MACHINE LEARNING ASSIGNMENT: TURKNET CHURN DATASET

SELAHADDİN HONİ • 040160046

MAHMUT EMİN ERTÜRK • 040160212

RANDOM SIGNALS AND NOISE
SPRING 2021

ABSTRACT

This report aims to give brief explanation on what we have covered in the presentation, first. Then, step-by-step implementation will be shown in the next section.

Our challenge is a binary classification problem on Turknet CHURN dataset having 192,000 rows of 125 features. The tricky issues of the project are selecting the meaningful features to make classification and balancing the instances of two targets to prevent the biased prediction on the class which has much more data than the other one.

In the classification, Random Forest Classifier, Support Vector Machine and Multi-Layer Perceptron algorithms have been used; we have observed very similar performance results.

The source code, presentation and even this report are available at our GitHub repository.

About the Feature Extraction

We proposed an automated method to select the best features. The dataset separated into two based on their target labels: 'ACIK' and 'KAPALI'. First-order statistics are generated for each class and the difference of these statistics values are obtained. The best features are the features whose mean differences are larger.

Table 1 Difference of first-order statistics of two class (not-sorted)

	count	mean	std	min	25%	50%	75%	max
KALDIGI_AY_SAYISI	168388.0	0.043037	0.034273	-0.037736	-0.018868	-0.094340	-0.075472	0.000000
RISKLIMUSTERI	168388.0	0.032751	-0.116186	0.000000	0.000000	0.000000	0.000000	0.000000
KAPASITE	168388.0	0.002163	-0.002285	0.000000	0.000000	0.000000	-0.040650	0.914634
currentDown	166777.0	0.004463	0.000216	0.000000	0.005356	0.008492	0.000001	0.947608
ARKADASINIGETIR	168388.0	0.000016	0.002170	0.000000	0.000000	0.000000	0.000000	0.987352
...	...	...	...	...	...	...	...	...
PORTERROR_SAYISI_3	168388.0	0.002634	-0.009692	0.000000	0.000318	-0.000318	-0.001910	0.195417
MAX_SESSIONTIME_3	168388.0	0.000758	0.011579	0.000000	0.000059	0.000003	0.000001	0.000035
MIN_SESSIONTIME_3	168388.0	0.000082	-0.012324	0.000000	0.000000	0.000000	0.005411	-0.006529
TOTALUPLOADGB_3	168388.0	0.001216	-0.004627	0.000000	0.000873	-0.000277	-0.001700	0.047570
TOTALDOWNLOADGB_3	168388.0	0.006408	-0.017388	0.000000	0.005670	0.000277	-0.008420	0.224918

The Gaussian curves below, represents the first-order statistics of the features, green for 'ACIK' and orange for 'KAPALI'. The features having larger mean difference could make the decision of classification more accurately. Therefore, all features are sorted again according to mean difference. The head of the feature-queue are the best features.

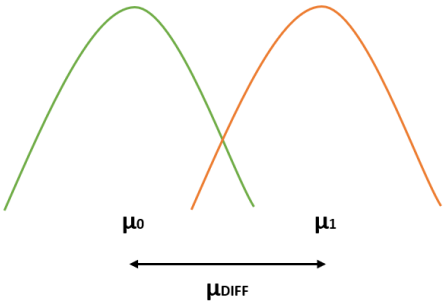


Figure 1 Mean difference illustration

IMPLEMENTATION DETAILS

The project has been constructed on Jupyter Notebook; if you have trouble launching '.ipynb' file, GitHub provides you the notebook support with pre-obtained outputs of each cells. Check the repository shared in the abstract section.

In the below cell outputs, 10 best features are picked to train and obtain inference. However, increasing the number of features up to 50 is increased the accuracy as well. Our observations can be found in appendix section.

***

Importing the dataset to Python environment over Pandas framework.

```
import pandas as pd
import numpy as np

dataset_path = "./dataset/turknetchurnekimanonim.xlsx"
dataset = pd.read_excel(dataset_path)
```

Min-Max Scaling for numeric values

First, choose the numerical features with select_dtypes() method. Apply, scaling on these features in other words columns.

```
dataset = dataset.select_dtypes(include=np.number)

for column in dataset.columns:
    dataset[column] = (dataset[column] - dataset[column].min()) / (dataset[column].max() - dataset[column].min())

dataset["DURUM"] = DURUM
dataset.tail()
```

