# comptetition-wp871q

May 22, 2024

#

Kiss Dániel Márk

##

WP871Q

## 1 Library import

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from sklearn.metrics import accuracy_score
```

## 2 Data import

[2]:
```
df_verseny_public_train = pd.read_csv('data/verseny_public_train.csv', sep=',',␣
↪low_memory=False)
```

## 3 Data familiarization

[642]:
```
#df_verseny_public_train
```

[3]:
```
df_verseny_public_train.describe()
```

[3]:
```
           cookie_id      Topic1_ic      Topic1_ec      Topic2_ic  \
count  100000.000000  100000.000000  100000.000000  100000.000000
mean   149999.500000       8.798000      16.085980       1.066320
std     28867.657797      23.308133      48.515646       5.824816
min    100000.000000       0.000000       0.000000       0.000000
25%    124999.750000       0.000000       0.000000       0.000000
50%    149999.500000       0.000000       0.000000       0.000000
```

```
75%    174999.250000       8.000000       8.000000       0.000000
max    199999.000000     477.000000    1548.000000     610.000000

            Topic2_ec      Topic3_ic      Topic3_ec      Topic4_ic  \
count  100000.000000  100000.000000  100000.000000  100000.000000
mean        7.923940      19.105870       8.563590      15.600520
std        50.279646      42.710725      40.371399      31.981042
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.000000       0.000000       0.000000
50%         0.000000       0.000000       0.000000       8.000000
75%         0.000000      22.000000       1.000000      15.000000
max      1576.000000     631.000000    1506.000000     603.000000

            Topic4_ec      Topic5_ic  …   Topic177_ec  Topic178_ic  \
count  100000.000000  100000.000000  …  100000.000000     100000.0
mean       23.712400       4.927330  …       0.002190          0.0
std        54.356458      17.719046  …       0.501804          0.0
min         0.000000       0.000000  …       0.000000          0.0
25%         0.000000       0.000000  …       0.000000          0.0
50%         1.000000       0.000000  …       0.000000          0.0
75%        22.000000       0.000000  …       0.000000          0.0
max      1632.000000     512.000000  …     134.000000          0.0

       Topic178_ec  Topic179_ic  Topic179_ec  Topic180_ic  Topic180_ec  \
count     100000.0     100000.0     100000.0     100000.0     100000.0
mean           0.0          0.0          0.0          0.0          0.0
std            0.0          0.0          0.0          0.0          0.0
min            0.0          0.0          0.0          0.0          0.0
25%            0.0          0.0          0.0          0.0          0.0
50%            0.0          0.0          0.0          0.0          0.0
75%            0.0          0.0          0.0          0.0          0.0
max            0.0          0.0          0.0          0.0          0.0

       Topic181_ic  Topic181_ec         target
count     100000.0     100000.0  100000.000000
mean           0.0          0.0       0.015000
std            0.0          0.0       0.121553
min            0.0          0.0       0.000000
25%            0.0          0.0       0.000000
50%            0.0          0.0       0.000000
75%            0.0          0.0       0.000000
max            0.0          0.0       1.000000

[8 rows x 258 columns]
```

[644]:
```
df_verseny_public_train.columns
```

[4]:
```python
len(df_verseny_public_train.columns)
```

[4]: 258

[645]:
```python
len(df_verseny_public_train)
```

[645]: 100000

# 4 Remove missing values

[646]:
```python
df_verseny_public_train = df_verseny_public_train.dropna()
```

[647]:
```python
len(df_verseny_public_train)
```

[647]: 100000

this means that there are no missing values

# 5 Selecting columns with the highest variance in the training set

- sorrting values by variance value ascending
- selecting only first 100

[648]:
```python
df_verseny_public_train.var().sort_values(ascending=False)
y = df_verseny_public_train['target']
df_verseny_public_train = df_verseny_public_train[df_verseny_public_train.var().
↪sort_values(ascending=False).index[:100]]
```

# 6 PCA

- trying dimension reduction using PCA
- there are 250+ features

[649]:
```python
"""from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

X = df_verseny_public_train.drop(['target', 'cookie_id'], axis=1)
y = df_verseny_public_train['target']

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

pca = PCA(n_components=50)

