Neuronowy model dawcy krwi

KWD

Krzysztof Michalski

Rafał Pokrywka

28.01.2020

Streszczenie

W tym projekcie zajmowaliśmy się przewidywaniem, czy pacjent odda krew w marcu 2007, na podstawie danych z Blood Transfusion Service Center Data Set. Było to zatem zadanie klasyfikacji binarnej. Zastosowaliśmy kilka modeli - regresję liniową, regresję z użyciem drzew decyzyjnych i lasów losowych oraz w pełni połączoną sieć neuronową. Modele zostały poddane analizie skuteczności i na jej podstawie najlepszy okazał się model sieci neuronowej z dokładnością na zbiorze testowym dla danych niezrównoważonych około 85.3% , a dla danych zrównoważonych 75%.

Spis treści

Rozdział 1

Wprowadzenie

Dane do projektu uzyskaliśmy ze strony Blood Transfusion Service Center Data Set. Składały się one z dwóch plików *transfusion.data* i *transfusion.names*.

W pliku *transfusion.data* znajdowało się 748 przykładów opisanych 4 cechami:

- R (Recency - months since last donation)
- F (Frequency - total number of donation)
- M (Monetary - total blood donated in c.c.)
- T (Time - months since first donation)

oraz jedną etykietą - binary variable representing whether he/she donated blood in March 2007 (1 stand for donating blood; 0 stands for not donating blood). Zatem zadanie można zaliczyć do kategorii klasyfikacji binarnej. Klasy te były słabo zrównoważone - klasę 1 posiadało około 24% przykładów, a klasę 0 - 76%.

W pliku *transfusion.names* znajdował się opis danych - źródło, opis atrybutów oraz autorzy.

1.1 Opis problemu

Zadanie polega na klasyfikacji binarnej przykładów na podstawie 4 cech numerycznych. Celem klasyfikacji, jest określenie, czy nowy pacjent kliniki odda krew po pewnym okresie czasu. W tym celu można wykorzystać różne techniki uczenia maszynowego takie jak regresja liniowa, drzewa decyzyjne, lasy losowe czy sieci neuronowe. Ze względu na niewielką ilość przykładów oraz cech je opisujących, skuteczność takiej klasyfikacji może być niewielka

- zbiory danych o takiej wielkości są niewystarczające dla dobrego wytrenowania na przykład sieci neuronowej. Ze względu na niezrównoważenie klas, modele mogą mieć tendencję do częstszego przewidywania dominującej klasy - bo zapewnia to dużą dokładność takiego modelu. Można spróbować zrównoważyć zbiór wybierając podzbiór przykładów o takiej samej reprezentacji klasy pozytywnej i negatywnej.

Rozdział 2

Opis metody

2.1 Wprowadzenie teoretyczne

W celu rozwiązania zadania wykorzystaliśmy kilka metod uczenia maszynowego:

Regresję logistyczną - czyli metodę służącą do klasyfikacji binarnej, pozwalającą określić prawdopodobieństo przynależności przykładu do jednej z dwóch klas. Jeżeli prawdopodobieństwo to przekracza 50% oznacza to, że model przewiduje, że dany przykład należy do tej klasy. Do nauki regregresora logistycznego wykorzystywana jest:

- Logarytmiczna funkcja straty, która wykonuje nieliniową transformacje wyjściowych wartości na przedział od 0 do 1, tak, żeby suma wartości obu klas wynosiła 1.
- Metoda spadku wzdłuż gradientu prostego - pozwalająca zbliżać się do minumum globalnego powyższej funkcji straty.
- Funkcję logistyczną - pozwalającą określić wartość prawdopodobieństwa dla każdej klasy.

Drzewa decyzyjne - czyli wszechstronny algorytm uczenia maszynowego, który można wykorzystywać do różnych zadań - zarówno klasyfikacji (binarnej i wieloklasowej) jak i do zadań regresji. Podejście tego modelu do klasyfikacji binarnej polega na dzieleniu każdej z cech danych na dwa zbiory w sposób liniowy - tak, żeby zbiory te były jak najlepiej rozdzielone. Oznacza to, że w podzbiorach przykładów po każdej ze stron tej tak zwanej granicy decyzyjnej, znadowało się jak najwięcej przykładów należących do jednej klasy - podzbiory te powinien być jak najmniej zanieczyszczone przykładami należącymi do drugiej grupy. Wysokość takiego drzewa określa ile takich liniowych podziałów cech przykładu może wykonać model.

Lasy losowe - jest to jedna z metod uczenia zespołowego. Polega ona na wytrenowaniu pewnej ilości drzew decyzyjnych, a następnie zastosowaniu jednej z technik głosowania drzew w celu określenia rezultatu. Głosowanie można przeprowadzać na przykład w sposób większościowy - w takim przypadku, każdy model określa, do której klasy według niego należy dany przykład - klasa określana przez zespół to ta, na którą głosowało najwięcej modeli.