	KALDIGI_AY_SAYISI	RISKLIMUSTERI	KAPASITE	currentDown	ARKADASINIGETIR	ARKADASINIGETIRILEGELENLER	FATURA_GECIKME_1	FATURA_GECIKME_UCRETI_1	CAGRIMERKEZIARAMASAYISI_1	DESTEKSAYISI_1	...	DENVERGIRISTARIHI_3	DENVERCIKISTARIHI_
192289	0.037736	0.0	0.019309	0.015647	0.000000	0.0	0.333333	0.020774	0.0	0.009479	...	NaN	Na
192290	0.037736	0.0	0.085366	NaN	0.000415	1.0	0.000000	0.000000	0.0	0.018957	...	NaN	Na
192291	0.037736	0.0	0.019309	0.015268	0.000207	1.0	0.000000	0.000000	0.0	0.000000	...	NaN	Na
192292	0.037736	0.0	0.019309	0.015645	0.000000	1.0	0.000000	0.000000	0.0	0.000000	...	NaN	Na
192293	0.037736	0.0	0.019309	0.015335	0.000000	0.0	0.000000	0.000000	0.0	0.000000	...	NaN	Na

5 rows × 112 columns

Separating the dataset for two target classes: 'ACIK' and 'KAPALI'

Find the row indices having 'K' instance in the dataset and storing them in Python list 'rows_durum_K'

Selecting 'rows_durum_K' from the dataset with 'iloc' method forms the 'dataset_K'
Dropping the 'rows_durum_K' from the dataset with 'drop' method forms the 'dataset_A'

```
rows_durum_K = []

for row in range(dataset.shape[0]):
    if dataset["DURUM"][row] == 'K':
        rows_durum_K.append(row)

dataset_K = dataset.iloc[rows_durum_K]
K = dataset_K.describe().transpose()

dataset_A = dataset.drop(rows_durum_K)
A = dataset_A.describe().transpose()
```

K: first-order statistics of class 'K'
A: first-order statistics of class 'A'

```
# An example table of first-order statistics
K
```

	count	mean	std	min	25%	50%	75%	max
KALDIGI_AY_SAYISI	11953.0	0.292397	0.213871	0.037736	0.113208	0.226415	0.415094	1.000000
RISKLIMUSTERI	11953.0	0.038651	0.192771	0.000000	0.000000	0.000000	0.000000	1.000000
KAPASITE	11953.0	0.031754	0.030740	0.000000	0.019309	0.019309	0.059959	0.085366
currentDown	11828.0	0.010086	0.009182	0.000000	0.001797	0.007153	0.015647	0.052392
ARKADASINIGETIR	11953.0	0.000172	0.000450	0.000000	0.000000	0.000000	0.000207	0.012648
...	...	...	...	...	...	...	...	...
PORTERROR_SAYISI_3	11953.0	0.006624	0.022126	0.000000	0.000318	0.001910	0.005729	0.804583
MAX_SESSIONTIME_3	11953.0	0.001264	0.020405	0.000000	0.000993	0.001081	0.001088	0.999965
MIN_SESSIONTIME_3	11953.0	0.016789	0.060534	0.000000	0.000000	0.000000	0.003354	1.000000
TOTALUPLOADGB_3	11953.0	0.006525	0.015491	0.000000	0.000587	0.003484	0.007755	0.952430
TOTALDOWNLOADGB_3	11953.0	0.035843	0.050390	0.000000	0.003352	0.020203	0.047469	0.775082

Difference of two class tables of statistics

Best features having more power to classify these two targets are the features larger mean difference. In the for loop, all values of the 'mean' column are converting to their absolute.

```
DIFF = A - K

# Convert the absolute values
for row in range(DIFF.shape[0]):
    val = DIFF["mean"][row]
    if val:
        DIFF["mean"][row] = abs(val)
```

DIFF table after absolute value conversion:

	count	mean	std	min	25%	50%	75%	max
KALDIGI_AY_SAYISI	168388.0	0.043037	0.034273	-0.037736	-0.018868	-0.094340	-0.075472	0.000000
RISKLIMUSTERI	168388.0	0.032751	-0.116186	0.000000	0.000000	0.000000	0.000000	0.000000
KAPASITE	168388.0	0.002163	-0.002285	0.000000	0.000000	0.000000	-0.040650	0.914634
currentDown	166777.0	0.004463	0.000216	0.000000	0.005356	0.008492	0.000001	0.947608
ARKADASINIGETIR	168388.0	0.000016	0.002170	0.000000	0.000000	0.000000	0.000000	0.987352
...	...	...	...	...	...	...	...	...
PORTERROR_SAYISI_3	168388.0	0.002634	-0.009692	0.000000	0.000318	-0.000318	-0.001910	0.195417
MAX_SESSIONTIME_3	168388.0	0.000758	0.011579	0.000000	0.000059	0.000003	0.000001	0.000035
MIN_SESSIONTIME_3	168388.0	0.000082	-0.012324	0.000000	0.000000	0.000000	0.005411	-0.006529
TOTALUPLOADGB_3	168388.0	0.001216	-0.004627	0.000000	0.000873	-0.000277	-0.001700	0.047570
TOTALDOWNLOADGB_3	168388.0	0.006408	-0.017388	0.000000	0.005670	0.000277	-0.008420	0.224918