X_pca = pca.fit_transform(X_scaled)
```

```
X_pca

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis')

plt.xlabel('First principal component')

plt.ylabel('Second principal component')

plt.show()"""
```

[649]: "from sklearn.decomposition import PCA\nfrom sklearn.preprocessing import StandardScaler\n\nX = df_verseny_public_train.drop(['target', 'cookie_id'], axis=1)\ny = df_verseny_public_train['target']\n\nscaler = StandardScaler()\n\nX_scaled = scaler.fit_transform(X)\n\npca = PCA(n_components=50)\n\nX_pca = pca.fit_transform(X_scaled)\n\nX_pca\n\nplt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis')\n\nplt.xlabel('First principal component')\n\nplt.ylabel('Second principal component')\n\nplt.show()"

# 7 Feature importance

- using feature importance to find the important features

[650]:
```
X = df_verseny_public_train.drop(['cookie_id'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

## 7.1 Decision Tree

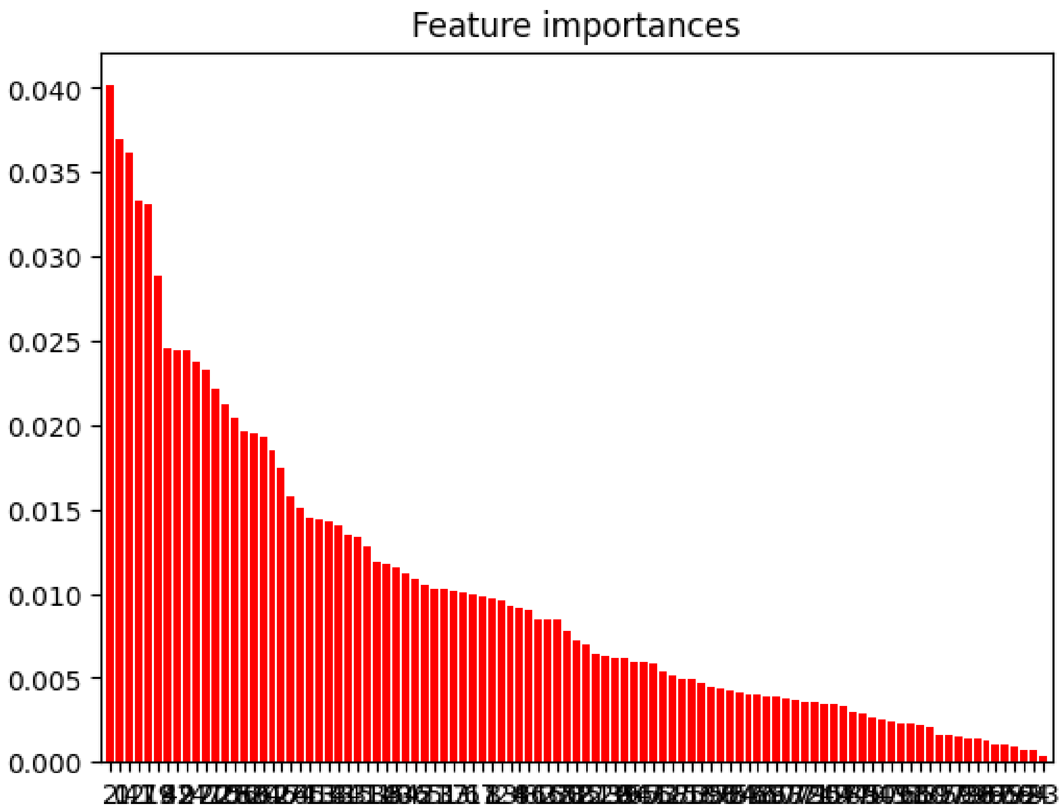
[651]:
```
clf = DecisionTreeClassifier()

clf.fit(X_train, y_train)

importances = clf.feature_importances_

indices = np.argsort(importances)[::-1]

print("Feature ranking:")
for f in range(X.shape[1]):
    print("%d. Feature %d (%f) %s" % (f + 1, indices[f],␣
↪importances[indices[f]], X.columns[indices[f]]))


plt.figure()
plt.title("Feature importances")
```

```
plt.bar(range(X.shape[1]), importances[indices],
       color="r", align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
```

```
Feature ranking:
1. Feature 2 (0.040136) Topic42_ec
2. Feature 0 (0.036941) Topic63_ec
3. Feature 12 (0.036203) Topic12_ec
4. Feature 41 (0.033347) Topic51_ec
5. Feature 7 (0.033043) Topic4_ec
6. Feature 19 (0.028929) Topic56_ec
7. Feature 8 (0.024541) Topic13_ec
8. Feature 42 (0.024462) Topic4_ic
9. Feature 9 (0.024455) Topic14_ec
10. Feature 27 (0.023779) Topic55_ic
11. Feature 47 (0.023310) Topic13_ic
12. Feature 22 (0.022138) Topic3_ic
13. Feature 10 (0.021189) Topic74_ec
14. Feature 28 (0.020392) Topic54_ec
15. Feature 56 (0.019589) Topic56_ic
16. Feature 58 (0.019578) Topic19_ic
17. Feature 20 (0.019245) Topic55_ec
18. Feature 62 (0.018520) Topic12_ic
19. Feature 46 (0.017497) Topic14_ic
20. Feature 26 (0.015786) Topic24_ec
21. Feature 74 (0.015090) Topic24_ic
22. Feature 45 (0.014526) Topic9_ec
23. Feature 13 (0.014343) Topic65_ec
24. Feature 14 (0.014324) Topic1_ec
25. Feature 33 (0.014028) Topic41_ec
26. Feature 34 (0.013468) Topic136_ec
27. Feature 15 (0.013343) Topic16_ec
28. Feature 31 (0.012801) Topic25_ec
29. Feature 11 (0.011857) Topic2_ec
30. Feature 38 (0.011807) Topic137_ec
31. Feature 40 (0.011535) Topic99_ec
32. Feature 54 (0.011150) Topic9_ic
33. Feature 36 (0.010821) Topic97_ec
34. Feature 5 (0.010519) Topic8_ec
35. Feature 21 (0.010329) Topic53_ec
36. Feature 51 (0.010316) Topic15_ic
37. Feature 37 (0.010145) Topic10_ec
38. Feature 3 (0.010039) Topic33_ec
39. Feature 6 (0.010007) Topic19_ec
40. Feature 17 (0.009875) Topic89_ec
```

