Neuronowy model klienta bankowego (Bank Marketing Data)

KWD
Grzegorz Podwika
Marek Ziaja

01.02.2021

Streszczenie

Projekt ma na celu stworzenie klasyfikatora realizującego przewidywanie chęci ulokowania swoich środków w lokacie terminowej w banku. Program komputerowy na podstawie wcześniej dostarczonych danych, zbieranych przez pracowników banku telefonujących do klientów, trenuje klasyfikator który w kolejnym kroku zostaje poddany testom. Naszym zadaniem jest analiza wyników programu i wybranie oraz opracowanie dającego najlepsze wyniki modelu klasyfikującego.

Spis treści

Rozdział 1

Wprowadzenie

Banki na całym świecie głowią się i trują, aby realizowane przez nich strategie przynosiły jak największe zyski. Jedną z obranych strategii przez bank było kilku/kilkunastrokrotne przeprowadzanie rozmów telefonicznych przez konsultantów z klientami banku z zamiarem założenia korzystnej lokaty terminowej. Zbierając dane z tego typu pracy możemy opracować model klienta bankowego, analizować go i ulepszać tą konkretną strategię z wymiernymi korzyściami dla banku. Dzięki temu następne kampanie przeprowadzane przez bank będą bardziej efektywne i przyniosą korzyści zarówno dla klientów jak i samego banku.

1.1 Opis problemu

Podstawowym problemem jest stworzenie klasyfikatora, który przy klasyfikowaniu danych, czyli w tym przypadku cech klienta banku, który będzie popełniał możliwie najmniejszą ilość błędów przy predykcji następnych danych. Pod uwagę należy wziąć odpowiedni dobór danych na których nasz klasyfikator będzie trenowany, ponieważ na ich podstawie będzie podejmować decyzję. Model musi także zostać dokładnie przetestowany na różne sposoby, uwzględniając zmianę danych testowych i trenujących, tak by zminimalizować błąd, a zmaksymalizować wiarygodność naszego modelu. Uwagę należy poświęcić także rozkładzie danych na treningowe i testujące. Tworząc każdy model należy zadbać o to, by nie był niedouczony, ale także aby nie był przeuczony, ponieważ w takim przypadku model za bardzo podąża za błędami/outlinerami niż za ogólnym trendem, którego chcemy uchwycić.

Rozdział 2

Realizacja

2.1 Wprowadzenie teoretyczne

Odmianą regresji używaną do rozwiązania naszego problemu jest regresja logistyczna. Najważniejszą cechą regresji logistycznej jest to, że etykieta przyjmuje tylko dwie wartości (najczęściej 0 lub 1). Używamy jej do określania wystąpienia jakiegoś zdarzenia lub zjawiska. Regresja logistyczna różni się zasadniczo od regresji liniowej. W przypadku regresji logistycznej w przeciwieństwie do regresji liniowej celem nie jest przewidzenie wartości zmiennej zależnej na podstawie wykorzystania predykatów, ale przewidzenie prawdopodobieństwa wystąpienia jakiegoś zdarzenia.

Wzór na równanie regresji logistycznej:

$$P(Y = 1 | x_1, x_2, \dots, x_k) = P(X) = \frac{e^{a_0 + \sum_{i=1}^{k} a_i x_i}}{1 + e^{a_0 + \sum_{i=1}^{k} a_i x_i}}$$

Gdzie:

$a_i, i = 0, \dots, k$ są współczynnikami regresji,

$x_1, x_2, \dots, x_k$ - zmienne niezależne (ilościowe lub jakościowe).

Analiza regresji logistycznej w porównaniu do regresji liniowej obarczona jest mniejszą ilością założeń. Najważniejszą cechą regresji logistycznej jest to, że predykatem jest zmienna dychotomiczna 0 - 1.

Metodą używaną podczas testowania naszego modelu jest walidacja krzyżowa. Jest to metoda statystyczna polegająca na

podziale próby statystycznej na podzbiory, a następnie przeprowadzeniu wszelkich analiz na zbiorze uczącym, podczas gdy zbiór testowy służy do potwierdzenia wiarygodności wyników. Metoda ta jest stosowana w przypadku niedostatecznie licznego zbioru uczącego.

2.2 Opis danych wejściowych

Zbiór danych na których działać będzie klasyfikator składa się z 41188 rekordów, z których każdy określa konkretnego klienta banku. W każdej krotce znajduje się 20 cech określających klienta. Na ich podstawie wyznaczana jest etykieta, która określa czy udało się namówić klienta na lokatę - niepowodzenie ("no"), bądź powodzenie ("yes").

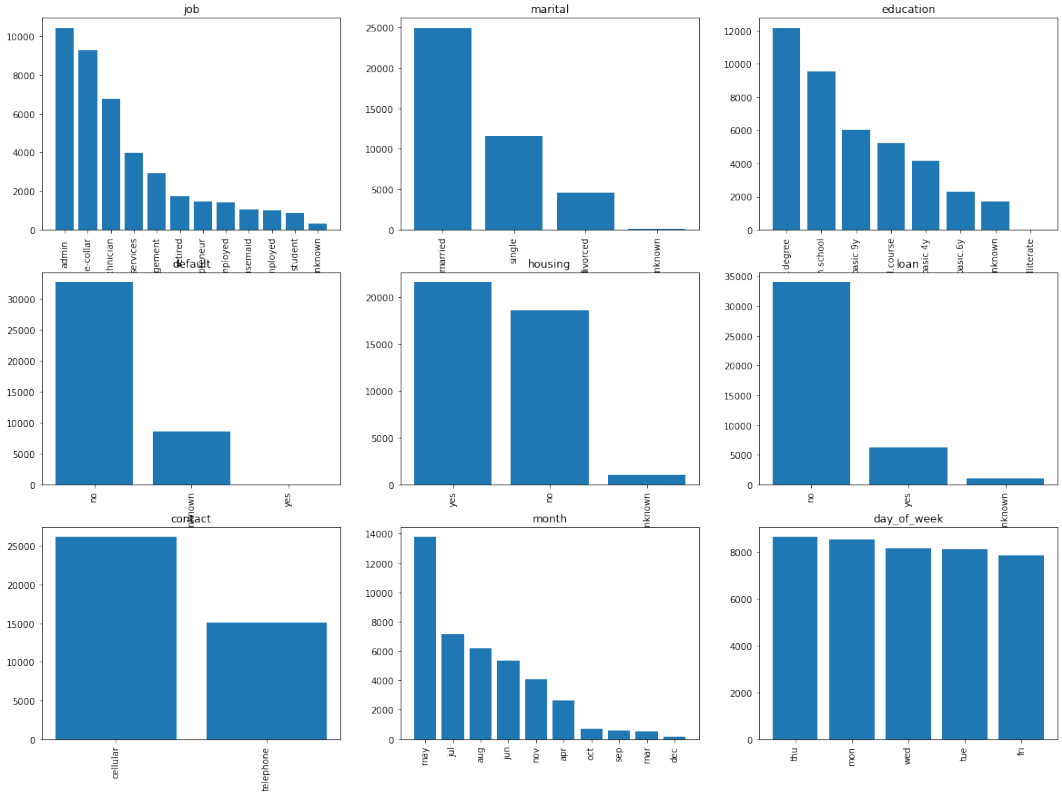
Info o zbiorze danych:

```
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             41188 non-null  int64
 1   job             41188 non-null  object
 2   marital         41188 non-null  object
 3   education       41188 non-null  object
 4   default         41188 non-null  object
 5   housing         41188 non-null  object
 6   loan            41188 non-null  object
 7   contact         41188 non-null  object
 8   month           41188 non-null  object
 9   day_of_week     41188 non-null  object
 10  duration        41188 non-null  int64
 11  campaign        41188 non-null  int64
 12  pdays           41188 non-null  int64
 13  previous        41188 non-null  int64
 14  poutcome        41188 non-null  object
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object
dtypes: float64(5), int64(5), object(11)
```

