PROJEKT

Uczenie sieci neuronowej dla problemu klasyfikacji klientów banku (Bank Marketing Data)

Wykonali:
Adrianna Baranowska
Bartłomiej Smalec

1 Spis treści:

2 Wstęp:

Celem projektu jest stworzenie modelu sieci neuronowej zdolnej do przewidywania czy dana osoba zostanie odbiorcą usług oferowanych przez dany bank.

3 Zbiór danych:

Zbiór danych na których jest trenowana nasza sieć neuromowa, pochodzi ze zbioru „UCI Machine Learning Repository” i zawiera realne dane z Portugalskiego Banku. Zbiór ten składa się z 17 kolumn i 45211 wierszy i zawiera w sobie 2 rodzaje danych numeryczne i słowne.

Numeryczne dane:

- Age (Wiek)

- Balance (Stan konta)

- Day (Ostatnni dzień tygodnia w których się kontaktował z bankiem)

- Duration (Czas trwana ostaniego kontaktu)

- Campaign (Liczba połączeń w trakcie trwania kampanii)

- Pdays (Liczba dni która upłyneła od ostatniego kontaku w poprzedniej kampanii marketingowej)

- Previous (Liczba kontaktów która upłyneła przed tą kampanią marketingową)

- y (Przybiera 1 jeśli klient został odbiorcą usług oferowanych przez bank, 0 jeśli nie. Jest to nasz target)

Słowne dane:

- Maritial (Stan cywilny)

- Job (Rodzaj pracy lub jej brak)

- Contact (Typ kontaktu komórka/stacjonarny)

- Education (Typ edukacjii potencjalnego klienta)

- Month (Miesiąc ostatniego kontaktu z klientem)

- Poutcome (Wynik poprzedniej kampanii marketingowej)

- Housing (Ma wartość 1 jeżeli klient posiada kredyt na nieruchomość, 0 jeśli nie posiada)

- Loan (Ma wartość 1 jeżeli klient posiada pożyczke, 0 jeśli nie posiada)

- Default (Ma wartość 1 jeżeli klient posiada już kredyt w banku, 0 jeśli nie posiada)

Ponieważ część z danych jest słowna, skorzystamy z narzędzia LabelEncoder() z biblioteki sklearn aby zaminić je na wartości numeryczne.

```
def preprocess_data(df):
    df.columns = [col.replace('"', '') for col in df.columns]
    df.drop(columns=['day', 'poutcome'], axis=1, inplace=True)

    le = preprocessing.LabelEncoder()
    df.job = le.fit_transform(df.job)
    df.education = le.fit_transform(df.education)
    df.housing = le.fit_transform(df.housing)
    df.loan = le.fit_transform(df.loan)
    df.month = le.fit_transform(df.month)
    df.contact = le.fit_transform(df.contact)
    df.marital = le.fit_transform(df.marital)
    df.default = le.fit_transform(df.default)
    df.y = le.fit_transform(df.y)
    return df
```

- Następne sprawdzamy czy któraś kolumna jest pusta „`pd.isnull(df).any()`”:

```
Is null

age          False
job          False
marital      False
education    False
default      False
balance      False
housing      False
loan         False
contact      False
month        False
duration     False
campaign     False
pdays        False
previous     False
y            False
```

- Kolejno sprawdzany dla każdej kolumny jej unikalność + jej opis za pomocą „df[i].nunique()” + „df[i].describe()” :

```
Number of unique values for "age"  is: 77
============================================================
Describe for "age" is: count    45211.000000
mean        40.936210
std         10.618762
min         18.000000
25%         33.000000
50%         39.000000
75%         48.000000
max         95.000000
Name: age, dtype: float64
Number of unique values for "job"  is: 12
============================================================
Describe for "job" is: count    45211.000000
mean         4.339762
std          3.272657
min          0.000000
25%          1.000000
50%          4.000000
75%          7.000000
max         11.000000
Name: job, dtype: float64
Number of unique values for "marital"  is: 3
============================================================
Describe for "marital" is: count    45211.000000
mean         1.167725
std          0.608230
min          0.000000
25%          1.000000
50%          1.000000
75%          2.000000
max          2.000000
Name: marital, dtype: float64
Number of unique values for "education"  is: 4
============================================================
Describe for "education" is: count    45211.000000
```