41. Feature 1 (0.009762) Topic52_ec
42. Feature 83 (0.009636) Topic10_ic
43. Feature 24 (0.009226) Topic3_ec
44. Feature 4 (0.009194) Topic5_ec
45. Feature 86 (0.009082) Topic8_ic
46. Feature 82 (0.008515) Topic41_ic
47. Feature 16 (0.008495) Topic35_ec
48. Feature 32 (0.008478) Topic20_ec
49. Feature 30 (0.007745) Topic91_ec
50. Feature 52 (0.007186) Topic86_ec
51. Feature 85 (0.006945) Topic5_ic
52. Feature 81 (0.006418) Topic34_ec
53. Feature 23 (0.006308) Topic28_ec
54. Feature 29 (0.006169) Topic40_ec
55. Feature 39 (0.006155) Topic15_ec
56. Feature 84 (0.006009) Topic88_ec
57. Feature 65 (0.005940) Topic1_ic
58. Feature 61 (0.005804) Topic72_ec
59. Feature 73 (0.005432) Topic131_ec
60. Feature 68 (0.005119) Topic61_ec
61. Feature 25 (0.004972) Topic27_ec
62. Feature 75 (0.004931) Topic82_ec
63. Feature 18 (0.004723) Topic23_ec
64. Feature 35 (0.004464) Topic78_ec
65. Feature 98 (0.004358) Topic66_ec
66. Feature 78 (0.004294) Topic108_ec
67. Feature 64 (0.004186) Topic62_ec
68. Feature 53 (0.004071) Topic68_ec
69. Feature 48 (0.004052) Topic87_ec
70. Feature 69 (0.003948) Topic26_ec
71. Feature 60 (0.003909) Topic117_ec
72. Feature 63 (0.003832) Topic67_ec
73. Feature 77 (0.003714) Topic107_ec
74. Feature 72 (0.003563) Topic134_ec
75. Feature 71 (0.003517) Topic130_ec
76. Feature 90 (0.003508) Topic127_ec
77. Feature 49 (0.003422) Topic85_ec
78. Feature 57 (0.003355) Topic36_ec
79. Feature 44 (0.002950) Topic60_ec
80. Feature 89 (0.002884) Topic133_ec
81. Feature 70 (0.002665) Topic73_ec
82. Feature 50 (0.002556) Topic92_ec
83. Feature 43 (0.002391) Topic37_ec
84. Feature 76 (0.002329) Topic57_ec
85. Feature 91 (0.002310) Topic70_ec
86. Feature 55 (0.002248) Topic98_ec
87. Feature 80 (0.002038) Topic64_ec
88. Feature 87 (0.001580) Topic128_ec

```
89. Feature 97 (0.001576) Topic58_ec
90. Feature 59 (0.001559) Topic155_ec
91. Feature 79 (0.001440) Topic100_ec
92. Feature 88 (0.001440) Topic104_ec
93. Feature 96 (0.001294) Topic103_ec
94. Feature 67 (0.001098) Topic106_ec
95. Feature 95 (0.001090) Topic105_ec
96. Feature 66 (0.000908) Topic93_ec
97. Feature 92 (0.000723) Topic71_ec
98. Feature 94 (0.000719) Topic140_ec
99. Feature 93 (0.000365) Topic83_ec
```