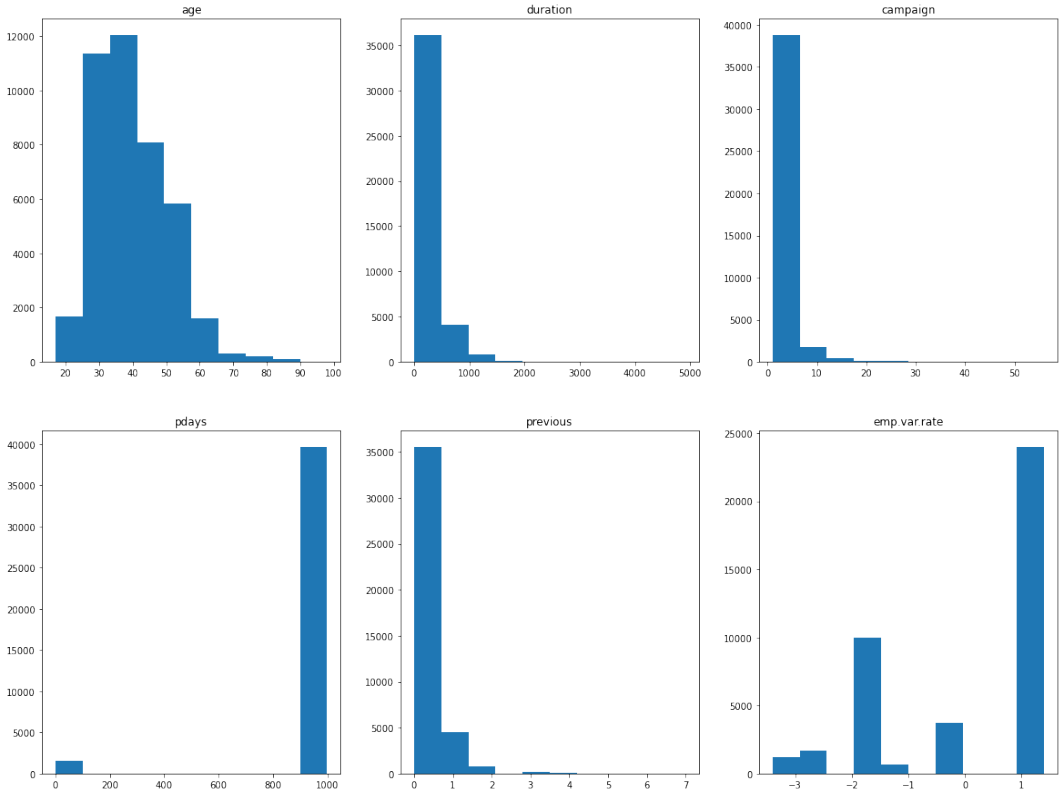
Dane określające klienta:

	0	1	2	3	4
age	56	57	37	40	56
job	housemaid	services	services	admin.	services
marital	married	married	married	married	married
education	basic.4y	high.school	high.school	basic.6y	high.school
default	no	unknown	no	no	no
housing	no	no	yes	no	no
loan	no	no	no	no	yes
contact	telephone	telephone	telephone	telephone	telephone
month	may	may	may	may	may
day_of_week	mon	mon	mon	mon	mon
duration	261	149	226	151	307
campaign	1	1	1	1	1
pdays	999	999	999	999	999
previous	0	0	0	0	0
poutcome	nonexistent	nonexistent	nonexistent	nonexistent	nonexistent
emp.var.rate	1.1	1.1	1.1	1.1	1.1
cons.price.idx	93.994	93.994	93.994	93.994	93.994
cons.conf.idx	-36.4	-36.4	-36.4	-36.4	-36.4
euribor3m	4.857	4.857	4.857	4.857	4.857
nr.employed	5191	5191	5191	5191	5191

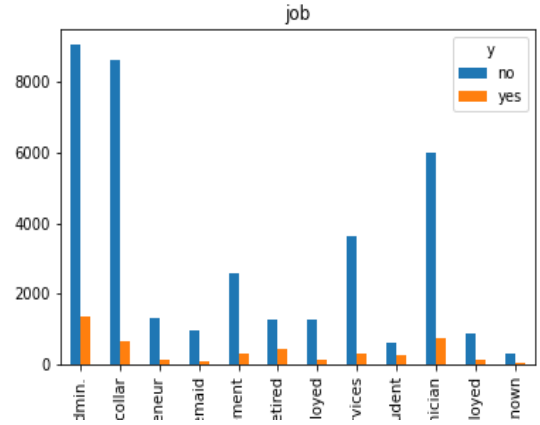
Eksploracja kolumn danych kategorycznych:

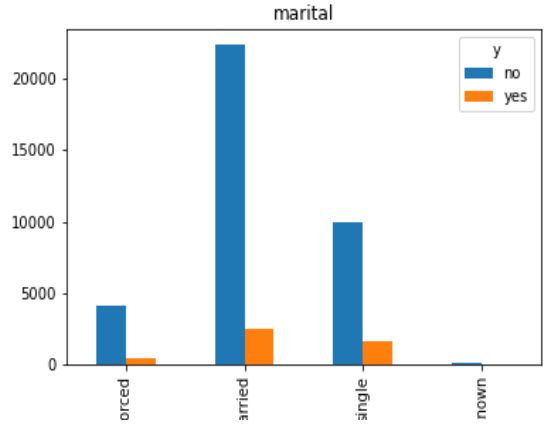


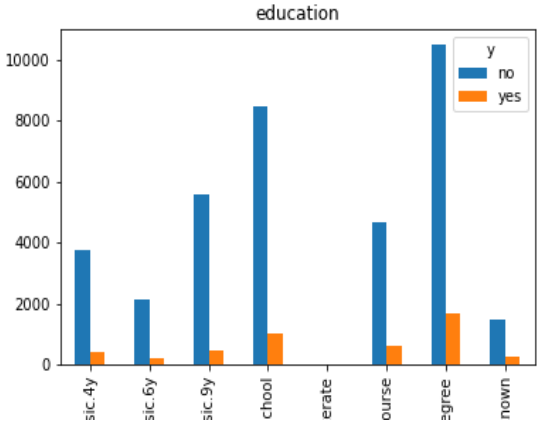
Eksploracja kolumn danych numerycznych:

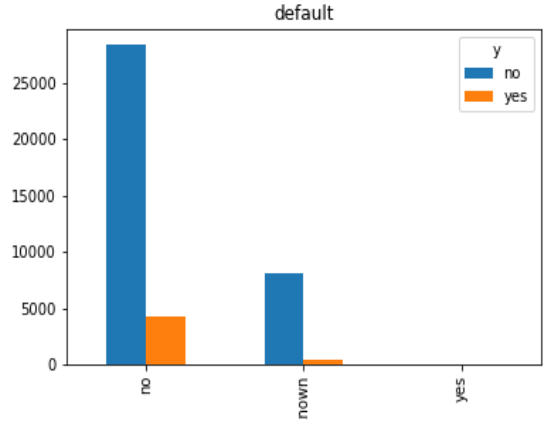


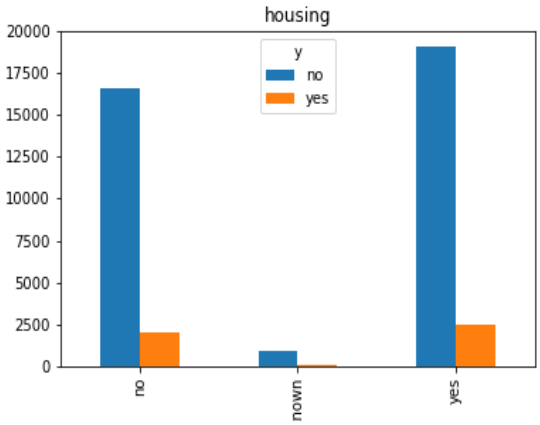
Rozkład danych kategorycznych wraz z etykietami

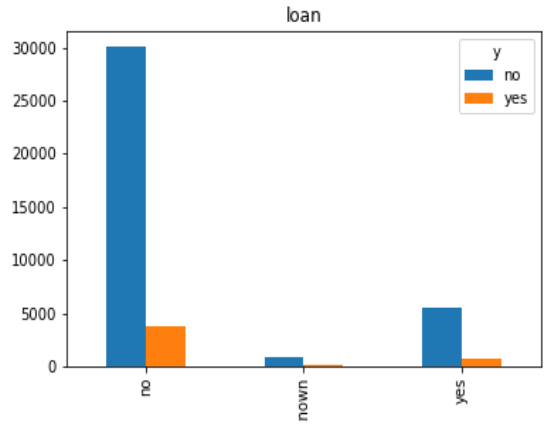


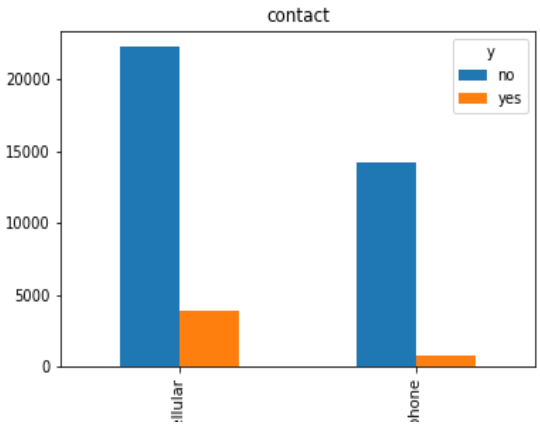


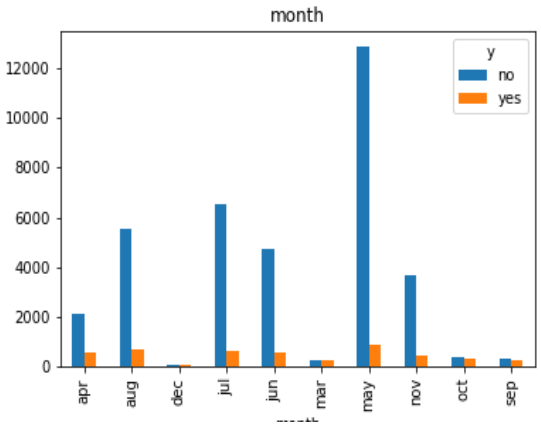


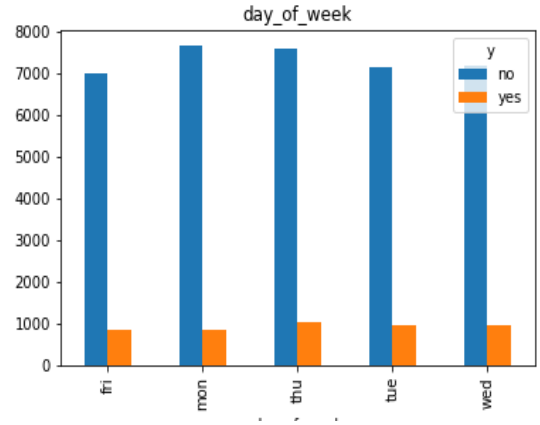


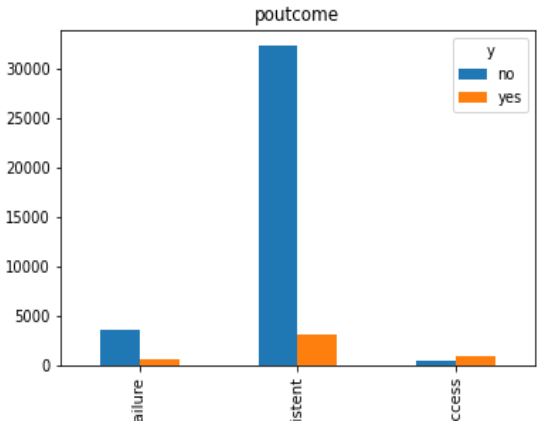




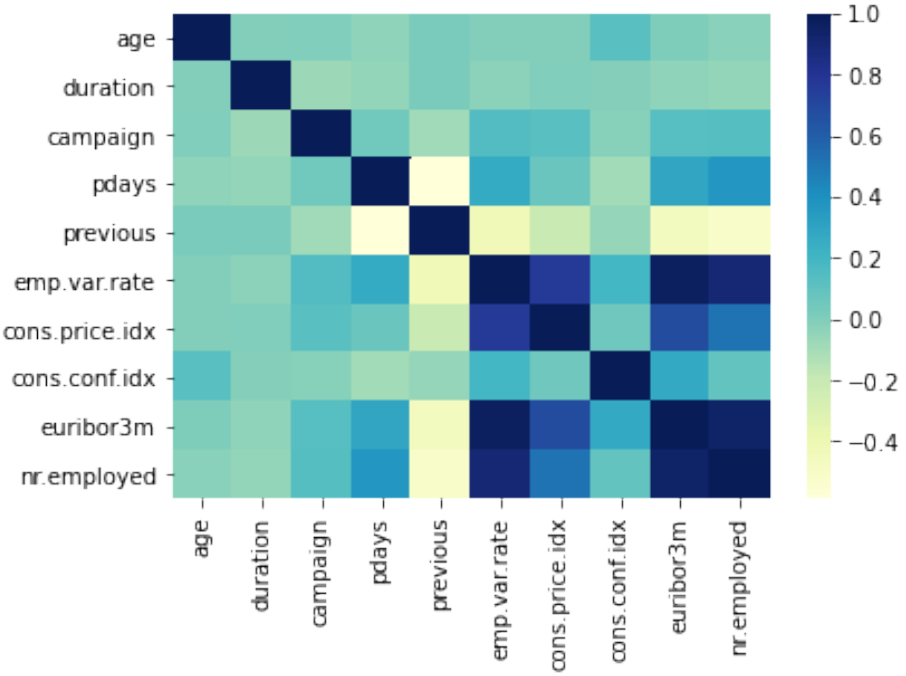

contact
y
no
yes
20000
15000
10000
5000
0


month
y
no
yes
12000
10000
8000
6000
4000
2000
0
apr
aug
dec
jul
jun
mar
may
nov
oct
sep

