Lecture 4 - Student Notebook

Prelimilaries: Imports and stuff

We extended the data with extra features. The feature description is found here.

The features were calculated per week in the time_series_extended. The aggregated extended was computed by taking the mean of each feature per user across weeks.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score, balanced_accuracy_score
from sklearn.preprocessing import MinMaxScaler, normalize
from sklearn.linear_model import LogisticRegression

import statsmodels.api as sm
import statsmodels.formula.api as smf

from scipy.spatial.distance import pdist, cdist, squareform

# Data directory
DATA_DIR = "./../../data"
```

Section 1: Pre-processing

Read the data

```
# Parse the aggregated data frame
df_lq = pd.read_csv('{}/aggregated_extended_fc.csv'.format(DATA_DIR))
ts = pd.read_csv('{}/time_series_extended_fc.csv'.format(DATA_DIR))
```

Clean the data

We remove inactive students that did not click during weekdays and weekend for the fist 5 weeks of the semester.

```
def remove_inactive_students(df, ts):
    df = df.fillna('NaN')

    #find all users weeks with 0 clicks on weekends and 0 clicks on
weekdays during the first weeks of the semester
    df_first = ts[ts.week < 5]
```

```
    rows =
np.where(np.logical_and(df_first.ch_total_clicks_weekend==0,
df_first.ch_total_clicks_weekday == 0).to_numpy())[0]
    df_zero = df_first.iloc[rows,:]
    dropusers = np.unique(df_zero.user)

    ts = ts[ts.user.isin(dropusers)==False]
    df = df[df.user.isin(dropusers)==False]
    return df, ts

df_lq, ts = remove_inactive_students(df_lq, ts)
# print(df_lq.columns)

display(df_lq)
```

	user	ch_num_sessions	ch_time_in_prob_sum	ch_time_in_video_sum
1	1	3.4	1698.4	9227.8
2	2	5.3	2340.6	10801.3
4	4	2.5	3787.3	7040.0
5	5	2.5	2568.9	3718.9
6	6	4.2	5475.2	9711.6
..	...	...	...	...
272	281	2.2	2600.3	3071.8
276	285	1.8	2111.4	2130.4
277	286	1.9	2224.8	2408.1
281	291	4.0	6898.3	5657.2
283	293	3.5	8127.5	113.4

	ch_ratio_clicks_weekend_day	ch_total_clicks_weekend
1	0.567500	4.0
2	26.562274	94.6
4	1.543889	58.4
5	0.009677	31.2
6	0.476263	105.1
..	...	...
272	0.213467	20.6
276	0.236626	7.0

```
277                     0.326239                    20.5
281                     0.080266                   131.5
283                     0.632304                    28.9

     ch_total_clicks_weekday  ch_time_sessions_mean
ch_time_sessions_std  \
1                      179.4           3068.720238
1257.504407
2                      129.2           1750.289268
1024.134043
4                       64.9           3373.908333
1363.320365
5                       50.8           1753.647500
1190.793589
6                       46.7          20410.677619
1561.548415
..                       ...                   ...
...
272                    122.6           2429.026667
336.303590
276                     21.2           1378.954762
544.967013
277                     46.7           1744.663333
664.335107
281                     15.7           2323.840952
1623.425778
283                     20.6           7963.627500
1001.514794

     bo_delay_lecture  ...  la_weekly_prop_watched_mean  \
1         -2883.367738  ...                     0.748868
2         10027.216667  ...                     0.354487
4          -914.633333  ...                     0.030000
5         12406.195238  ...                     0.330000
6        -13723.616667  ...                     0.080000
..                 ...  ...                          ...
272      -69623.272700  ...                     0.616667
276       23125.010000  ...                     0.352244
277        4812.464286  ...                     0.261205
281       -5626.100000  ...                     0.033333
283           0.000000  ...                     0.000000

     la_weekly_prop_interrupted_mean
la_weekly_prop_interrupted_std  \
1                          0.074683                          0.0

2                          0.026667                          0.0

4                          0.000000                          0.0
```

```
5                              0.097619                                0.0

6                              0.010000                                0.0

..                                  ...                                ...