### 7.2 Random forest

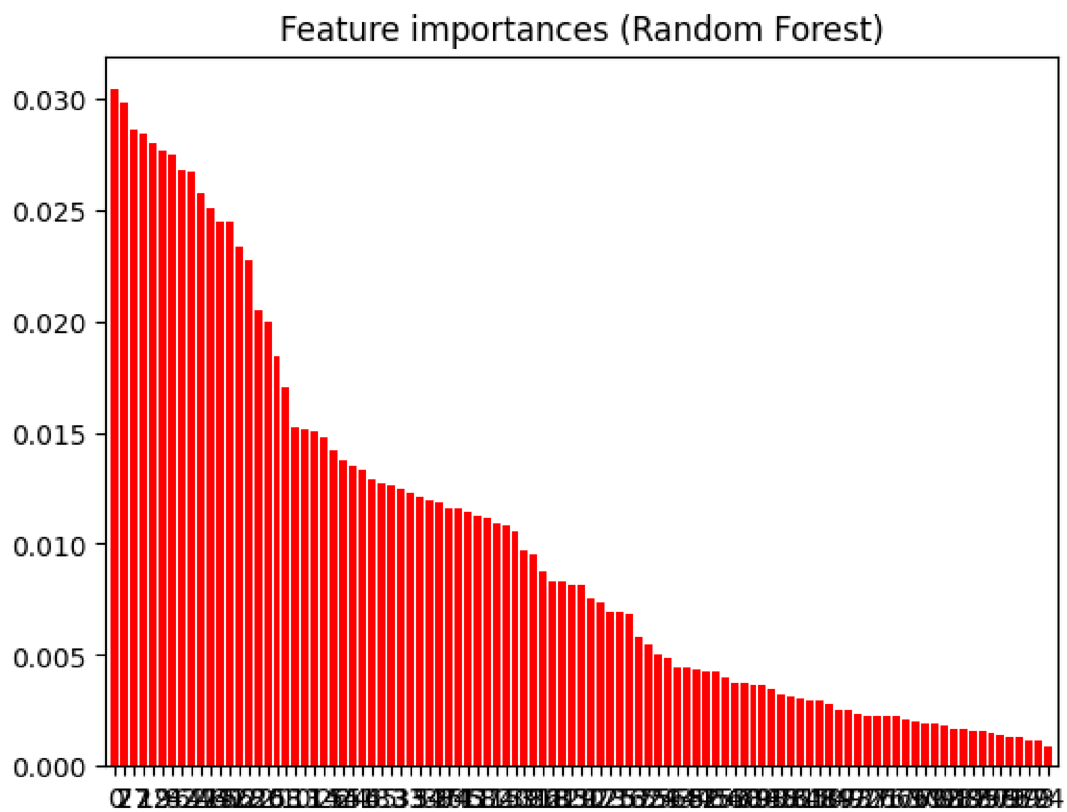
[652]:
```
clf_rf = RandomForestClassifier()

clf_rf.fit(X_train, y_train)

importances_rf = clf_rf.feature_importances_
indices_rf = np.argsort(importances_rf)[::-1]
```

```
print("Feature ranking:")
for f in range(X.shape[1]):
    print("%d. Feature %d (%f) %s" % (f + 1, indices_rf[f],␣
 ↪importances_rf[indices_rf[f]], X.columns[indices_rf[f]]))

plt.figure()
plt.title("Feature importances (Random Forest)")
plt.bar(range(X.shape[1]), importances_rf[indices_rf],
       color="r", align="center")
plt.xticks(range(X.shape[1]), indices_rf)
plt.xlim([-1, X.shape[1]])
plt.show()
```

```
Feature ranking:
1. Feature 0 (0.030399) Topic63_ec
2. Feature 7 (0.029850) Topic4_ec
3. Feature 27 (0.028577) Topic55_ic
4. Feature 2 (0.028430) Topic42_ec
5. Feature 12 (0.027965) Topic12_ec
6. Feature 9 (0.027683) Topic14_ec
7. Feature 8 (0.027458) Topic13_ec
8. Feature 42 (0.026783) Topic4_ic
9. Feature 62 (0.026677) Topic12_ic
10. Feature 47 (0.025747) Topic13_ic
11. Feature 46 (0.025063) Topic14_ic
12. Feature 19 (0.024500) Topic56_ec
13. Feature 41 (0.024496) Topic51_ec
14. Feature 56 (0.023310) Topic56_ic
15. Feature 28 (0.022780) Topic54_ec
16. Feature 22 (0.020458) Topic3_ic
17. Feature 20 (0.019946) Topic55_ec
18. Feature 6 (0.018387) Topic19_ec
19. Feature 58 (0.017079) Topic19_ic
20. Feature 10 (0.015261) Topic74_ec
21. Feature 13 (0.015128) Topic65_ec
22. Feature 1 (0.015066) Topic52_ec
23. Feature 14 (0.014834) Topic1_ec
24. Feature 26 (0.014146) Topic24_ec
25. Feature 54 (0.013734) Topic9_ic
26. Feature 24 (0.013479) Topic3_ec
27. Feature 51 (0.013336) Topic15_ic
28. Feature 4 (0.012869) Topic5_ec
29. Feature 65 (0.012689) Topic1_ic
30. Feature 83 (0.012630) Topic10_ic
31. Feature 3 (0.012499) Topic33_ec
32. Feature 33 (0.012303) Topic41_ec
33. Feature 5 (0.012089) Topic8_ec
```

