Analiza skupa podataka *Online Shoppers Purchasing Intention*

Olivera Popović

Jun 19, 2019

Sadržaj

Analiza skupa podataka *Online Shoppers Purchasing Intention*

1. Uvod

Online Shoppers Purchasing Intention skup podataka sadrži informacije o aktivnostima korisnika na internetu, a podaci su prikupljeni sa sajta za maloprodaju. Konstruisan od strane Google Analytics servisa za sakupljanje statističkih podataka o aktivnostima korisnika na internetu, skup se sastoji od 12330 pristupa korisnika. Skup je pažljivo formiran tako da svaki pristup odgovara različitom korisniku u periodu od jedne godine, kako bi se izbegla sklonost ka specifičnom profilu korisnika, danu ili periodu.

2. Opis skupa podataka

Skup podataka *Online Shoppers Purchasing Intention* se sastoji od 10 numeričkih i 8 kategoričkih atributa. Njihov detaljniji opis sledi u nastavku.

U tabeli 1 dat je kratak opis numeričkih atributa, a **u tabeli 3** je prikazan kratak opis kategoričkih atributa.

Tabela 1: Numerički atributi

Naziv atributa	Opis atributa	Min	Max	Std. dev
Administrative	Broj posećenih veb strana o upravljanju profilom	0	27	3.32
Administrative Duration	Vreme (u sekundama) provedeno na veb stranama o upravljanju profilom	0	3398	176.70
Informational	Broj posećenih veb strana sa informacijama o veb sajtu, komunikaciji i adresi sajta za kupovinu	0	24	1.26
Informational Duration	Vreme (u sekundama) provedeno na veb stranama za informacije	0	2549	140.64
Product Related	Broj posećenih veb strana vezanih za proizvode	0	705	44.45
Product Related Duration	Vreme (u sekundama) provedeno na veb stranama vezanim za proizvode	0	63973	1912.25
Bounce Rate	Procenat korisnika koji nakon ulaska na veb sajt izađu bez pokretanja drugih zahteva ka serveru	0	0.2	0.04
Exit Rate	Procenat koliko je puta veb strana bila poslednja u jednom pristupu korisnika internetu, u odnosu na ukupan broj pregleda	0	0.2	0.05
Special Day	Pokazuje koliko je vreme posete veb sajtu blizu nekog specijalnog dana u godini, u kojima je veća verovatnoća da se uspešno izvrši transakcija	0	1.0	0.19

Naziv atributa	Opis atributa	Min	Max	Std. dev
Page Value	Prosek vrednosti veb strana koje je korisnik posetio	0	361	18.55

Primer kako se određuje tip veb strane dat je **u tabeli 2**.

Tabela 2: Određivanje tipa veb strane

Page Type	URL
Administrative	/?login
Administrative	/?logout
Administrative	/customer
Administrative	/passwordrecovery
Product Related	/search
Product Related	/c
Product Related	/cart
Informational	/Catalog
Informational	/contactus
Informational	/stores

Tabela 3: Kategorički atributi

Naziv atributa	Opis atributa	Broj kategoričkih vrednosti
Operating Systems	Operativni sistem korisnika	8
Browser	Veb pregledač korisnika	13
Region	Geografski region iz kog se korisnik prijavio	9
Traffic Type	Izvor, odakle je korisnik pristupio veb sajtu	20
Visitor Type	Tip korisnika koji može biti: “New Visitor”, “Returning Visitor” i “Other”	3
Weekend	Indikator da li je datum posete vikend ili nije	2
Month	Mesec u kom je korisnik pristupio veb sajtu	12
Revenue	Oznaka klase koja pokazuje da li je poseta bila završena transakcijom ili ne	2

U nastavku sledi programski kod analize skupa i odgovarajući rezultat izvršavanja. Biće korišćen programski jezik Python, kao i biblioteke: Numpy, Pandas, Seaborn, Matplotlib i Sklearn.

Programski kod:

```
import numpy as np
import pandas as pd
import seaborn as sns
```

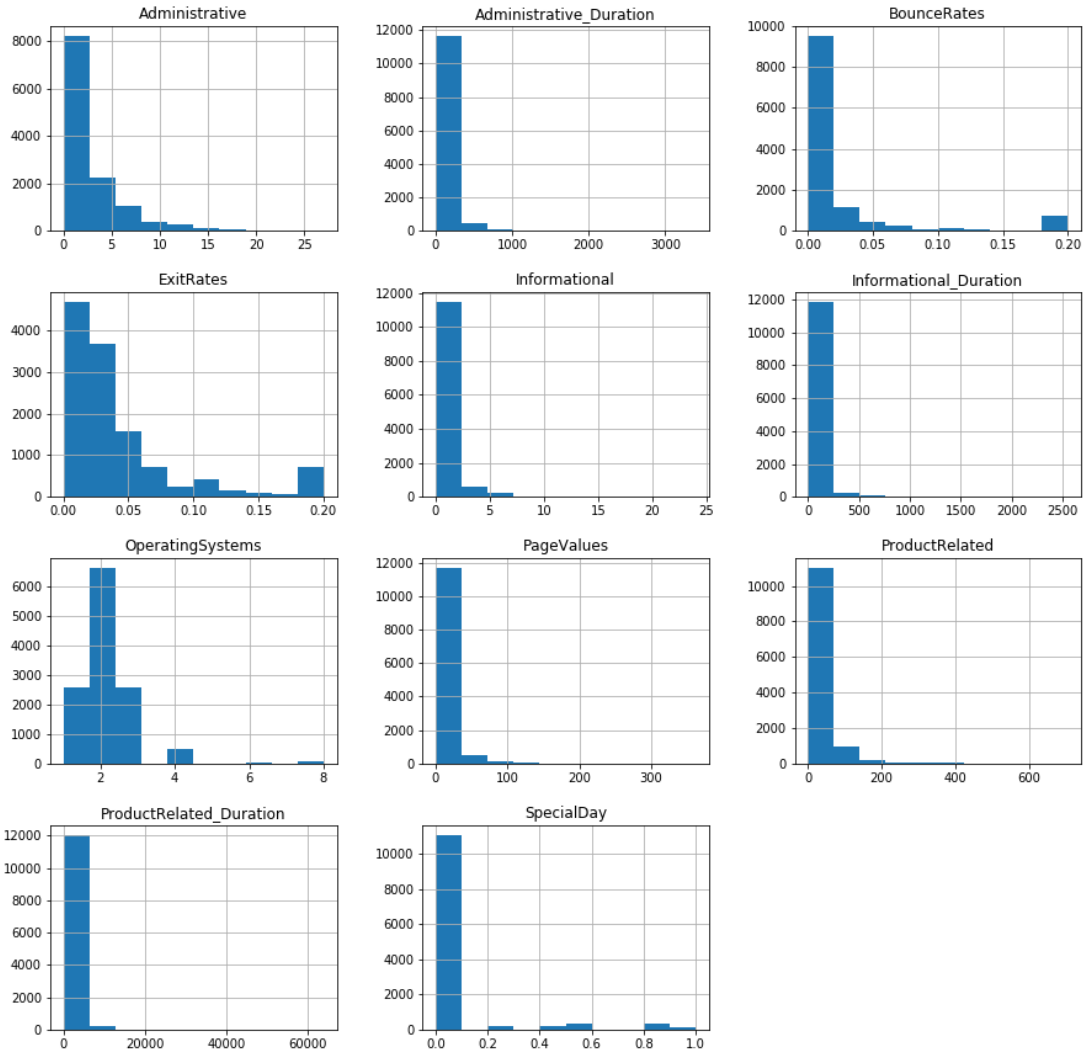
```
from matplotlib import pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn import preprocessing
from sklearn.decomposition import PCA
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from imblearn.over_sampling import SMOTE
import xgboost as xgb

# Učitavamo podatke
df_original = pd.read_csv('online_shoppers_intention.csv')
print(df_original.sample(10), '\n')
print(df_original.describe(), '\n')

print('Atributi:')
for c in df_original.columns:
    print(c)

print()
print(f'Broj instanci: {df.shape[0]}\nBroj atributa: {df.shape[1]}')
```