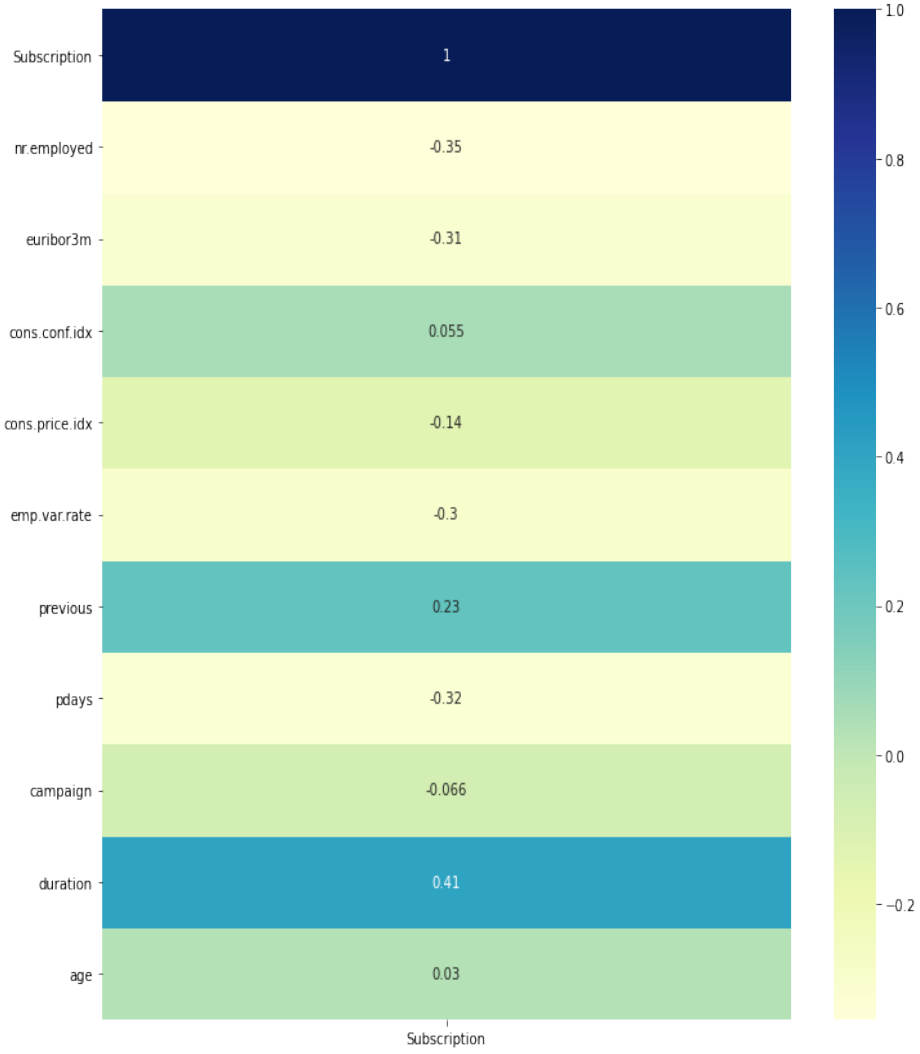

day_of_week
y
no
yes
8000
7000
6000
5000
4000
3000
2000
1000
0
fri
mon
thu
tue
wed


poutcome
y
no
yes
30000
25000
20000
15000
10000
5000
0

Wzajemna korelacja cech



Wzajemna korelacja cech z subskrypcją



2.3 Przekształcenie zbioru danych

Zapoznawszy się z powyższymi zdjęciami obrazującymi zbiór danych klientów banku możemy od razu zauważyć, że co po niektóre kolumny są zapisane w postaci kategorycznych co uniemożliwia uczenie naszego modelu, więc musimy zakodować cechy tych kolumn w postaci numerycznej. To samo musimy uczynić z etykietami, przekształcając je z "yes" oraz "no" na odpowiednio 1 i 0.

Enkodowanie danych:

```
# Transform 'yes' and 'no'  into (0,1) dependency
bank_data['y']=[1 if i=='yes' else 0 for i in bank_data['y']]
```

```
# Create dummy variable for the categorical data and drop first column
# to avoid dummy varibale trap
bank_data = pd.get_dummies(bank_data,columns = ['job','marital','education',
                                                'default', 'housing','loan',
                                                'month', 'contact',
                                                'day_of_week','poutcome'],
                                                drop_first = True)
```

2.4 Normalizacja i standaryzacja danych

Biorąc pod uwagę zróżnicowanie zakresu danych w zbiorze, w obliczeniach konieczne jest uwzględnienie normalizacji oraz standaryzacji. Standaryzacja danych do modeli machine learning polega na przekształceniu danych pierwotnych, aby ich rozkład miał średnią wartość równą 0 i odchylenie standardowe równe 1. Od każdej wartość w kolumnie danych będzie odejmowana średnia wartość kolumny, a następnie to co wyjdzie będzie podzielona przez odchylenie standardowe kolumny danych. Opisany proces dotyczy każdej kolumny oddzielnie. Wiele klasyfikatorów oblicza odległość między dwoma punktami na podstawie odległości euklidesowej. Jeśli jedna z cech ma szeroki zakres wartości, odległość będzie zależała od tej konkretnej cechy. Dlatego zakres wszystkich cech powinien zostać znormalizowany, tak aby każda cecha miała udział w przybliżeniu proporcjonalnie do końcowej odległości.

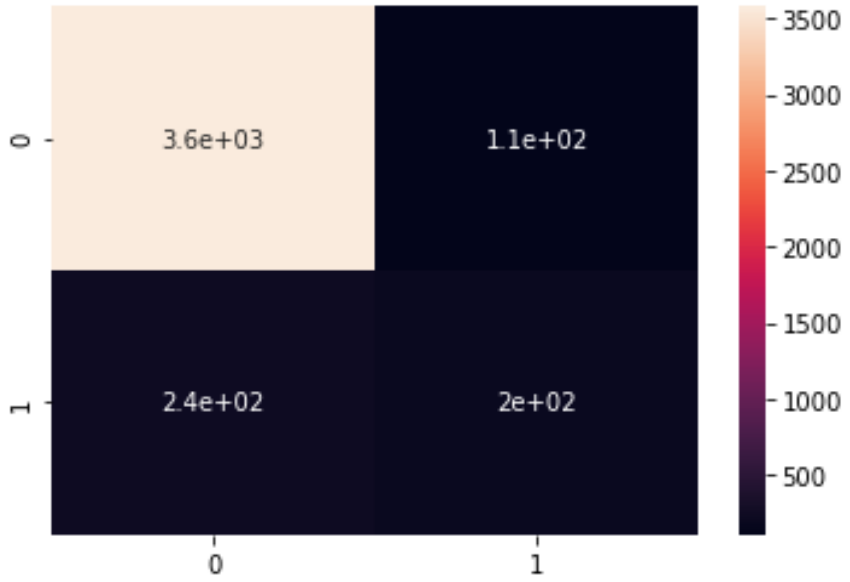
Standaryzacja danych dla modelu:

```
# scale data
sc = StandardScaler()
bank_train_data = sc.fit_transform(bank_train_data)
bank_test_data = sc.transform(bank_test_data)
```

2.5 Ewaluacja uzyskanego modelu

Wytrenowany przez nas model za pomocą regresji logistycznej dla przeskalowanych danych daje następujące wyniki:

```
Basic Logistic Regression with scaled data
Basic model accuracy of scaled data= 0.9169701383831027
[[3579  107]
 [ 235  198]]
```



