FT PARK z.ú.

Identifikátor výsledku: FT08082020
Typ výsledku dle číselku RVVI: software (R)
Název výsledku: Neurální síť na predikci diabetes
Způsob zveřejnění: http://www.ftpark.cz/
Vedoucí řešitel: Radek Jirkovský
Řešitel: Rudolf Holík
Datum dokončení: 8. srpna 2020

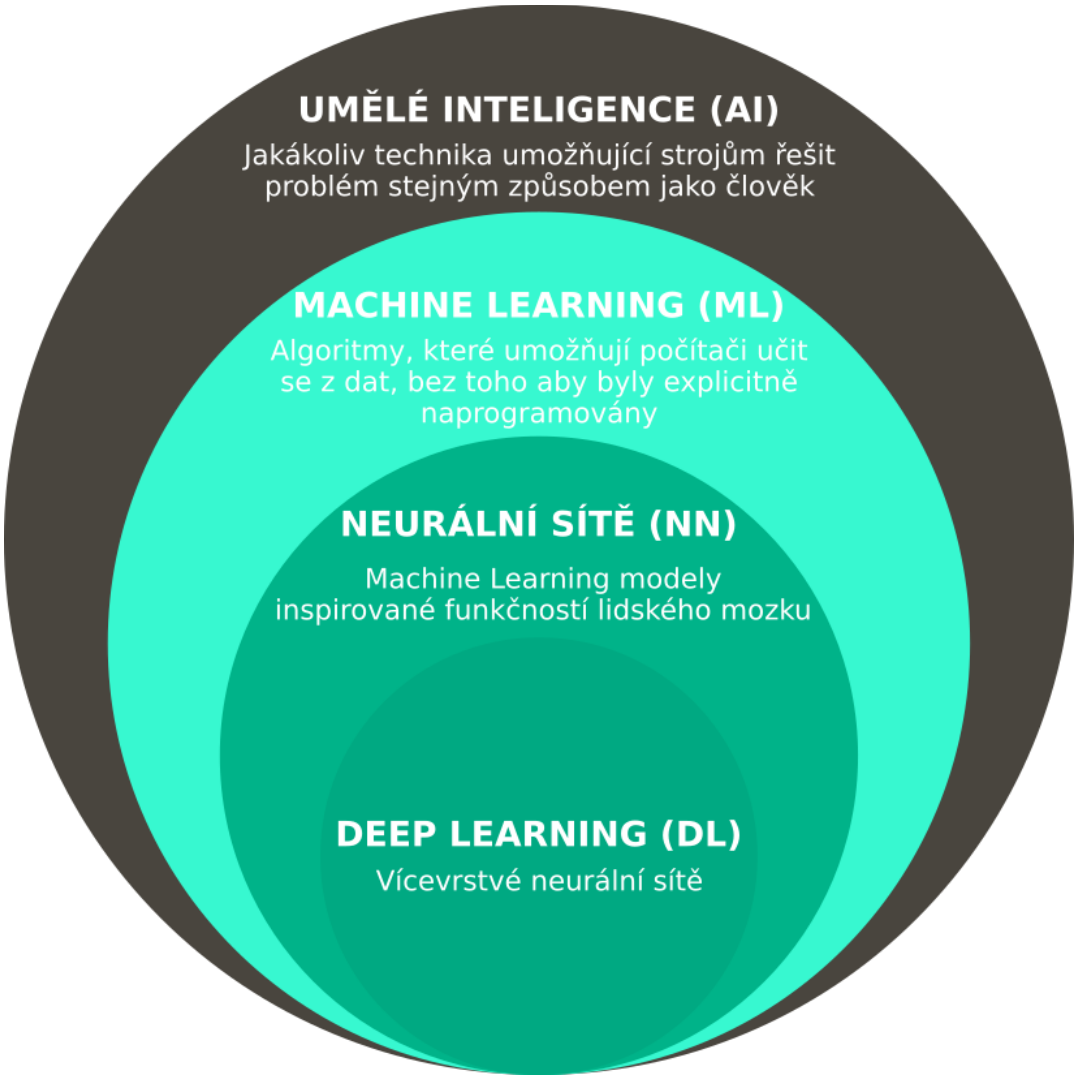
Součástí balíčku souborů výsledku FT08082020 je:

1. Model Neurální sítě na predikci diabetes ve formátu .pkl
2. Doprovodná dokumentace ve formátu .pdf

1. ÚVOD

1.1. Umělé inteligence a hierarchie

Umělé inteligence (AI) je založena na principu simulování lidské inteligence a řešit tak převážně nealgoritmické problémy podobným způsobem jako člověk. Flexibilita této technologie umožňuje uplatnění AI téměř v každém oboru. AI je již široce využívané v mnoha oborech a v mnoha případech vykazuje výsledků, které předčí i profesionály svého oboru. Například Kheiron medical vytvořil model na základě generativní adversiální sítě, který byl schopen vytvořit syntetické obrazy mammogramu s kvalitou nerozeznatelnou od opravdových snímků [1]. Cílem FT Parku z.ú. je vývoj produktu na základě AI v oblasti zdravotnictví.