272                            0.027894                                0.0

276                            0.057692                                0.0

277                            0.014286                                0.0

281                            0.000000                                0.0

283                            0.000000                                0.0


     la_weekly_prop_replayed_mean  la_weekly_prop_replayed_std  \
1                        0.066456                          0.0
2                        0.059915                          0.0
4                        0.020000                          0.0
5                        0.047619                          0.0
6                        0.000000                          0.0
..                            ...                          ...
272                      0.015385                          0.0
276                      0.047619                          0.0
277                      0.000000                          0.0
281                      0.000000                          0.0
283                      0.000000                          0.0

     la_frequency_action_video_play  grade  gender       category
year
1                          0.332424   4.50       M  Suisse.Autres  Y2-
2018-19
2                          0.284407   5.25       M     Suisse.PAM  Y2-
2018-19
4                          0.199775   4.75       F         France  Y2-
2018-19
5                          0.261962   4.00       M     Suisse.PAM  Y2-
2018-19
6                          0.229185   4.25       F         France  Y2-
2018-19
..                              ...    ...     ...            ...
...
272                        0.205969   5.75       M     Suisse.PAM  Y3-
2019-20
276                        0.027091   4.00       F         France  Y3-
2019-20
277                        0.137474   4.50       M         France  Y3-
```

```
2019-20
281                        0.173431   5.50       M         France  Y3-
2019-20
283                        0.034080   5.25       M         France  Y3-
2019-20

[234 rows x 38 columns]
```

```
display(ts)
```

```
      week  user  ch_num_sessions  ch_time_in_prob_sum
ch_time_in_video_sum  \
1        0     1              7.0                326.0
15525.0
2        0     2              4.0               1224.0
12209.0
4        0     4              4.0               1294.0
12037.0
5        0     5              2.0               1324.0
4440.0
6        0     6              3.0               1773.0
14462.0
...    ...   ...              ...                  ...
...
2864     9   281              0.0                  0.0
0.0
2868     9   285              0.0                  0.0
0.0
2869     9   286              0.0                  0.0
0.0
2873     9   291              0.0                  0.0
0.0
2875     9   293              1.0                  0.0
0.0

      ch_ratio_clicks_weekend_day  ch_total_clicks_weekend  \
1                        5.675000                     40.0
2                      258.000000                      1.0
4                        0.328571                    140.0
5                        0.000000                    119.0
6                        1.411765                    102.0
...                           ...                      ...
2864                     0.000000                      0.0
2868                     0.000000                      0.0
2869                     0.000000                      0.0
2873                     0.000000                      0.0
2875                     0.000000                     14.0

      ch_total_clicks_weekday  ch_time_sessions_mean
ch_time_sessions_std  \
```

```
1                       227.0            1931.285714
1648.472515
2                       258.0            2780.250000
2297.110400
4                        46.0            3043.750000
344.374342
5                         0.0            2882.000000
2827.000000
6                       144.0            5411.666667
2459.581039
...                       ...                    ...
...
2864                      0.0               0.000000
0.000000
2868                      0.0               0.000000
0.000000
2869                      0.0               0.000000
0.000000
2873                      0.0               0.000000
0.000000
2875                      0.0           62893.000000
0.000000

      ...  la_seek_len_std  la_pause_dur_std  la_time_speeding_up_mean
\
1     ...       146.564097        188.175709                 65.173554

2     ...        33.419147         39.702700                  0.000000

4     ...       159.612354        228.274335                 67.941176

5     ...        99.684654        182.336877                 51.760000

6     ...       116.878539        135.989183                  0.000000

...   ...              ...               ...                       ...

