Technická univerzita v Košiciach

Fakulta elektrotechniky a informatiky

Katedra kybernetiky a umelej inteligencie

# Klasifikácia zákrytových premenných hviezd pomocou hlbokého učenia

Bakalárska práca

# BINARY STARS

Systémová príručka

Vedúci bakalárskej práce:

Doc. Ing. Peter Butka, PhD.

Bakalár:

Maximilián Revický

Konzultant bakalárskej práce:

Ing. Viera Maslej Krešnáková, PhD.

Košice 2022

# Obsah

# Zoznam obrázkov

# 1 Funkcia programu

Úlohou daných skriptov je predpripraviť vstupné dátové množiny do `.pkl` alebo `.csv` súborov a klasifikovať svetelné krivky zákrytových premenných hviezd. Konkrétne vyriešiť následujúce klasifikačné úlohy: (1) Binárna klasifikácia dvojhviezd na oddelené a dotykové systémy. (2) Binárna klasifikácia na dvojhviezdy s kritickým sklonom dráhy a škvrnité hviezdy. Úlohy programu sú rozdelené do viacerých skríptov:

V priečinku `data-preparation/` sa nachádzajú:

- Načítanie syntetických dát oddelených kriviek s kritickým sklonom dráhy z databázy a uloženie do formátu `.pkl`: `binaries_detached_bellow_i_crit_10000.ipynb`
- Načítanie syntetických dát oddelených kriviek z databázy a uloženie do formátu `.pkl`: `binaries_detached_random.ipynb`
- Načítanie syntetických dát dotykových kriviek s kritickým sklonom dráhy z databázy a uloženie do formátu `.pkl`: `binaries_overcontact_bellow_i_crit_10000.ipynb`
- Načítanie syntetických dát dotykových kriviek z databázy a uloženie do formátu `.pkl`: `binaries_overcontact_random.ipynb`
- Načítanie syntetických dát dotykových kriviek z databázy a uloženie do formátu `.pkl`: `single_spotty.ipynb`
- Načítanie observačných dát dvojhviezd z JSON súborov a uloženie do formátu `.csv`: `data_observed_binary.ipynb`
- Načítanie observačných dát dvojhviezd s kritickým sklonom dráhy a škvrnitých hviezd z textových súborov a uloženie do formátu `.csv`: `data_observed_binary_spotty.py`:

V priečinku `modeling/` sa nachádzajú:

- Binárna klasifikácia dvojhviezd na oddelené a dotykové systémy, 3 experiemnty, 2 modely 1D CNN a BiLSTM RNN, vyhodnotenie modelov na syntetických krivkách: `model_3_experiments_binary_stars_50000.ipynb`
- Binárna klasifikácia dvojhviezd na oddelené a dotykové systémy, vyhodnotenie na observačných dátach: `model_observed_binary.ipynb`
- Binárna klasifikácia na dvojhviezdy s kritickým sklonom dráhy a škvrnité hviezdy, 1 experiment, 1 model 1D CNN a BiLSTM RNN, vyhodnotenie modelu na syntetických krivkách: `model_experiment_single_spotty_critical_binary.ipynb`
- Binárna klasifikácia na dvojhviezdy s kritickým sklonom dráhy a škvrnité hviezdy, vyhodnotenie modelu na observačných krivkách: `model_observed_spotty_binary.ipynb`
- Funkcia šumu `stochastic_noise_generator`, ktorá sa aplikuje na syntetické dáta, aby sa priblížila viac observačným dátam zo skriptu `noise_generator.py`
- Funkcia na náhodné zarovnanie kriviek oddelených a dotykových systémov s kritickým sklonom dráhy `curve_alignement_randomizer` zo skriptu `curve_alignement.py`

# 2 Inštalácia programu

## 2.1 Požiadavky na programové prostriedky

- Python 3.8.6
- Anaconda Navigator
- Jupyter Notebook
- Knižnice: `numpy`, `pandas`, `json`, `pickle`, `sklearn`, `keras`, `tensorflow`, `imblearn`, `collections`, `scikitplot`

# 3 Popis programu

V tejto práci sme vykonali viacero experimentov, pracovali s rôznymi modelmi a vstupnými dátami. Rozhodli sme sa opísať skript `modeling/model_experiment_single_spotty_critical_binary.ipynb` . Celý projekt je dostupný na github: `github.com/MaxRevicky/Classification-of-eclipsing-binary-stars`