Rezultat izvršavanja:

```
       Administrative  Administrative_Duration  ...  Weekend  Revenue
10842               0                    0.000  ...    False    False
7804                6                  132.025  ...    False    False
8540                4                   28.500  ...     True    False
7346                0                    0.000  ...    False    False
2014                2                   36.000  ...    False    False
5731                0                    0.000  ...    False     True
1230                0                    0.000  ...    False    False
12025               0                    0.000  ...     True    False
9468                0                    0.000  ...    False    False
1387                3                   63.000  ...     True    False

[10 rows x 18 columns]

       Administrative  Administrative_Duration  ...        Region   TrafficType
count    12330.000000             12330.000000  ...  12330.000000  12330.000000
mean         2.315166                80.818611  ...      3.147364      4.069586
std          3.321784               176.779107  ...      2.401591      4.025169
min          0.000000                 0.000000  ...      1.000000      1.000000
25%          0.000000                 0.000000  ...      1.000000      2.000000
50%          1.000000                 7.500000  ...      3.000000      2.000000
75%          4.000000                93.256250  ...      4.000000      4.000000
max         27.000000              3398.750000  ...      9.000000     20.000000

[8 rows x 14 columns]


Broj instanci: 12330
Broj atributa: 18
```

Iz rezultata izvršavanja vidi se da zaista u skupu ima 12330 instanci i 18 atributa. Takođe, vidi se da su podaci retki i da su atributi različito skalirani, što će biti rešeno u fazi preprocesiranja podataka.

Raspodela nekih atributa u vidu histograma **prikazana je na slici 1**.



Slika 1: Histogrami raspodela nekih atributa

Ciljna promenljiva u procesu klasifikacije biće *Revenue*, odnosno da li je korisnik kupio neki proizvod ili nije, jer je to informacija koju želimo da dobijemo nakon istraživanja.

Programski kod:

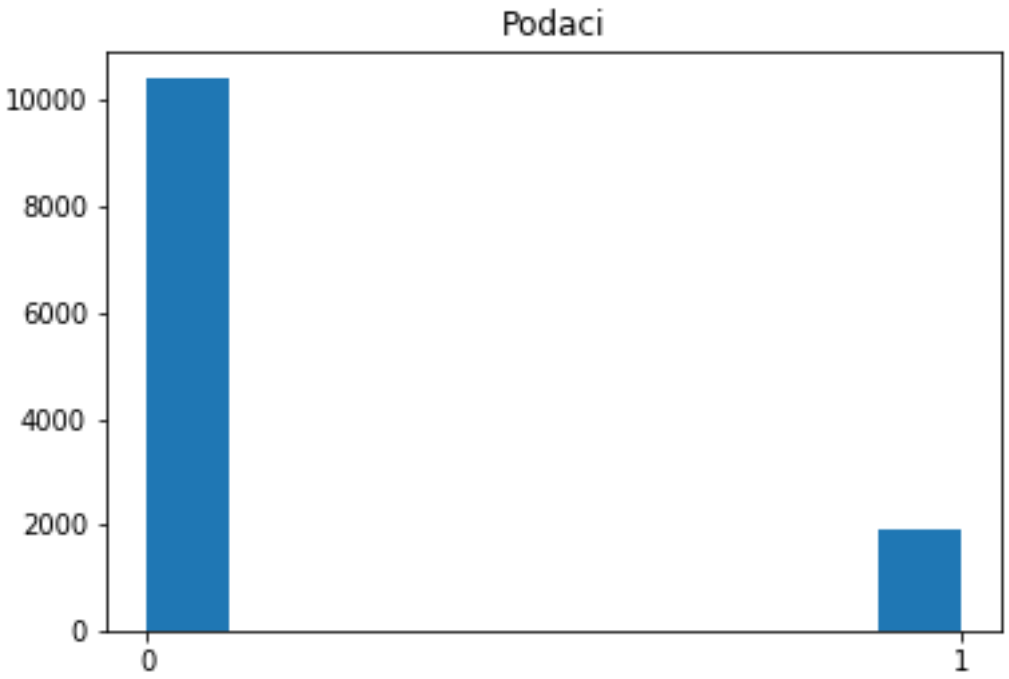
```
all_revenue = df_original['Revenue']
revenue = df_original['Revenue'].unique()
n_classes = revenue.shape[0]
```

```
print(f'Broj klasa: {n_classes}\n')

print('Klase:')
for c in revenue:
    print(c)

print()
print('False: {}\nTrue: {}'.format(np.sum(all_revenue == False), np.sum(all_revenue == True)))
```

Rezultat izvršavanja:

```
Broj klasa: 2

Klase:
False
True

False: 10422
True: 1908
```

Može se primetiti da klase nisu balansirane, što će predstavljati problem u procesu klasifikacije. To je prikazano i sledećim histogramom **koji je ilustrovan na slici 2**.



Slika 2: Histogram zastupljenosti klasa

3. Preprocesiranje podataka

U ovom odeljku će biti opisani sprovedeni koraci preprocesiranja nad podacima.

3.1 Rad sa nedostajućim vrednostima

Potrebno je proveriti da li u podacima postoje nedostajuće vrednosti i ako postoje treba ih na odgovarajući način ukloniti, odnosno zameniti.

Programski kod:

```
print('Nedostajuce vrednosti u podacima:\n')
print(df_original.isnull().sum())
```

Rezultat izvršavanja:

```
Nedostajuce vrednosti u podacima:

Administrative             0
Administrative_Duration    0
Informational              0
Informational_Duration     0
ProductRelated             0
ProductRelated_Duration    0
BounceRates                0
ExitRates                  0
PageValues                 0
SpecialDay                 0
Month                      0
OperatingSystems           0
Browser                    0
Region                     0
TrafficType                0
VisitorType                0
Weekend                    0
Revenue                    0
dtype: int64
```

Iz rezultata izvršavanja može se primetiti da među podacima nema nedostajućih vrednosti.

3.2 Izbor atributa

U ovom odeljku biće opisan način za početni odabir atributa. Ovo će biti samo početni odabir jer će u narednom odeljku atributi biti ponovo birani, kao pokušaj da se unapredi rezultat algoritama.

Da bismo odabrali atribute potrebna nam je matrica korelacije. Matricom korelacije dobićemo informacije o tome koliko atributi utiču jedni na druge, odnosno koliko su korelirani. **Na slici 3** tamnije plavom bojom su prikazani visoko korelirani atributi. Prateći tamno plavu boju može se uočiti da atributi *Exit Rates* i *Bounce Rates* veoma utiču jedan na drugi, te da jedan možemo isključiti. Slično je i sa *Product Related* i *Product Related Duration* atributima.

