Semestrální práce – Game of Thrones (4IZ569 Data mining v cloudu)

Dataset: Game of Thrones – which characters will die

Odkaz: https://www.kaggle.com/datasets/kapturovalexander/which-game-of-thrones-characters-will-die?resource=download

Tým: Matěj Volf (volm08), Filip Janeba (janf11)

Úvod:

Tento dataset se zaměřuje na postavy ze světa seriálu a knižní série Game of Thrones. Jeho hlavním cílem je předpovědět, zda se daná postava v průběhu děje dožije určitého bodu (např. konce knih/seriálu), nebo jestli v příběhu zemře. Data jsme stáhli z kaggle.com. Data jsou rovnou rozděleny na trénovací a testovací v poměru 80/20. V práci využíváme pouze dataset game_of_thrones_train.csv

První část semestrální práce

Co v datech najdeme:

- **Identifikační údaje**: Jméno postavy (Name), její titul (title) nebo např. její rod (house)
- **Demografické informace**: Pohlaví (male – binární 0/1), kultura (Culture – textový popis, např. Northmen, Dornish, Andal atp.), věk (Age).
- **Rodinné a společenské vazby**: Např. zda je postava členem šlechtického stavu (isNoble), kolik má mrtvých příbuzných (numDeadRelations).
- **Záznam výskytu v knihách**: Sloupce typu book1 – book5, které značí, zda se daná postava v určité knize skutečně objevila.
- **Cílová proměnná**: isAlive (0 = mrtvá, 1 = živá), která určuje, zda postava zemře.

Zde je přehled a stručný popis všech sloupců datasetů:

- **name**: Jméno postavy
  zitle: Společenské postavení nebo šlechtický titul
  house: Rod, ke kterému postava patří
  culture: Sociální skupina spojená s postavou
  book1/2/3/4/5: Indikátor, zda se postava objevila v dané knize
  isNoble: Šlechtický status podle titulu
  age: Věk postavy (rok 305 AC jako referenční bod)
  male: Pohlaví postavy (1 = muž, 0 = žena)
  dateOfBirth: Rok narození postavy
  spouse: Jméno manžela/manželky postavy
  father: Jméno otce postavy
  mother: Jméno matky postavy

heir: Jméno dědice postavy
isMarried: Zda je postava vdaná/ženatá (1 = ano, 0 = ne)
isSpouseAlive: Zda je manžel/manželka naživu (1 = ano, 0 = ne)
isMotherAlive: Zda je matka naživu (1 = ano, 0 = ne)
isHeirAlive: Zda je dědic naživu (1 = ano, 0 = ne)
isFatherAlive: Zda je otec naživu (1 = ano, 0 = ne)
numberDeadRelations: Počet známých zesnulých příbuzných
popularity: Počet interních příchozích a odchozích odkazů na stránku postavy na wiki

Popis dat – game_of_thrones_train:

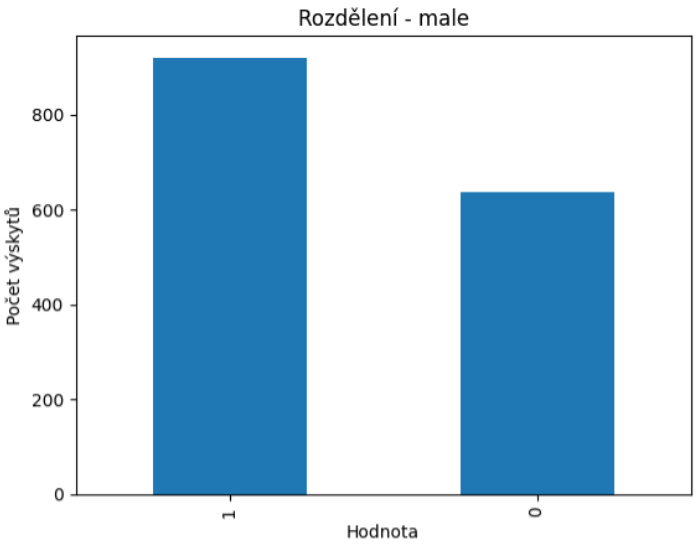
Informace o datasetu:

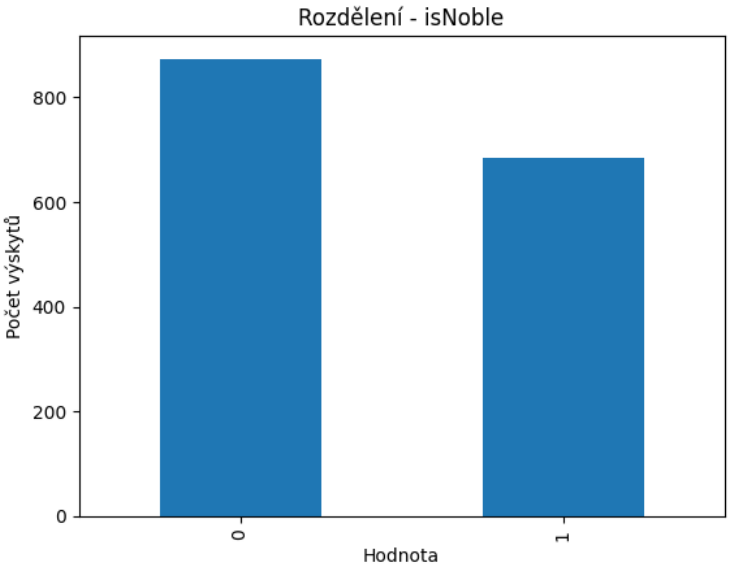
```
Shape (rows, columns): (1557, 26)

Info o datech:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1557 entries, 0 to 1556
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   S.No              1557 non-null   int64
 1   name              1557 non-null   object
 2   title             717 non-null    object
 3   male              1557 non-null   int64
 4   culture           488 non-null    object
 5   dateOfBirth       279 non-null    float64
 6   mother            18 non-null     object
 7   father            22 non-null     object
 8   heir              21 non-null     object
 9   house             1176 non-null   object
 10  spouse            200 non-null    object
 11  book1             1557 non-null   int64
 12  book2             1557 non-null   int64
 13  book3             1557 non-null   int64
 14  book4             1557 non-null   int64
 15  book5             1557 non-null   int64
 16  isAliveMother     18 non-null     float64
 17  isAliveFather     22 non-null     float64
 18  isAliveHeir       21 non-null     float64
 19  isAliveSpouse     200 non-null    float64
 20  isMarried         1557 non-null   int64
 21  isNoble           1557 non-null   int64
 22  age               279 non-null    float64
 23  numDeadRelations  1557 non-null   int64
 24  popularity        1557 non-null   float64
 25  isAlive           1557 non-null   int64
dtypes: float64(7), int64(11), object(8)
memory usage: 316.4+ KB
```

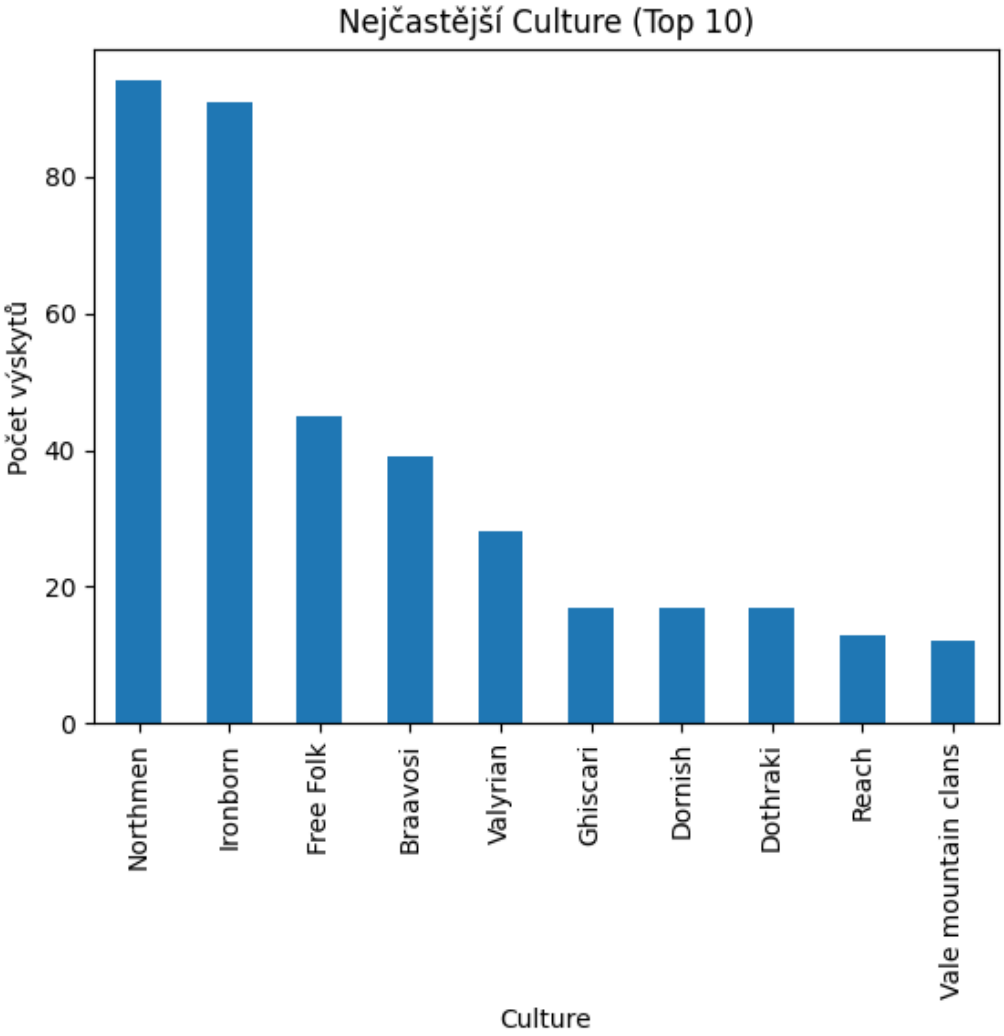
Popisné statistky:

```
Popisné statistiky (číselné sloupce):
              S.No         male  dateOfBirth        book1        book2  \
count  1557.000000  1557.000000   279.000000  1557.000000  1557.000000
mean    779.000000     0.590880   247.551971     0.138728     0.327553
std     449.611499     0.491829    61.550441     0.345774     0.469472
min       1.000000     0.000000   -25.000000     0.000000     0.000000
25%     390.000000     0.000000   241.000000     0.000000     0.000000
50%     779.000000     1.000000   272.000000     0.000000     0.000000
75%    1168.000000     1.000000   286.000000     0.000000     1.000000
max    1557.000000     1.000000   299.000000     1.000000     1.000000

             book3        book4        book5  isAliveMother  isAliveFather  \
count  1557.000000  1557.000000  1557.000000      18.000000      22.000000
mean      0.431599     0.562620     0.330122       0.666667       0.227273
std       0.495458     0.496223     0.470408       0.485071       0.428932
min       0.000000     0.000000     0.000000       0.000000       0.000000
25%       0.000000     0.000000     0.000000       0.000000       0.000000
50%       0.000000     1.000000     0.000000       1.000000       0.000000
75%       1.000000     1.000000     1.000000       1.000000       0.000000
max       1.000000     1.000000     1.000000       1.000000       1.000000

       isAliveHeir  isAliveSpouse    isMarried      isNoble         age  \
count    21.000000      200.00000  1557.000000  1557.000000  279.000000
mean      0.666667        0.79000     0.128452     0.439306   35.290323
std       0.483046        0.40833     0.334700     0.496462   26.364864
min       0.000000        0.00000     0.000000     0.000000    0.000000
25%       0.000000        1.00000     0.000000     0.000000   16.000000
50%       1.000000        1.00000     0.000000     0.000000   24.000000
75%       1.000000        1.00000     0.000000     1.000000   49.000000
max       1.000000        1.00000     1.000000     1.000000  100.000000

       numDeadRelations   popularity      isAlive
count       1557.000000  1557.000000  1557.000000
mean           0.187540     0.062400     0.778420
std            1.114648     0.121416     0.415443
min            0.000000     0.000000     0.000000
25%            0.000000     0.013378     1.000000
50%            0.000000     0.023411     1.000000
75%            0.000000     0.063545     1.000000
max           15.000000     1.000000     1.000000
```

Grafy pro podstatné kategoriální atributy:







Popis dat – game_of_thrones_test:

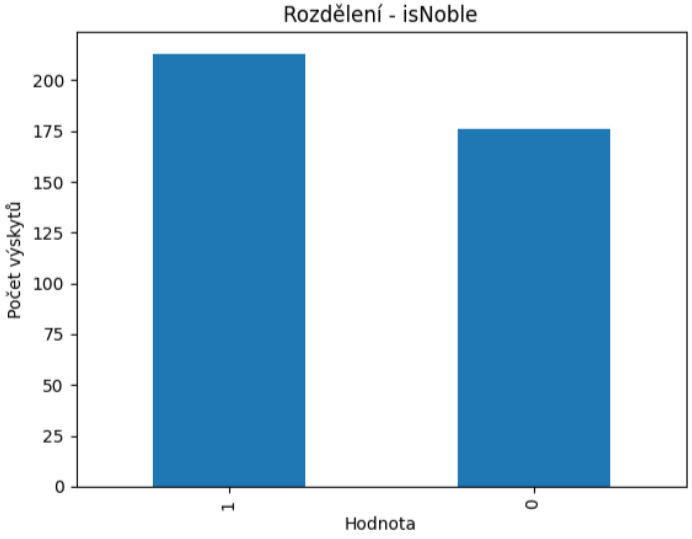
Informace o datasetu:

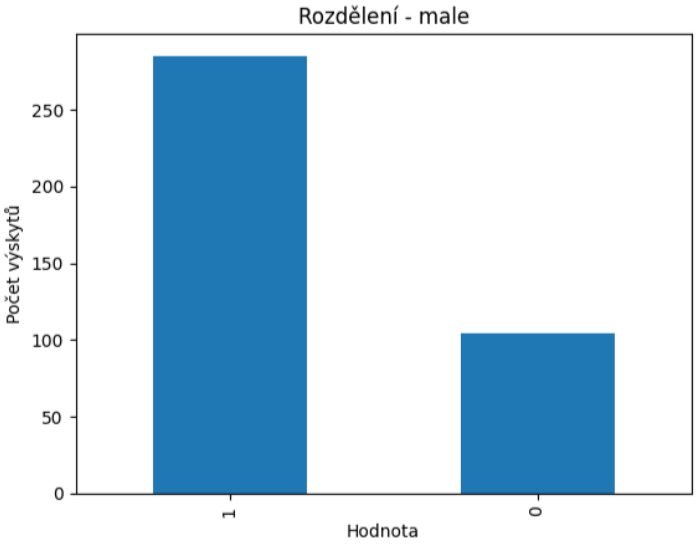
```
Shape (rows, columns): (389, 25)

Informace o datasetu:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 389 entries, 0 to 388
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   S.No              389 non-null    int64
 1   name              389 non-null    object
 2   title             221 non-null    object
 3   male              389 non-null    int64
 4   culture           189 non-null    object
 5   dateOfBirth       154 non-null    float64
 6   mother            3 non-null      object
 7   father            4 non-null      object
 8   heir              2 non-null      object
 9   house             343 non-null    object
 10  spouse            76 non-null     object
 11  book1             389 non-null    int64
 12  book2             389 non-null    int64
 13  book3             389 non-null    int64
 14  book4             389 non-null    int64
 15  book5             389 non-null    int64
 16  isAliveMother     3 non-null      float64
 17  isAliveFather     4 non-null      float64
 18  isAliveHeir       2 non-null      float64
 19  isAliveSpouse     76 non-null     float64
 20  isMarried         389 non-null    int64
 21  isNoble           389 non-null    int64
 22  age               154 non-null    float64
 23  numDeadRelations  389 non-null    int64
 24  popularity        389 non-null    float64
dtypes: float64(7), int64(10), object(8)
memory usage: 76.1+ KB
```

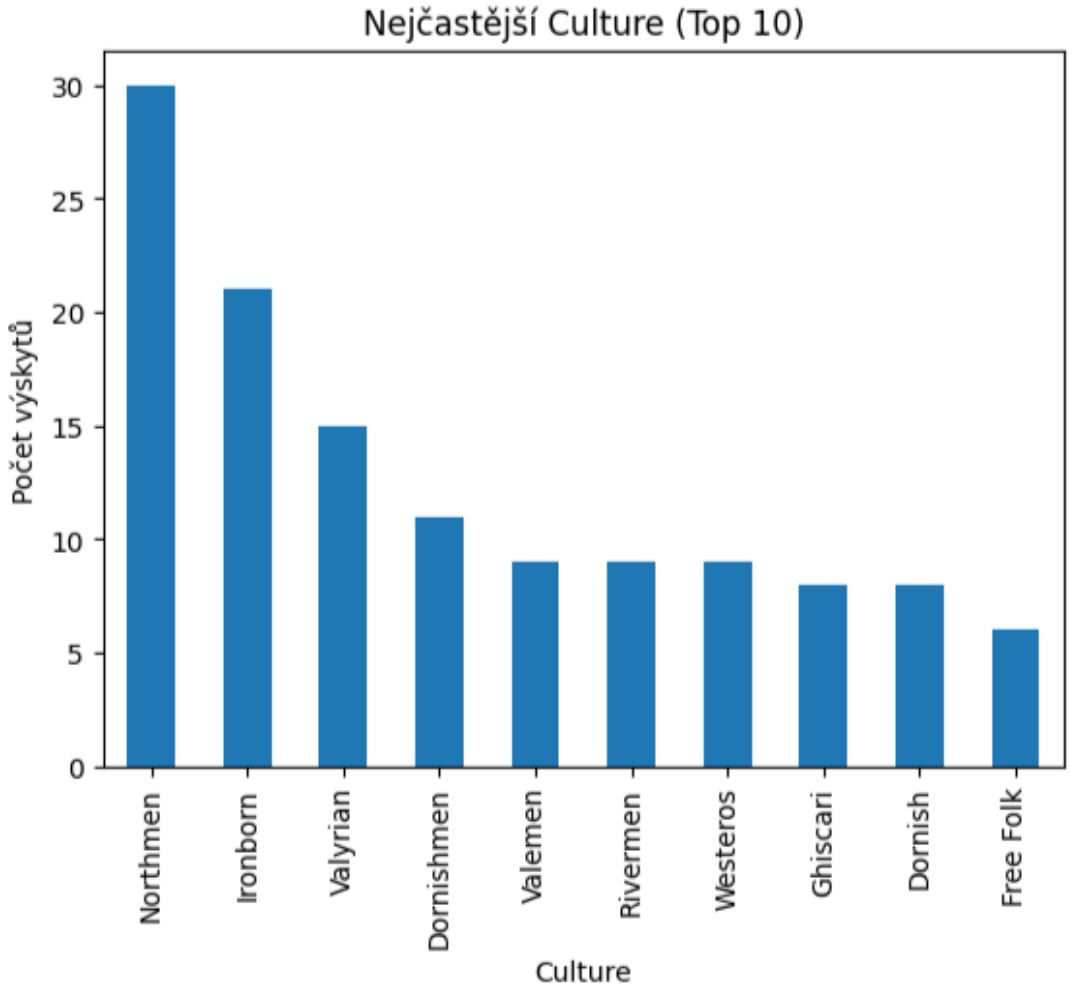
Popisné statistky:

```
Popisné statistiky (číselné sloupce):
              S.No         male  dateOfBirth        book1        book2  \
count  1557.000000  1557.000000   279.000000  1557.000000  1557.000000
mean    779.000000     0.590880   247.551971     0.138728     0.327553
std     449.611499     0.491829    61.550441     0.345774     0.469472
min       1.000000     0.000000   -25.000000     0.000000     0.000000
25%     390.000000     0.000000   241.000000     0.000000     0.000000
50%     779.000000     1.000000   272.000000     0.000000     0.000000
75%    1168.000000     1.000000   286.000000     0.000000     1.000000
max    1557.000000     1.000000   299.000000     1.000000     1.000000

             book3        book4        book5  isAliveMother  isAliveFather  \
count  1557.000000  1557.000000  1557.000000      18.000000      22.000000
mean      0.431599     0.562620     0.330122       0.666667       0.227273
std       0.495458     0.496223     0.470408       0.485071       0.428932
min       0.000000     0.000000     0.000000       0.000000       0.000000
25%       0.000000     0.000000     0.000000       0.000000       0.000000
50%       0.000000     1.000000     0.000000       1.000000       0.000000
75%       1.000000     1.000000     1.000000       1.000000       0.000000
max       1.000000     1.000000     1.000000       1.000000       1.000000

       isAliveHeir  isAliveSpouse    isMarried      isNoble          age  \
count    21.000000      200.00000  1557.000000  1557.000000  279.000000
mean      0.666667        0.79000     0.128452     0.439306   35.290323
std       0.483046        0.40833     0.334700     0.496462   26.364864
min       0.000000        0.00000     0.000000     0.000000    0.000000
25%       0.000000        1.00000     0.000000     0.000000   16.000000
50%       1.000000        1.00000     0.000000     0.000000   24.000000
75%       1.000000        1.00000     0.000000     1.000000   49.000000
max       1.000000        1.00000     1.000000     1.000000  100.000000

       numDeadRelations   popularity      isAlive
count       1557.000000  1557.000000  1557.000000
mean           0.187540     0.062400     0.778420
std            1.114648     0.121416     0.415443
min            0.000000     0.000000     0.000000
25%            0.000000     0.013378     1.000000
50%            0.000000     0.023411     1.000000
75%            0.000000     0.063545     1.000000
max           15.000000     1.000000     1.000000
```

Grafy pro podstatné kategoriální atributy:



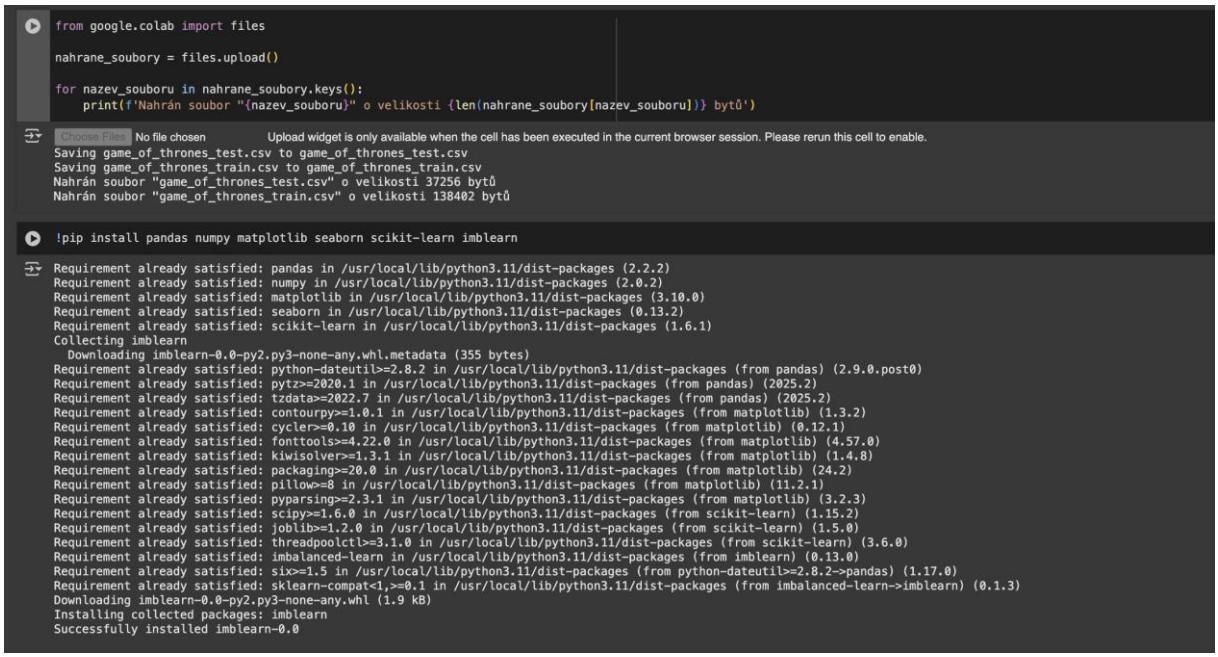




Preprocessing

Nejprve jsme připravili pracovní prostředí: naimportovali jsme knihovny pandas, NumPy, matplotlib, seaborn a scikit-learn, které zajišťují načítání dat, vizualizaci i následný preprocessing. Poté jsme z .csv souborů načetli data o postavách Game of Thrones. Následovala kontrola kvality dat. Spočítali jsme absolutní i relativní četnost chybějících hodnot a identifikovali sloupce, které vyžadovaly další ošetření. Zároveň jsme si ověřili rozměry obou datasetů a pomocí grafu jsme zjistili distribuci cílové proměnné isAlive, abychom zjistili vyváženost tříd živých a mrtvých postav. U kategoriálních atributů jsme přistoupili k dichotomizaci: z proměnných Cultrue a Title jsme vybrali deset, respektive pět nejčastějších hodnot a pro každou z nich vytvořili binární indikátorový sloupec. Všechny ostatní, méně frekventované či prázdné, jsme sloučili do kategorie other, pro kterou jsme rovněž zavedli vlastní binární příznak. Číselné atributy prošly dvěma úpravami. Age jsme diskretizovali do čtyř věkových skupin (mladiství, mladí, dospělí a senioři), chybějící hodnoty jsme doplnili nejčastěji se vyskytující kategorií. Popularity jsme normalizovali metodou Min-Max na interval <0;1>, aby žádná proměnná nedominovala ostatním. Ze vzniklé tabulky jsme vybrali pouze číselné a nově vytvořené binární sloupce, které mají skutečný predikční potenciál. Ve vybrané podmnožině jsme zbylé chybějící hodnoty nahradili nulami, abychom nenaráželi na NaN. Před trénovámín dat jsme ještě aplikovali metodu SMOTE, která pomocí syntetických vzorků dorovnala počet dorovnala počet zástupců minoritní třídy a vyvážila tak trénovací data. Nakonec jsme soubory opět uložili do formátu .csv, abychom je mohli využít k následnému modelování a vyhodnocení.