Sieci neuronowe - czyli algorytm wielokrotnej zmiany reprezentacji cech określających przykłady, w celu znalezienia reprezentacji pozwalającej - w przypadku klasyfikacji - w jak największym stopniu rozdzielić od siebie przykłady należące do różnych klas. Sieci składają się z:

- Neuronów - które są elementami wykonującymi liniową transformację danych otrzymywanych na wejściu - sumę ważoną wejść wraz z tak zwanym członem obciążenia, niezależnym od wartości wejść.
- Warstw - które są złożone z neuronów. Neurony w ramach jednej warstwy nie są ze sobą połączone.
- Połączeń pomiedzy warstwami - czyli połączeniem danych wejściowych do neuronów pierwszej warstwy, a następnie wyjść neuronów jednej warstwy z wejściami neuronów kolejnej wastwy.
- Funkcji aktywacji - określających sposób modyfikacji reprezentacji danych przekazywanych pomiędzy warstwami. W przypadku klasyfikacji ostatnia warstwa stosuje funkcję softmax zwracającą prawdopodobieństwa przynależności danego przykładu do danej klasy.

Sieci neuronowe są trenowane z użyciem metody spadku wzdłuż gradientu prostego. Każdy krok takiej nauki określa się jako epokę danej sieci.

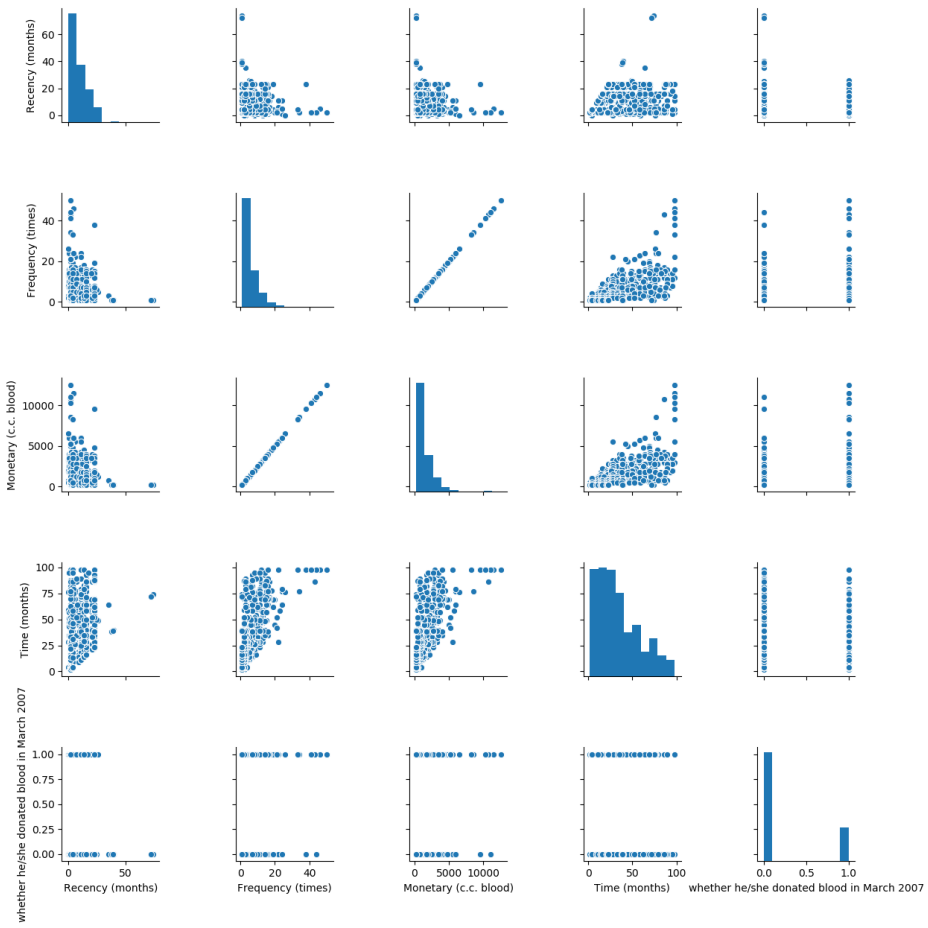
2.2 Badania symulacyjne

Na początku programu, w funkcji load_and_analyze_data, wczytaliśmy zbiór danych i wyświetliliśmy informacje na jego temat. Żeby zapoznać się z rozkładem danych, sprawdzilismy czy i jakie są korelacje pomiędzy poszczególnymi kolumnami, oraz jakie są minimalne, średnie i maksymalne wartości.