mean   1.224813

std   0.747997

min   0.000000

25%   1.000000

50%   1.000000

75%   2.000000

max   3.000000

Name: education, dtype: float64

Number of unique values for "default" is: 2

==========

Describe for "**default**" is: count   45211.000000

mean   0.018027

std   0.133049

min   0.000000

25%   0.000000

50%   0.000000

75%   0.000000

max   1.000000

Name: default, dtype: float64

Number of unique values for "balance" is: 7168

==========

Describe for "**balance**" is: count   45211.000000

mean   1362.272058

std   3044.765829

min   -8019.000000

25%   72.000000

50%   448.000000

75%   1428.000000

max   102127.000000

Name: balance, dtype: float64

Number of unique values for "housing" is: 2

==========

Describe for "**housing**" is: count   45211.000000

mean   0.555838

std   0.496878

min   0.000000

25%   0.000000

50%   1.000000

75%   1.000000

max   1.000000

Name: housing, dtype: float64

Number of unique values for "loan" is: 2

==========

Describe for "**loan**" is: count   45211.000000

mean   0.160226

std   0.366820

min   0.000000

25%   0.000000

50%   0.000000

75%   0.000000

max   1.000000

Name: loan, dtype: float64

Number of unique values for "contact" is: 3

==========

Describe for "**contact**" is: count   45211.000000

mean   0.640242

std   0.897951

min   0.000000

25%   0.000000

50%   0.000000

75%   2.000000

max   2.000000

Name: contact, dtype: float64

Number of unique values for "month" is: 12

==========

Describe for "**month**" is: count   45211.000000

mean   5.523014

std   3.006911

min   0.000000

25%   3.000000

50%   6.000000

75%   8.000000

max   11.000000

Name: month, dtype: float64

Number of unique values for "duration" is: 1573

==========

Describe for "**duration**" is: count   45211.000000

mean   258.163080

std   257.527812

min   0.000000

25%   103.000000

50%   180.000000

75%   319.000000

max   4918.000000

Name: duration, dtype: float64

Number of unique values for "campaign" is: 48

==========

Describe for "**campaign**" is: count   45211.000000

mean   2.763841

std   3.098021

min   1.000000

25%   1.000000

50%   2.000000

75%   3.000000

max   63.000000

Name: campaign, dtype: float64

Number of unique values for "pdays" is: 559

==========

Describe for "**pdays**" is: count   45211.000000

mean   40.197828

std   100.128746

min   -1.000000

25%   -1.000000

50%   -1.000000

75%   -1.000000

max   871.000000

Name: pdays, dtype: float64

Number of unique values for "previous" is: 41

==========

Describe for "**previous**" is: count   45211.000000

mean   0.580323

std   2.303441

min   0.000000

25%   0.000000

```
50%          0.000000
75%          0.000000
max        275.000000
Name: previous, dtype: float64
Number of unique values for "y"  is: 2
============================================================
Describe for "y" is: count    45211.000000
mean         0.116985
std          0.321406
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: y, dtype: float64
```

Po wykonaniu tych operacji nasz zbiór danych to macierz o rozmiarach (45211, 15)

4 Normalizacja danych:

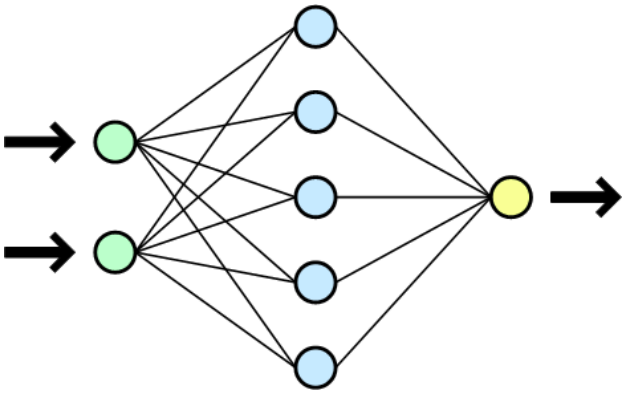
Normalizacja to proces skalowania pojedynczych próbek w celu otrzymania małego, specyficznego przedziału. Przykładowo przekształcamy dane wejściowe w taki sposób, aby mieściły się w przedziale [-1, 1] lub [0, 1]. Normalizacje wykonuje za pomocą polecenia:

```
scaler = Normalizer()
X = scaler.fit_transform(X)
```