Ukázka preprocessingu



```
from google.colab import files

nahrane_soubory = files.upload()

for nazev_souboru in nahrane_soubory.keys():
    print(f'Nahrán soubor "{nazev_souboru}" o velikosti {len(nahrane_soubory[nazev_souboru])} bytů')
```

```
Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving game_of_thrones_test.csv to game_of_thrones_test.csv
Saving game_of_thrones_train.csv to game_of_thrones_train.csv
Nahrán soubor "game_of_thrones_test.csv" o velikosti 37256 bytů
Nahrán soubor "game_of_thrones_train.csv" o velikosti 138402 bytů
```

```
!pip install pandas numpy matplotlib seaborn scikit-learn imblearn
```

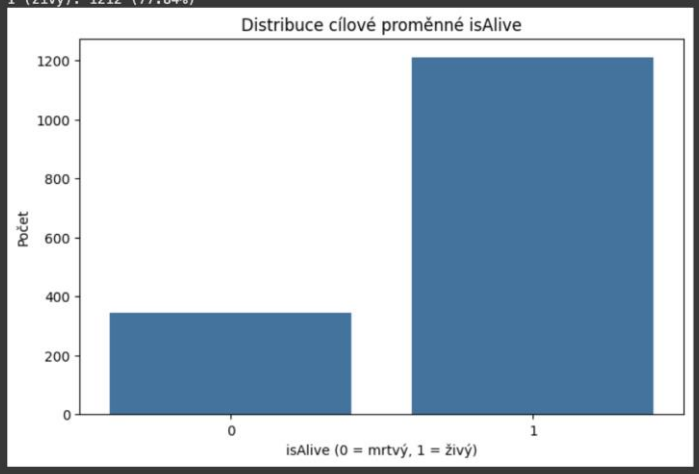
```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Collecting imblearn
  Downloading imblearn-0.0-py2.py3-none-any.whl.metadata (355 bytes)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.3)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.15.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.5.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: imbalanced-learn in /usr/local/lib/python3.11/dist-packages (from imblearn) (0.13.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
Requirement already satisfied: sklearn-compat<1,>=0.1 in /usr/local/lib/python3.11/dist-packages (from imbalanced-learn->imblearn) (0.1.3)
Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Installing collected packages: imblearn
Successfully installed imblearn-0.0
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from imblearn.over_sampling import SMOTE

# Načtení dat
train_data = pd.read_csv('game_of_thrones_train.csv')
test_data = pd.read_csv('game_of_thrones_test.csv')

# Kontrola chybějících hodnot
missing_train = train_data.isnull().sum()
missing_percent_train = (missing_train / len(train_data)) * 100
missing_df_train = pd.DataFrame({'Missing Values': missing_train,
                                 'Percentage': missing_percent_train})
print("Chybějící hodnoty v trénovacích datech:")
print(missing_df_train[missing_df_train['Missing Values'] > 0])
```

```
Chybějící hodnoty v trénovacích datech:
                Missing Values  Percentage
title                      840   53.949904
culture                   1069   68.657675
dateOfBirth               1278   82.080925
mother                    1539   98.843931
father                    1535   98.587026
heir                      1536   98.651252
house                      381   24.470135
spouse                    1357   87.154785
isAliveMother             1539   98.843931
isAliveFather             1535   98.587026
isAliveHeir               1536   98.651252
isAliveSpouse             1357   87.154785
age                       1278   82.080925
```

```python
# Načtení trénovacích a testovacích dat
train_data = pd.read_csv('game_of_thrones_train.csv')
test_data = pd.read_csv('game_of_thrones_test.csv')

print(f"Trénovací data: {train_data.shape}")
print(f"Testovací data: {test_data.shape}")

# Kontrola distribuce cílové proměnné isAlive
isalive_counts = train_data['isAlive'].value_counts()
print(f"\nDistribuce cílové proměnné 'isAlive':")
print(f"0 (mrtvý): {isalive_counts[0]} ({isalive_counts[0]/len(train_data)*100:.2f}%)")
print(f"1 (živý): {isalive_counts[1]} ({isalive_counts[1]/len(train_data)*100:.2f}%)")

# Vizualizace distribuce cílové proměnné
plt.figure(figsize=(8, 5))
sns.countplot(x='isAlive', data=train_data)
plt.title('Distribuce cílové proměnné isAlive')
plt.xlabel('isAlive (0 = mrtvý, 1 = živý)')
plt.ylabel('Počet')
plt.savefig('isalive_distribution.png')
```

```
Trénovací data: (1557, 26)
Testovací data: (389, 25)

Distribuce cílové proměnné 'isAlive':
0 (mrtvý): 345 (22.16%)
1 (živý): 1212 (77.84%)
```




```python
# Dichotomizace kategorických atributů
# Identifikace 10 nejčastějších kultur
top_cultures = train_data['culture'].value_counts().head(10).index.tolist()
print("10 nejčastějších kultur:", top_cultures)

# Vytvoření binárních sloupců pro top 10 kultur
for culture in top_cultures:
    train_data[f'culture_{culture}'] = (train_data['culture'] == culture).astype(int)
    test_data[f'culture_{culture}'] = (test_data['culture'] == culture).astype(int)

# Vytvoření sloupce other_culture pro kultury mimo top 10
train_data['culture_other'] = (~train_data['culture'].isin(top_cultures) & train_data['culture'].notna()).astype(int)
test_data['culture_other'] = (~test_data['culture'].isin(top_cultures) & test_data['culture'].notna()).astype(int)

# Identifikace 5 nejčastějších titulů
top_titles = train_data['title'].value_counts().head(5).index.tolist()
print("5 nejčastějších titulů:", top_titles)

# Vytvoření binárních sloupců pro top 5 titulů
for title in top_titles:
    train_data[f'title_{title}'] = (train_data['title'] == title).astype(int)
    test_data[f'title_{title}'] = (test_data['title'] == title).astype(int)

# Vytvoření sloupce other_titles pro tituly mimo top 5
train_data['title_other'] = (~train_data['title'].isin(top_titles) & train_data['title'].notna()).astype(int)
test_data['title_other'] = (~test_data['title'].isin(top_titles) & test_data['title'].notna()).astype(int)
```

```
10 nejčastějších kultur: ['Northmen', 'Ironborn', 'Free Folk', 'Braavosi', 'Valyrian', 'Ghiscari', 'Dornish', 'Dothraki', 'Reach', 'Vale mountain clans']
5 nejčastějších titulů: ['Ser', 'Maester', 'Archmaester', 'Lord', 'Septon']
```

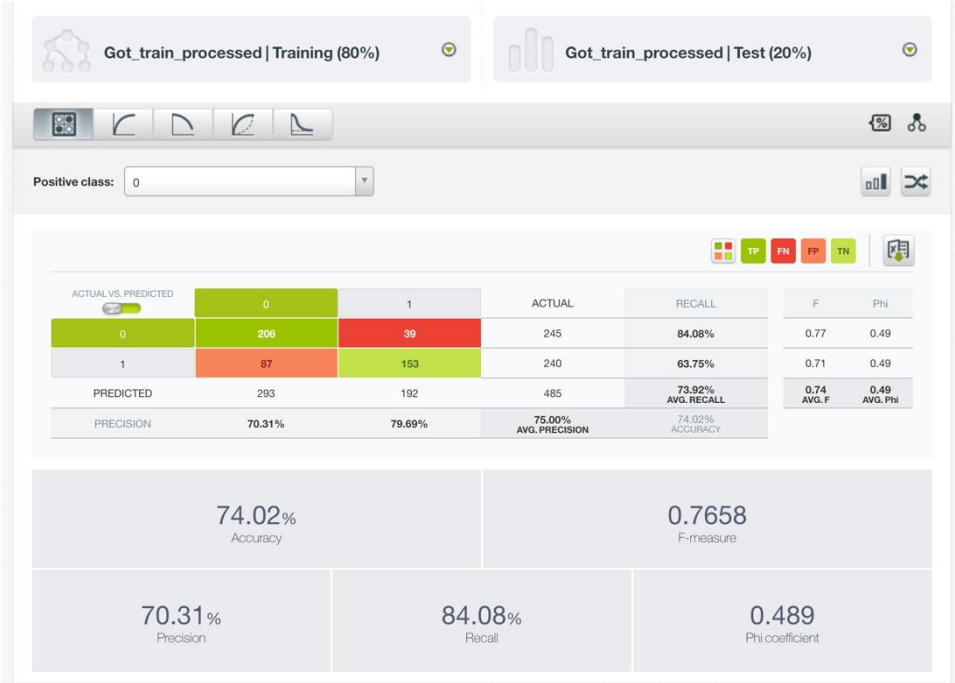
```python
# Diskretizace numerického atributu 'age'
def discretize_age(age):
    if pd.isna(age):
        return np.nan
    elif age < 18:
        return 0  # mladistvý
    elif age < 40:
        return 1  # mladý dospělý
    elif age < 60:
        return 2  # dospělý
    else:
        return 3  # senior

train_data['age_category'] = train_data['age'].apply(discretize_age)
test_data['age_category'] = test_data['age'].apply(discretize_age)

# Min-max normalizace numerického atributu 'popularity'
scaler = MinMaxScaler()
train_data['popularity_normalized'] = scaler.fit_transform(train_data[['popularity']])
test_data['popularity_normalized'] = scaler.transform(test_data[['popularity']])

# Nahrazení chybějících hodnot průměrem / modem
if train_data['age_category'].isnull().sum() > 0:
    age_mode = train_data['age_category'].mode()[0]
    train_data['age_category'] = train_data['age_category'].fillna(age_mode)
    test_data['age_category'] = test_data['age_category'].fillna(age_mode)
```

```
[ ]  # Vybereme relevantní numerické sloupce pro SMOTE
     numeric_cols = ['male', 'book1', 'book2', 'book3', 'book4', 'book5',
                     'isMarried', 'isNoble', 'numDeadRelations',
                     'age_category', 'popularity_normalized']

     # Přidáme sloupce kultur
     for culture in top_cultures:
         numeric_cols.append(f'culture_{culture}')
     numeric_cols.append('culture_other')

     # Přidáme sloupce titulů
     for title in top_titles:
         numeric_cols.append(f'title_{title}')
     numeric_cols.append('title_other')

     # Nahradíme missing hodnoty nulami před aplikací SMOTE
     X_train = train_data[numeric_cols].fillna(0)
     y_train = train_data['isAlive']

     smote = SMOTE(random_state=42)
     X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

     # Vytvoření finálního datasetu
     train_final = pd.DataFrame(X_train_balanced, columns=numeric_cols)
     train_final['age_category'] = train_final['age_category'].round().astype(int)
     train_final['isAlive'] = y_train_balanced

     # Uložení preprocessovaných dat
     train_final.to_csv('got_train_processed.csv', index=False)
     test_data[numeric_cols].to_csv('got_test_processed.csv', index=False)

     print("\nPreprocessing dokončen a data uložena.")
```

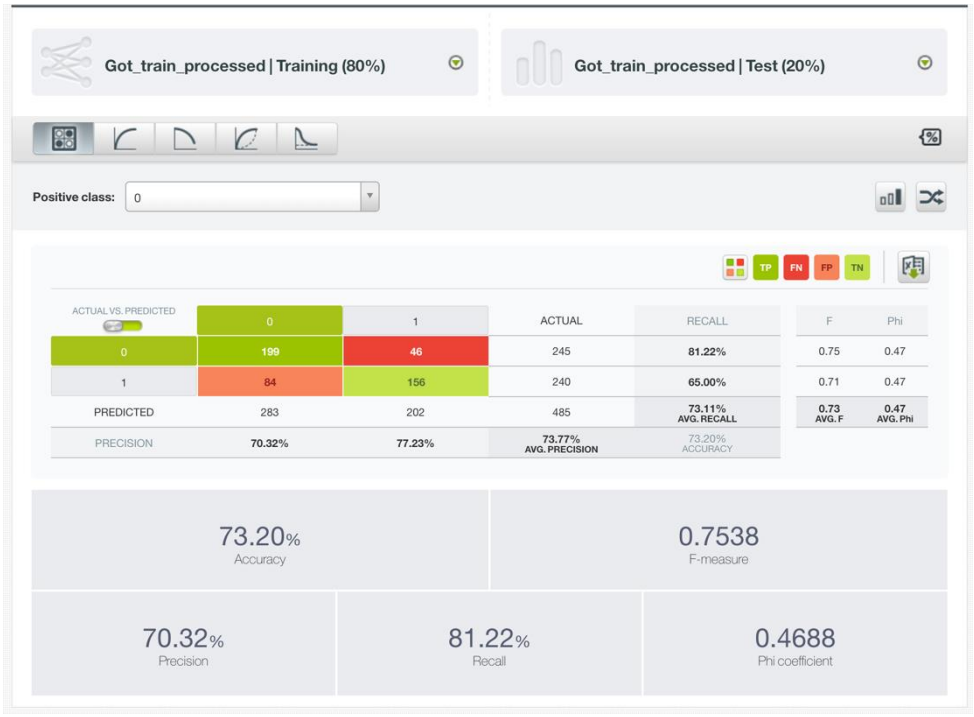
```
Preprocessing dokončen a data uložena.
```