Analiza zbioru danych w wersji niezrównoważonej:

	R	F	M	T	D
R	1	-0.182745	-0.182745	0.160618	-0.279869
F	-0.182745	1	1	0.63494	0.218633
M	-0.182745	1	1	0.63494	0.218633
T	0.160618	0.63494	0.63494	1	-0.0358544
D	-0.279869	0.218633	0.218633	-0.0358544	1

	R	F	M	T	D
count	748	748	748	748	748
mean	9.50668	5.51471	1378.68	34.2821	0.237968
std	8.0954	5.83931	1459.83	24.3767	0.426124
min	0	1	250	2	0
25%	2.75	2	500	16	0
50%	7	4	1000	28	0
75%	14	7	1750	50	0
max	74	50	12500	98	1

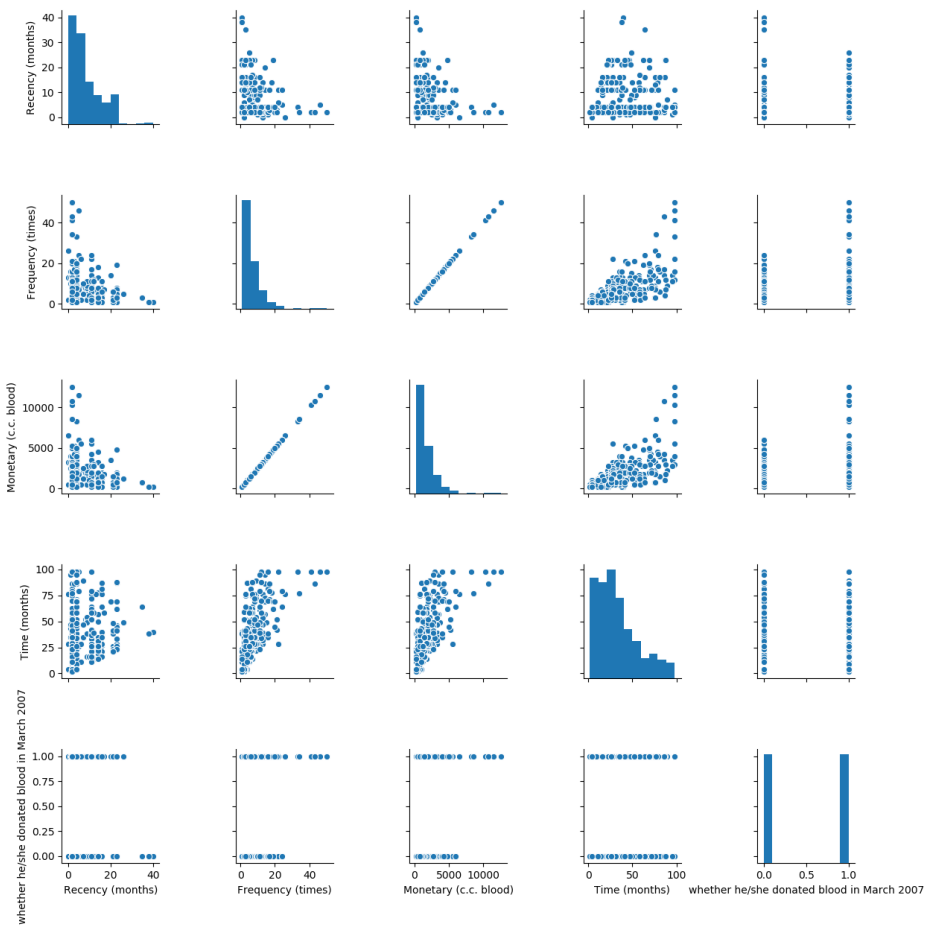


Ponieważ średnia wartość kolumny D (0 lub 1) jest równa 0.237968, to znaczy że tylko 24% wierszy ma jedynkę, czyli ilość przykładów dla klas jest niezrównoważona. Stworzyliśmy więc dodatkowy zbiór danych, który składa się z równej ilości przykładów dla każdej z klas.

Analiza zbioru danych w wersji zrównoważonej:

	R	F	M	T	D
R	1	-0.218071	-0.218071	0.0949278	-0.348375
F	-0.218071	1	1	0.685435	0.216987
M	-0.218071	1	1	0.685435	0.216987
T	0.0949278	0.685435	0.685435	1	-0.0152334
D	-0.348375	0.216987	0.216987	-0.0152334	1

	R	F	M	T	D
count	356	356	356	356	356
mean	7.95787	6.34551	1586.38	33.0815	0.5
std	7.19436	6.70222	1675.55	23.8207	0.500704
min	0	1	250	2	0
25%	2	2	500	16	0
50%	4	5	1250	28	0.5
75%	13.25	8	2000	46	1
max	40	50	12500	98	1



Ponieważ okazało się, że kolumny F i M są ze sobą idealnie skorelowane (F to liczba oddań krwi, a M to suma obiętości oddanej krwi), to możemy pozbyć się jednej z nich, a przewidywania programu nie ulegną zmianie. Następnie dane należało przeskalować, aby znajdowały się w jednym przedziale wartości, a ich średnia wynosiła zero. Dzięki temu każda cecha będzie traktowana w równy sposób przez algorytmy uczące.