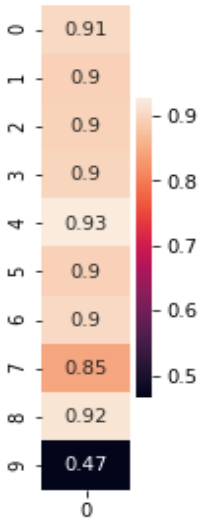
Wynik modelu jest przedstawiony za pomocą współczynnika dokładności - accuracy score. Obliczamy go dzieląc liczbę poprawnych dopasowań przez wszystkie próby dopasowania. W naszym przypadku dokładność wyniosła około 91 %, co jest przyzwoitym wynikiem. Macierz konfuzji przedstawia ilość poprawnych

i niepoprawnych dopasowań. W pierwszym wierszu znajdują się informacje o 3569 poprawnych dopasowaniach braku subskrypcji (0), oraz o 98 pomyłkach, gdzie model przypisał błędnie zdecydowanie się na otwarcie lokaty, a w rzeczywistości klient nie zdecydował się na ten krok. W drugim wierszu znajdują się informację o 206 poprawnych dopasowaniach subskrypcji lokaty (1), oraz o 246 pomyłkach kiedy model przypisał brak subskrypcji, a w rzeczywistości klient zdecydował się na założenie lokaty. Podczas tworzenia modelu 90% danych było zbiorem trenującym, natomiast 10% danych było zbiorem testującym.

```
# split data and target sets into 4 sets for train and test purposes
bank_train_data, bank_test_data, \
bank_train_target, bank_test_target = \
train_test_split(data_np, target_np, test_size=0.1)
```

Dla naszych danych przeprowadziliśmy również kroswalidację, której wyniki są zbliżone do modelu wytrenowanego przez nas.

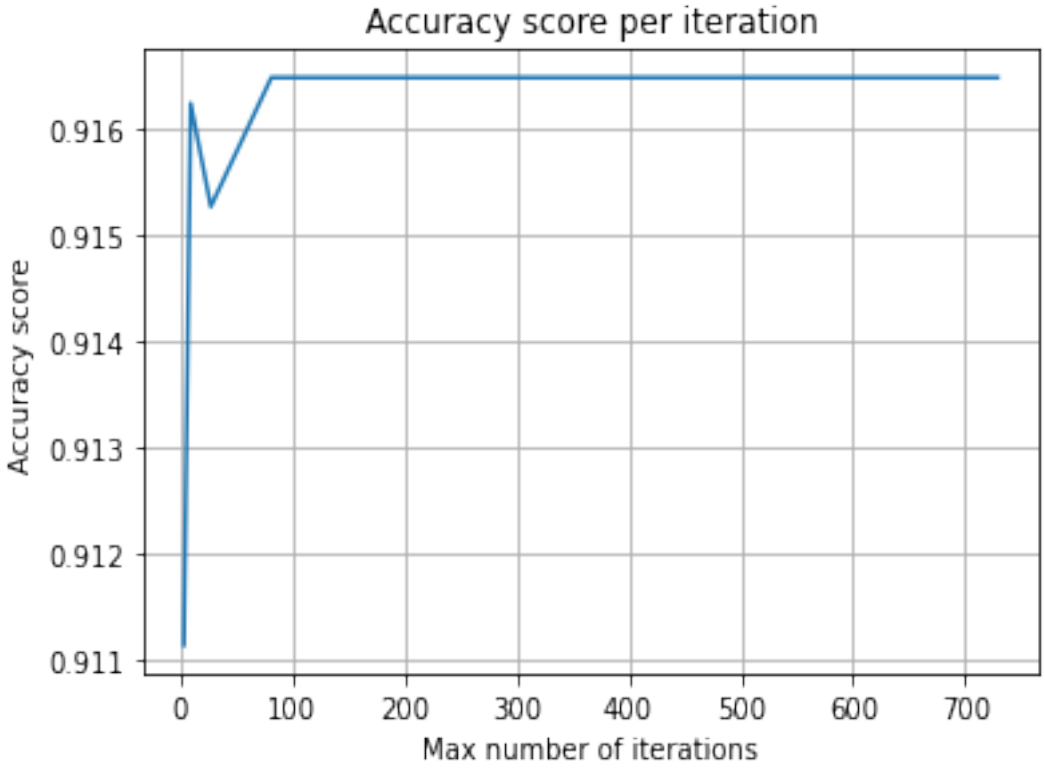
Wyniki z kroswalidacji wyglądają następująco:



2.6 Badania symulacyjne

Badania wyniku zmienności naszego modelu rozpoczęliśmy od określania maksymalnej liczby iteracji dla algorytmu budującego klasyfikator.

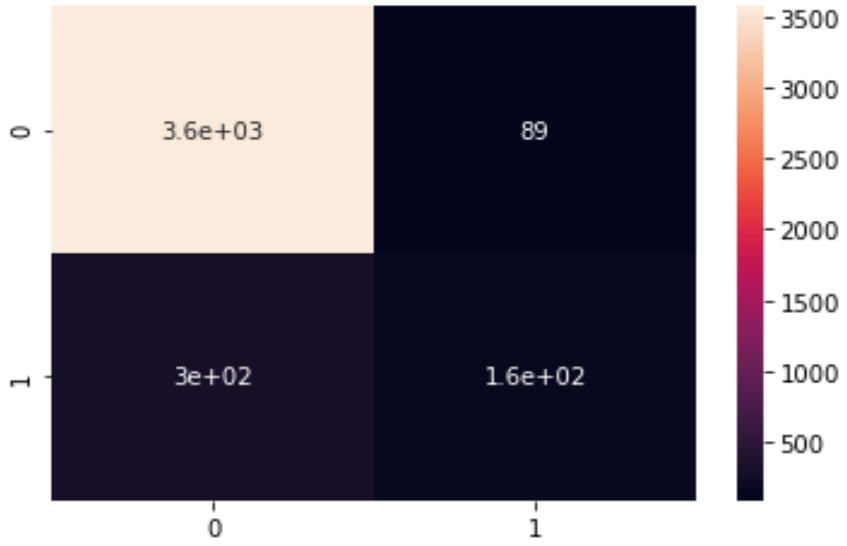
Wykres przedstawiający dokładność modelu w zależności od maksymalnej liczby iteracji:



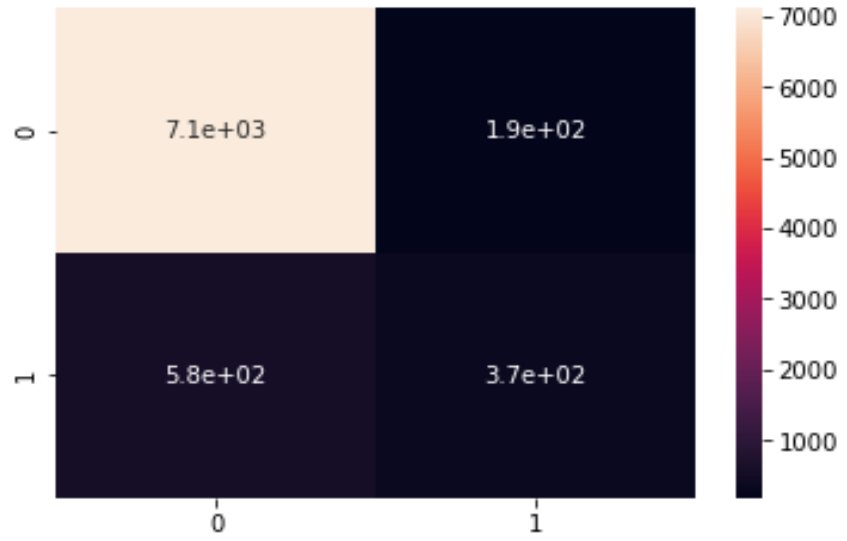
Maksymalne iteracje są potęgami liczby 3 do 729 włącznie.