BigML

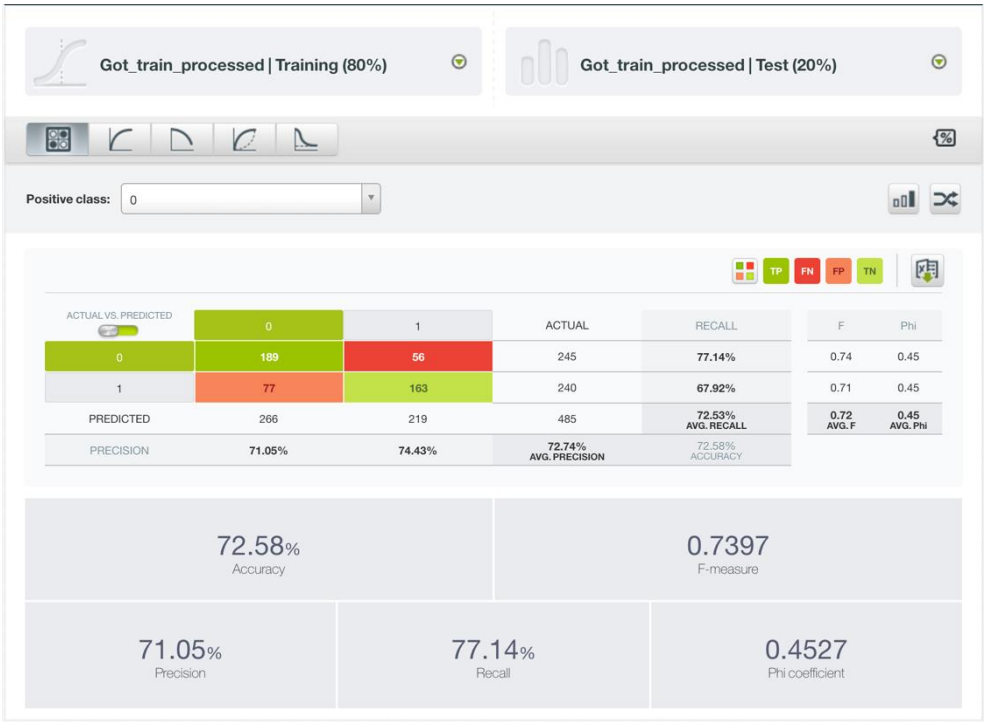
Evaluace Decision Tree



Evaluce Neuronové sítě



Evaluace Logistické regrese



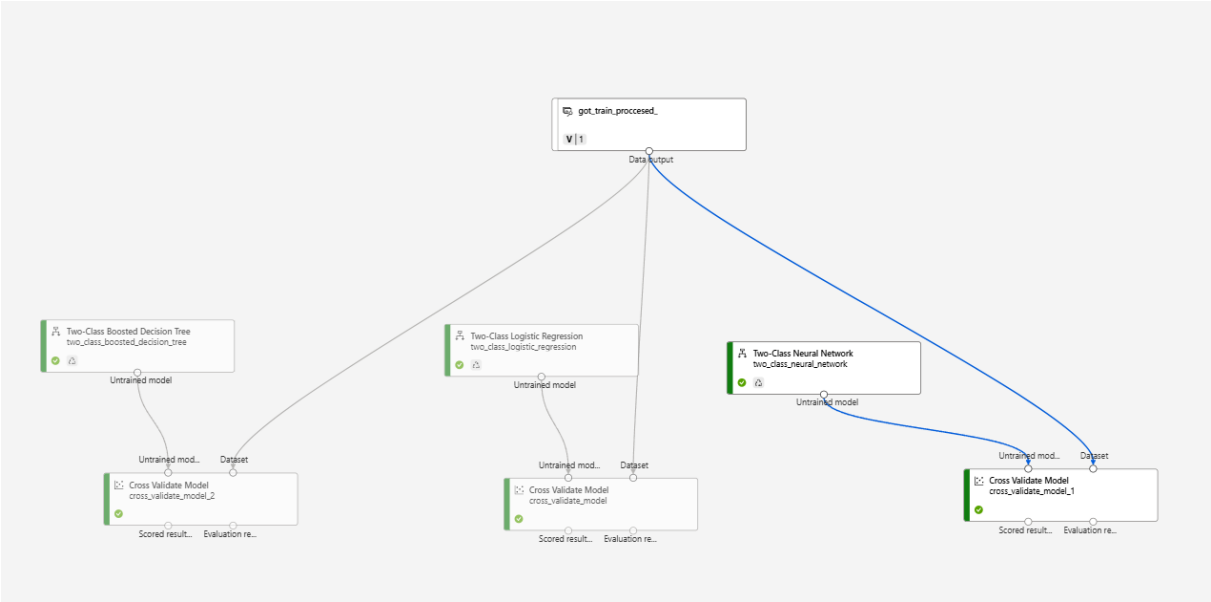
Metrika	**Rozhodovací strom**	**Logistická regrese**	**Neuronová síť**
Accuracy	74,02 %	72,58 %	73,20 %

Precision (0)	70,31 %	71,05 %	70,32 %
Precision (1)	79,69 %	74,43 %	77,23 %
Recall (0)	84,08 %	77,14 %	81,22 %
Recall (1)	63,75 %	67,92 %	65,00 %
F-measure	0,7658	0,7397	0,7538
Phi koeficient	0,489	0,4527	0,4688

Nejlepšího výsledku dosáhl rozhodovací strom, zejména díky vyšší přesnosti a F-míře. Neuronová síť je těsně za ním a logistická regrese mírně zaostává.

Azure

Pipeline



Evaluace Decision Tree

Fold Number	Number of examples in fold	AUC	Accuracy	Precision	Recall	F1	True Negative	False Positive	False Negative
0	242	0.950111	0.880165	0.886525	0.905797	0.896057	88	16	13
1	242	0.930441	0.838843	0.809917	0.859649	0.834043	105	23	16
2	243	0.94424	0.839506	0.804878	0.868421	0.835443	105	24	15
3	242	0.919575	0.81405	0.833333	0.785124	0.808511	102	19	26
4	243	0.927345	0.864198	0.901639	0.839695	0.869565	100	12	21
5	242	0.932306	0.863636	0.876033	0.854839	0.865306	103	15	18
6	242	0.946315	0.880165	0.903509	0.85124	0.876596	110	11	18
7	242	0.901844	0.805785	0.820513	0.786885	0.803347	99	21	26
8	243	0.927097	0.831276	0.795276	0.87069	0.831276	101	26	15
9	243	0.952873	0.888889	0.85	0.918919	0.883117	114	18	9
Mean	2424	0.933215	0.850651	0.848162	0.854126	0.850326	102.7	18.5	17.7
Standard Deviation	2424	0.01569	0.02896	0.041221	0.043304	0.03207	6.929005	5.060742	5.417051

Evaluace Logistické regrese

Fold Number	Number of examples in fold	AUC	Accuracy	Precision	Recall	F1	True Negative	False Positive	False Negative
0	242	0.79104	0.710744	0.757576	0.724638	0.740741	72	32	38
1	242	0.791907	0.731405	0.724771	0.692982	0.70852	98	30	35
2	243	0.79879	0.707819	0.697248	0.666667	0.681614	96	33	38
3	242	0.748958	0.690083	0.701754	0.661157	0.680851	87	34	41
4	243	0.785851	0.699588	0.758929	0.648855	0.699588	85	27	46
5	242	0.841068	0.752066	0.796296	0.693548	0.741379	96	22	38
6	242	0.80138	0.72314	0.741071	0.68595	0.712446	92	29	38
7	242	0.732923	0.677686	0.707547	0.614754	0.657895	89	31	47
8	243	0.828197	0.744856	0.721311	0.758621	0.739496	93	34	28
9	243	0.769758	0.703704	0.669565	0.693694	0.681416	94	38	34
Mean	2424	0.788987	0.714109	0.727607	0.684087	0.704395	90.2	31	38.3
Standard Deviation	2424	0.032723	0.023666	0.036799	0.039919	0.029419	7.627436	4.396969	5.558777

Evaluce Neuronové sítě

Fold Number	Number of examples in fold	AUC	Accuracy	Precision	Recall	F1	True Negative	False Positive	False Negative
0	242	0.813406	0.731405	0.806723	0.695652	0.747082	81	23	42
1	242	0.825212	0.752066	0.75	0.710526	0.72973	101	27	33
2	243	0.835645	0.736626	0.745098	0.666667	0.703704	103	26	38
3	242	0.797657	0.727273	0.757009	0.669421	0.710526	95	26	40
4	243	0.827426	0.744856	0.834951	0.656489	0.735043	95	17	45
5	242	0.871275	0.77686	0.857143	0.677419	0.756757	104	14	40
6	242	0.82105	0.731405	0.791667	0.628099	0.700461	101	20	45
7	242	0.790642	0.706612	0.747573	0.631148	0.684444	94	26	45
8	243	0.848018	0.748971	0.73913	0.732759	0.735931	97	30	31
9	243	0.807364	0.72428	0.689655	0.720721	0.704846	96	36	31
Mean	2424	0.82377	0.738035	0.771895	0.67889	0.720852	96.7	24.5	39
Standard Deviation	2424	0.023955	0.019011	0.050216	0.035801	0.023293	6.58365	6.363961	5.617433

Nejlepšího výsledku v Azure dosáhl rozhodovací strom, jak v přesnosti, tak správnosti i úplnosti.

Pokud porovnáme všechny modely, jak BigML tak Azure, tak nejlépe na tom je Decision Tree v Azure, kde všechny metriky přesahují hodnotu 0.84