1. Na obrázku 3 – 1 vidíme príkazy na importovanie knižníc, s ktorými sa v skripte pracuje.

```
import numpy as np
import pandas as pd
import json
import pickle
np.random.seed(1234)

from noise_generator import stochastic_noise_generator
from curve_alignement import curve_alignement_randomizer
from sklearn.model_selection import train_test_split

from keras.utils import np_utils
from keras.models import load_model
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import precision_recall_fscore_support
from sklearn.model_selection import train_test_split

from keras.layers import Conv1D, GlobalMaxPooling1D, MaxPooling1D, SpatialDropout1D, GlobalAveragePooling1D
from keras.layers import Input, Dense, concatenate, Activation, LSTM, Bidirectional, Flatten, Dropout
from keras.models import Model
from keras.models import Sequential
from keras.layers.merge import Concatenate
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
import matplotlib.pyplot as plt

import imblearn
from imblearn import under_sampling, over_sampling
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter

import scikitplot as skplt
```

**Obr. 3 – 1** Importovanie knižníc

2. V tejto časti môžeme vidieť načítanie dát škvrnitých hviezd, dotykových hviezd a oddelených hviezd. V datasete s oddelenými hviezdami sme preklasifikovali hodnoty v stĺpci `overcontact` na hodnoty 1 (pozri obr. 3 – 2).

3. Aplikovanie funkcie `curve_alignement_randomizer` na dotykové a oddelené krivky a spojenie načítaných datasetov do jedného datasetu (pozri obr. 3 – 3).

```
data_single_spotty_0 = pd.read_pickle("single_spotty.pkl")
data_single_spotty_0.head(1)
```

• • •

```
data_overcontact_1 = pd.read_pickle("overcontact_bellow_i_crit_10000.pkl")
data_overcontact_1.head(1)
```

• • •

```
data_detached_1 = pd.read_pickle("detached_bellow_i_crit_10000.pkl")
data_detached_1['overcontact'].values[:] = 1
data_detached_1.head(1)
```

**Obr. 3 – 2** Načítanie dát

```
data_overcontact_1= pd.DataFrame(curve_alignement_randomizer(data_overcontact_1),
                                 columns=['id', 'Bessell_U', 'Bessell_B', 'Bessell_V', 'Bessell_R', 'Bessell_I', 'overcontact'])

data_detached_1= pd.DataFrame(curve_alignement_randomizer(data_detached_1),
                                 columns=['id', 'Bessell_U', 'Bessell_B', 'Bessell_V', 'Bessell_R', 'Bessell_I', 'overcontact'])

data = pd.concat([data_single_spotty_0, data_overcontact_1, data_detached_1])
data.head(2)
```

**Obr. 3 – 3** Aplikovanie funkcie `curve_alignement_randomizer` a spojenie datasetov

4. Rozdelenie datasetu na trénovaciu a testovaciu množinu v pomere 80:20 (pzori obr. 3 – 4)

```
X_train, X_test, y_train, y_test = train_test_split(data, data['overcontact'], test_size=0.2, random_state=42)

# Check shapes after split
print(X_train.shape)
print("########")
print(y_train)
print("########")
print(X_test.shape)
print("########")
print(y_test)
print("########")
```

**Obr. 3 – 4** Rozdelenie datasetu na trénovaciu a testovaciu množinu

5. Vytvorenie funkcie, ktorá transformuje všetky krivky v 1 stĺpci (400 čiarkami oddelených čísel) na `numpy` pole (pozri obr. 3 – 5).

6. Aplikovanie funkcií `stochastic_noise_generator` a `bassell_to_array` na trénovaciu a testovaciu množinu (pozri obr. 3 – 6).

```
def bessel_to_array(column_name, df):
    """Model expects array input.
    This function transforms all curves in 1 column (400 comma seperated numbers) to array"""
    newData_bessell = []
    for row in df[column_name]:
        newRow = []
        for valueIndex in range(len(row)):

            newRow.append([row[valueIndex]])
        newData_bessell.append(newRow)
    newData_bessell = np.array(newData_bessell)
    return newData_bessell
```

**Obr. 3 – 5** Funkcia na transformovanie kriviek na `numpy` pole

```
X_train_with_arrays_noise = []
for column in ['Bessell_U','Bessell_B','Bessell_V','Bessell_R','Bessell_I']:
    X_train_with_arrays_noise.append(stochastic_noise_generator(bessel_to_array(column, X_train)))
    print(column + " " + "prossessed")
```

•••

```
X_test_with_arrays_noise = []
for column in ['Bessell_U','Bessell_B','Bessell_V','Bessell_R','Bessell_I']:
    X_test_with_arrays_noise.append(stochastic_noise_generator(bessel_to_array(column, X_test)))
    print(column + " " + "prossessed")
```