Po przeanalizowaniu wykresu można wywnioskować, że dokładność wzrasta wraz ze wzrostem liczby iteracji, aż zatrzymuje się na poziomie 91,64% i utrzymuje się pomimo zwiększania liczby iteracji. Przeprowadzone zostały również badania wpływu rozkładu danych na zbiory testujące i trenujące:

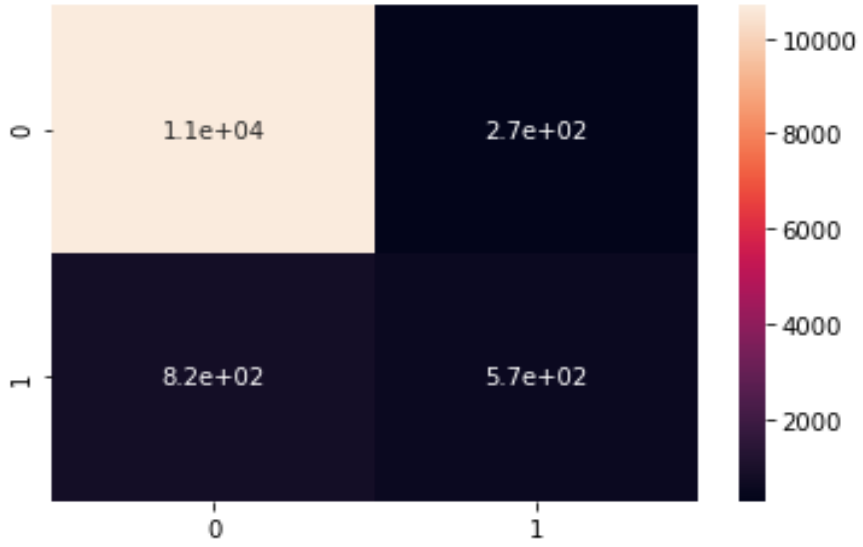
```
Model LogisticRegression basic - Test data = 10 %
Model accuracy is 0.91
[[3570   89]
 [ 295  165]]
```



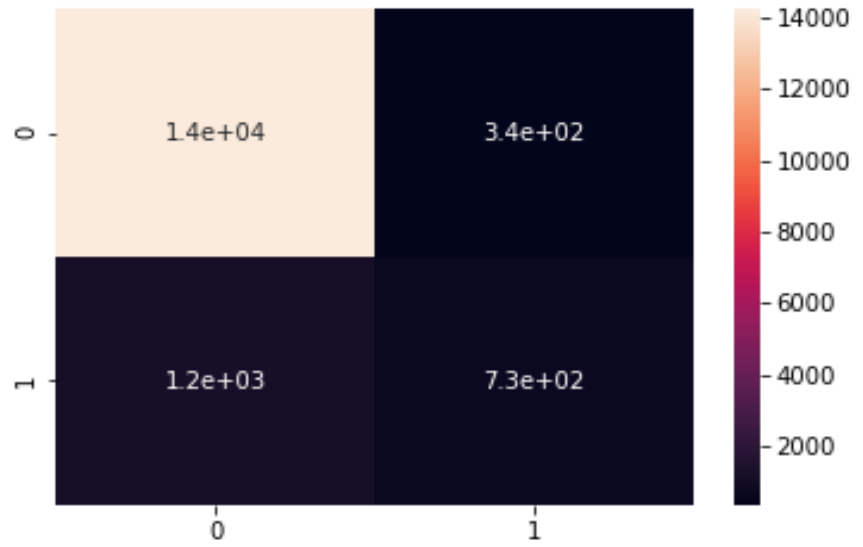

```
Model LogisticRegression basic - Test data = 20 %
Model accuracy is 0.91
[[7107  189]
 [ 575  367]]
```



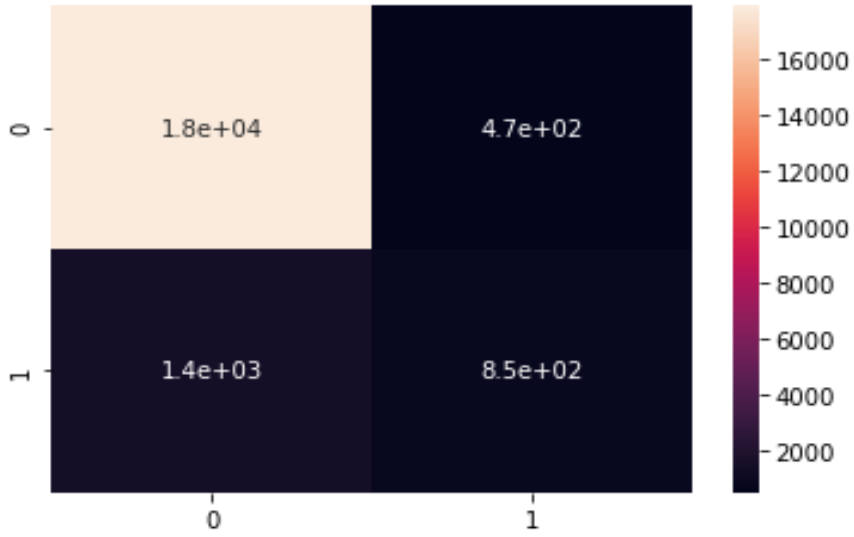
```
Model LogisticRegression basic - Test data = 30 %
Model accuracy is 0.91
[[10689   274]
 [  820   574]]
```




```
Model LogisticRegression basic - Test data = 40 %
Model accuracy is 0.91
[[14241   341]
 [ 1167   727]]
```



```
Model LogisticRegression basic - Test data = 50 %
Model accuracy is 0.91
[[17869   473]
 [ 1399   853]]
```



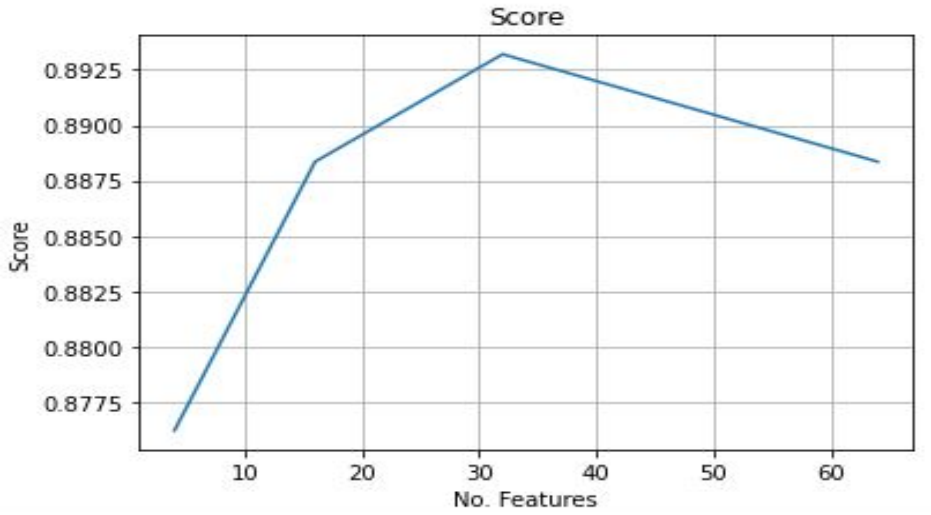
Zwiększając zbiór danych testujących wynik pozostaje bez zmian. Sprawdzony został także wpływ rodzaju algorytmu optymaliza-