Gdy dane są już odpowiednio przygotowane, można je podzielić na dwa zbiory: trenujący i testujący. W naszym programie 90% to będą dane trenujące, a 10% testujące.

W celu porównania różnych klasyfikatorów, sprawdzamy w pętli wszystkie podane modele (LogisticRegression, DecisionTreeClassifier, RandomForestClassifier). Wszystkie te klasyfikatory uczymy z różnymi metaparametrami, np. głębokością drzewa, ilością powtórzeń itp. Po wyuczeniu każdego z modeli, wypisujemy na wyjście podstawowe informacje o nim (np. dokładność, macierz konfuzji, ...) oraz wykres zależności dokładności od różnych parametrów danego modelu.

Później sprawdzamy również jak z tym problemem poradzi sobie sieć

neuronowa. Tworzymy sieć neuronową składającą się z 3 warstw gęstych, mających kolejno 8, 5 i 1 neuronów. Pierwsze dwie warstwy mają aktywację relu, a ostatnia sigmoid. Dodatkowo środkowa warstwa ma ustawiony dropout na 10% - w celu ograniczenia ryzyka przeuczenia sieci. Metaparametr learning rate ustawiamy na 0.003 - im mniejsza wartość, tym dokładniejsze rozwiązanie można znaleźć, ale jest również ryzyko utknięcia w minimum lokalnym.

Tak utworzoną sieć, zaczynamy uczyć na danych trenujących, z 1000 epok, oraz podając na których danych mają odbywać się testy. Ponieważ niektóre algorytmy najlepiej się uczą gdy w każdej epoce dane są podawane w innej kolejnści, to ustawiamy również parametr shuffle=True.

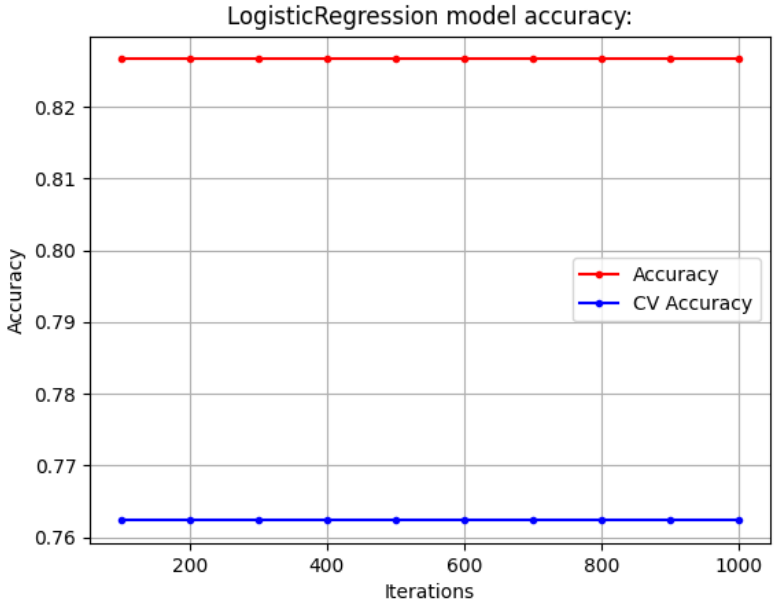
Po zakończonym trenowaniu, wypisujemy informacje o efektach: - jaka jest dokładność przewidywań na zbiorze treningowym, oraz na zbiorze testującym - jak wygląda macierz konfuzji - żeby zobaczyć jakiego rodzaju błędy najczęściej zostały popełnione (false positives, false negatives) oraz rysujemy wykresy dokładności w zależności od liczby epok uczenia.

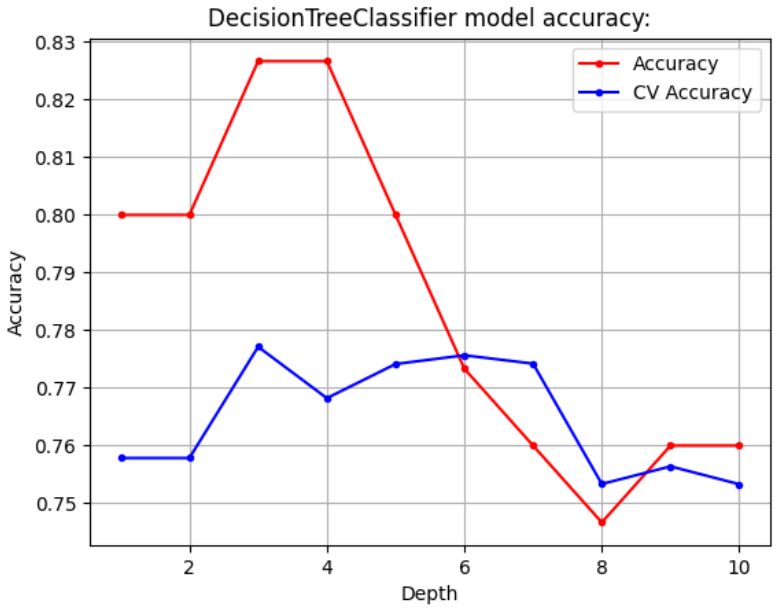
Wszystkie powyższe operacje wykonujemy dla obu zbiorów danych (niezrównoważonego i zrównoważonego).