Sorting the features according to descending-order of 'mean' column

Because obtained features will be used for training, the rows having 'N/A' values are dropped.

```
mean_sorted = DIFF.sort_values(['mean'],ascending=False)['mean']
best_features = mean_sorted[0:10].index
dataset = dataset.dropna(subset=best_features)
```

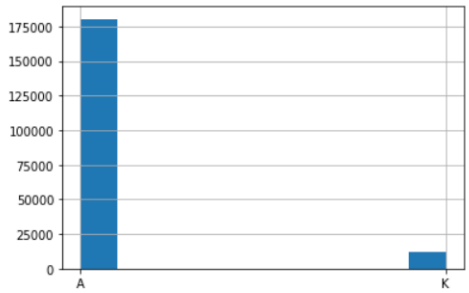
Number of the features is a new hyper-parameter now. Here are the first 10 features of the sorted-table:

```
Index(['TDU_TICKETSL_1', 'ADSLARIZA_TICKETSL_1', 'KALDIGI_AY_SAYISI',
       'RISKLIMUSTERI', 'IKNATICKET_1', 'TDU_TICKETKAPANMASURESI_1',
       'ADSLARIZA_TICKETSL_2', 'ADSLARIZA_TICKETKAPANMASURESI_1',
       'TDU_TICKETSL_2', 'TDU_DESTEKSAYISI_1'],
      dtype='object')
```

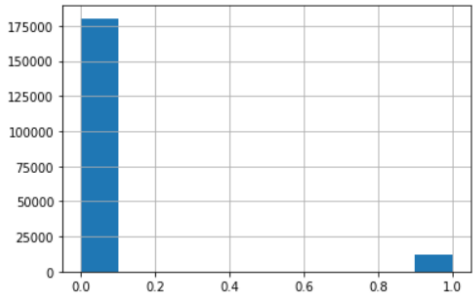
Target label-encoding

```
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
dataset["DURUM"] = labelencoder.fit_transform(dataset["DURUM"])
```

Histogram of target classes before label-encoding:



Histogram of target classes after label-encoding:

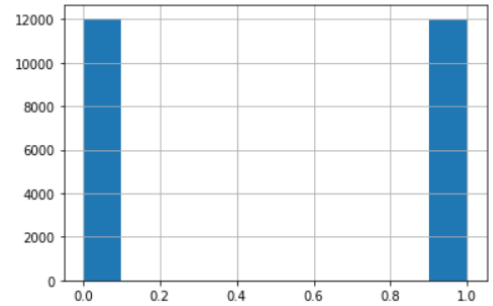


Balancing the instances of two classes

The number of rows of two target classes are not balanced. Therefore, rows of '0' will be reduced by resampling. In the for loop, if condition checks the row is an instance of first class '0'; appends the index of the current dataset to 'drop_rows' list with an approximate resampling of 1/15.

It is a programming must reset the dataset indices before this process.

```
dataset = dataset.reset_index(drop=True)

drop_rows = []
for row in range(dataset.shape[0]):
    if dataset["DURUM"][row] == 0:
        if not row%15 == 0:
            drop_rows.append(row)

Qset = dataset.drop(drop_rows)
Qset["DURUM"].hist()
```



Train-Validation Split

Selecting the best features as X and "DURUM" column as target y.

```
from sklearn.model_selection import train_test_split

X = Qset[best_features]
y = Qset["DURUM"]

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.33, random_state=1)
```

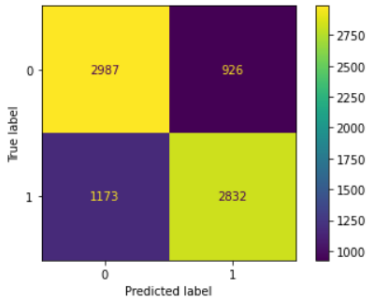
Random Forest Classifier

It is an easy concept that train a model and get predictions. Every classifier we have used are already implemented in 'sklearn' library. Random forest is an ensemble machine learning method and we engage this algorithm to an object named 'clf'.