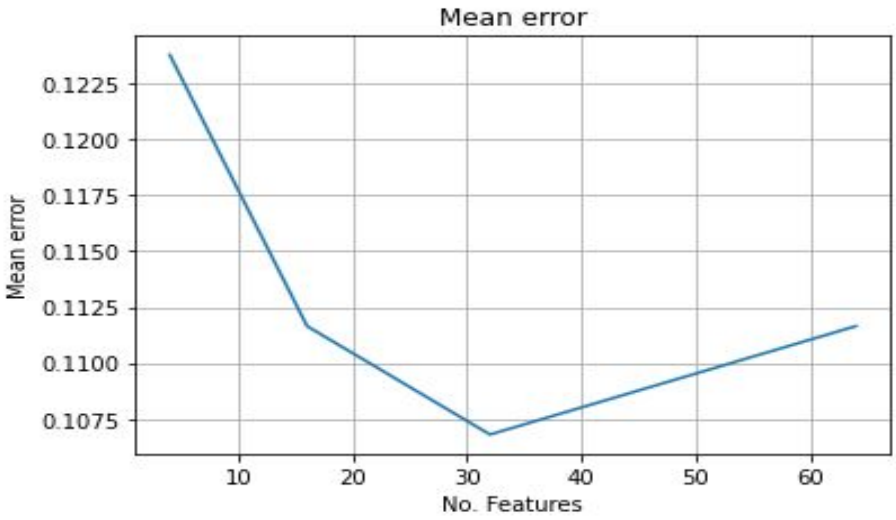
cyjnego na rezultat działania modelu:

Zmiana algorytmu optymalizacyjnego nie wpłynęła na wynik modelu.

Do testów z wykorzystaniem Polynomial Features i RFE (rekurencyjnej eliminacji cech) użyliśmy mniejszego zestawu danych, ponieważ czas wykonywania testu rośnie wykładniczo wraz z rozmiarem zbioru.
Za pomocą Polynomial Features zestaw cech jest podnoszony do potęgi (w tym przypadku jest to 2).Rekurencyjna eliminacja atrybutów polega na wybieraniu podzbioru najważniejszych cech dla zbioru danych. Mniejsza liczba cech może umożliwić bardziej wydajne działanie algorytmów uczenia maszynowego (mniejsza złożoność przestrzenna lub czasowa) i większą skuteczność. Niektóre algorytmy uczenia maszynowego mogą być wprowadzane w błąd przez nieistotne połączenia atrybutów.

Na poniższych wykresach można dostrzec zależność wyniku (dokładności) oraz średniego błędu. Test był przeprowadzony podając różne ilości atrybutów (4, 16, 32, 64). Można zauważyć minimum błędu przy 32 atrybutach, dalej dla 64 atrybutów niewielki wzrost. Należy mieć na uwadze jednak wyskalowanie wykresu - różnice błędów zależne od ilości wybranych atrybutów (co za tym idzie też wyników) są niewielkie, zauważalne zmiany na drugim miejscu po przecinku, czyli procentowy rząd wielkości.
Analizując wyniki testów należy stwierdzić, że rekurencyjna eliminacja cech nie poprawia wyniku trenowania modelu.




Mean error
Mean error
0.1225
0.1200
0.1175
0.1150
0.1125
0.1100
0.1075
10
20
30
40
50
60
No. Features

Rozdział 3

Podsumowanie

Klasyfikator wytrenowany przez nas dobrze radzi sobie z analizą zadanych danych. Najlepszym uzyskanym wynikiem dokładności było około 92 procent, co jest zadowalające.
Skalowanie zbioru nieznacznie polepszyło wynik modelu.
Dla podstawowej regresji logistycznej optymalną ilością iteracji było około 80, zwiększanie iteracji nie dawało lepszego efektu.
Manipulowanie rozmiarami zbiorów testowych i trenujących nie niesie za sobą zmian w dokładności.
Algorytm optymalizujący również nie ma wpływu na skuteczność klasyfikatora.
RFE poskutkowało nieznacznym spadkiem dokładności klasyfikatora.

Mimo doskonalenia procesu uczenia maszynowego, należy zawsze brać pod uwagę tzw. "czynnik ludzki", czyli możliwość pojedynczych odstępstw od ogólnie wypracowanego modelu.