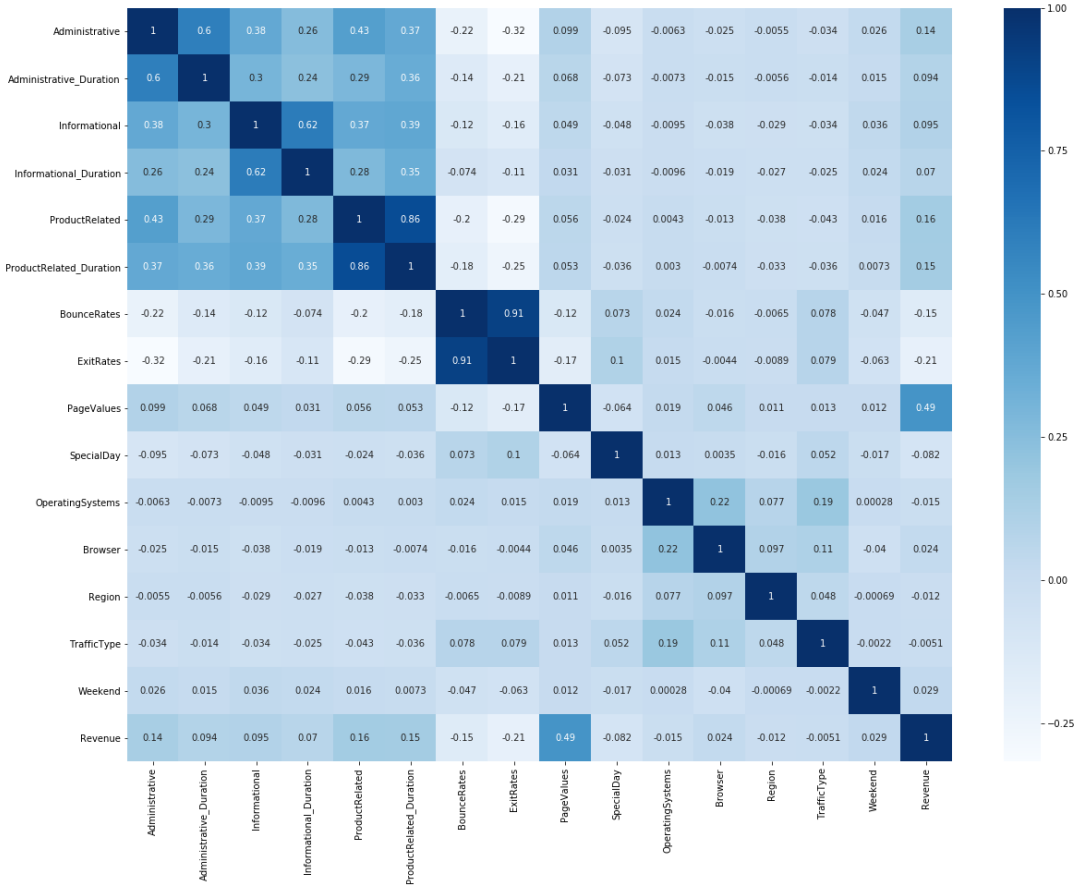
Slika 3 je rezultat izvršavanja sledećeg koda:

Programski kod:

```
plt.figure(figsize = (20,15))
correlation_matrix = df_original.corr()
sns.heatmap(correlation_matrix, annot = True, cmap = plt.cm.Blues)
```

3.3 Transformacija kategoričkih atributa

Da bi se mogla vršiti izračunavanja u algoritmima klasifikacije potrebno je da atributi budu numeričkog tipa. U obrađivanom skupu postoji nekoliko takvih atributa. Takođe, iako numerički, atributi mogu uzimati vrednosti iz nekog diskretnog skupa, što bi takođe moglo dovesti do problema pri izračunavanju ako neki atribut ima veći broj kao



Slika 3: Matrica korelacije

oznaku a ne predstavlja nužno veću vrednost. Primer takvog problema bio bi na primer atribut *Operating Systems* koji uzima vrednosti iz skupa $\{1, 2, \ldots, 8\}$, iako ne znači da je neki operativni sistem "veći" od nekog drugog. Zbog tih problema izvršićemo takozvanu **binarizaciju**. Ako kategorički atribut ima n različitih vrednosti formira se n novih, različitih binarnih atributa. Svaki binarni atribut odgovaraće jednoj mogućoj vrednosti kategoričkog atributa. U jednom redu tačno jedan od n atributa imaće vrednost 1, dok će ostali imati vrednost 0.

Binarni kategoričke atribute koji uzimaju logičke vrednosti *True* ili *False* biće zamenjeni sa 0 i 1, jer su binarni podaci specijalni slučaj i kategoričkih i numeričkih atributa.

Programski kod:

```
df = pd.get_dummies(df_original, columns = ['OperatingSystems', 'Browser', 'Region',
'TrafficType', 'VisitorType', 'Month', 'Weekend'])

revenue = df_original['Revenue'].unique()
n_classes = revenue.shape[0]
changes = dict( zip(revenue, range(n_classes)))

df = df.replace(changes)
```

Primer atributa koji su dobijeni od atributa *Operating Systems*:

Programski kod:

```
print('Novi atributi:')
print(list(df.columns[9:17]))

print()
print('Novi izgled skupa:')
print(df.head())
```

Rezultat izvršavanja:

```
Novi atributi:
['OperatingSystems_1', 'OperatingSystems_2', 'OperatingSystems_3', 'OperatingSystems_4',
'OperatingSystems_5', 'OperatingSystems_6', 'OperatingSystems_7', 'OperatingSystems_8']

Novi izgled skupa:
   Administrative  Administrative_Duration  ...  Weekend_False  Weekend_True
0               0                      0.0  ...              1             0
1               0                      0.0  ...              1             0
2               0                      0.0  ...              1             0
3               0                      0.0  ...              1             0
4               0                      0.0  ...              0             1

[5 rows x 74 columns]
```

3.4 Standardizacija

Kao što je prethodno napomenuto, atributi su različito skalirani, što znači da ih je nemoguće međusobno upoređivati. Kako bi se rešio ovaj problem biće korišćena standardizacija. To se postiže tako što se od atributa oduzme njegova srednja vrednost i to se podeli njegovom standardnom devijacijom, odnosno: $X_s = \frac{X-\mu}{\delta}$.

Pre nego što se izvrši standardizacija potrebno je podeliti skup podataka na trening i test skup, koji će biti korišćeni u procesu klasifikacije, kako bi se izbegao uticaj podatak iz test skupa na statistike trening skupa. Na primer, ako uradimo standardizaciju na celom skupu i onda ih podelimo, biće uzeti i podaci iz test skupa za računanje srednje vrednosti i standardne devijacije, što, ako test skup ima autlajere, neće biti željeno ponašanje jer će i oni biti uračunati.