34. Feature 34 (0.011931) Topic136_ec
35. Feature 38 (0.011829) Topic137_ec
36. Feature 36 (0.011573) Topic97_ec
37. Feature 74 (0.011568) Topic24_ic
38. Feature 45 (0.011438) Topic9_ec
39. Feature 11 (0.011230) Topic2_ec
40. Feature 37 (0.011194) Topic10_ec
41. Feature 82 (0.010944) Topic41_ic
42. Feature 15 (0.010792) Topic16_ec
43. Feature 40 (0.010573) Topic99_ec
44. Feature 39 (0.009677) Topic15_ec
45. Feature 31 (0.009548) Topic25_ec
46. Feature 86 (0.008752) Topic8_ic
47. Feature 18 (0.008336) Topic23_ec
48. Feature 21 (0.008280) Topic53_ec
49. Feature 85 (0.008139) Topic5_ic
50. Feature 29 (0.008103) Topic40_ec
51. Feature 30 (0.007560) Topic91_ec
52. Feature 17 (0.007367) Topic89_ec
53. Feature 23 (0.006948) Topic28_ec
54. Feature 25 (0.006932) Topic27_ec
55. Feature 16 (0.006826) Topic35_ec
56. Feature 57 (0.005800) Topic36_ec
57. Feature 32 (0.005425) Topic20_ec
58. Feature 55 (0.004997) Topic98_ec
59. Feature 59 (0.004827) Topic155_ec
60. Feature 43 (0.004461) Topic37_ec
61. Feature 68 (0.004438) Topic61_ec
62. Feature 44 (0.004304) Topic60_ec
63. Feature 52 (0.004291) Topic86_ec
64. Feature 75 (0.004284) Topic82_ec
65. Feature 64 (0.004022) Topic62_ec
66. Feature 53 (0.003747) Topic68_ec
67. Feature 60 (0.003696) Topic117_ec
68. Feature 69 (0.003639) Topic26_ec
69. Feature 84 (0.003616) Topic88_ec
70. Feature 98 (0.003497) Topic66_ec
71. Feature 35 (0.003195) Topic78_ec
72. Feature 78 (0.003144) Topic108_ec
73. Feature 81 (0.003026) Topic34_ec
74. Feature 61 (0.002966) Topic72_ec
75. Feature 48 (0.002934) Topic87_ec
76. Feature 50 (0.002790) Topic92_ec
77. Feature 89 (0.002558) Topic133_ec
78. Feature 49 (0.002544) Topic85_ec
79. Feature 73 (0.002399) Topic131_ec
80. Feature 72 (0.002306) Topic134_ec
81. Feature 77 (0.002300) Topic107_ec

```
82. Feature 71 (0.002286) Topic130_ec
83. Feature 66 (0.002231) Topic93_ec
84. Feature 63 (0.002100) Topic67_ec
85. Feature 76 (0.002024) Topic57_ec
86. Feature 90 (0.001963) Topic127_ec
87. Feature 70 (0.001957) Topic73_ec
88. Feature 92 (0.001801) Topic71_ec
89. Feature 91 (0.001698) Topic70_ec
90. Feature 88 (0.001695) Topic104_ec
91. Feature 87 (0.001585) Topic128_ec
92. Feature 95 (0.001557) Topic105_ec
93. Feature 80 (0.001478) Topic64_ec
94. Feature 96 (0.001389) Topic103_ec
95. Feature 79 (0.001337) Topic100_ec
96. Feature 97 (0.001335) Topic58_ec
97. Feature 67 (0.001165) Topic106_ec
98. Feature 93 (0.001124) Topic83_ec
99. Feature 94 (0.000879) Topic140_ec
```