Nasazení modelu

V BigML webovém rozhraní jsme po natrénování modelu využili možnost stáhnout tzv. "bindings ready" model, což je automaticky vygenerovaný Python skript připravený pro použití s BigML Python knihovnou. Vybrali jsme model rozhodovacího stromu, jelikož měl nejlepší výsledky.

```python
# Requires BigML Python bindings
#
!pip install bigml
#
# or clone it:
#   git clone https://github.com/bigmlcom/python.git

from bigml.model import Model
from bigml.api import BigML

def predict_isalive(data={}):
    """
    Predicts whether a character is alive or dead.

    Args:
        data (dict): Dictionary containing character attributes

    Returns:
        str: '1' for alive, '0' for dead
    """
    # Downloads and generates a local version of the model, if it
    # hasn't been downloaded previously.
    model = Model('model/67fe95f2554200fbb97d570d',
                  api=BigML("FilaJeFrajer",
                            "f71a5a905b2e10ad6f2117544e5c1420a551752c",
                            domain="bigml.io",
                            project="project/67bf0f8683014742ecf2710f"))

    # Format input data properly - ensure all keys use lowercase
    formatted_data = {}
    for key, value in data.items():
        formatted_key = key.lower()
        formatted_data[formatted_key] = value

    # Make prediction and return the result
    prediction = model.predict(formatted_data, full=True)

    # Return the prediction outcome
    if isinstance(prediction, dict) and 'prediction' in prediction:
        return prediction['prediction']
    elif isinstance(prediction, str):
        return prediction
    else:
        return str(prediction)
```

```python
test_instances = [
    {
        'male': 0, 'book1': 0, 'book2': 0, 'book3': 0, 'book4': 1, 'book5': 0,
        'isMarried': 0, 'isNoble': 0, 'numDeadRelations': 3, 'age_category': 1,
        'popularity_normalized': 0.050167224, 'culture_Northmen': 0, 'culture_Ironborn': 0,
        'culture_Free Folk': 0, 'culture_Braavosi': 0, 'culture_Valyrian': 0,
        'culture_Ghiscari': 0, 'culture_Dornish': 0, 'culture_Dothraki': 0,
        'culture_Reach': 0, 'culture_Vale mountain clans': 0, 'culture_other': 0,
        'title_Ser': 0, 'title_Maester': 0, 'title_Archmaester': 0, 'title_Lord': 0,
        'title_Septon': 0, 'title_other': 0
    },
    {
        'male': 0, 'book1': 0, 'book2': 1, 'book3': 1, 'book4': 1, 'book5': 0,
        'isMarried': 0, 'isNoble': 0, 'numDeadRelations': 1, 'age_category': 3,
        'popularity_normalized': 0.745819398, 'culture_Northmen': 0, 'culture_Ironborn': 0,
        'culture_Free Folk': 0, 'culture_Braavosi': 0, 'culture_Valyrian': 0,
        'culture_Ghiscari': 0, 'culture_Dornish': 0, 'culture_Dothraki': 0,
        'culture_Reach': 0, 'culture_Vale mountain clans': 1, 'culture_other': 0,
        'title_Ser': 0, 'title_Maester': 0, 'title_Archmaester': 0, 'title_Lord': 0,
        'title_Septon': 0, 'title_other': 0
    },
    {
        'male': 0, 'book1': 0, 'book2': 0, 'book3': 1, 'book4': 1, 'book5': 0,
        'isMarried': 0, 'isNoble': 0, 'numDeadRelations': 1, 'age_category': 1,
        'popularity_normalized': 0.010033445, 'culture_Northmen': 0, 'culture_Ironborn': 0,
        'culture_Free Folk': 0, 'culture_Braavosi': 0, 'culture_Valyrian': 0,
        'culture_Ghiscari': 0, 'culture_Dornish': 0, 'culture_Dothraki': 0,
        'culture_Reach': 0, 'culture_Vale mountain clans': 0, 'culture_other': 0,
        'title_Ser': 0, 'title_Maester': 0, 'title_Archmaester': 0, 'title_Lord': 0,
        'title_Septon': 0, 'title_other': 0
    },
    {
        'male': 1, 'book1': 1, 'book2': 1, 'book3': 1, 'book4': 1, 'book5': 1,
        'isMarried': 0, 'isNoble': 1, 'numDeadRelations': 1, 'age_category': 2,
        'popularity_normalized': 0.220735786, 'culture_Northmen': 0, 'culture_Ironborn': 0,
        'culture_Free Folk': 0, 'culture_Braavosi': 0, 'culture_Valyrian': 0,
        'culture_Ghiscari': 0, 'culture_Dornish': 0, 'culture_Dothraki': 0,
        'culture_Reach': 0, 'culture_Vale mountain clans': 0, 'culture_other': 0,
        'title_Ser': 0, 'title_Maester': 1, 'title_Archmaester': 0, 'title_Lord': 0,
        'title_Septon': 0, 'title_other': 0
    },
    {
        'male': 1, 'book1': 1, 'book2': 1, 'book3': 1, 'book4': 1, 'book5': 1,
        'isMarried': 1, 'isNoble': 0, 'numDeadRelations': 1, 'age_category': 1,
        'popularity_normalized': 0.434782609, 'culture_Northmen': 0, 'culture_Ironborn': 0,
        'culture_Free Folk': 0, 'culture_Braavosi': 0, 'culture_Valyrian': 0,
        'culture_Ghiscari': 0, 'culture_Dornish': 0, 'culture_Dothraki': 0,
        'culture_Reach': 0, 'culture_Vale mountain clans': 0, 'culture_other': 0,
        'title_Ser': 1, 'title_Maester': 0, 'title_Archmaester': 0, 'title_Lord': 0,
        'title_Septon': 0, 'title_other': 0
    },
    {
        'male': 1, 'book1': 0, 'book2': 0, 'book3': 0, 'book4': 0, 'book5': 0,
        'isMarried': 0, 'isNoble': 0, 'numDeadRelations': 11, 'age_category': 1,
        'popularity_normalized': 0.605351170568561, 'culture_Northmen': 0, 'culture_Ironborn': 0,
        'culture_Free Folk': 0, 'culture_Braavosi': 0, 'culture_Valyrian': 0,
        'culture_Ghiscari': 0, 'culture_Dornish': 0, 'culture_Dothraki': 0,
        'culture_Reach': 0, 'culture_Vale mountain clans': 0, 'culture_other': 0,
        'title_Ser': 1, 'title_Maester': 0, 'title_Archmaester': 0, 'title_Lord': 0,
        'title_Septon': 0, 'title_other': 0
    }
]
```

```python
for i, instance in enumerate(test_instances, 1):
    prediction = predict_isalive(instance)
    print(f"Instance {i}: {'Živý' if int(prediction) == 1 else 'Mrtvý'} (predikce: {prediction})")
```

```
Instance 1: Živý (predikce: 1)
Instance 2: Živý (predikce: 1)
Instance 3: Živý (predikce: 1)
Instance 4: Živý (predikce: 1)
Instance 5: Živý (predikce: 1)
Instance 6: Mrtvý (predikce: 0)
```

Druhá část seminární práce

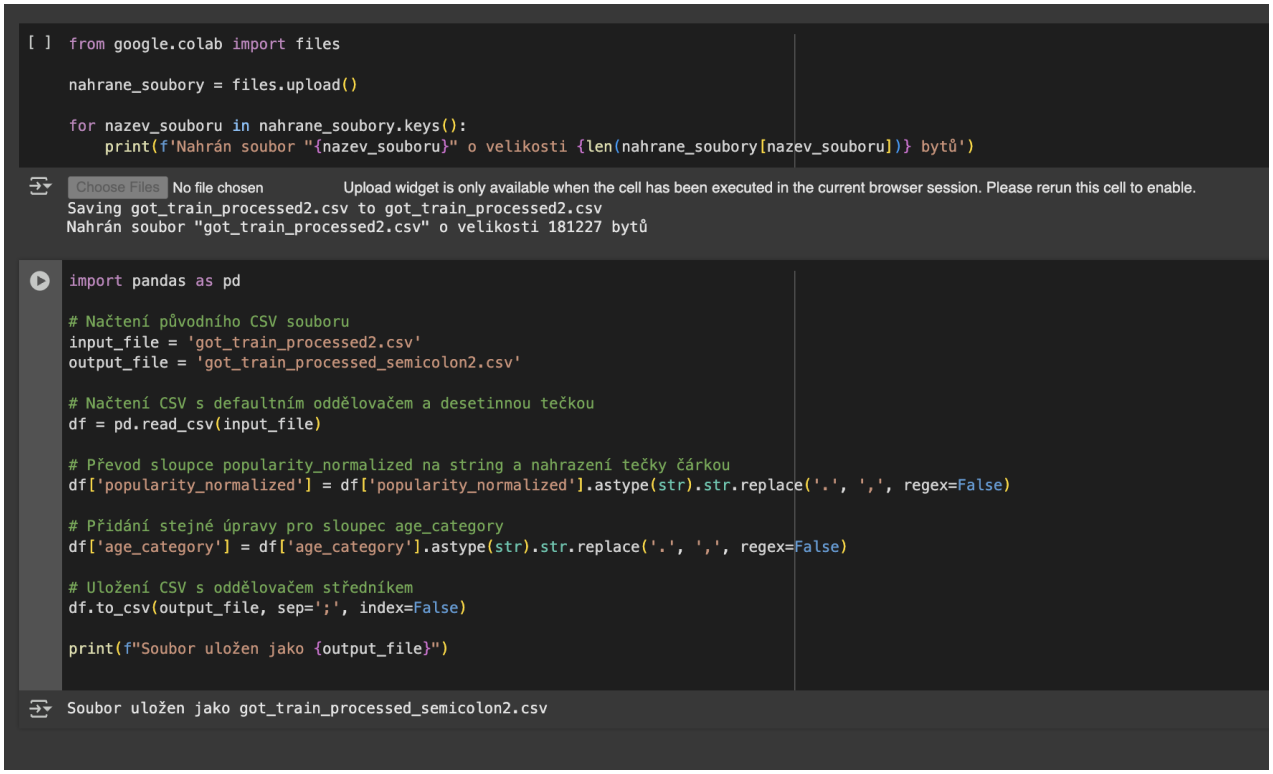
Strategie pro vylepšení hodnot

V rámci strategie pro zlepšení výsledků jsme se rozhodli zkusit kombinovaný manuální postup a následně funkcionalitu AutoML. U manuálního postupu bychom nejprve doplnili drobný preprocessing, ale hlavní důraz budeme klást na úpravu hyperparametrů u zvolených modelů.

Preprocessing/Úprava hyperparametrů

Pro druhou část semestrální práce jsme přidali ještě drobný preprocessing. Konkrétně jsme u atributů age_category a popularity_normalized vyměnili tečku za čárku.

Konkrétní úpravy hyperparametrů u zvolených modelů jsme popsali v dalších částech.



```
[ ]  from google.colab import files

     nahrane_soubory = files.upload()

     for nazev_souboru in nahrane_soubory.keys():
         print(f'Nahrán soubor "{nazev_souboru}" o velikosti {len(nahrane_soubory[nazev_souboru])} bytů')
```

```
Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving got_train_processed2.csv to got_train_processed2.csv
Nahrán soubor "got_train_processed2.csv" o velikosti 181227 bytů
```

```
import pandas as pd

# Načtení původního CSV souboru
input_file = 'got_train_processed2.csv'
output_file = 'got_train_processed_semicolon2.csv'

# Načtení CSV s defaultním oddělovačem a desetinnou tečkou
df = pd.read_csv(input_file)

# Převod sloupce popularity_normalized na string a nahrazení tečky čárkou
df['popularity_normalized'] = df['popularity_normalized'].astype(str).str.replace('.', ',', regex=False)

# Přidání stejné úpravy pro sloupec age_category
df['age_category'] = df['age_category'].astype(str).str.replace('.', ',', regex=False)

# Uložení CSV s oddělovačem středníkem
df.to_csv(output_file, sep=';', index=False)

print(f"Soubor uložen jako {output_file}")
```

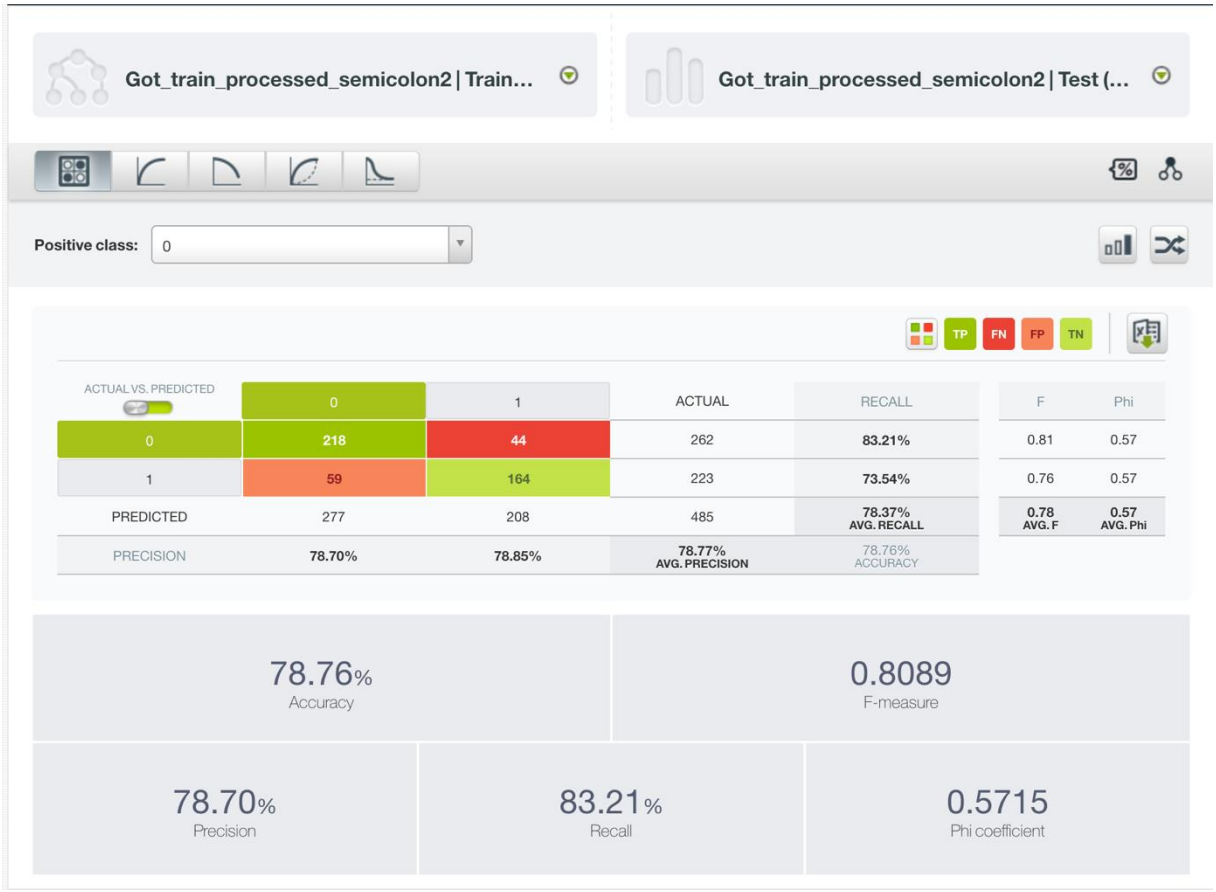
```
Soubor uložen jako got_train_processed_semicolon2.csv
```