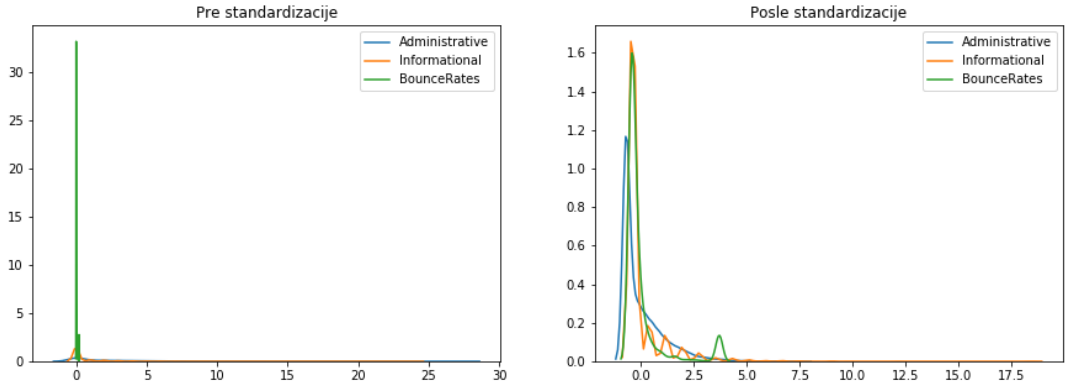
Skup podataka se prvo deli na skup atributa i na specijalni atribut koji će biti korišćen kao oznaka klase. Zatim se oba skupa dele na trening i test podskup.

Naredni programski kod ilustruje opisani postupak.

Programski kod:

```
# X - skup atributa
# Y - specijalni atribut određen za oznaku klase
X = df.drop('Revenue', axis = 1)
Y = df['Revenue']

x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.3,
stratify = Y, random_state = 7)

scaler = preprocessing.StandardScaler()
scaler.fit(x_train)

x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

Rezultat standardizacije na nekim atributima **prikazan je na slici** 4.



Slika 4: Grafik raspodele atributa pre i posle standardizacije

4. Klasifikacija

U ovom odeljku biće opisani korišćeni algoritmi klasifikacije.

4.1 Logistička regresija

Prvi algoritam koji će biti primenjen na skup podataka je logistička regresija, upravo zato što je jedna od najkorišćenijih metoda. Jednostavna je i pruža efikasno treniranje. Ova metoda primenljiva je samo na binarnu klasifikaciju, što jeste slučaj u opisanom skupu podataka.

4.1.1 Prvi pokušaj

Prvi pokušaj biće bez balansiranja klasa i bez PCA metode za smanjivanje dimenzionalnosti, kako bi se mogli uporediti rezultati pri svakom novom koraku. Takođe, biće prikazani i rezultati metode sa drugačijim izborom atributa. Za odabir regularizacionog parametra korišćen je Grid Search algoritam iscrpne pretrage, koji se koristi za pronalaženje optimalnih parametara za prosleđeni metod. Algoritam za evaluaciju koristi unakrsnu validaciju.

Opisani postupak prikazan je narednim kodom, dok se odgovarajuća matrica konfuzije nad test podacima **nalazi na slici** 5.

Programski kod:

```
model = LogisticRegression(C = 1.0, penalty = 'l2', solver = 'lbfgs', max_iter = 200)
model.fit(x_train_scaled, y_train)
```

Naredni kod definiše funkcije za evaluaciju modela i iscrtavanje matrice konfuzije, a iste funkcije će biti korišćene u nastavku za evaluaciju narednih modela.

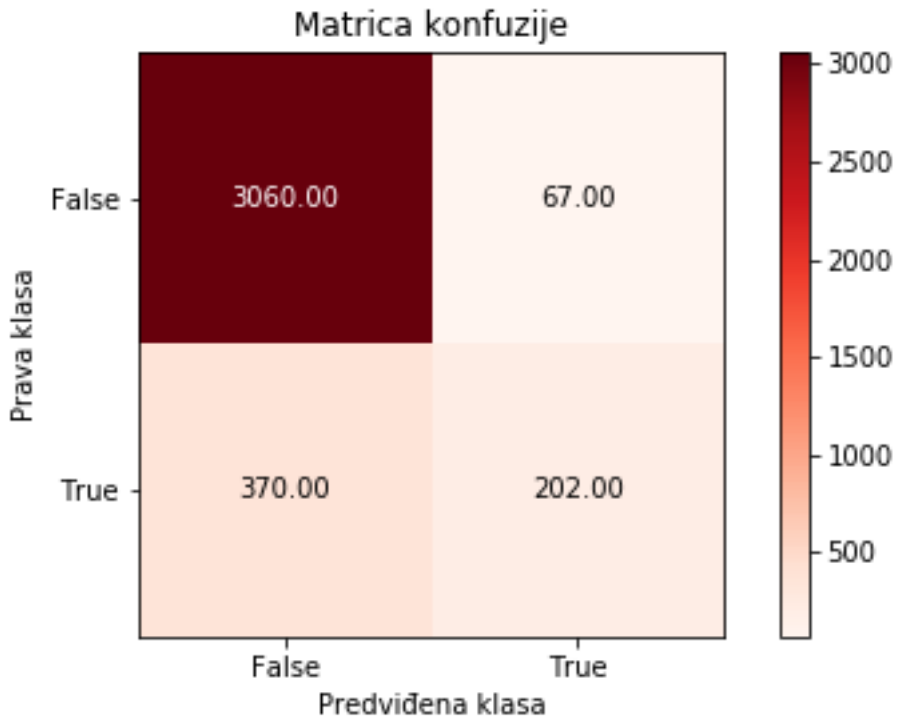
Programski kod:

```
def evaluate(model, x_tr, x_tst):
    y_test_predicted = model.predict(x_tst)
    test_score = metrics.accuracy_score(y_test, y_test_predicted)
    y_train_predicted = model.predict(x_tr)
    train_score = metrics.accuracy_score(y_train, y_train_predicted)
    print("Train score: {train}\nTest score: {test}".format(train = train_score, test = test_score))

    print()
    print(metrics.classification_report(y_test, y_test_predicted))

    return y_test_predicted

def draw_matrix(y_tst_predicted, cmap):
    confusion_matrix = metrics.confusion_matrix(y_tst, y_tst_predicted)

    fig, ax = plt.subplots()
    img = ax.imshow(confusion_matrix, interpolation = 'nearest', cmap = cmap)
    ax.figure.colorbar(img, ax = ax)

    ax.set(xticks = np.arange(confusion_matrix.shape[1]),
           yticks = np.arange(confusion_matrix.shape[0]),
           xticklabels = ['False', 'True'], yticklabels = ['False', 'True'],
           title = 'Matrica konfuzije',
           ylabel = 'Prava klasa',
           xlabel = 'Predviđena klasa')

    fmt = '.2f'
    thresh = confusion_matrix.max() / 2.
    for i in range(confusion_matrix.shape[0]):
        for j in range(confusion_matrix.shape[1]):
            ax.text(j, i, format(confusion_matrix[i, j], fmt),
                    ha = "center", va = "center",
                    color = "white" if confusion_matrix[i, j] > thresh else "black")

    fig.tight_layout()

# Pozivamo funkciju za trenutni skup
y_test_predicted = evaluate(lr, x_train_scaled, x_test_scaled)
```

Rezultat izvršavanja:

```
Train score: 0.8870351060132082
Test score: 0.881859962151933

              precision    recall  f1-score   support

           0       0.89      0.98      0.93      3127
           1       0.75      0.35      0.48       572
```

```
    accuracy                           0.88      3699
   macro avg       0.82      0.67      0.71      3699
weighted avg       0.87      0.88      0.86      3699
```



Slika 5: Matrica konfuzije

4.1.2 Popravka performansi

Iz prethodnih rezultata može se zaključiti da se dobijaju znatno lošiji rezultati za klasifikaciju podataka koji pripadaju klasi tačno, jer u toj klasi postoji manje podataka. Balansiranjem klasa trebalo bi da se popravi taj rezultat. Za balansiranje klasa biće korišćena *SMOTE (Synthetic Minority Oversampling TEchnique)* tehnika, koja počiva na sintezi elemenata manje klase od onih koji već postoje u skupu. Funkcioniše tako što na slučajan način bira tačku iz manje klase i računa k najbližih suseda za tu tačku. Sintetisane tačke se dodaju između odabrane tačke i njenih suseda.

Takođe, biće primenjen i algoritam PCA, kako bi se smanjila dimenzionalnost i time povećala efikasnost algoritma.

Naredni kod ilustruje taj postupak, a odgovarajuća matrica konfuzije nad test podacima prikazana je **na slici 6**.

Programski kod:

```
print('Broj instanci u klasi "1" pre balansiranja: {}'.format(sum(y_train == 1)))
print('Broj instanci u klasi "0" pre balansiranja: {}'.format(sum(y_train == 0)))

sm = SMOTE(sampling_strategy = 'minority', random_state = 2)
x_train_balanced, y_train = sm.fit_sample(x_train_scaled, y_train.ravel())

print()
print('Broj instanci u klasi "1" nakon balansiranja: {}'.format(sum(y_train == 1)))
print('Broj instanci u klasi "0" pre balansiranja: {}'.format(sum(y_train == 0)))
```

Rezultat izvršavanja:

```
Broj instanci u klasi "1" pre balansiranja: 1336
```

```
Broj instanci u klasi "0" pre balansiranja: 7295

Broj instanci u klasi "1" nakon balansiranja: 7295
Broj instanci u klasi "0" pre balansiranja: 7295
```

Primena algoritma PCA:

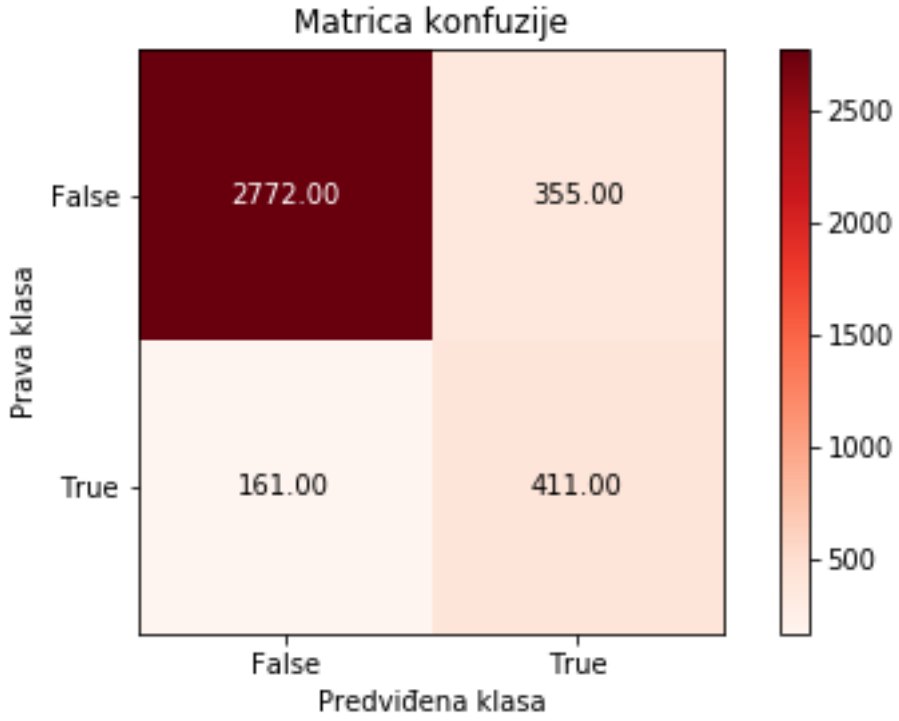
Programski kod:

```
pca = PCA(0.95)
pca.fit(x_train_balanced)

x_train = pca.transform(x_train_balanced)
x_test = pca.transform(x_test_scaled)
```

Pokretanjem identičnog koda za dobijanje modela i evaluaciju dobijamo naredni rezultat izvršavanja, a odgovarajuća matrica konfuzije nad test podacima nalazi se **na slici 6**.