Dodatek A

Kod programu

```
#imports
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.feature_selection import RFE

----------------------------------------

# load data set
bank_data = pd.read_csv("bank-additional-full.csv", sep=';')
print("Shape of DataFrame is ", bank_data.shape)

----------------------------------------

# Closer look at data
bank_data.head().T

bank_data.info()

# Statistical summary of our dataset
bank_data.describe().T
```

```
# iterated through bank_data and stored data with datatype as 'object'
#to new variable cat_col

cat_col = [n for n in bank_data.columns if bank_data[n].dtypes == 'object']
cat_col.remove('y')

----------------------------------------

fig, axs = plt.subplots(3, 3, sharex=False, sharey=False, figsize=(20, 15))

counter = 0
for col in cat_col:
    value_counts = bank_data[col].value_counts()

    trace_x = counter // 3
    trace_y = counter % 3
    x_pos = np.arange(0, len(value_counts))

    axs[trace_x, trace_y].bar(x_pos, value_counts.values, tick_label
    = value_counts.index)

    axs[trace_x, trace_y].set_title(col)

    for tick in axs[trace_x, trace_y].get_xticklabels():
        tick.set_rotation(90)

    counter += 1

plt.show()
#plt.savefig('dataset_cat_columns_exploration.png')

----------------------------------------

num_columns = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate',
              'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr_employed']

fig, axs = plt.subplots(2, 3, sharex=False, sharey=False, figsize=(20, 15))

counter = 0
for num_column in num_columns:

    trace_x = counter // 3
    trace_y = counter % 3
```

```
    axs[trace_x, trace_y].hist(bank_data[num_column])

    axs[trace_x, trace_y].set_title(num_column)

    counter += 1

plt.show()
#plt.savefig("dataset_num_columns_exploration.png")

----------------------------------------

# column y has unbalanced data
bank_data['y'].value_counts().plot.bar()
----------------------------------------

for col in cat_col:
    pd.crosstab(bank_data[col], bank_data.y).plot(kind = 'bar')
    plt.title(col)
    #plt.savefig(col + ".png")
----------------------------------------

plt.figure(figsize = (10, 6))
plt.grid(b = True)
sns.distplot(a = bank_data['age'], kde = False)

----------------------------------------

data_heatmap = bank_data.loc[:, bank_data.columns != 'y']
sns.heatmap(data_heatmap.corr(),cmap="YlGnBu")
plt.show()

----------------------------------------

target_heatmap = bank_data.loc[:, bank_data.columns == 'y']
data_heatmap['Subscription'] = target_heatmap
fig, ax = plt.subplots(figsize=(12,12))
sns.heatmap(data_heatmap.corr()[['Subscription']], cmap='YlGnBu', annot=True);
ax.invert_yaxis()

----------------------------------------

# Transform 'yes' and 'no'  into (0,1) dependency
bank_data['y']=[1 if i=='yes' else 0 for i in bank_data['y']]
```

```
-----------------------------------------

# Create dummy variable for the categorical data and drop first column
# to avoid dummy varibale trap
bank_data = pd.get_dummies(bank_data,columns = ['job','marital','education',
                                                'default', 'housing','loan',
                                                'month', 'contact',
                                                'day_of_week','poutcome'],
                                                drop_first = True)
bank_data.shape
bank_data.head().T

-----------------------------------------

# split dataset into data and target sets
data = bank_data.loc[:, bank_data.columns != 'y']
target = bank_data.loc[:, bank_data.columns == 'y']

data_np = np.array(data)
target_np = np.array(target)
#target_np = target_np.ravel()

print(data_np.shape)
print(target_np.shape)

-----------------------------------------

# split data and target sets into 4 sets for train and test purposes
bank_train_data, bank_test_data, \
bank_train_target, bank_test_target = \
train_test_split(data_np, target_np, test_size=0.1)

bank_train_data_basic = np.copy(bank_train_data)
bank_test_data_basic = np.copy(bank_test_data)

-----------------------------------------

# scale data
sc = StandardScaler()
bank_train_data = sc.fit_transform(bank_train_data)
bank_test_data = sc.transform(bank_test_data)

-----------------------------------------
```

```
# Training our model
logistic_regression = LogisticRegression()
logistic_regression.fit(bank_train_data, bank_train_target)

log_reg_basic = LogisticRegression()
log_reg_basic.fit(bank_train_data_basic, bank_train_target.ravel())

-----------------------------------------

# Evaluate accuracy of our model
print("Basic Logistic Regression")
accu_score = accuracy_score(bank_test_target,
log_reg_basic.predict(bank_test_data_basic))
print("Basic model accuracy = {}".format(accu_score)

-----------------------------------------

# Evaluate accuracy of our model
print("Basic Logistic Regression with scaled data")
accu_score = accuracy_score(bank_test_target,
logistic_regression.predict(bank_test_data))
print("Basic model accuracy of scaled data= {}".format(accu_score))

# confusion matrix
conf_matrix = confusion_matrix(bank_test_target,
logistic_regression.predict(bank_test_data))
print(conf_matrix)

sns.heatmap(conf_matrix, annot=True)
plt.show()

-----------------------------------------

def cross_valid_data(numbOfIteration):
    print("\nCross walidacja:")
    scores = cross_val_score(LogisticRegression(),
    data_np, target_np.ravel(), cv=numbOfIteration)
    plt.figure(figsize=(1, 5))
    sns.heatmap(scores[:, np.newaxis],annot=True)
    plt.show()

def compare_given_result_with_real(left_bound, right_bound):
    for i in range(left_bound, right_bound):
        pred = logistic_regression.predict(bank_test_data[i, :].reshape(1, -1))
```

```
        print("\nModel predicted for client {0} value {1}".format(i, pred))
        print("Real value for client \"{0}\" is {1}".format(i, bank_test_target[i]))
        #Wyświetlenie
        print(logistic_regression.predict_proba(bank_test_data[i,:].reshape(1,-1)))

compare_given_result_with_real(0, 10)
cross_valid_data(10)

-----------------------------------------

%matplotlib inline
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

bank_data2 = pd.read_csv("bank-additional.csv", sep=';')
data2 = bank_data2.loc[:, bank_data2.columns != 'y']

sns.pairplot(data2, diag_kind = "kde")

-----------------------------------------

# Mean squared error and variance score of a basic logistic model
from sklearn.metrics import mean_squared_error

m_squared_error = mean_squared_error(bank_test_target,
logistic_regression.predict(bank_test_data))
print("Mean squared error = %.2f" % m_squared_error)

var_score = logistic_regression.score(bank_test_data, bank_test_target)
print("Logistic Regression variance score: %.2f" % var_score)


-----------------------------------------

i = 3
x = []
y_acc = []

while i <= 729:
    x.append(i)
    logistic_regression = LogisticRegression(max_iter=i)
    logistic_regression.fit(bank_train_data, bank_train_target.ravel())
```

```
    acc = accuracy_score(bank_test_target,
    logistic_regression.predict(bank_test_data))
    y_acc.append(acc)
    i *= 3

plt.plot(x, y_acc)
plt.title("Accuracy score per iteration")
plt.grid()
plt.xlabel("Max number of iterations")
plt.ylabel("Accuracy score")

----------------------------------------

for size in range (1,6):
    target_np = target_np.ravel()

    bank_train_data, bank_test_data, \
    bank_train_target, bank_test_target = \
    train_test_split(data_np, target_np, test_size=(size/10))

    log_reg = LogisticRegression()
    log_reg.fit(bank_train_data, bank_train_target)

    #Skuteczność naszego modelu
    print("Model LogisticRegression basic - Test data = {} %".format(size*10))
    acc = accuracy_score(bank_test_target, log_reg.predict(bank_test_data))
    print("Model accuracy is {0:0.2f}".format(acc))
    conf_matrix = confusion_matrix(bank_test_target,
     log_reg.predict(bank_test_data))
    print(conf_matrix)
    sns.heatmap(conf_matrix,annot=True)
    plt.show()
    plt.savefig("dataset" + str(size*10) + ".png")

----------------------------------------

solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']


for sol in solvers:
    scores = cross_val_score(LogisticRegression(solver=sol),
     bank_train_data, bank_train_target, cv=10)
    plt.figure(figsize=(1,4))
    sns.heatmap(scores[:, np.newaxis],annot=True)
```

```
    print("Solver:" + sol)
    plt.show()

----------------------------------------

from sklearn.preprocessing import PolynomialFeatures
from sklearn.feature_selection import RFE
from sklearn.metrics import mean_squared_error

bank_data2 = pd.read_csv("bank-additional.csv", sep=';')

bank_data2['y']=[1 if i=='yes' else 0 for i in bank_data2['y']]

bank_data2 = pd.get_dummies(bank_data2,columns =
['job','marital','education','default',
   'housing','loan','month', 'contact',
   'day_of_week','poutcome'], drop_first = True)

data2 = bank_data2.loc[:, bank_data2.columns != 'y']
target2 = bank_data2.loc[:, bank_data2.columns == 'y']

data2 = np.array(data2)
target2 = np.array(target2)

bank_train_data, bank_test_data, \
bank_train_target, bank_test_target = \
train_test_split(data2, target2, test_size=0.1)

pf = PolynomialFeatures(2, )

bank_train_poly = pf.fit_transform(bank_train_data)
bank_test_poly = pf.fit_transform(bank_test_data)

print("train poly size {}".format(bank_train_poly.size))
print("test poly size {}".format(bank_test_poly.size))

----------------------------------------

x = [4, 16, 32, 64]
y_mean_error = []
y_r2_score = []

for feat in x:
    sel_ = RFE(estimator=LogisticRegression(solver='lbfgs',
```

```
    max_iter=300), n_features_to_select=feat)
    sel_.fit(bank_train_poly, bank_train_target)
    y_mean_error.append(mean_squared_error(bank_test_target,
    sel_.predict(bank_test_poly)))
    y_r2_score.append(sel_.score(bank_test_poly, bank_test_target))

plt.plot(x, y_mean_error)
plt.title("Mean error")
plt.grid()
plt.xlabel("No. Features")
plt.ylabel("Mean error")
plt.show()

plt.plot(x,y_r2_score)
plt.title("Score")
plt.grid()
plt.xlabel("No. Features")
plt.ylabel("Score")
plt.show()
```