By using the 'fit' method to train our model and 'predict' method to get inference. Furthermore, this object stores all performance evaluations with a help of 'sklearn.metrics' class; confusion matrix can be visualized with 'plot_confusion_matrix()' function.

```
from sklearn.ensemble import RandomForestClassifier

clf=RandomForestClassifier(n_estimators=100)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_val)

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

plot_confusion_matrix(clf, X_val, y_val)
print("Accuracy:", accuracy_score(y_val, y_pred))
print(classification_report(y_val, y_pred, target_names=['ACIK','KAPALI']))
```

```
              precision    recall  f1-score   support

        ACIK       0.72      0.76      0.74      3913
      KAPALI       0.75      0.71      0.73      4005

    accuracy                           0.73      7918
   macro avg       0.74      0.74      0.73      7918
weighted avg       0.74      0.73      0.73      7918
```

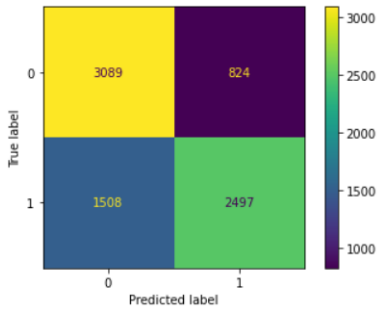


SVM Classifier

As explained before the algorithm calls with an object ‘clf’.

```
from sklearn.svm import SVC

clf = SVC(kernel='rbf')

clf.fit(X_train, y_train)
y_pred = clf.predict(X_val)

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

plot_confusion_matrix(clf, X_val, y_val)
print("Accuracy:", accuracy_score(y_val, y_pred))
print(classification_report(y_val, y_pred, target_names=['ACIK','KAPALI']))
```

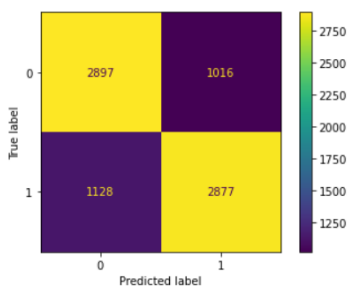
```
              precision    recall  f1-score   support

        ACIK       0.67      0.79      0.73      3913
      KAPALI       0.75      0.62      0.68      4005

    accuracy                           0.71      7918
   macro avg       0.71      0.71      0.70      7918
weighted avg       0.71      0.71      0.70      7918
```



MLP Classifier – 32 Units

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(hidden_layer_sizes=32, activation='relu', solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200,
shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999,
epsilon=1e-08, n_iter_no_change=10, max_fun=15000)

clf.fit(X_train, y_train)
y_pred = clf.predict(X_val)

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

plot_confusion_matrix(clf, X_val, y_val)
print("Accuracy:", accuracy_score(y_val, y_pred))
print(classification_report(y_val, y_pred, target_names=['ACIK','KAPALI']))
```

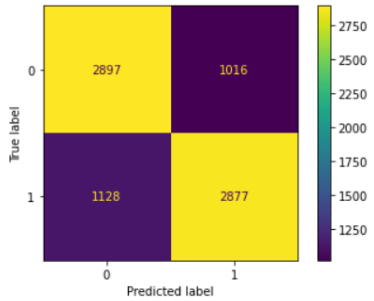
```
              precision    recall  f1-score   support