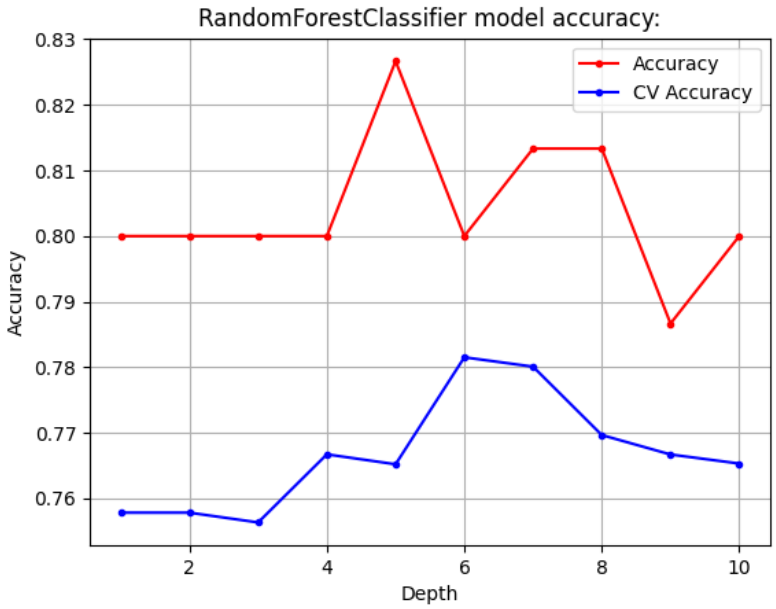
2.3 Wyniki symulacji dla różnych modeli

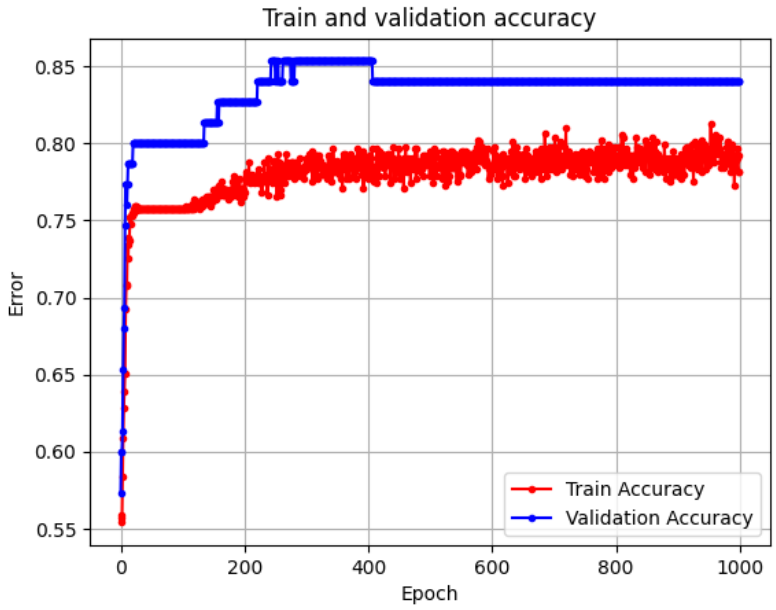
2.3.1 Dane niezrównoważone

Nazwa modelu	Dokładność	Dokładność CV
LogisticRegression	0.82666666	0.76233538191
DecisionTreeClassifier	0.82666666	0.77712906057
RandomForestClassifier	0.82666666	0.78151887620
NeuralNetwork	0.81277859	0.85333335399


LogisticRegression model accuracy:
Accuracy
CV Accuracy
Accuracy
Iterations
0.82
0.81
0.80
0.79
0.78
0.77
0.76
200
400
600
800
1000