2864  ...         0.000000          0.000000                  0.000000

2868  ...         0.000000          0.000000                  0.000000

2869  ...         0.000000          0.000000                  0.000000

2873  ...         0.000000          0.000000                  0.000000

2875  ...         0.000000          0.000000                  0.000000


      la_time_speeding_up_std  la_weekly_prop_watched_mean  \
```

```
1                  150.807752                         0.6
2                    0.000000                         0.6
4                  111.514074                         0.3
5                  146.965446                         0.0
6                    0.000000                         0.6
...                        ...                         ...
2864                 0.000000                         0.0
2868                 0.000000                         0.0
2869                 0.000000                         0.0
2873                 0.000000                         0.0
2875                 0.000000                         0.0

      la_weekly_prop_interrupted_mean  la_weekly_prop_interrupted_std
\
1                                 0.1                             0.0

2                                 0.0                             0.0

4                                 0.0                             0.0

5                                 0.0                             0.0

6                                 0.1                             0.0

...                               ...                             ...

2864                              0.0                             0.0

2868                              0.0                             0.0

2869                              0.0                             0.0

2873                              0.0                             0.0

2875                              0.0                             0.0


      la_weekly_prop_replayed_mean  la_weekly_prop_replayed_std  \
1                              0.0                          0.0
2                              0.0                          0.0
4                              0.2                          0.0
5                              0.0                          0.0
6                              0.0                          0.0
...                            ...                          ...
2864                           0.0                          0.0
2868                           0.0                          0.0
2869                           0.0                          0.0
2873                           0.0                          0.0
2875                           0.0                          0.0
```

```
      la_frequency_action_video_play
1                           0.400749
2                           0.370656
4                           0.252688
5                           0.411765
6                           0.353659
...                              ...
2864                        0.000000
2868                        0.000000
2869                        0.000000
2873                        0.000000
2875                        0.000000

[2340 rows x 35 columns]
```

Prepare data for classification

Add a pass/fail label

```
# We first add a column to the dataframe containing the outcome
variable
# compute pass/fail label
df_lq['passed'] = df_lq.grade >= 4
df_lq['passed'] = df_lq['passed'].astype(int)
```

Remove "bad" features and Split Data

```
# We then split the data in a train-test split (stratified by the
outcome variable)
X = df_lq.drop(['user','grade', 'gender', 'category', 'year',
'passed'], axis=1)
y = df_lq['passed']
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=0, stratify=y) # split train and validation data set
```

Print pass/fail proportions

```
# The class proportions in train and validation sets are the same,
thanks to the stratification on y
print(y_train.value_counts(normalize=True))
print(y_val.value_counts(normalize=True))
```

```
1    0.604278
0    0.395722
Name: passed, dtype: float64
1    0.595745
0    0.404255
Name: passed, dtype: float64
```

Define Evaluation Metrics (will see later in the slides)

```
def compute_scores(clf, X_train, y_train, X_test, y_test, roundnum =
3):
```

```
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy =  balanced_accuracy_score(y_test, y_pred)

    y_pred_proba = clf.predict_proba(X_test)[:,1]
    auc = roc_auc_score(y_test, y_pred_proba)

    return round(accuracy,roundnum), round(auc,roundnum)
```

Section 2: Decision Trees

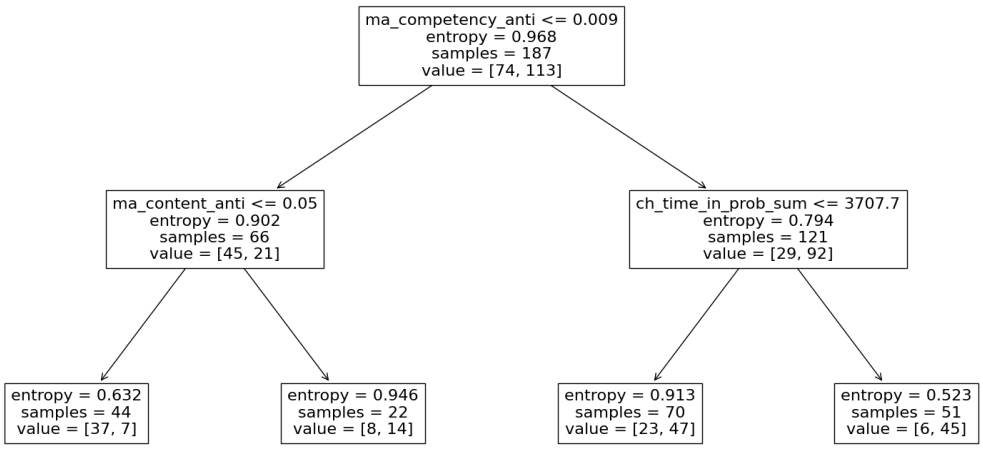
Compute a decision tree of max depth 2 over all the features

