	0.208424	0.213666

[83]:
```
sns.set_context('talk')
sns.set_style('white')

ax = oob_df.plot(legend=True, marker='o', figsize=(14, 7), linewidth=5)
ax.set(ylabel='out-of-bag error');
```



[88]:
```
print('The minimum out of bag error is:', oob_df['RandomForest OOB'].min(),␣
↪'with n of trees equal to:',
      oob_df['RandomForest OOB'].idxmin())
```

```
The minimum out of bag error is: 0.20553145336225598 with n of trees equal to:
450.0
```

[89]:
```
best_RF=RandomForestClassifier(n_estimators=450)
best_RF.fit(X_train_s, y_train)
```

[89]:
```
RandomForestClassifier(n_estimators=450)
```

[94]:
```
y_train_pred_best_RF = best_RF.predict(X_train_s)
y_pred_best_RF = best_RF.predict(X_test_s)
```

[97]:
```
print(classification_report(y_test, y_pred_best_RF))
```

```
              precision    recall  f1-score   support

           0       0.82      0.89      0.85      1101
           1       0.60      0.48      0.54       399

    accuracy                           0.78      1500
   macro avg       0.71      0.68      0.69      1500
weighted avg       0.77      0.78      0.77      1500
```

[ ]:
```

```