[653]:
```
dt_feature_importances = pd.DataFrame({'Feature': indices, 'Importance_DT':␣
↪importances[indices]})
```

```
rf_feature_importances = pd.DataFrame({'Feature': indices_rf, 'Importance_RF':␣
↪importances_rf[indices_rf]})

merged_feature_importances = pd.merge(dt_feature_importances,␣
↪rf_feature_importances, on='Feature')

print("Merged Feature Importances:")
print(merged_feature_importances)
```

```
Merged Feature Importances:
    Feature  Importance_DT  Importance_RF
0         2       0.040136       0.028430
1         0       0.036941       0.030399
2        12       0.036203       0.027965
3        41       0.033347       0.024496
4         7       0.033043       0.029850
..      …            …              …
94       95       0.001090       0.001557
95       66       0.000908       0.002231
96       92       0.000723       0.001801
97       94       0.000719       0.000879
98       93       0.000365       0.001124

[99 rows x 3 columns]
```

[654]:
```
percentile_threshold = 0.5

importance_threshold_dt = merged_feature_importances['Importance_DT'].
↪quantile(percentile_threshold)
importance_threshold_rf = merged_feature_importances['Importance_RF'].
↪quantile(percentile_threshold)

print("Threshold value based on the top", int(percentile_threshold * 100),␣
↪"percentileDT:", importance_threshold_dt, "percentileRF:",␣
↪importance_threshold_rf)
```

```
Threshold value based on the top 50 percentileDT: 0.007185629188621481
percentileRF: 0.008103335083112865
```

[655]:
```
merged_feature_importances =␣
↪merged_feature_importances[(merged_feature_importances['Importance_DT'] >␣
↪importance_threshold_dt) & (merged_feature_importances['Importance_RF'] >␣
↪importance_threshold_rf)]
print("Merged Feature Importances:")
print(merged_feature_importances)
```

```
Merged Feature Importances:
```