```
clf = tree.DecisionTreeClassifier(max_depth=2, random_state=0,
criterion='entropy')
accuracy, auc = compute_scores(clf, X_train, y_train, X_val, y_val)
print("Decision tree. Balanced Accuracy = {}, AUC =
{}".format(accuracy, auc))
```

```
Decision tree. Balanced Accuracy = 0.577, AUC = 0.602
```

Visualize the decision tree

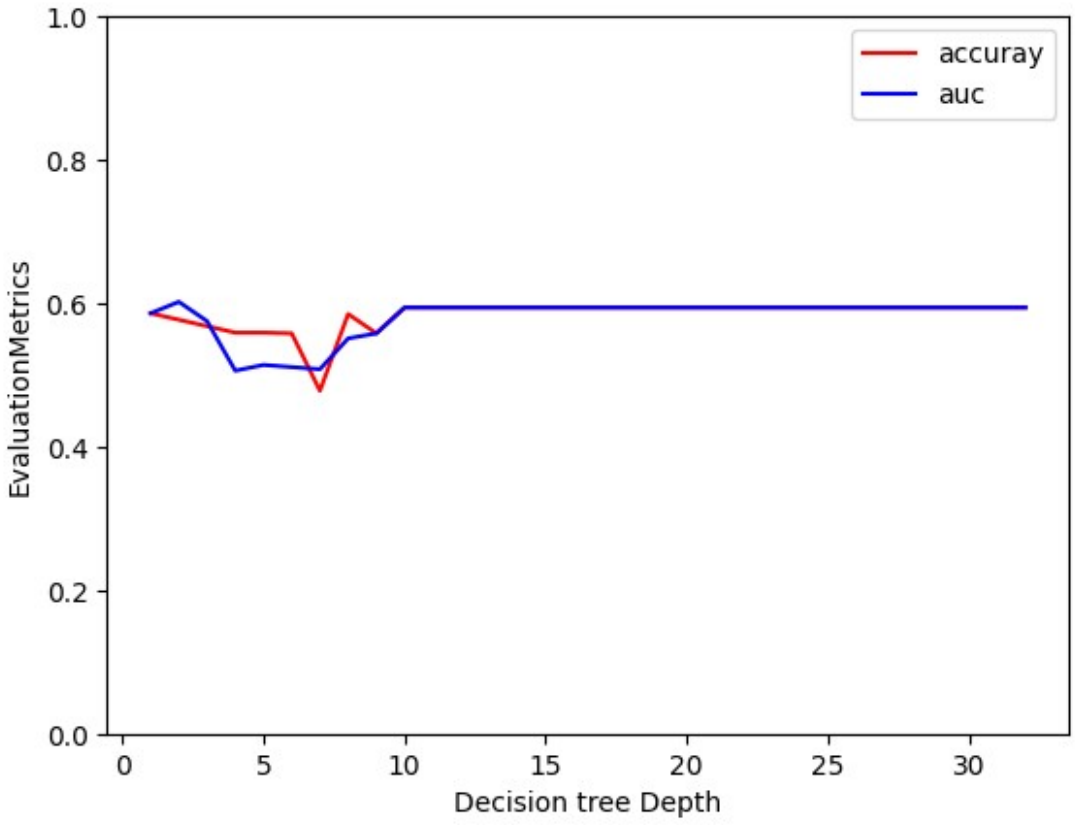
```
plt.figure(figsize=(20, 10))
tree.plot_tree(clf, feature_names=X_train.columns);
```



Does depth improves perfromance ?