```
Train score: 0.8503084304318026
Test score: 0.8605028386050284
```

	precision	recall	f1-score	support
0	0.95	0.89	0.91	3127
1	0.54	0.72	0.61	572
accuracy			0.86	3699
macro avg	0.74	0.80	0.76	3699
weighted avg	0.88	0.86	0.87	3699



Slika 6: Popravljena matrica konfuzije

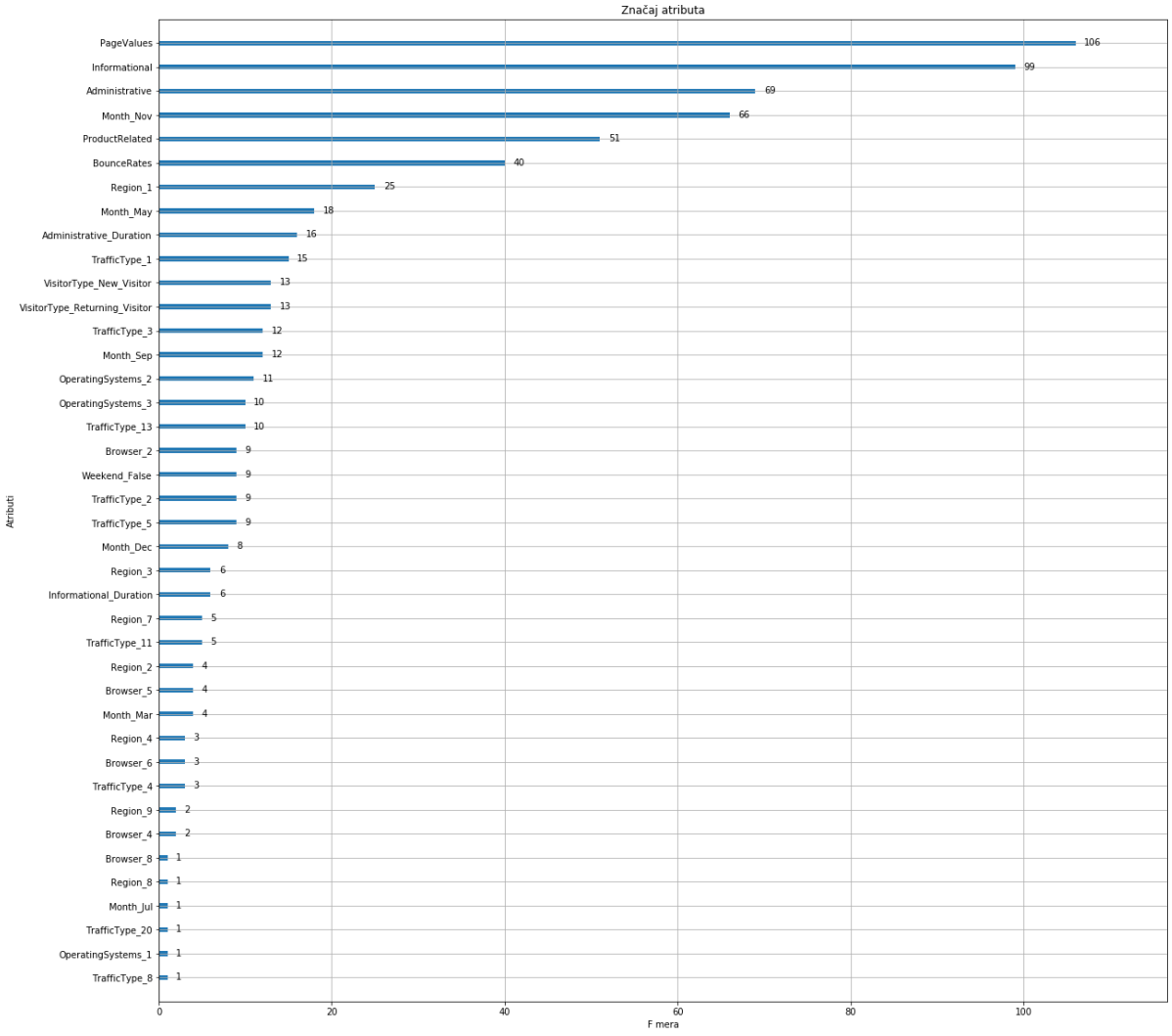
Na prvi pogled drugi model deluje lošiji, jer je dobijena manja ukupna tačnost. Međutim, odziv za klasu '1' je sada čak duplo bolji, što znači da će model manje grešiti kada je ta klasa u pitanju.

U prilagođavanju narednih modela biće korišćen skup dobijen procesom standardizacije, bez primene algoritma PCA.

4.1.3 Promena atributa

Sledeće što će biti pokušano, zajedno sa svim prethodnim, jeste promena atributa koji se koriste. Za određivanje najbitnijih atributa biće korišćen algoritam XGBoost. XGBoost je jedan od najpopularnijih algoritama danas za regresiju i klasifikaciju, a baziran je na stablima odlučivanja. U ovom slučaju biće korišćen za selekciju važnih atributa.

Kao rezultat dobijamo grafik **na slici 7**.



Slika 7: Grafik značajnosti atributa

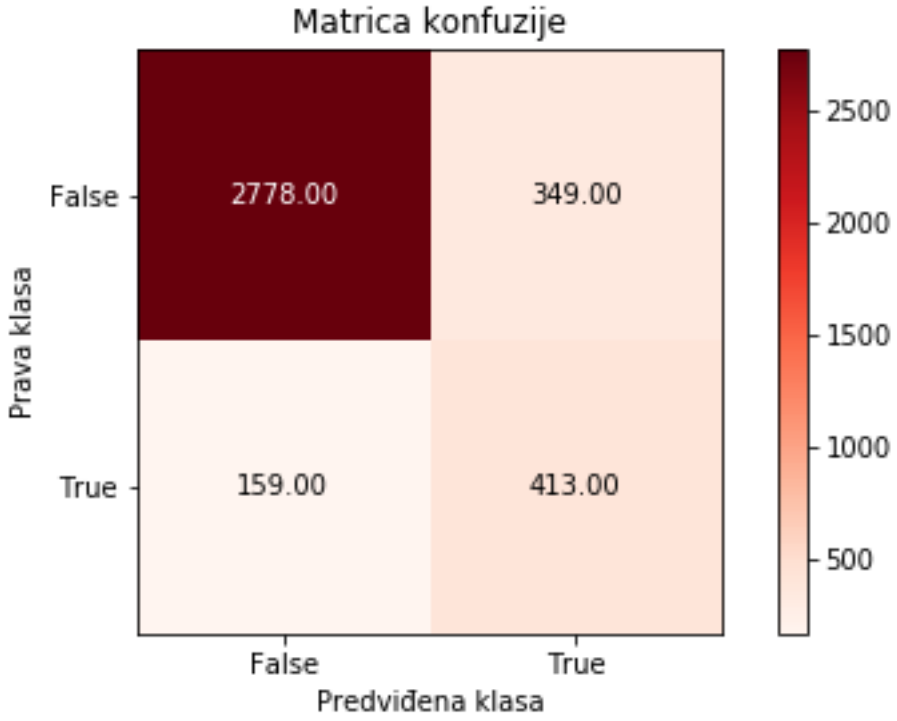
Na osnovu grafika može se zaključiti da atributi *Informational Duration*, *Browser* i *Weekend* nisu mnogo značajni,

pa ćemo ih isključiti. Atribut *Special Day* ima iznenađujuće malu značajnost, jer intuitivno veoma utiče na kupovinu, ali ćemo ga ipak isključiti.

Ponovnim pokretanjem identičnog koda za dobijanje modela i evaluaciju, samo na novom skupu podataka, dobijamo naredni rezultat izvršavanja.

Rezultat izvršavanja:

```
Train score: 0.8465387251542152
Test score: 0.8626655852933225

              precision    recall  f1-score   support

           0       0.95      0.89      0.92      3127
           1       0.54      0.72      0.62       572

    accuracy                           0.86      3699
   macro avg       0.74      0.81      0.77      3699
weighted avg       0.88      0.86      0.87      3699
```



Može se primetiti da se dobijaju neznatno bolji rezultati, tako da će nadalje biti korišćen stari skup, jer je bogatiji, odnosno nosi više informacija.

4.2 Stabla odlučivanja

Sledeći algoritam koji će biti primenjen na opisani skup jesu stabla odlučivanja. To je metoda u kojoj se proces klasifikacije modeluje pomoću skupa hijerarhijskih odluka donetih na osnovu atributa trening podataka, uređenih u obliku drvolike strukture. Kao kriterijum podele biće korišćena Entropija, a čvorovi će se nalaziti na dubini najviše 7. Parametri su, kao i kod Logističke regresije, odabrani korišćenjem Grid Search algoritma.

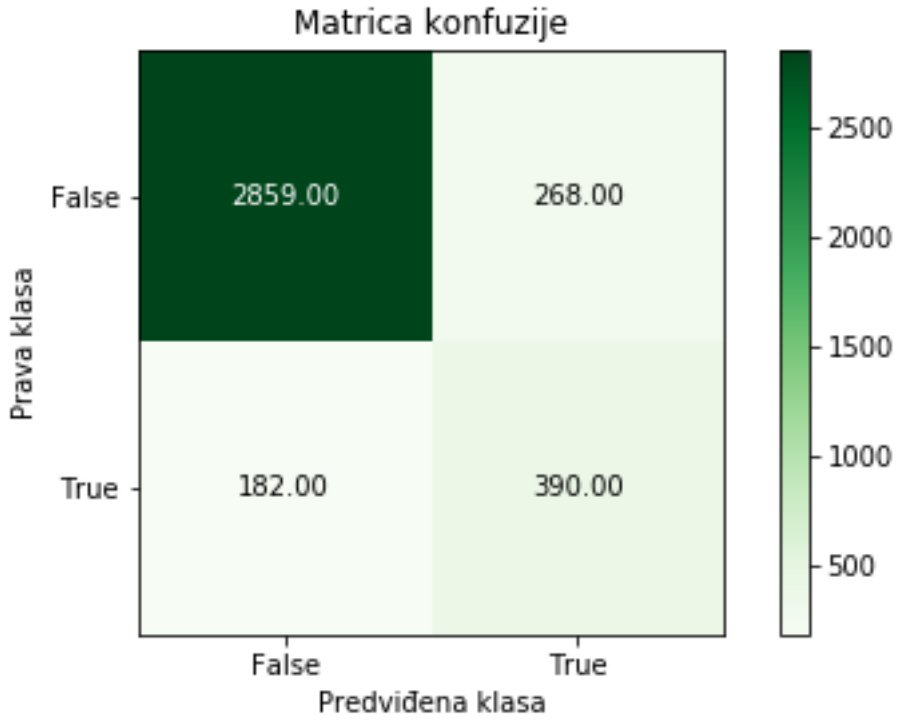
Naredni programski kod ilustruje pravljenje modela i njegovu evaluaciju, dok se odgovarajuća matrica konfuzije nalazi **na slici** 8.

Programski kod:

```
dt = DecisionTreeClassifier(criterion = 'entropy', max_depth = 7)
dt.fit(x_train_balanced, y_train)

y_test_predicted = evaluate(dt, x_train_balanced, x_test_scaled)
```

Rezultat izvršavanja:

```
Train score: 0.9229609321453051
Test score: 0.878345498783455

              precision    recall  f1-score   support

           0       0.94      0.91      0.93      3127
           1       0.59      0.68      0.63       572

    accuracy                           0.88      3699
   macro avg       0.77      0.80      0.78      3699
weighted avg       0.89      0.88      0.88      3699
```



Slika 8: Matrica konfuzije za stablo odlučivanja

4.3 Metod potpornih vektora (SVM)

Naredna tehnika koja će biti primenjena je *SVM (Support Vector Machine)*, odnosno tehnika potpornih vektora. To je tehnika za klasifikaciju zasnovana na ideji vektorskih prostora. Prvo će biti prikazan linearni SVM, a zatim SVM sa kernel funkcijom.

4.3.1 Linearni SVM

Za regularizacioni parametar biće uzeta vrednost 1.0, što je dobijeno Grid Search algoritmom.

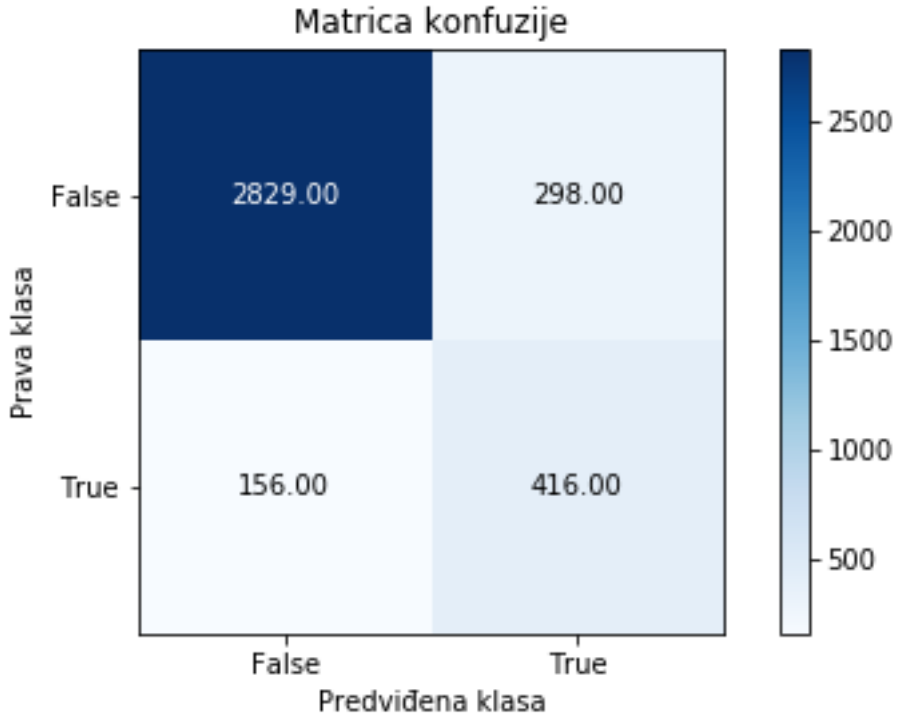
Naredni kod ilustruje primer pravljenja modela i njegove evaluacije, dok je odgovarajuća matrica konfuzije na test skupu prikazana **na slici 9**.

Programski kod:

```
lin_svm = svm.LinearSVC(loss = 'hinge', C = 1.0)
lin_svm.fit(x_train_balanced, y_train)

y_test_predicted = evaluate(lin_svm, x_train_balanced, x_test_scaled)
```

Rezultat izvršavanja:

```
Train score: 0.8613433858807402
Test score: 0.8772641254393079

              precision    recall  f1-score   support

           0       0.95      0.90      0.93      3127
           1       0.58      0.73      0.65       572

    accuracy                           0.88      3699
   macro avg       0.77      0.82      0.79      3699
weighted avg       0.89      0.88      0.88      3699
```



Slika 9: Matrica konfuzije za linearni SVM

4.3.2 SVM sa kernel funkcijom

Za regularizacioni parametar biće uzeta vrednost 1.0, a za kernel je odabran rbf kernel.

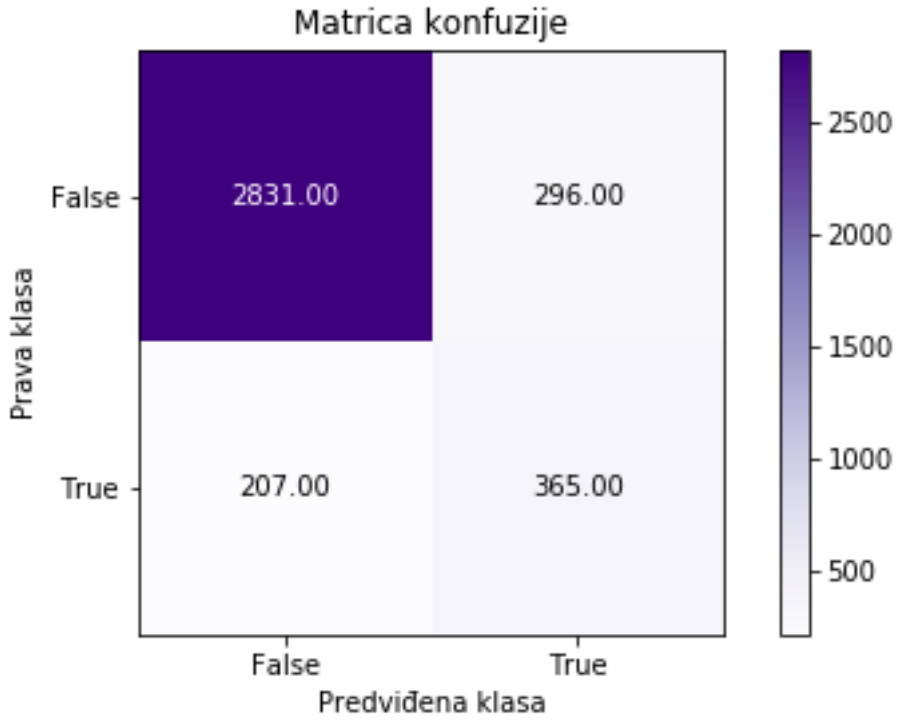
Naredni kod ilustruje primer pravljenja modela i njegove evaluacije, dok je odgovarajuća matrica konfuzije na test skupu prikazana **na slici 10**.

Programski kod:

```
kernelized_svm = svm.SVC(kernel = 'rbf', C = 2.0)
kernelized_svm.fit(x_train_balanced, y_train)

y_test_predicted = evaluate(kernelized_svm, x_train_balanced, x_test_scaled)
```

Rezultat izvršavanja:

```
Train score: 0.9372172721041809
Test score: 0.8640173019735063

              precision    recall  f1-score   support

           0       0.93      0.91      0.92      3127
           1       0.55      0.64      0.59       572

    accuracy                           0.86      3699
   macro avg       0.74      0.77      0.76      3699
weighted avg       0.86      0.87      0.86      3699
```



Slika 10: Matrica konfuzije za SVM sa kernel funkcijom

4.4 Random Decision Forests

Poslednji metod koji će biti primenjen na skup je *Random Forest* algoritam. Ovaj metod spada u grupu ansambala. Ansambli koriste više algoritama za učenje kako bi postigli bolje rezultate u predikciji klasa. Konkretno, Random Forest metod funkcioniše tako što izgrađuje mnoštvo stabala odlučivanja pri treniranju i dodeljuje instanci onu klasu koja se najčešće pojavljivala.

Kao kriterijum podele biće korišćena Entropija, a broj stabala u šumi biće jednak 15. Ti parametri su još jednom dobijeni iz Grid Search algoritma.

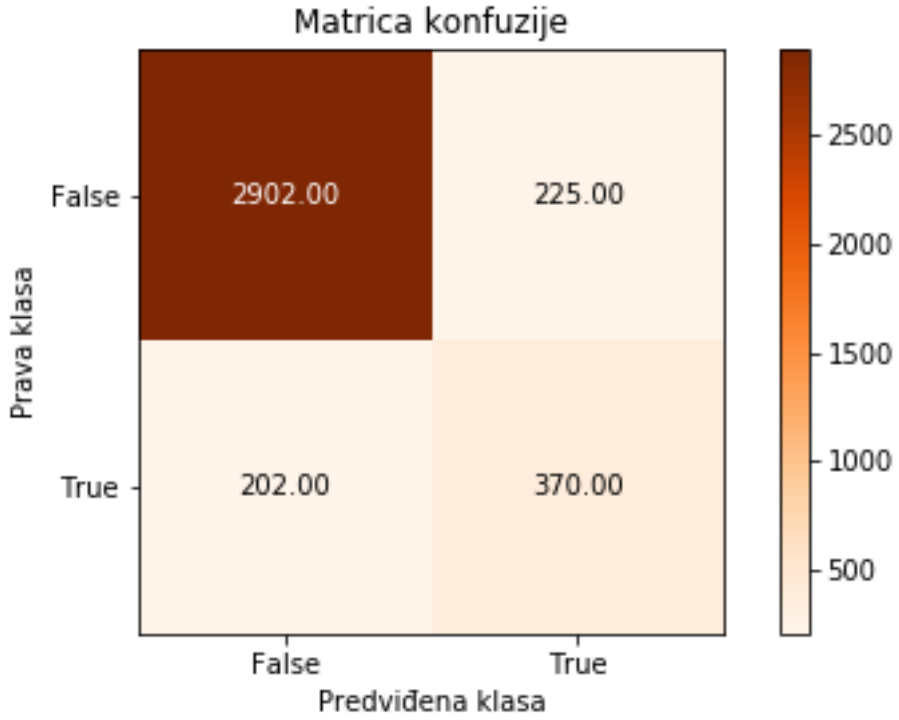
Naredni kod ilustruje opisani postupak, a odgovarajuća matrica konfuzije na test skupu biće prikazana **na slici 11**.

Programski kod:

```
rfc = RandomForestClassifier(n_estimators = 15, criterion = 'entropy')
rfc.fit(x_train_balanced, y_train)

y_test_predicted = evaluate(rfc, x_train_balanced, x_test_scaled)
```

Rezultat izvršavanja:

```
Train score: 0.997943797121316
Test score: 0.8821303054879697

              precision    recall  f1-score   support

           0       0.93      0.93      0.93      3127
           1       0.61      0.64      0.63       572

    accuracy                           0.88      3699
   macro avg       0.77      0.78      0.78      3699
weighted avg       0.88      0.88      0.88      3699
```



Slika 11: Matrica konfuzije za Random Forest metod

5. Zaključak

Rezultati svih primenjenih metoda bez balansiranja prikazani su **u tabeli** 4, a sa balansiranjem **u tabeli** 5.

Tabela 4: Rezultati metoda bez balansiranja klasa

Naziv metode	Tačnost na trening skupu	Tačnost na test skupu	Preciznost klase ‘0’	Preciznost klase ‘1’	Odziv klase ‘0’	Odziv klase ‘1’
Logistička regresija	0.85	0.86	0.95	0.54	0.89	0.72
Stabla odlučivanja	0.90	0.88	0.92	0.66	0.95	0.57
Linearni SVM	0.88	0.88	0.89	0.73	0.98	0.36
SVM sa kernel funkcijom	0.91	0.88	0.90	0.74	0.98	0.38
Random Forest	0.99	0.88	0.91	0.68	0.96	0.49

Tabela 5: Rezultati metoda sa balansiranjem klasa

Naziv metode	Tačnost na trening skupu	Tačnost na test skupu	Preciznost klase ‘0’	Preciznost klase ‘1’	Odziv klase ‘0’	Odziv klase ‘1’
Logistička regresija	0.85	0.86	0.95	0.54	0.89	0.72
Stabla odlučivanja	0.92	0.87	0.94	0.59	0.91	0.68
Linearni SVM	0.86	0.88	0.95	0.58	0.90	0.73
SVM sa kernel funkcijom	0.94	0.86	0.93	0.55	0.91	0.64
Random Forest	0.99	0.88	0.93	0.61	0.93	0.64

Svi prikazani metodi se veoma slično ponašaju na opisanom skupu. Takođe, kao što se vidi iz tabela 4 i 5, na svaki metod je balansiranje klasa uticalo na isti način: tako što je opala tačnost na test skupu, ali se povećao odziv klase ‘1’, što svakako znači da je model dobijen balansiranjem klasa bolji.

Iz tabela se takođe vidi da najbolje mere ima metod Random Forest, što je i bilo očekivano zbog načina na koji ovaj metod radi. Dakle, krajnji rezultat istraživanja bio bi zaključak da iako su samo nijanse u pitanju u rezulatatima metoda, za klasifikaciju ovog skupa, ukoliko se uzimaju u obzir mere klasifikacije a ne i druge osobine poput jednostavnosti i brzine, trebalo bi koristiti algoritam Random Forest.