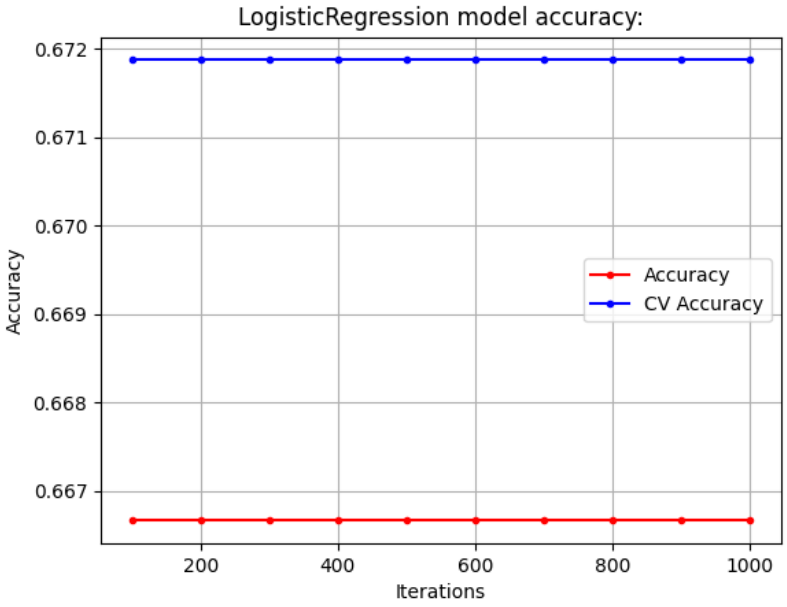

DecisionTreeClassifier model accuracy:
Accuracy
CV Accuracy
Accuracy
Depth
0.83
0.82
0.81
0.80
0.79
0.78
0.77
0.76
0.75
2
4
6
8
10

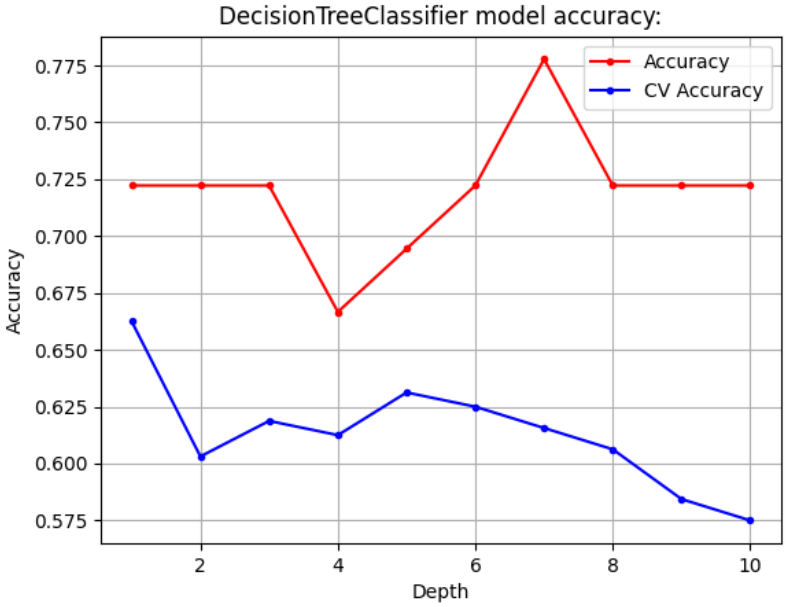

RandomForestClassifier model accuracy:
Accuracy
CV Accuracy
Accuracy
Depth
0.83
0.82
0.81
0.80
0.79
0.78
0.77
0.76
2
4
6
8
10

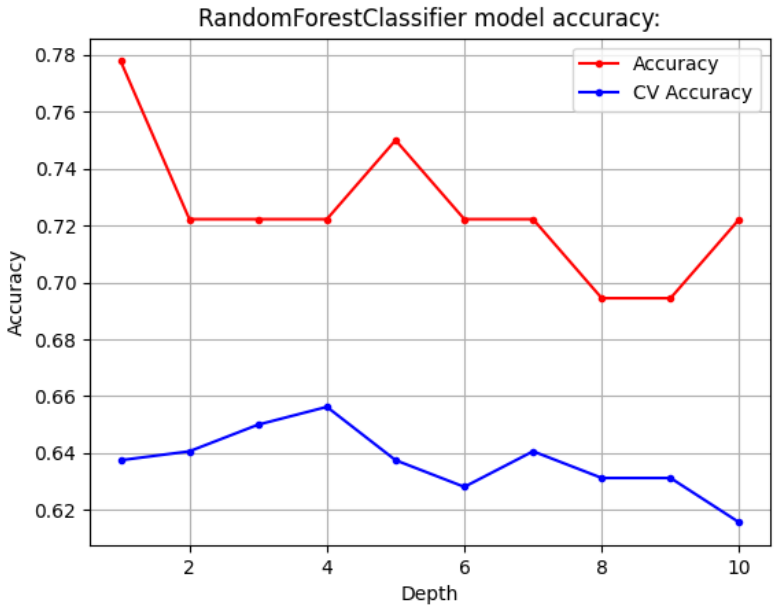

Train and validation accuracy
Train Accuracy
Validation Accuracy
Error
Epoch
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0
200
400
600
800
1000