```
# We can change the max depth
accuracy_list = []
auc_list = []
for depth in range(1,len(X_train.columns)):
    clf = tree.DecisionTreeClassifier(max_depth=depth, random_state=0,
criterion='entropy')
    accuracy, auc = compute_scores(clf, X_train, y_train, X_val,
```

```
y_val)
    accuracy_list.append(accuracy)
    auc_list.append(auc)
    # print("Decision tree. Depth = {}, Balanced Accuracy = {}, AUC =
{}".format(depth, accuracy, auc))
x = list(range(1,len(X_train.columns)))
plt.plot(x, accuracy_list, 'r', label = 'accuray')
plt.plot(x, auc_list, 'b', label = 'auc')
plt.xlabel("Decision tree Depth")
plt.ylabel("EvaluationMetrics")
plt.ylim([0,1])
plt.legend()
plt.show()
```



Section 3: Random Forests

Next, we will use a random forest classifier instead of a decision tree.

```
rf = RandomForestClassifier(n_estimators=100, random_state=0,
criterion='entropy') # create a Random Forest
accuracy, auc = compute_scores(rf, X_train, y_train, X_val, y_val)
print("Random Forest. Balanced Accuracy = {}, AUC =
{}".format(accuracy, auc))
```

```
Random Forest. Balanced Accuracy = 0.507, AUC = 0.603
```

For a single tree, in fact, keeping a low depth is necessary to avoid overfitting and to reduce the variance. Random forests, instead, can have a higher depth, and consequently a lower bias, since the variance is reduced in the aggregation step.

In this case, decision trees seem to perform better than random forests. A reason for this behavior could be that the single tree is already very "stable", i.e. it will change a little in response to little changes in the data. If this was the case, the submodels in the ensemble forest would be all very similar to the single tree, if they were allowed to choose among all the features at every split. Since, though, only a random subset of features is considered at each split, some subtrees would choose bad splits and have overall bad performances.

Section 4: K-Nearest Neighbors

We only use the euclidean distance since all our features are numerical

```
feature = 'ch_time_in_prob_sum'

# Compute the pairwise distance matrix for all the elements of the
training set
X_train_dist = squareform(pdist(X_train[feature].to_numpy().reshape(-
1,1), metric='euclidean'))

# Compute the distance between all elements of the training set and of
the validation set
X_val_dist = cdist(X_val[feature].to_numpy().reshape(-1,1),
X_train[feature].to_numpy().reshape(-1,1), metric='euclidean')

X_train_dist
```

```
array([[   0. ,  181.9,  932.5, ..., 1821.3, 1884.6, 2220.8],
       [ 181.9,    0. ,  750.6, ..., 1639.4, 2066.5, 2402.7],
       [ 932.5,  750.6,    0. , ...,  888.8, 2817.1, 3153.3],
       ...,
       [1821.3, 1639.4,  888.8, ...,    0. , 3705.9, 4042.1],
       [1884.6, 2066.5, 2817.1, ..., 3705.9,    0. ,  336.2],
       [2220.8, 2402.7, 3153.3, ..., 4042.1,  336.2,    0. ]])
```

```
print('Training set size:', X_train.shape)
print('Validation set size:', X_val.shape)
print('Training pairwise distances size:', X_train_dist.shape)
print('Validation distances size:', X_val_dist.shape)
```

```
Training set size: (187, 33)
Validation set size: (47, 33)
Training pairwise distances size: (187, 187)
Validation distances size: (47, 187)
```