        ACIK       0.72      0.74      0.73      3913
      KAPALI       0.74      0.72      0.73      4005

    accuracy                           0.73      7918
   macro avg       0.73      0.73      0.73      7918
weighted avg       0.73      0.73      0.73      7918
```



MLP Classifier – 250 Units

Same with MLP-32 code; only difference is 'hidden_layer_size' argument is now 250.

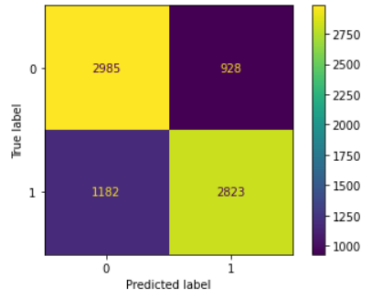
ACIK	0.72	0.74	0.73	3913
KAPALI	0.74	0.72	0.73	4005
accuracy			0.73	7918
macro avg	0.73	0.73	0.73	7918
weighted avg	0.73	0.73	0.73	7918



MLP Classifier – 500 Units

Same with MLP-32 code; only difference is 'hidden_layer_size' argument is now 500.

ACIK	0.72	0.76	0.74	3913
KAPALI	0.75	0.70	0.73	4005
accuracy			0.73	7918
macro avg	0.73	0.73	0.73	7918
weighted avg	0.73	0.73	0.73	7918



APPENDIX

Comparison of confusion matrices obtained from Random Forest Classifier (100 estimators) for the change of the number of features (10, 25 and 50) from 'best_features' list.

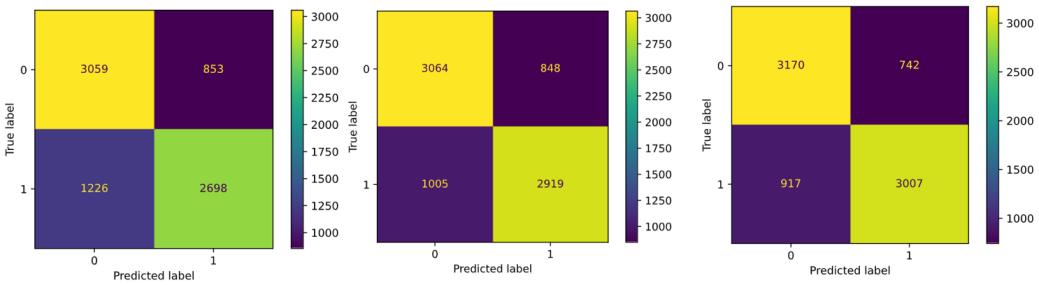
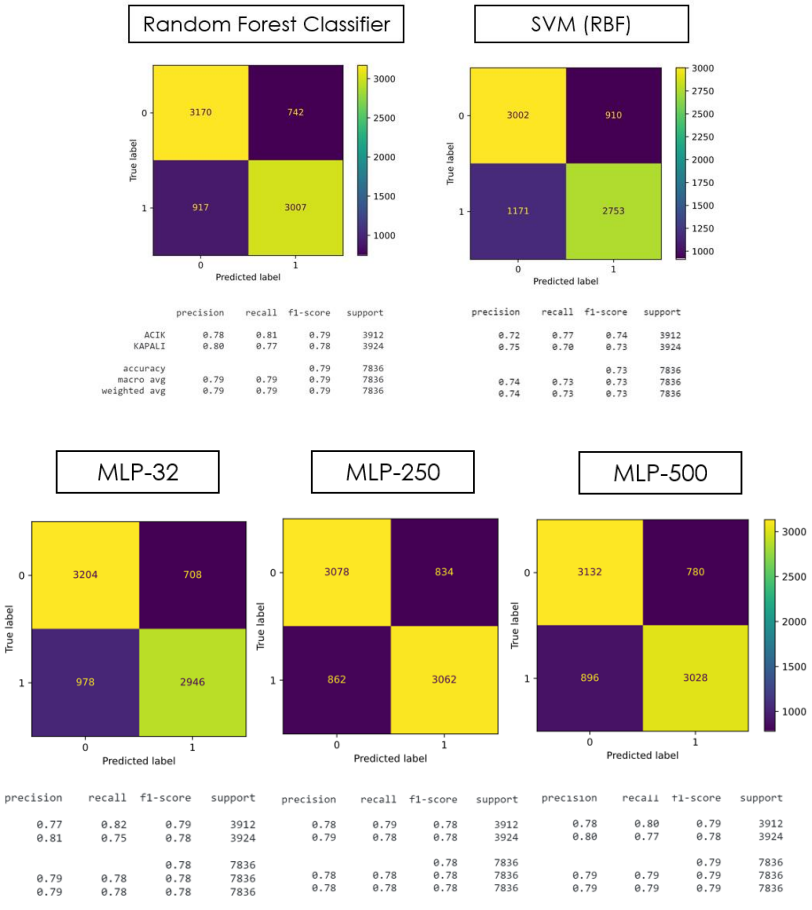


Figure 2 Confusion matrices for 10, 25 and 50 features respectively

By looking the matrices, we can say that increasing the number of features up to 50 also increased the precision score of the second class. Therefore, we choose this hyper-parameter as 50 for further validations.



Obtained results are very similar any model can be picked for an application needs approximate %70 to %80 accuracy. However, for the 'KAPALI' class, models for best precision is MLP-32 and best recall is MLP-250.

A better performance model may be acquired by constructing a deeper neural model as a suggestion.