BigML

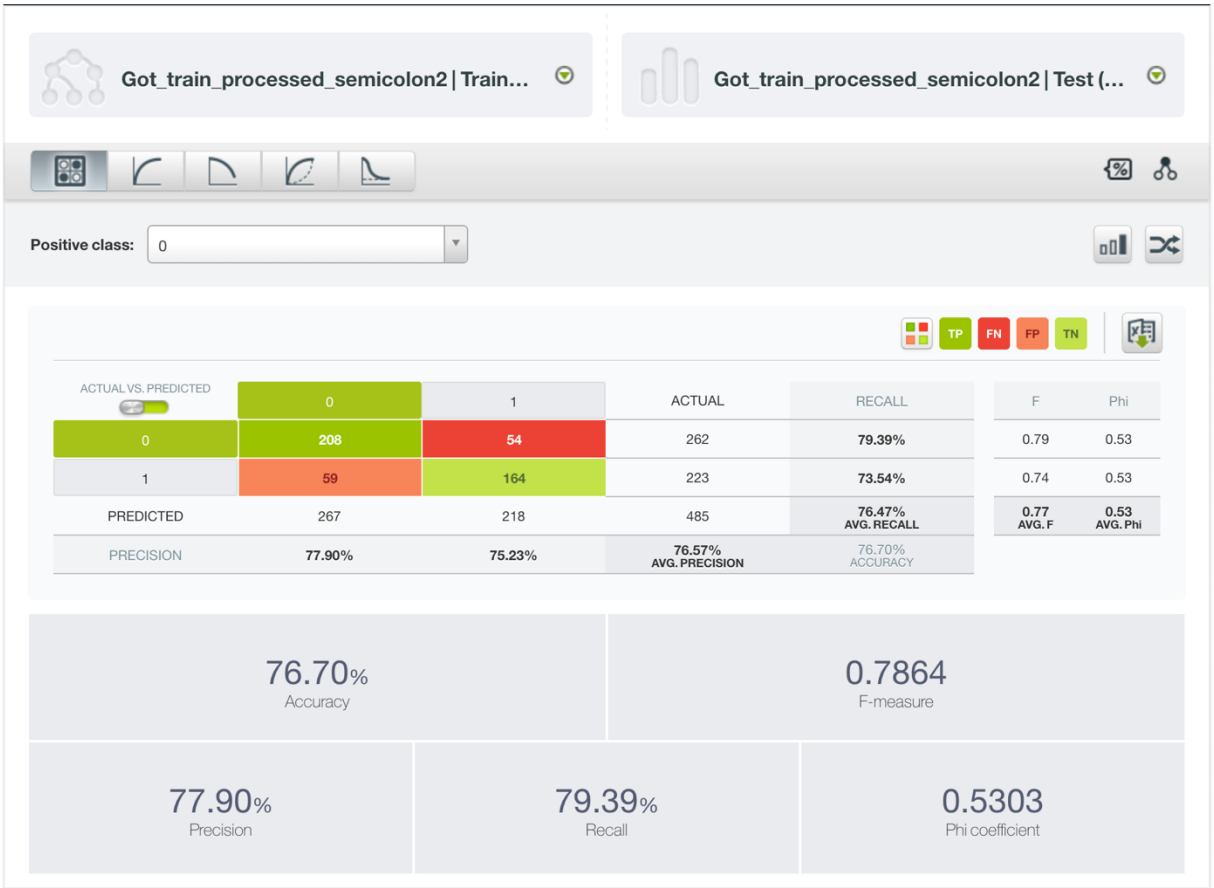
Rozhodovací strom

U rozhodovacího stromu už pracujeme s daty, které prošli preprocessingem z první části práce i doplňujícím drobným preprocessingem v druhé části. Nejprve jsme evaluovali rozhodovací strom bez úpravy hyperparametrů a následně s úpravou. V tomto případě jsme upravili hodnotu Node Treshold a to z 512 na 100. Hodnotu jsme upravili na 100 z důvodu nízkému počtu vzorků

Evaluace rozhodovacího stromu bez úpravy hyperparametru



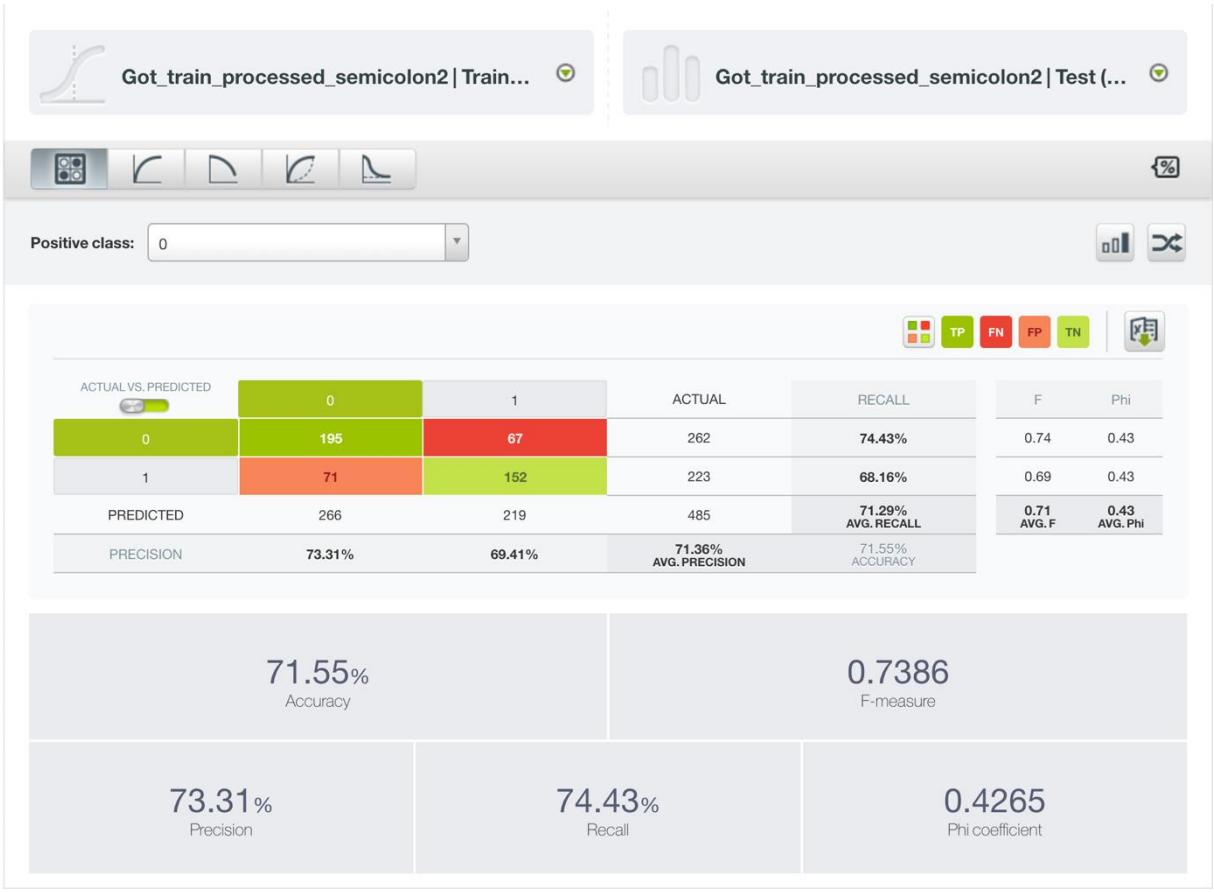
Evaluace rozhodovacího stromu s úpravou hyperparametru



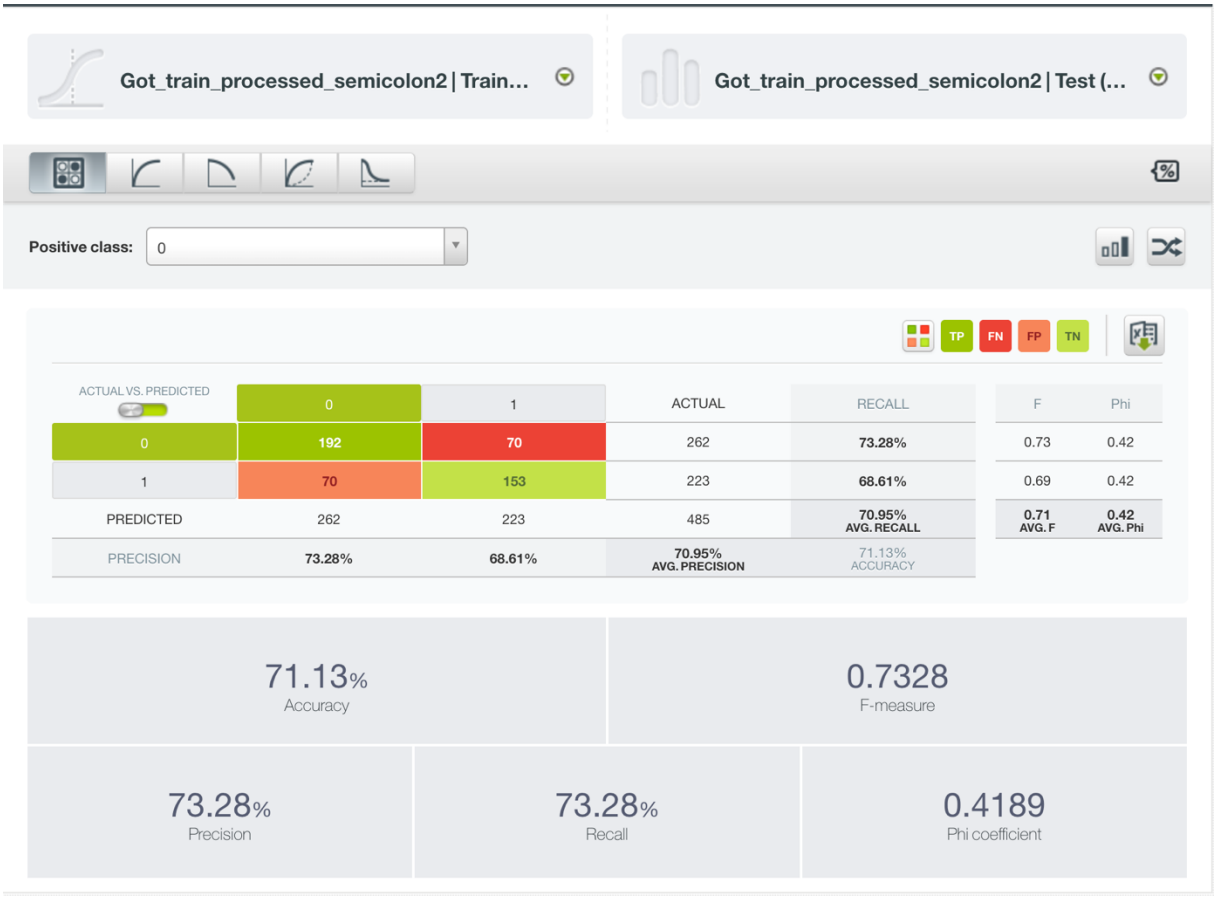
Logistická regrese

U logistické regrese jsme postupovali stejným postupem. Nejdříve jsme evaluovali bez úpravy hyperparametrů a následně se změnami hyperparametrů. U logistické regrese jsme upravili hodnotu C, konkrétně z 0,5 na 1 pro lepší přizpůsobení trénovacím datům, a hodnotu Eps z 0,0001 na 0,00005. Hodnotu Eps jsme zmenšili na polovinu pro zpřísnění kritéria pro ukončení trénování, se snahou o minimalizování ztrátové funkce a pro větší přesnost modelu.