| | Feature | Importance_DT | Importance_RF |
|---|---|---|---|
| 0 | 2 | 0.040136 | 0.028430 |
| 1 | 0 | 0.036941 | 0.030399 |
| 2 | 12 | 0.036203 | 0.027965 |
| 3 | 41 | 0.033347 | 0.024496 |
| 4 | 7 | 0.033043 | 0.029850 |
| 5 | 19 | 0.028929 | 0.024500 |
| 6 | 8 | 0.024541 | 0.027458 |
| 7 | 42 | 0.024462 | 0.026783 |
| 8 | 9 | 0.024455 | 0.027683 |
| 9 | 27 | 0.023779 | 0.028577 |
| 10 | 47 | 0.023310 | 0.025747 |
| 11 | 22 | 0.022138 | 0.020458 |
| 12 | 10 | 0.021189 | 0.015261 |
| 13 | 28 | 0.020392 | 0.022780 |
| 14 | 56 | 0.019589 | 0.023310 |
| 15 | 58 | 0.019578 | 0.017079 |
| 16 | 20 | 0.019245 | 0.019946 |
| 17 | 62 | 0.018520 | 0.026677 |
| 18 | 46 | 0.017497 | 0.025063 |
| 19 | 26 | 0.015786 | 0.014146 |
| 20 | 74 | 0.015090 | 0.011568 |
| 21 | 45 | 0.014526 | 0.011438 |
| 22 | 13 | 0.014343 | 0.015128 |
| 23 | 14 | 0.014324 | 0.014834 |
| 24 | 33 | 0.014028 | 0.012303 |
| 25 | 34 | 0.013468 | 0.011931 |
| 26 | 15 | 0.013343 | 0.010792 |
| 27 | 31 | 0.012801 | 0.009548 |
| 28 | 11 | 0.011857 | 0.011230 |
| 29 | 38 | 0.011807 | 0.011829 |
| 30 | 40 | 0.011535 | 0.010573 |
| 31 | 54 | 0.011150 | 0.013734 |
| 32 | 36 | 0.010821 | 0.011573 |
| 33 | 5 | 0.010519 | 0.012089 |
| 34 | 21 | 0.010329 | 0.008280 |
| 35 | 51 | 0.010316 | 0.013336 |
| 36 | 37 | 0.010145 | 0.011194 |
| 37 | 3 | 0.010039 | 0.012499 |
| 38 | 6 | 0.010007 | 0.018387 |
| 40 | 1 | 0.009762 | 0.015066 |
| 41 | 83 | 0.009636 | 0.012630 |
| 42 | 24 | 0.009226 | 0.013479 |
| 43 | 4 | 0.009194 | 0.012869 |
| 44 | 86 | 0.009082 | 0.008752 |
| 45 | 82 | 0.008515 | 0.010944 |

### 7.3 Dropping the feature which are not in the percentile

[656]:
```
X = X.drop(X.columns.difference(X.
↪columns[merged_feature_importances['Feature']]), axis=1)
```

[657]:
```
column_names = list(X.columns)
```

[658]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

## 8 PCA train test dataset

[659]:
```
#X_train_pca, X_test_pca, y_train_pca, y_test_pca = train_test_split(X_pca, y,␣
↪test_size=0.2, random_state=42)
```

## 9 Modell building - Random forest and AdaBoost with Voting

### 9.1 Using 80 percentil dataset

[660]:
```
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,␣
↪VotingClassifier
from sklearn.metrics import accuracy_score

base_rf = RandomForestClassifier(
    n_jobs=-1,
    n_estimators=150,
    max_depth=12,
    random_state=42,
    criterion='entropy',
    max_features='log2',
    oob_score=True,
    verbose=1
)

base_ada = AdaBoostClassifier(
    n_estimators=150,
    random_state=42,
    learning_rate=1.5
)

voting_clf = VotingClassifier(
    estimators=[
        ('rf', base_rf),
        ('ada', base_ada),
    ],
    voting='soft',
```

```
    verbose=True
)

voting_clf.fit(X_train, y_train)

y_pred_voting = voting_clf.predict(X_test)

accuracy_voting = accuracy_score(y_test, y_pred_voting)
print("Accuracy (Voting Classifier):", accuracy_voting)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 12 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:    0.8s
[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed:    3.1s finished
```

```
[Voting] … (1 of 2) Processing rf, total=   4.5s
```

```
/Users/kissdanielmark/Documents/01.Iskola/MSc/3/Customer
Analytics/Competition/CustomerAnalytics_Competition/.venv/lib/python3.9/site-
packages/sklearn/ensemble/_weight_boosting.py:519: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
```

```
[Voting] … (2 of 2) Processing ada, total=  10.6s
```

```
[Parallel(n_jobs=12)]: Using backend ThreadingBackend with 12 concurrent
workers.
[Parallel(n_jobs=12)]: Done  26 tasks      | elapsed:    0.0s
[Parallel(n_jobs=12)]: Done 150 out of 150 | elapsed:    0.1s finished
```

```
Accuracy (Voting Classifier): 0.98525
```