**Obr. 3 – 6** Aplikovanie funkcií `stochastic_noise_generator` a `bassell_to_array`

7. Transformovanie `y_train` na `numpy` pole (pozri obr. 3 – 7).

```
# target is overcontact
target = np.array(y_train)
print(target)

# just to be sure that both types are in dataset
exists = 0 in target
print(exists)
exists = 1 in target
print(exists)
```

**Obr. 3 – 7** Transformovanie `y_train` na `numpy` pole

8. Pretransformovanie hodnôt `y_train` na kategorické (pozri obr. 3 – 8).

```
# transform to categorical
y_train = np_utils.to_categorical(y_train, 2)
y_train
```

**Obr. 3 – 8** Pretransformovanie hodnôt `y_train` na kategorické

```
inputs = Input(shape=(400,1))

a = Bidirectional(LSTM(64, return_sequences=True,dropout=0.1,recurrent_dropout=0.1))(inputs)
a = Flatten()(a)

b = Conv1D(32, kernel_size = 3, padding = "valid", input_shape=(400,1))(inputs)
b = MaxPooling1D(2)(b)
b = Conv1D(32, kernel_size = 3, padding = "valid")(b)
b = MaxPooling1D(2)(b)
b = Flatten()(b)

x = concatenate([a,b])
x = Dropout(0.2)(x)
x = Dense(32, activation='relu')(x)

output = Dense(2, activation='softmax')(x)
classifier = Model(inputs=inputs, outputs=output)
classifier.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
print(classifier.summary())
```

**Obr. 3 – 9** Vytvorenie modelu

9. Vytvorenie modelu 1D CNN a BiLSTM (pozri obr. 3 – 9).

10. Nastavenie modelu, aby sa pri trénovaní modelu vytvárali checkpointy (pozri obr. 3 – 10).

```
saved_model = "model_experiment_4.hdf5"
checkpoint = ModelCheckpoint(saved_model, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
early = EarlyStopping(monitor="val_acc", mode="max", patience=3)
callbacks_list = [checkpoint, early]
```

**Obr. 3 – 10** Vytváranie checkpointov

11. Pripravenie dát do modelu na trénovanie (pozri obr. 3 – 11).

```
x_train = np.concatenate((X_train_with_arrays_noise[0][0],
                          X_train_with_arrays_noise[1][0],
                          X_train_with_arrays_noise[2][0],
                          X_train_with_arrays_noise[3][0],
                          X_train_with_arrays_noise[4][0]))
```

```
y_train_mixed = np.concatenate((y_train,
                                y_train,
                                y_train,
                                y_train,
                                y_train))
```

```
history = classifier.fit(x_train, y_train_mixed, validation_split=0.2, epochs=10, batch_size=128, verbose=1, callbacks = callbacks_list)
```

**Obr. 3 – 11** Pripravenie dát do modelu na trénovanie

12. Vytvorenie grafu z priebehu trénovania (pozri obr. 3 – 12).

```
# Define function for plot loss and accuracy during training
def trainingLoss(history):
    plt.style.use('ggplot')
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('')
    plt.ylabel('Accuracy')
    plt.xlabel('Epochs')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.savefig('tarin1.png', bbox_inches='tight')
    plt.show()
    plt.style.use('ggplot')
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('')
    plt.ylabel('Loss')
    plt.xlabel('Epochs')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.savefig('tarin2.png', )
    plt.show()
```

Plot training graphs

```
trainingLoss(history)
```

**Obr. 3 – 12** Vytvorenie grafu z priebehu trénovania

13. Vytvorenie univerzálnej funkcie na vyhodnotenie modelu (Klasifikačný report, kontingenčná tabuľka klasifikácie, ROC krivka, vizualizácia vzorky nesprávne predikovaných kriviek) (pozri obr. 3 – 13 a 3 – 14).