Evaluace logistické regrese bez úpravy hyperparametrů



Evaluace logistické regrese s úpravami hyperparametrů

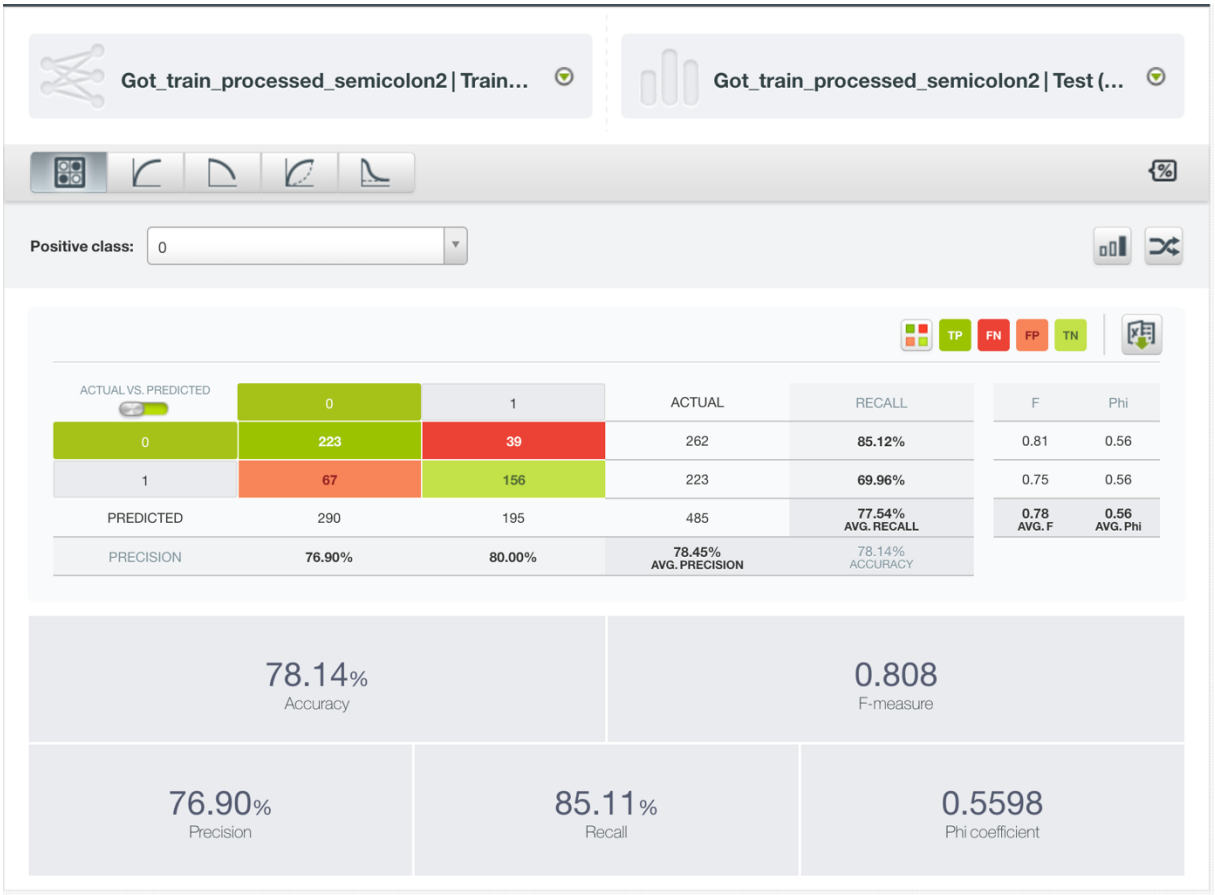


Neuronová síť

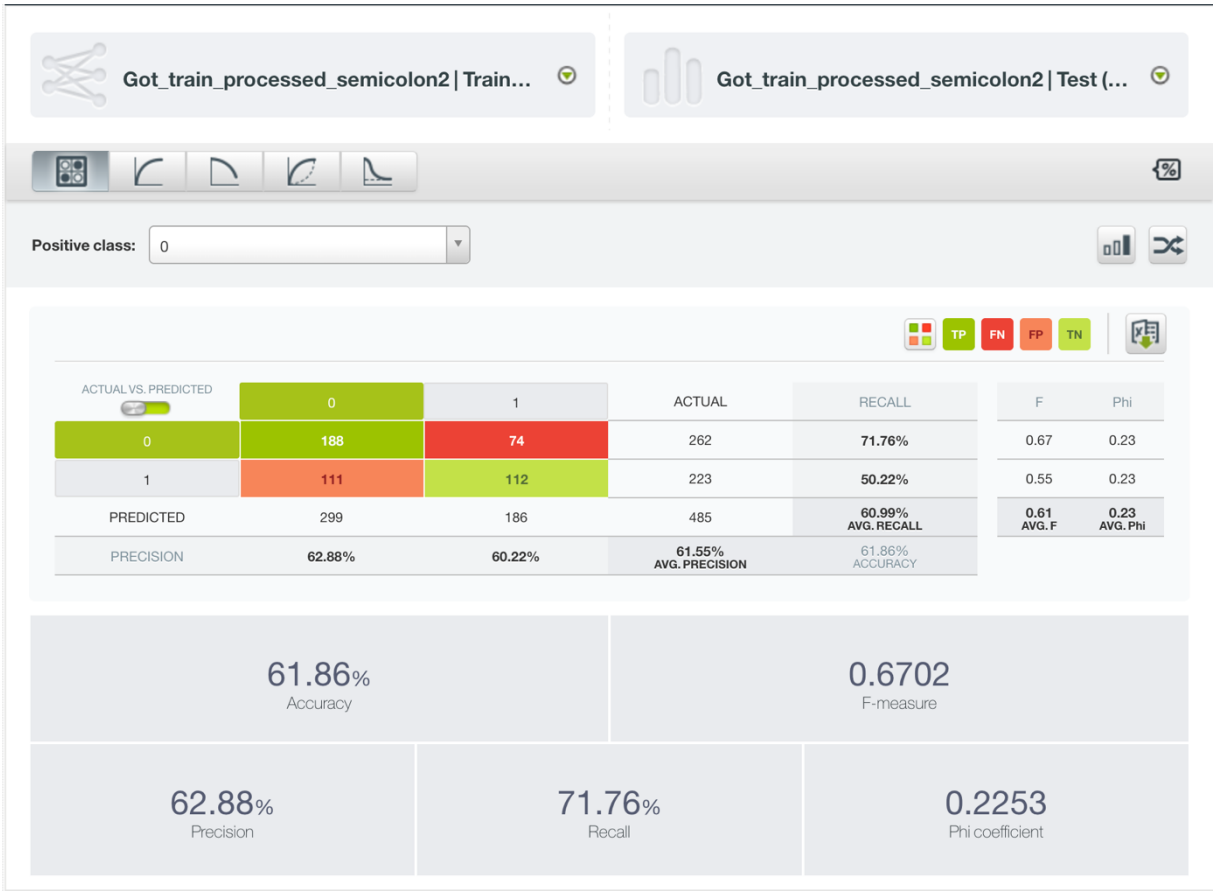
U neuronové sítě jsme také nejdříve evaluovali bez úprav hyperparametrů a následně jsme upravili tyto hypermarametry na následující hodnoty:

- Max iterations = 1000
- Missing numerics = yes
- 3 skryté vrstvy ReLu (128,64,32)
- Learn residuals = Yes
- Batch normalization = Yes
- Tree embedding = Yes
- Algorithm = ADAM
- Dropout rate = 25%
- Learning rate = 0.01%

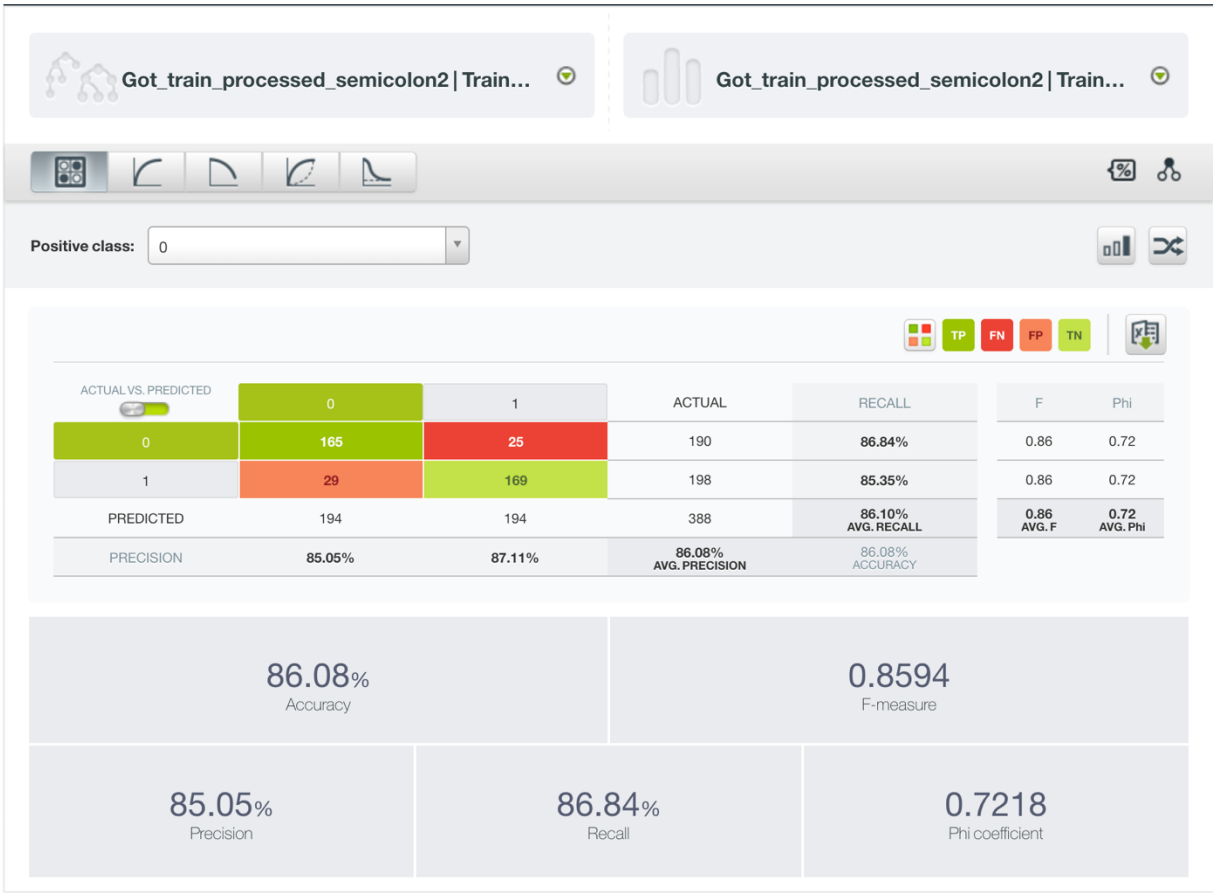
Evaluace neuronové sítě bez úpravy hyperparametrů



Evaluace neuronové sítě s úpravami hyperparametrů



OptiML



OptiML bylo vytvořeno pomocí 1-click funkce a po 28 minutách bylo vyhodnoceno 66 modelů, přičemž nejlepších výsledků dosáhla metoda boosted trees s těmito hyperparametry:

- Node threshold = 1121
- Missing splits = No
- Iterations = 33
- Early stopping = Early out of bag
- Learning rate = 28.679462981533028%

Zhodnocení BigML

Tabulka evaluace z první části semestrální práce

Metrika	Rozhodovací strom	Logistická regrese	Neuronová síť
Accuracy	74,02 %	72,58 %	73,20 %
Precision (0)	70,31 %	71,05 %	70,32 %
Precision (1)	79,69 %	74,43 %	77,23 %
Recall (0)	84,08 %	77,14 %	81,22 %
Recall (1)	63,75 %	67,92 %	65,00 %
F-measure	0,7658	0,7397	0,7538
Phi koeficient	0,489	0,4527	0,4688

Tabulka evaluace z druhé části práce

	Bez úprav hyperparametrů			S úpravami hyperparametrů			OptiML
	Rozhodovací strom	Logistická regrese	Neuronová síť	Rozhodovací strom	Logistická regrese	Neuronová síť	
Accuracy	78,76%	71,55%	78,14%	76,70%	71,13%	61,86%	86,08%
Precision	78,70%	73,31%	76,90%	77,90%	73,28%	62,88%	85,05%
Recall	83,21%	74,43%	85,11%	79,39%	73,28%	71,76%	86,84%
F-measure	0,8089	0,7386	0,8080	0,7864	0,7328	0,6702	0,8594
Phi koeficient	0,5715	0,4265	0,5598	0,5303	0,4189	0,2253	0,7218

Pokud porovnáme tyto dvě tabulky tak můžeme vidět, že se nám vylepšení modelu vcelku povedlo. V první části si nejlépe vedl rozhodovací strom, stejně tomu tak je i v druhé části a jeho výsledky se ještě o několik procent zlepšily. I s úpravami hyperparametrů dopadl rozhodovací strom lépe než v první části semestrální práce. Logistická regrese se oproti rozhodovacímu stromu o tolik nezlepšila, v některých metrikách naopak zhoršila. Pokud porovnáme logistickou regresi pouze z druhé části práce, tak můžeme vidět, že bez úprav hyperparametrů dopadla evaluace lépe než s úpravami. U neuronové sítě můžeme opět vidět zlepšení, ale pouze u pokusů bez úprav hyperparametrů. Bohužel, neuronová síť s úpravami hyperparametrů je o dost horší, jak bez úprav, tak i oproti neuronové síti z první části práce s méně obsáhlým preprocessingem. Absolutní vítěz je OptiML, který dosáhl nejlepších výsledků ze všech modelů. Nejspíše to bude tím, že OptiML zvolil metodu boosted trees s hyperparametry, které zmiňujeme u postupu práce s OptiML.