### 9.2 Using PCA

[661]:
```
"""from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,␣
 ↪VotingClassifier
from sklearn.metrics import accuracy_score

base_rf = RandomForestClassifier(n_jobs=-1, n_estimators=150, max_depth=12,␣
 ↪random_state=42, criterion='entropy', max_features='log2', oob_score=True,␣
 ↪verbose=1)
base_ada = AdaBoostClassifier(n_estimators=150, random_state=42,␣
 ↪learning_rate=1.5)

voting_clf = VotingClassifier(estimators=[('rf', base_rf), ('ada', base_ada)],␣
 ↪voting='soft', verbose=True)

voting_clf.fit(X_train_pca, y_train_pca)
```

```
y_pred_voting = voting_clf.predict(X_test_pca)

accuracy_voting = accuracy_score(y_test_pca, y_pred_voting)
print("Accuracy (Voting Classifier):", accuracy_voting)"""
```

[661]:
```
'from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
VotingClassifier\nfrom sklearn.metrics import accuracy_score\n\nbase_rf =
RandomForestClassifier(n_jobs=-1, n_estimators=150, max_depth=12,
random_state=42, criterion=\'entropy\', max_features=\'log2\', oob_score=True,
verbose=1)\nbase_ada = AdaBoostClassifier(n_estimators=150, random_state=42,
learning_rate=1.5)\n\nvoting_clf = VotingClassifier(estimators=[(\'rf\',
base_rf), (\'ada\', base_ada)], voting=\'soft\',
verbose=True)\n\nvoting_clf.fit(X_train_pca, y_train_pca)\n\ny_pred_voting =
voting_clf.predict(X_test_pca)\n\naccuracy_voting = accuracy_score(y_test_pca,
y_pred_voting)\nprint("Accuracy (Voting Classifier):", accuracy_voting)'
```

# 10 Loading test set

[662]:
```
df_verseny_public_test = pd.read_csv('data/verseny_public_test.csv', sep=',',␣
↪low_memory=False)
```

# 11 Evaluation

## 11.1 Using 80 percentil

[663]:
```
X_test_test = df_verseny_public_test.drop(['cookie_id'], axis=1)
X_test_test = X_test_test[column_names]
```

[664]:
```
y_pred_rf = voting_clf.predict_proba(X_test_test)[:, 1]

df_verseny_public_test['target'] = y_pred_rf

df_verseny_public_test = df_verseny_public_test[['cookie_id', 'target']]

df_verseny_public_test.to_csv('data/
↪prediction_random_forest_w_adaboost_voting_80_entropy.csv', index=False)
```

```
[Parallel(n_jobs=12)]: Using backend ThreadingBackend with 12 concurrent
workers.
[Parallel(n_jobs=12)]: Done  26 tasks      | elapsed:    0.4s
[Parallel(n_jobs=12)]: Done 150 out of 150 | elapsed:    0.6s finished
```

### 11.2 Using PCA

[665]:
```
"""from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

X_test_test = df_verseny_public_test.drop(['cookie_id'], axis=1)

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X_test_test)

pca = PCA(n_components=50)

X_pca = pca.fit_transform(X_scaled)

y_pred_rf = voting_clf.predict_proba(X_pca)[:, 1]

df_verseny_public_test['target'] = y_pred_rf

df_verseny_public_test = df_verseny_public_test[['cookie_id', 'target']]

df_verseny_public_test.to_csv('data/
↪prediction_random_forest_w_adaboost_voting_PCA.csv', index=False)"""
```

[665]:
```
"from sklearn.decomposition import PCA\nfrom sklearn.preprocessing import
StandardScaler\n\nX_test_test = df_verseny_public_test.drop(['cookie_id'],
axis=1)\n\nscaler = StandardScaler()\n\nX_scaled =
scaler.fit_transform(X_test_test)\n\npca = PCA(n_components=50)\n\nX_pca =
pca.fit_transform(X_scaled)\n\ny_pred_rf = voting_clf.predict_proba(X_pca)[:,
1]\n\ndf_verseny_public_test['target'] = y_pred_rf\n\ndf_verseny_public_test =
df_verseny_public_test[['cookie_id', 'target']]\n\ndf_verseny_public_test.to_csv
('data/prediction_random_forest_w_adaboost_voting_PCA.csv', index=False)"
```