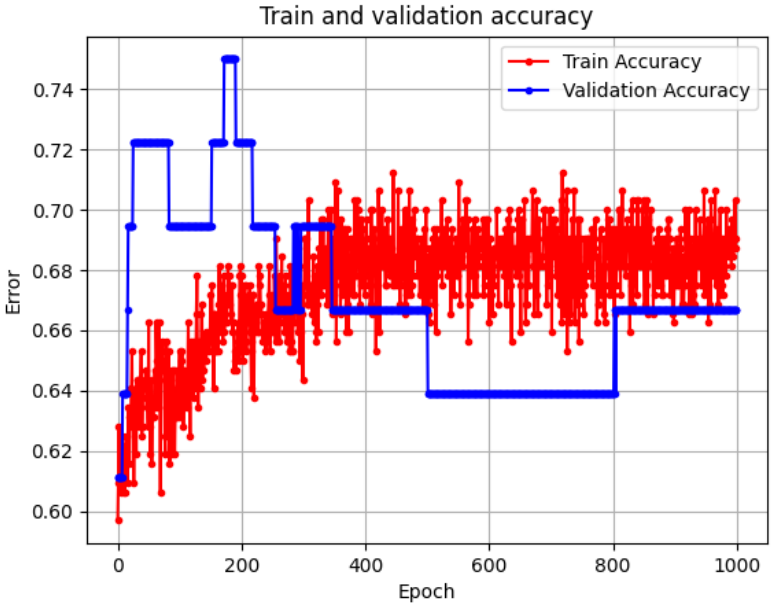
2.3.2 Dane zrównoważone

Nazwa modelu	Dokładność	Dokładność CV
LogisticRegression	0.66666666	0.671875
DecisionTreeClassifier	0.77777777	0.6625
RandomForestClassifier	0.77777777	0.65625
NeuralNetwork	0.71249997	0.75




DecisionTreeClassifier model accuracy:
Accuracy
CV Accuracy
0.775
0.750
0.725
0.700
0.675
0.650
0.625
0.600
0.575
Accuracy
2
4
6
8
10
Depth


RandomForestClassifier model accuracy:
Accuracy
CV Accuracy
0.78
0.76
0.74
0.72
0.70
0.68
0.66
0.64
0.62
Accuracy
2
4
6
8
10
Depth


Train and validation accuracy
Train Accuracy
Validation Accuracy
Error
Epoch
0.74
0.72
0.70
0.68
0.66
0.64
0.62
0.60
0
200
400
600
800
1000

Rozdział 3

Podsumowanie

W przypadku użycia oryginalnego zbioru danych klasyfikator logistyczny, drzewo decyzyjne oraz las miały dokładność między 76 a 78%. Najlepsza okazała się być sieć neuronowa, z dokładnością (na zbiorze testowym) wynoszącą 85%.

W przypadku użycia zrównoważonego zbioru danych, klasyfikator logistyczny, drzewo decyzyjne oraz las miały dokładność między 65 a 67%. Sieć neuronowa znów okazała się być najlepsza, osiągając dokładność równą 75%.

Dodatek A

Kod programu

Link do repozytorium: Repozytorium

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tabulate import tabulate
from sklearn.model_selection import cross_validate
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, accuracy_score
from keras.layers import Dropout
from numpy.random import seed
import random as rn
import tensorflow as tf
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD

PATH_ORIGINAL = 'data/transfusion.data'
PATH_MODIFIED = 'data/transfusion2.data'

paths = [PATH_ORIGINAL, PATH_MODIFIED]
accu_tuples = [('Nazwa modelu', 'Dokładność', 'Dokładność CV')]

def load_and_analyze_data(path, verbose=False):
    # Wczytywanie danych
    input_data = pd.read_csv(path)
```

```
    if verbose:
        # Analiza danych
        sns.pairplot(input_data)
        plt.show()
        print(input_data.head())
        print(input_data.shape)
        sns.heatmap(input_data.corr())
        plt.show()
        columns_copy = input_data.columns
        input_data.columns = ['R', 'F', 'M', 'T', 'D']
        print(tabulate(input_data.corr(), headers='keys', tablefmt='psql'))
        print(tabulate(input_data.describe(), headers='keys', tablefmt='psql'))
        input_data.columns = columns_copy
    return input_data


def preprocess_data(input_data):
    # F (Frequency - total number of donation)
    # M (Monetary - total blood donated in c.c.) - mililitry krwi
    # ponieważ Monetary wynika z Frequency, to pozbywamy się tego drugiego

    input_data = input_data.drop(columns="Monetary (c.c. blood)")

    input_features = input_data.iloc[:, :-1]
    target = input_data.iloc[:, -1:]