MS Azure

V MS Azure jsme využili pouze doplnění drobného preprocessingu a neupravovali jsme hyperparametry.

Logistická regrese

Řádky 12 Sloupce 11

Fold Number	Number of examples in fold	AUC	Accuracy	Precision	Recall	F1	True Negative	False Positive	False Negative	True Positive
0	242	0.791945	0.710744	0.753731	0.731884	0.742647	71	33	37	101
1	242	0.791632	0.731405	0.724771	0.692982	0.70852	98	30	35	79
2	243	0.797906	0.707819	0.697248	0.666667	0.681614	96	33	38	76
3	242	0.749437	0.690083	0.701754	0.661157	0.680851	87	34	41	80
4	243	0.78476	0.699588	0.758929	0.648855	0.699588	85	27	46	85
5	242	0.840384	0.752066	0.796296	0.693548	0.741379	96	22	38	86
6	242	0.802336	0.72314	0.741071	0.68595	0.712446	92	29	38	83
7	242	0.732514	0.677686	0.707547	0.614754	0.657895	89	31	47	75
8	243	0.828672	0.748971	0.727273	0.758621	0.742616	94	33	28	88
9	243	0.769758	0.703704	0.669565	0.693694	0.681416	94	38	34	77
Mean	2424	0.788934	0.714521	0.727819	0.684811	0.704897	90.2	31	38.2	83
Standard Deviation	2424	0.032713	0.024288	0.036411	0.040793	0.030107	7.941452	4.371626	5.573748	7.717225

Neuronová síť

Řádky 12 Sloupce 11

Fold Number	Number of examples in fold	AUC	Accuracy	Precision	Recall	F1	True Negative	False Positive	False Negative	True Positive
0	242	0.814033	0.731405	0.806723	0.695652	0.747082	81	23	42	96
1	242	0.826172	0.752066	0.75	0.710526	0.72973	101	27	33	81
2	243	0.835033	0.740741	0.752475	0.666667	0.706977	104	25	38	76
3	242	0.798477	0.727273	0.757009	0.669421	0.710526	95	26	40	81
4	243	0.826881	0.748971	0.836538	0.664122	0.740426	95	17	44	87
5	242	0.870797	0.77686	0.857143	0.677419	0.756757	104	14	40	84
6	242	0.82105	0.731405	0.791667	0.628099	0.700461	101	20	45	76
7	242	0.790574	0.702479	0.740385	0.631148	0.681416	93	27	45	77
8	243	0.848018	0.757202	0.747826	0.741379	0.744589	98	29	30	86
9	243	0.807569	0.72428	0.689655	0.720721	0.704846	96	36	31	80
Mean	2424	0.82386	0.739268	0.772942	0.680515	0.722281	96.8	24.4	38.8	82.4
Standard Deviation	2424	0.023683	0.020657	0.049947	0.036861	0.02474	6.762642	6.292853	5.672546	6.168018

Rozhodovací strom

Řádky 12 Sloupce 11

Fold Number	Number of examples in fold	AUC	Accuracy	Precision	Recall	F1	True Negative	False Positive	False Negative	True Positive
										
0	242	0.943701	0.85124	0.864286	0.876812	0.870504	85	19	17	121
1	242	0.933046	0.834711	0.808333	0.850877	0.82906	105	23	17	97
2	243	0.940024	0.82716	0.795082	0.850877	0.822034	104	25	17	97
3	242	0.909808	0.805785	0.813559	0.793388	0.803347	99	22	25	96
4	243	0.934501	0.855967	0.887097	0.839695	0.862745	98	14	21	110
5	242	0.934151	0.859504	0.862903	0.862903	0.862903	101	17	17	107
6	242	0.947749	0.900826	0.929204	0.867769	0.897436	113	8	16	105
7	242	0.900615	0.818182	0.825	0.811475	0.818182	99	21	23	99
8	243	0.924111	0.831276	0.8	0.862069	0.829876	102	25	16	100
9	243	0.953215	0.909465	0.886957	0.918919	0.902655	119	13	9	102
Mean	2424	0.932092	0.849412	0.847242	0.853478	0.849874	102.5	18.7	17.8	103.4
Standard Deviation	2424	0.016503	0.033857	0.045356	0.034597	0.034297	9.095176	5.638164	4.417138	7.734483

AutoML

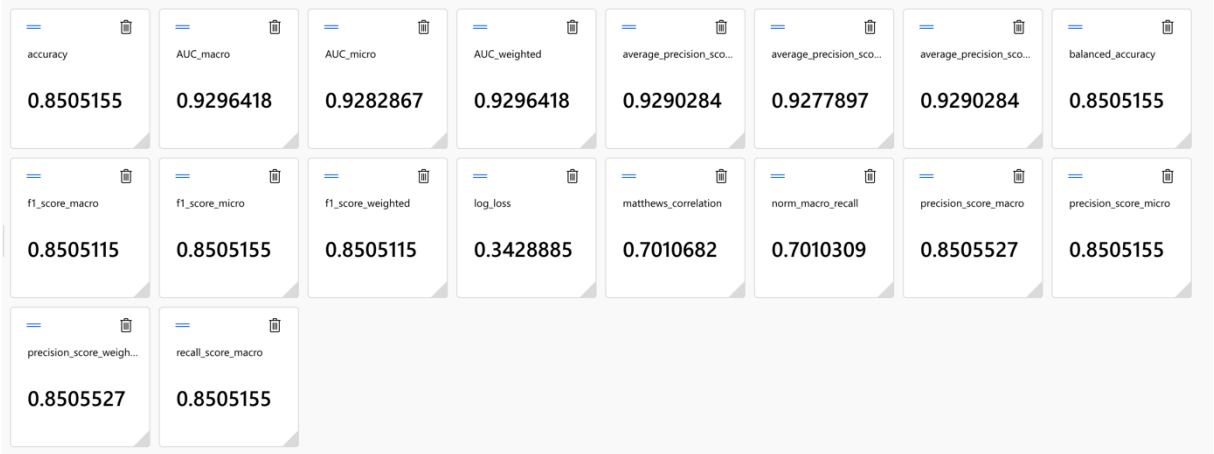
Hyperparametry

Transformace dat:

```
{
    "class_name": "StandardScaler",
    "module": "sklearn.preprocessing",
    "param_args": [],
    "param_kwargs": {
        "with_mean": false,
        "with_std": false
    },
    "prepared_kwargs": {},
    "spec_class": "preproc"
}
```

Trénovací algoritmus:

```
    "class_name": "XGBoostClassifier",
    "module": "automl.client.core.common.model_wrappers",
    "param_args": [],
    "param_kwargs": {
        "booster": "gbtree",
        "colsample_bytree": 1,
        "eta": 0.5,
        "gamma": 0,
        "max_depth": 6,
        "max_leaves": 31,
        "n_estimators": 25,
        "objective": "reg:logistic",
        "reg_alpha": 2.5,
        "reg_lambda": 0,
        "subsample": 0.9,
        "tree_method": "auto"
    },
    "prepared_kwargs": {},
    "spec_class": "sklearn"
}
```



V rámci semestrální práce byl využit nástroj AutoML v Microsoft Azure pro nalezení optimálního klasifikačního modelu. Po nastavení klasifikační úlohy s cílovým sloupcem isAlive a spuštění automatického procesu byl vyhodnocen jako nejúspěšnější model XGBoostClassifier s přesností 85%. Model využíval booster typu "gbtree" s learning rate 0,5, maximální hloubkou stromů 6, 25 estimators a subsamplováním 0,9. Na data byla před trénováním aplikována transformace StandardScaler. Výsledný model dosáhl metrik s AUC 0,9296 a F1 skóre 0,85.

Zhodnocení MS Azure

V MS Azure nedošlo k výraznému zlepšení ani zhoršení. Hodnoty evaluace se liší o desetiny procent. Zde tedy drobný preprocessing model nezlepšil. Z trojice rozhodovací strom, neuronová síť a logistická regrese dosáhl nejlepších čísel rozhodovací strom. Tak tomu bylo i v první části práce. Nejhorších výsledků po doplnění drobného preprocessingu dosáhla logistická regrese. To, že zde nedošlo ke zlepšení výsledků, ale v BigML ano, si vysvětlujeme tím, že BigML může mít rozdílné zacházení s kategoriálními atributy, automatické imputace a interní přístup k datové přípravě. Zatímco BigML některé kroky provádí automaticky a reaguje na změny ve struktuře dat, Azure vyžaduje přesnou ruční konfiguraci. Při použití AutoML vyšly výsledky, které se mohly měřit pouze s rozhodovacím stromem, neuronovou síť i logistickou regresi AutoML předčil.

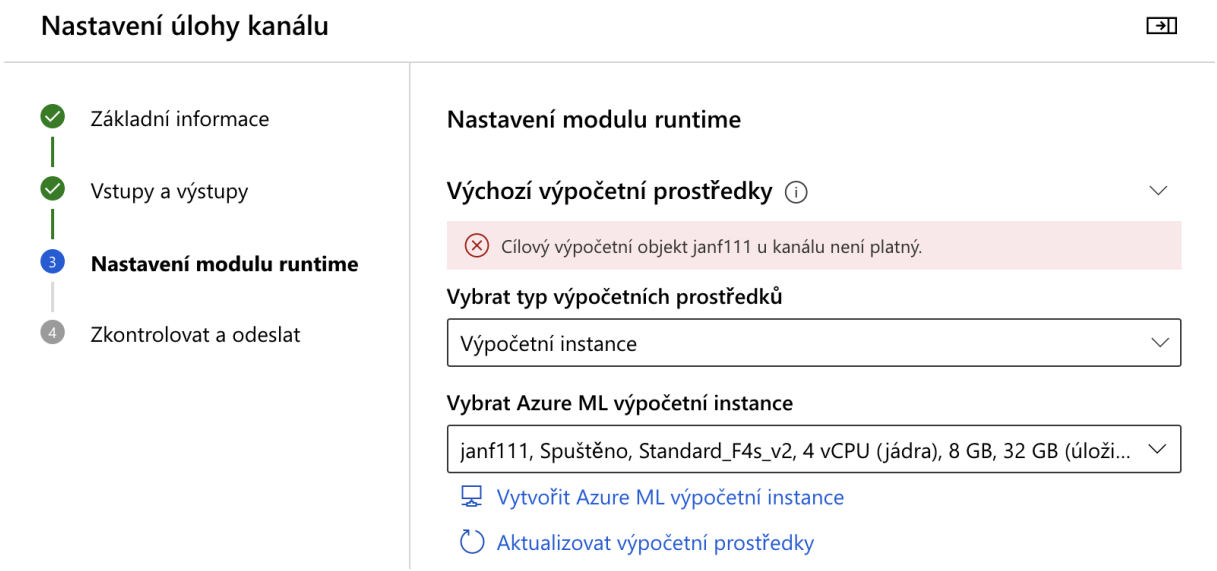
Závěr práce

Z provedené analýzy je patrné, že nejlepších výsledků v rámci klasických algoritmů dosahoval rozhodovací strom, který vykazoval nejvyšší přesnost (accuracy) 74,02% v první části a 78,76% bez úprav hyperparametrů v druhé části. Rozhodovací strom byl stále nejlepší i po optimalizaci hyperparametrů s hodnotou 76,70%, což je překvapivě mírně nižší než verze bez optimalizace, ale stále lepší než výsledky z první části práce.

Logistická regrese vykazovala nejstabilnější, ale ne nejlepší výsledky (72,58% v první části, 71,55% bez úprav a 71,13% s úpravami hyperparametrů v druhé části). Neuronová síť dosáhla smíšených výsledků, zatímco varianta bez úprav hyperparametrů se zlepšila na 78,14%, verze s upravenými hyperparametry výrazně propadla na pouhých 61,86%, což je horší než základní model z první části práce (73,20%).

Práce s platformou Microsoft Azure byla provázena několika významnými problémy:

- Technické komplikace - Opakovaně jsme se potýkali s chybou "Cílový výpočetní objekt není platný", která se objevovala i po vytvoření nové instance. Toto významně komplikovalo a zpomalovalo práci, protože často neexistoval jednoduchý způsob, jak tuto chybu opravit, kromě vytvoření nové instance nebo čekání.
- Finanční omezení - V průběhu experimentování jsme narazili na problém s dostupnými kredity na platformě Azure, které se postupně vyčerpávaly. Toto limitovalo možnosti důkladnějšího testování a optimalizace modelů, zejména při využití výpočetně náročnějších algoritmů.



V našich experimentech dosáhl nejvyšších výsledků model vytvořený pomocí OptiML s přesností 86%. Tento model má následující metriky:

- F-measure: 0,8594
- Precision: 85,05%
- Recall: 86,84%
- Phi coefficient: 0,7218

Tento výsledek překonává nejlepší model z Azure AutoML (85%), přičemž oba modely používají Boosted trees algoritmus.