5 Opis stosowanego narzędzia i metod jego uczenia:

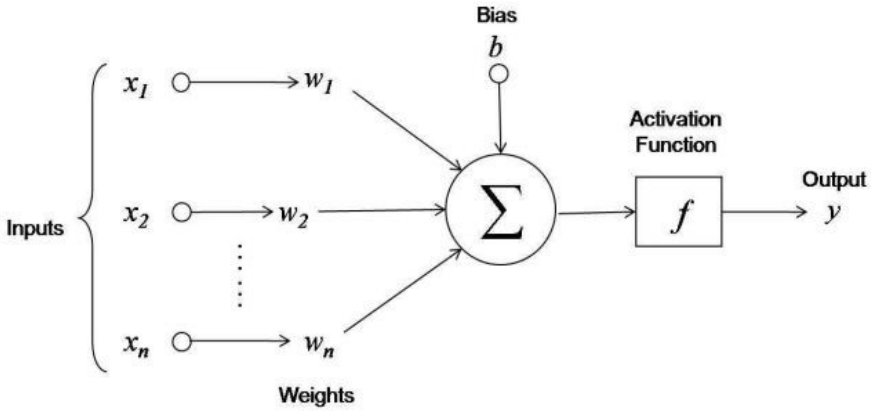
Sieć neuronowa to połączenie elementów zwanych sztucznymi neuronami, które tworzą co najmniej trzy warstwy: wejściową, ukrytą i wyjściową, przy czym warstw ukrytych może być wiele. Neurony sieci przetwarzają informacje dzięki temu, że ich połączeniom nadaje się parametry, zwane wagami, które modyfikuje się podczas działania sieci. Modyfikowanie wag nazywane jest „uczeniem się” sieci. Istnieją różne sieci neuronowe: rekurencyjne, konwulencyjne, „feedforward”.

Przykładowy schemat sieci neuronowej:



Neuron: jest to prosty system przetwarzający wartości sygnałów wprowadzanych na jego wejścia w pojedynczą wartość wyjściową, wysyłaną na jego jedynym wyjściu (dokładny sposób funkcjonowania określony jest przez przyjęty model neuronu). Jest to podstawowy element sieci neuronowych jednej z metod sztucznej inteligencjii, inspiracją dla budowy i dzialania sztucznego neuronu był neuron biologiczny.

Przykładowy schemat neuronu:



Jak widać neuron składa się z wejść, wag, biasu, funkcjii sumującej, funkcjii aktywacjii i wyjśćia.

Proces uczenia się sieci neuronowej:

1. Losowa inicjalizacja wag.

2. Wprowadzenie danych do sieci neuronowej i wykonianie „feedforward”

3. Porównanie tych wartośći z poprawnymi wynikami i obliczenia błędu.

4. Wykonanie wstecznej propagacjii błędu

5. Aktualizowanie wartości wag za pomocą gradientu prostego lub jego wariacjii, tak aby minimalizować otrzymany błąd.

6. Powtarzać powyższe punkty, aż nasz błąd zadowalająco mały.

6 Cross Validation:

```
start = time()
model = KerasClassifier(build_fn=nnmodel.create_model, verbose=1)
optimizers = ['rmsprop', 'adam','sgd']
activations = ['relu', 'selu', 'tanh']
neurons_one = [10, 32, 64]
neurons_two = [20, 64, 128]
epochs = np.array([10, 20, 30])
batches = np.array([5, 10, 20])
param_grid = dict(optimizer=optimizers, nb_epoch=epochs, batch_size=batches,
                  neuron_in_first=neurons_one, neuron_in_second=neurons_two)
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1)
grid_result = grid.fit(x_train, y_train)

print("total time:", time() - start)
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
for params, mean_score, scores in grid_result.cv_results_:
    print("%f (%f) with: %r" % (scores.mean(), scores.std(), params))
```

Wynik corss walidacjii:

```
Best: 0.886210 using {'batch_size': 5, 'nb_epoch': 30, 'optimizer': 'rmsprop', 'neuron_in_first': 32, 'neuron_in_second': 64}
```