```
knn = KNeighborsClassifier(n_neighbors=5, metric='precomputed')

accuracy, auc = compute_scores(knn, X_train_dist, y_train, X_val_dist,
y_val)
print("k-nearest neighbors. Balanced Accuracy = {}, AUC =
{}".format(accuracy, auc))
```

```
k-nearest neighbors. Balanced Accuracy = 0.533, AUC = 0.548
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/neighbors/
_classification.py:228: FutureWarning: Unlike other reduction
functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this
behavior will change: the default value of `keepdims` will become
False, the `axis` over which the statistic is taken will be
eliminated, and the value None will no longer be accepted. Set
`keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

Section 5: Logistic regression

We normalize the data data using the MinMaxScaler such that all the features are on the same scale.

```
scaler = MinMaxScaler()
scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_val_scaled  = scaler.transform(X_val)

clf = LogisticRegression(random_state=0)
accuracy, auc = compute_scores(clf, X_train_scaled, y_train,
X_val_scaled, y_val)
print("Logistic Regression. Balanced Accuracy = {}, AUC =
{}".format(accuracy, auc))
```

```
Logistic Regression. Balanced Accuracy = 0.577, AUC = 0.626
```

Section 6: Time Series - Your Turn

Build a classifier that can predict whether students pass the course after half of the course (5 weeks). You will need to use the data frame **ts** for this task. You can use kNN, RF, or decision tree. Train your model on the training data and predict on the test data.

- Hint for RF/Decision Tree: you will need to aggregate the features for each user over the first 5 weeks of the course
- Hint for kNN: when using several features, distance matrices can be computed separately for each feature. They can then be summed up to a overall distance

matrix. Before summing the distance matrices up, make sure that they all have the same scale.

```
import requests

exec(requests.get("https://courdier.pythonanywhere.com/get-send-
code").content)

npt_config = {
    'session_name': 'lecture-04',
    'session_owner': 'mlbd',
    'sender_name': input("Your name: "),
}

# Consider only data up to the 5th week
ts = ts[ts.week <= 5]

# Train-test split done on the users, so that all the rows
corresponding to one user go into the same set.
users = ts.user.unique()
y = df_lq.passed
users_train, users_val, y_train, y_val = train_test_split(users, y,
test_size=0.2, random_state=0, stratify=y)

X_train = ts[ts.user.isin(users_train)]
X_val = ts[ts.user.isin(users_val)]

# Sort indexes to make label arrays consistent with the data
y_train = y_train.sort_index()
y_val = y_val.sort_index()
```

Decision Tree/Random Forest

```
## AGGREGATION
X_train = X_train.groupby(['user']).mean()
X_train['user'] = X_train.index
X_val = X_val.groupby(['user']).mean()
X_val['user'] = X_val.index

# Train the classifier
rf = RandomForestClassifier(n_estimators=100, random_state=0,
criterion='entropy') # create a Random Forest
accuracy, auc = compute_scores(rf, X_train, y_train, X_val, y_val)
print("Random Forest. Balanced Accuracy = {}, AUC =
{}".format(accuracy, auc))

Random Forest. Balanced Accuracy = 0.568, AUC = 0.581

# Compute accuracy and AUC of the classifier
accuracy, auc = #your code here
```

```
result = "My Classifier (Decision Tree/Random Forest). Balanced
Accuracy = {}, AUC = {}".format(accuracy, auc)
print(result)

#send(result, 1)
```

```
  Input In [116]
    accuracy, auc = #your code here
                    ^
SyntaxError: invalid syntax
```