14. Aplikovanie univerzálnej funkcie na vyhodnotenie modelu (pozri obr. 3 – 15).

```python
def evaluate_all(model_path, input_data, y_test):
    classifier = load_model(model_path)
    # y_pred returns probability of detached and overcontact
    # for example [1.4059671e-04, 9.9985933e-01],
    # means detached with prob. 0.0001405  and overcontact with prob 0.998
    y_pred = classifier.predict(input_data)
    # print(y_pred)

    # for example [1.4059671e-04, 9.9985933e-01] -> [0, 1], it is marked as detached
    y_pred2 = np.where(y_pred > 0.5, 1, 0)

    # test data [0,1,...] is converted to categorical [[0,1],[1,0]]
    target_test = np.array(y_test)
    target_test = np_utils.to_categorical(target_test, 2)

    # confusion matrix -old
    # cm = confusion_matrix(target_test.argmax(axis=1), y_pred2.argmax(axis=1))
    # print("Confusion matrix: \n" + str(cm))

    # confusion matrix - nice print
    matrix=skplt.metrics.plot_confusion_matrix(target_test.argmax(axis=1), y_pred2.argmax(axis=1))
    matrix.xaxis.set_ticklabels(['single spotty', 'binary star'])
    matrix.yaxis.set_ticklabels(['single spotty', 'binary star'])
    matrix

    target_names= ['single spotty', 'overcontact']
    print("Classification report: \n" + classification_report(target_test.argmax(axis=1), y_pred2.argmax(axis=1),target_names=target_names))

    # from categorical [0, 1] -> 1
    y_true = target_test.argmax(axis=1)

    skplt.metrics.plot_roc(y_true, y_pred)
    plt.savefig('roc_auc.png')
    plt.show()

    input_data = np.array(input_data)
```

**Obr. 3 – 13** Vytvorenie univerzálnej funkcie na vyhodnotenie modelu (časť 1)

```python
limit_plot=0
for j in range(len(y_pred2)):
    if (y_pred2[j].argmax(axis=0) == 1) and (target_test[j].argmax(axis=0) == 0):
        if input_data.ndim == 4:
            plt.plot(input_data[0][j],label = str(y_test.iloc[[j]].index[0]) + ", " +str(y_test.iloc[[j]].values[0]))
            fig1 = plt.figure(1)
            fig1.text(0.45, 0.9, "single spotty 0", ha="center", va="bottom", color="green")
            fig1.text(0.53, 0.9, "/", ha="center", va="bottom",  color="black")
            fig1.text(0.62, 0.9, "overcontact 1", ha="center", va="bottom",  color="red")
            limit_plot=limit_plot+1
        else:
            plt.plot(input_data[j],label = str(y_test.iloc[[j]].index[0]) + ", " +str(y_test.iloc[[j]].values[0]))
            fig1 = plt.figure(1)
            fig1.text(0.45, 0.9, "single spotty 0", ha="center", va="bottom" ,color="green")
            fig1.text(0.53, 0.9, "/", ha="center", va="bottom",  color="black")
            fig1.text(0.62,0.9, "overcontact 1", ha="center", va="bottom",  color="red")
            limit_plot=limit_plot+1
        if limit_plot == 2:
            break


plt.legend(loc='upper right')
plt.savefig('image.png')
plt.show()


limit_plot2=0
for j in range(len(y_pred2)):
    if (y_pred2[j].argmax(axis=0) == 0) and (target_test[j].argmax(axis=0) == 1):
        if input_data.ndim == 4:
            plt.plot(input_data[0][j],label = str(y_test.iloc[[j]].index[0]) + ", " +str(y_test.iloc[[j]].values[0]))
            fig1 = plt.figure(1)
            fig1.text(0.45, 0.9, "single spotty 0", ha="center", va="bottom", color="red")
            fig1.text(0.53, 0.9, "/", ha="center", va="bottom",  color="black")
            fig1.text(0.62, 0.9, "overcontact 1", ha="center", va="bottom",  color="green")
            limit_plot2=limit_plot2+1

        else:
            plt.plot(input_data[j],label = str(y_test.iloc[[j]].index[0]) + ", " +str(y_test.iloc[[j]].values[0]))
            fig1 = plt.figure(1)
            fig1.text(0.45, 0.9, "single spotty 0", ha="center", va="bottom", color="red")
            fig1.text(0.53, 0.9, "/", ha="center", va="bottom",  color="black")
            fig1.text(0.62, 0.9, "overcontact 1", ha="center", va="bottom",  color="green")
            limit_plot2=limit_plot2+1
        if limit_plot2 == 2:
            break

plt.legend(loc='upper right')
plt.savefig('image2.png')
plt.show()
```

**Obr. 3 – 14** Vytvorenie univerzálnej funkcie na vyhodnotenie modelu (časť 2)

```
processed_test_data= np.concatenate((
 X_test_with_arrays_noise[0][0],
 X_test_with_arrays_noise[1][0],
 X_test_with_arrays_noise[2][0],
 X_test_with_arrays_noise[3][0],
 X_test_with_arrays_noise[4][0]))

y_test_mixed = pd.concat([y_test,
                          y_test,
                          y_test,
                          y_test,
                          y_test])
print(y_test_mixed.shape)
print(processed_test_data.shape)

evaluate_all('model_experiment_4.hdf5', processed_test_data, y_test_mixed)
```

**Obr. 3 – 15** Aplikovanie univerzálnej funkcie na vyhodnotenie modelu