    # Normalizacja
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(input_features)

    train_data, test_data, train_target, test_target = \
        train_test_split(scaled_data, target.T.squeeze().to_numpy(),
                         test_size=0.1, random_state=10)
    return train_data, test_data, train_target, test_target


def train_and_score_model(train_data, test_data, train_target, test_target, model,
                          model_params, x):
    model_instance = model(**model_params)
    model_instance.fit(train_data, train_target)

    # Sprawdzanie poprawności
    acc = accuracy_score(test_target, model_instance.predict(test_data))
    # print("Model accuracy is {0:0.2f}".format(acc))
```

```
    conf_matrix = confusion_matrix(test_target, model_instance.predict(test_data))
    # print(conf_matrix)

    cv_results = cross_validate(model_instance, train_data, train_target,
                    scoring=('accuracy'), cv=10)
    accuracy = cv_results['test_score'].mean()
    # print("Model accuracy via cross-validation is {0:0.2f}".format(accuracy))

    return [x, [acc, accuracy]], [x, conf_matrix]


for path in paths:
    input_data = load_and_analyze_data(path)
    train_data, test_data, train_target, test_target = preprocess_data(input_data)

    models = [LogisticRegression, DecisionTreeClassifier, RandomForestClassifier]

    models_params = [{'max_iter': range(100, 1001, 100), 'random_state': 42},
                     {'criterion': 'entropy', 'max_depth': range(1, 11),
                     'random_state': 42},
                     {'criterion': 'entropy', 'max_depth': range(1, 11),
                     'random_state': 42}]
    models_accuracy = {}
    models_conf_matrices = {}
    for i, model in enumerate(models):
        model_name = model().__class__.__name__
        models_accuracy[model_name] = []
        models_conf_matrices[model_name] = []
        params = models_params[i]
        iterations = range(1)
        if 'max_iter' in params:
            iterations = params['max_iter']
        elif 'max_depth' in params:
            iterations = params['max_depth']
        for j in iterations:
            if 'max_iter' in params:
                params['max_iter'] = j
            elif 'max_depth' in params:
                params['max_depth'] = j
            current_accuracy, current_conf_matrix =
                     train_and_score_model(train_data, test_data, train_target,
                      test_target, model, params, j)
            models_accuracy[model_name].append(current_accuracy)
```

```
            models_conf_matrices[model_name].append(current_conf_matrix)

    print(models_accuracy)
    print(models_conf_matrices)

    for key, value in models_accuracy.items():
        indexes = [element[0] for element in value]
        accuracy = [element[1][0] for element in value]
        cv_accuracy = [element[1][1] for element in value]

        plt.plot(indexes, accuracy, 'r', marker='.', label="Accuracy")
        plt.plot(indexes, cv_accuracy, 'b', marker='.', label="CV Accuracy")
        plt.title(key + " model accuracy:")
        plt.legend()
        if 'Regression' in key:
            plt.xlabel('Iterations'), plt.ylabel('Accuracy')
        elif 'Tree' in key or 'Forest' in key:
            plt.xlabel('Depth'), plt.ylabel('Accuracy')
        plt.grid()
        plt.show()

        # print(f"Max for {key}; acc={np.max(accuracy)};
                    cv_acc={np.max(cv_accuracy)}")
        accu_tuples.append((key, np.max(accuracy), np.max(cv_accuracy)))

    seed(1)
    rn.seed(12345)
    tf.random.set_seed(1234)

    neural_model = Sequential([
        Dense(8, input_shape=(3,), activation="relu"),
        Dense(5, activation="relu"),
        Dropout(0.1),
        Dense(1, activation="sigmoid")
    ])
    neural_model.summary()
    neural_model.compile(SGD(lr=.003), "binary_crossentropy", metrics=["accuracy"])

    np.random.seed(0)
    run_hist = neural_model.fit(train_data, train_target, epochs=1000,
                                validation_data=(test_data, test_target),
                                verbose=True, shuffle=True)

    print("Training neural network...\n")
```

```
    print('Accuracy over training data is ',
                    accuracy_score(train_target,
                    neural_model.predict_classes(train_data)))
    print('Accuracy over testing data is ',
                    accuracy_score(test_target,
                    neural_model.predict_classes(test_data)))

    conf_matrix = confusion_matrix(test_target,
                    neural_model.predict_classes(test_data))
    print(conf_matrix)

    plt.plot(run_hist.history["accuracy"], 'r', marker='.',
                    label="Train Accuracy")
    plt.plot(run_hist.history["val_accuracy"], 'b', marker='.',
                    label="Validation Accuracy")
    plt.title("Train and validation accuracy")
    plt.legend()
    plt.xlabel('Epoch'), plt.ylabel('Error')
    plt.grid()
    plt.show()
    accu_tuples.append(("NeuralNetwork", np.max(run_hist.history["accuracy"]),
                    np.max(run_hist.history["val_accuracy"])))


print(tabulate(accu_tuples, floatfmt=".3f", tablefmt='psql'))
```