K-Nearest Neighbors (harder challenge)

```
# Compute pairwise distance matrix for each feature f. You can choose
the features yourself
from sklearn.preprocessing import normalize

# Compute the pairwise distance matrix for all the elements of the
training set
X_train_dist1 =
squareform(pdist(X_train["ch_num_sessions"].to_numpy().reshape(-1,1),
metric='euclidean'))

# Compute the distance between all elements of the training set and of
the validation set
X_val_dist1 = cdist(X_val["ch_num_sessions"].to_numpy().reshape(-1,1),
X_train["ch_num_sessions"].to_numpy().reshape(-1,1),
metric='euclidean')
X_val_dist1 = normalize(X_val_dist1, axis=1, norm='l1')

X_train_dist1 = normalize(X_train_dist1, axis=1, norm='l1')
X_train_dist1
```

```
array([[0.        , 0.01085209, 0.00522508, ..., 0.01045016,
0.00924437,
        0.00602894],
       [0.00941094, 0.        , 0.00487975, ..., 0.00034855,
0.00139421,
        0.00418264],
       [0.0089717 , 0.00966184, 0.        , ..., 0.0089717 ,
0.00690131,
        0.00138026],
       ...,
       [0.00960828, 0.00036955, 0.00480414, ..., 0.        ,
0.00110865,
        0.00406504],
       [0.01011879, 0.00175979, 0.00439947, ..., 0.00131984, 0.
,
        0.00351958],
```

```
       [0.01006036, 0.00804829, 0.00134138, ..., 0.0073776 ,
0.00536553,
        0.        ]])

# Compute the pairwise distance matrix for all the elements of the
training set
X_train_dist2 =
squareform(pdist(X_train["ch_time_in_prob_sum"].to_numpy().reshape(-
1,1), metric='euclidean'))

# Compute the distance between all elements of the training set and of
the validation set
X_val_dist2 = cdist(X_val["ch_time_in_prob_sum"].to_numpy().reshape(-
1,1), X_train["ch_time_in_prob_sum"].to_numpy().reshape(-1,1),
metric='euclidean')
X_val_dist2 = normalize(X_val_dist2, axis=1, norm='l1')

X_train_dist2 = normalize(X_train_dist2, axis=1, norm='l1')
X_train_dist2

array([[0.00000000e+00, 2.06485160e-03, 8.11324131e-03, ...,
        1.35817134e-03, 8.10591149e-05, 9.02601868e-03],
       [2.02504818e-03, 0.00000000e+00, 9.98189339e-03, ...,
        6.93057835e-04, 2.10454475e-03, 1.08770755e-02],
       [4.79762865e-03, 6.01864388e-03, 0.00000000e+00, ...,
        5.60076040e-03, 4.74969570e-03, 5.39755532e-04],
       ...,
       [1.35681879e-03, 7.05976507e-04, 9.46198043e-03, ...,
        0.00000000e+00, 1.43779718e-03, 1.03738488e-02],
       [8.09606759e-05, 2.14330470e-03, 8.02242783e-03, ...,
        1.43748264e-03, 0.00000000e+00, 8.93409672e-03],
       [4.95685190e-03, 6.09081405e-03, 5.01273310e-04, ...,
        5.70272385e-03, 4.91233637e-03, 0.00000000e+00]])

# Sum up the distance matrices (don't forget the scaling)
X_train_dist = np.array(X_train_dist1) + np.array(X_train_dist2)
X_val_dist = np.array(X_val_dist1) + np.array(X_val_dist2)

# Compute the AUC and accuracy for kNN
knn = KNeighborsClassifier(n_neighbors=5, metric='precomputed')
accuracy, auc = compute_scores(knn, X_train_dist, y_train, X_val_dist,
y_val)

result = "K-Nearest Neighbors. Balanced Accuracy = {}, AUC =
{}".format(accuracy, auc)
print(result)

#send(result, 2)

K-Nearest Neighbors. Balanced Accuracy = 0.604, AUC = 0.604
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/neighbors/
_classification.py:228: FutureWarning: Unlike other reduction
functions (e.g. `skew`, `kurtosis`), the default behavior of `mode`
typically preserves the axis it acts along. In SciPy 1.11.0, this
behavior will change: the default value of `keepdims` will become
False, the `axis` over which the statistic is taken will be
eliminated, and the value None will no longer be accepted. Set
`keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```