Schemat modelu sieci neuronowej:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 32)                480
_________________________________________________________________
dense_2 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65
=================================================================
Total params: 2,657
Trainable params: 2,657
Non-trainable params: 0
```

7 Analiza jak się zmieni jakoś narzędzie jeśli zmienimy parametry konfiguracyjne:

Jakość sieci neuronowej nie zmienia się znacznie pomimo zman liczby neuronów w każdej warstwie, dodanie warstwy dropout, zmian funcjii aktywacji i sposobu liczenia błędu. Skuteczność sieci to prawie zawsze **0.88.**

8 Kod

main.py

```
def main():
    data = bm.load_data('bank-full.csv')
    data = bm.preprocess_data(data)
    X,y = bm.split_data(data)

    scaler = Normalizer()
    X = scaler.fit_transform(X)

    x_train, x_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0.1, random_state=0)


    start = time()
    model = KerasClassifier(build_fn=nnmodel.create_model, verbose=1)
    optimizers = ['rmsprop', 'adam','sgd']
    activations = ['relu', 'selu', 'tanh']
    neurons_one = [10, 32, 64]
    neurons_two = [20, 64, 128]
    epochs = np.array([10, 20, 30])
    batches = np.array([5, 10, 20])
    param_grid = dict(optimizer=optimizers, nb_epoch=epochs, batch_size=batches,
                      neuron_in_first=neurons_one, neuron_in_second=neurons_two)
    grid = GridSearchCV(estimator=model, param_grid=param_grid_, n_jobs=_-1)
    grid_result = grid.fit(x_train, y_train)

    print("total time:", time() - start)
    print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
    for params, mean_score, scores in grid_result.cv_results_:
        print("%f (%f) with: %r" % (scores.mean(), scores.std(), params))

    bestmodel = nnmodel.best_model()
    bestmodel.fit(x_train, y_train, epochs=30, batch_size=5)
    scores = bestmodel.evaluate(x_test, y_test)
    print("{} , {}".format(bestmodel.metrics_names[1], scores[1] * 100))

if __name__ == "__main__":
    main()
```

NNmodel.py

```
def create_model(optimizer='rmsprop', neuron_in_first=32, neuron_in_second=64, dropout_in_first=0, dropout_in_second=0,
                 activation_in_first='relu', activation_in_second='relu'):
    nn = Sequential()
    nn.add(Dense(neuron_in_first, input_dim=14, activation=activation_in_first))
    nn.add(Dense(neuron_in_second, activation=activation_in_second))

    nn.add(Dense(1, activation='sigmoid'))

    nn.compile(loss=keras.losses.binary_crossentropy,
               optimizer=optimizer,
               metrics=['accuracy'])
    return nn


def best_model():
    nn = Sequential()
    nn.add(Dense(32, input_dim=14, activation='relu'))
    nn.add(Dense(64, activation='relu'))

    nn.add(Dense(1, activation='sigmoid'))

    nn.compile(loss=keras.losses.binary_crossentropy,
               optimizer='rmsprop',
               metrics=['accuracy'])
    return nn
```

BankMarketingData.py

```
def load_data(name):
    df = pd.read_csv(name, sep=';')
    print(df.head())

    print(df.shape)

    print(df.columns)
    print(df.info)
    print(df.head())
    print(df.shape)
    return df

def preprocess_data(df):
    df.columns = [col.replace('"', '') for col in df.columns]
    df.drop(columns=['day', 'poutcome'], axis=1, inplace=True)

    le = preprocessing.LabelEncoder()
    df.job = le.fit_transform(df.job)
    df.education = le.fit_transform(df.education)
    df.housing = le.fit_transform(df.housing)
    df.loan = le.fit_transform(df.loan)
    df.month = le.fit_transform(df.month)
    df.contact = le.fit_transform(df.contact)
    df.marital = le.fit_transform(df.marital)
    df.default = le.fit_transform(df.default)
    # df.poutcome = le.fit_transform(df.poutcome)
    df.y = le.fit_transform(df.y)
    print(df.shape)
    return df
def split_data(df):
    X = df.iloc[:, 0:14]
    y = df.iloc[:, 14]
    X = np.array(X, dtype="float64")
    y = np.array(y, dtype="float64")
    return X,y

def describe_data(df):
    print("Is null")
    print(pd.isnull(df).any())
    for i in df.columns:
        #print("Value counts for \"{}\" is: {}".format(i, df[i].value_counts()))
        print("Number of unique values for \"{}\"  is: {}".format(i, df[i].nunique()))
        print("============================================================")
        print("Describe for \"{}\" is: {}".format(i, df[i].describe()))
```