1.2. Rozsah a cíl projektu

Cílem projektu je vytvořit neurální síť která by byla schopna predikovat pravděpodobnost zda pacient trpí cukrovkou na základě několika metrik za využití Deep Learningu (DL).

1.3. Užité technologie

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, Dropout, BatchNormalization
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from numpy import loadtxt
from sklearn.externals import joblib
import pickle
```

Model byl vytrénovaný za pomoci anacondy, což je jedna z distribucí programovacího jazyka Python.

Užité knihovny a frameworky:

- Tensorflow: open-source knihovna pro ML, DL a stavbu NN
- Keras: open-source knihovna pro DL a stavbu NN
- Scikit-learn: knihovna pro ML
- Pandas: knihovna pro manipulaci s daty
- Numpy: knihovna na operování s multidimenzionální maticemi
- Matplotlib: knihovna na vizualizaci dat
- Pickle: knihovna na exportování hodnot z developerského prostředí
- Django: framework na implementaci Pythonu do backendu webu

2. DATA

2.1. Importování datasetu

Dataset je třeba importovat do python prostředí. Pro další manipulaci nelze data nechat v xls, json, nebo csv formátu ale musí být konvertována do proměnných.

Python syntax pro importování datasetu do developerského prostředí:

```
data = pd.read_csv("data/diabetes.csv")
```

2.2. Dataset

Dataset využitý pro trénování NN byl získán z veřejného datasetu [2]. Dataset je ve formátu xls, má 769 různých subjektů a obsahuje 9 atributů, z čehož je jeden výsledný. S datasetem se musí manipulovat před tím než jsou data užita k trěnování NN, v tomto případě data musela být sanitizována, normalizována a rozdělena na trénovací a testovací data.

2.3. Sanitizace dat

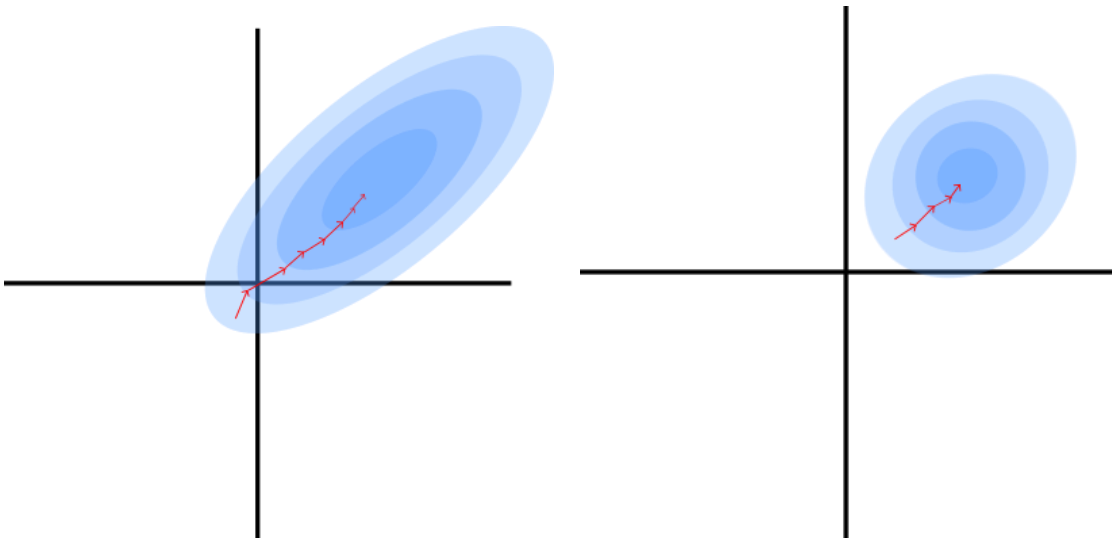
Pokud dataset obsahuje data které by negativně mohly ovlivnit trénovací proces NN, tak se musí sanitizovat, což znamená vyloučení určitých dat z datasetu, nebo kompletaci chybějících/poškozených dat . Ne všechna data byly však užita k trénování naší neurální sítě, jelikož u některých subjektů data chyběla, byla v nevhodném formátu nebo byla špatně vyplněna.

Python syntax pro sanitizaci atributů, které by se neměly rovnat nule:

```
data = data[data['SkinThickness'] != 0]
data = data[data['Insulin'] != 0]
data = data[data['BloodPressure'] != 0]
data = data[data['BMI'] != 0]
```

2.4. Normalizace a Standardizace dat

Data se po sanitizaci musí normalizovat nebo standardizovat pro zrychlení procesu trénování. Neurální sítě a různé regrese používají proces zvaný “gradient descent” pro hledání lokálního minima. Standardizací/normalizací se distribuce dat v prostoru zmenší a dosáhnutí lokálního minima je možné dosáhnout v méně krocích.

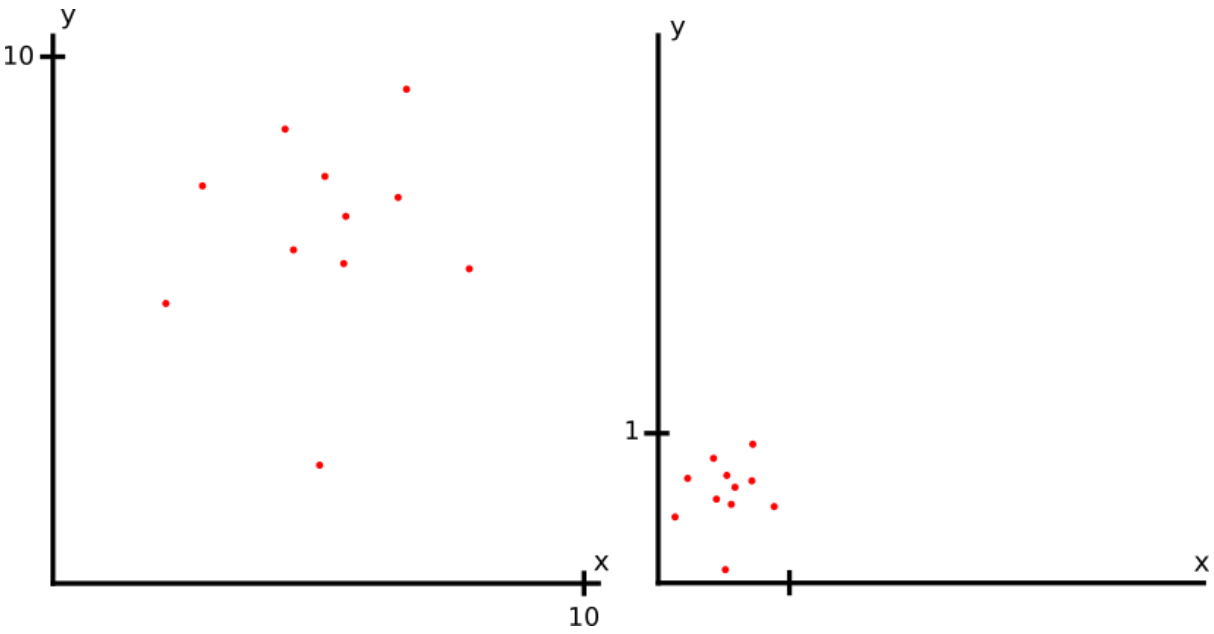


Gradient descent - Neoptimalizovaná data (vlevo), Optimalizovaná data (vpravo)

Při normalizaci se data vyškálují do rozsahu od 1 do 0. Tento proces je také známý jako min-max škálování.

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Reprezentace dat před a po normalizaci dat:



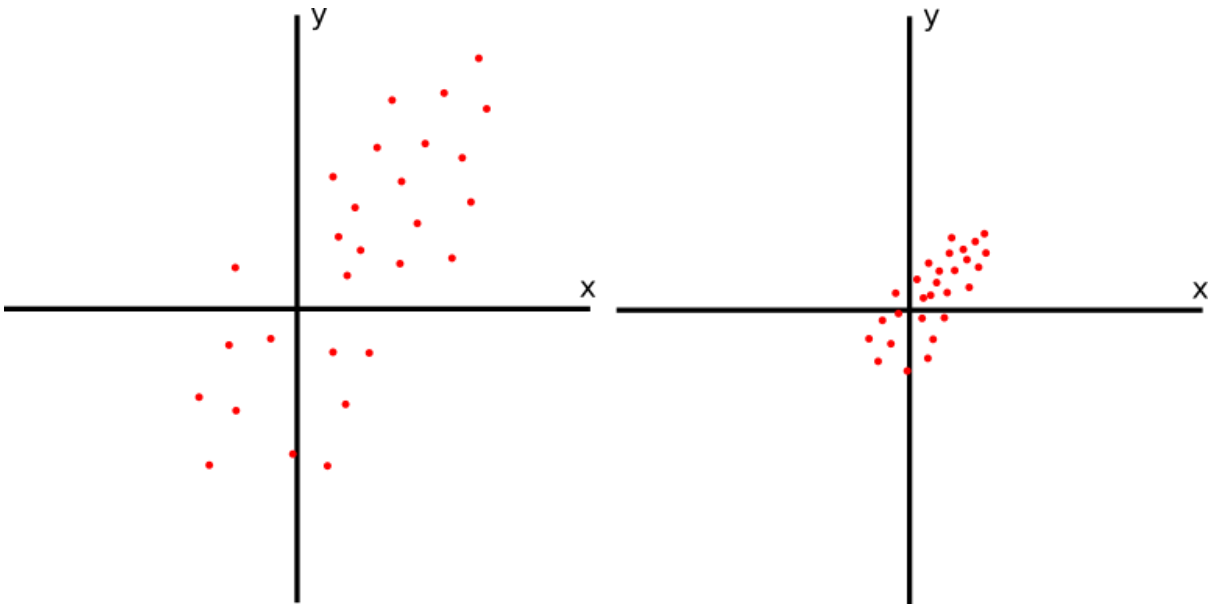
Distribuce dat před (vlevo) a po (vpravo) normalizaci

Při standardizaci se data škálují na základě centra, které je průměrem standardní odchylky datových bodů.

$$X' = \frac{X - \mu}{\sigma}$$

μpředstavuje průměr všech hodnot v datasetu a σpředstavuje standardní odchylku všech hodnot datasetu.

Reprezentace dat před a po standardizaci dat:



Distribuce dat před (vlevo) a po (vpravo) standardizaci

Python syntax pro standardizaci dat:

```
df_x = preprocessing.scale(df_x)
```

Python syntax pro normalizaci dat (nebylo užito):

```
df_x = preprocessing.normalize(df_x)
```

3. TRÉNOVÁNÍ MODELU

3.1. Trénovací a testovací data

Sanitizovaný dataset nelze užít celý k trénování neurální sítě. Data musí být rozdělena na trénovací a testovací. Dataset pro tento model byl rozdělen na 80% trénovacích a 20% testovacích dat.

Python syntax pro rozdělení Pandas data frame na výsledná data df_y a atributy df_x na základě kterých se bude predikovat hodnoty v df_y:

```
df_y = sanitized_data['Outcome']
df_x = sanitized_data.drop('Outcome', 1)
```

Python syntax pro konverzi pandas data frame na Numpy pole dat (neurální sítě nelze trénovat z pandas data frame formátu):

```
x = np.array(df_x)
y = np.array(df_y)
```

Python syntax pro rozdělení dat na trénovací a testovací:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

x_train - trénovací data atributů
x_test - testovací data atributů
y_train - trénovací data predikovaných hodnot
y_test - testovací data predikovaných hodnot

3.2. Struktura modelu

Neurální síť má 1 vstupní vrstvu tvořenou z 8 vstupů, 3 skryté vrstvy kde každá vrstva obsahuje 50 neuronů a jednu výstupní vrstvu. Aktivační funkce pro neurony skrytých vrstev je relu a pro výstupní vrstvu je užita sigmoid funkce.

Vrstvy neurální sítě byly optimalizovány “Batch Normalization”, což je jedna z forem normalizace (viz. 2.4.). Dále pro optimalizaci byla užita “Dropout” vrstva, která při každé propagaci dat náhodně vypne určitý počet neuronů dané vrstvy (V modelu F.T. parku z.ú. 30%), což zabraňuje overfittingu (viz 4.1.).

Python syntax implementace struktury modelu:

```
model = Sequential()
model.add(Dense(50, input_dim=8, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.3))
model.add(Dense(50, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(50, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))
```

3.3. Trénování modelu

Model byl trénovaný na 100 epochů (počet propagací dat neurální sítí). S batch size 70 (velikost jedné propagované várky).

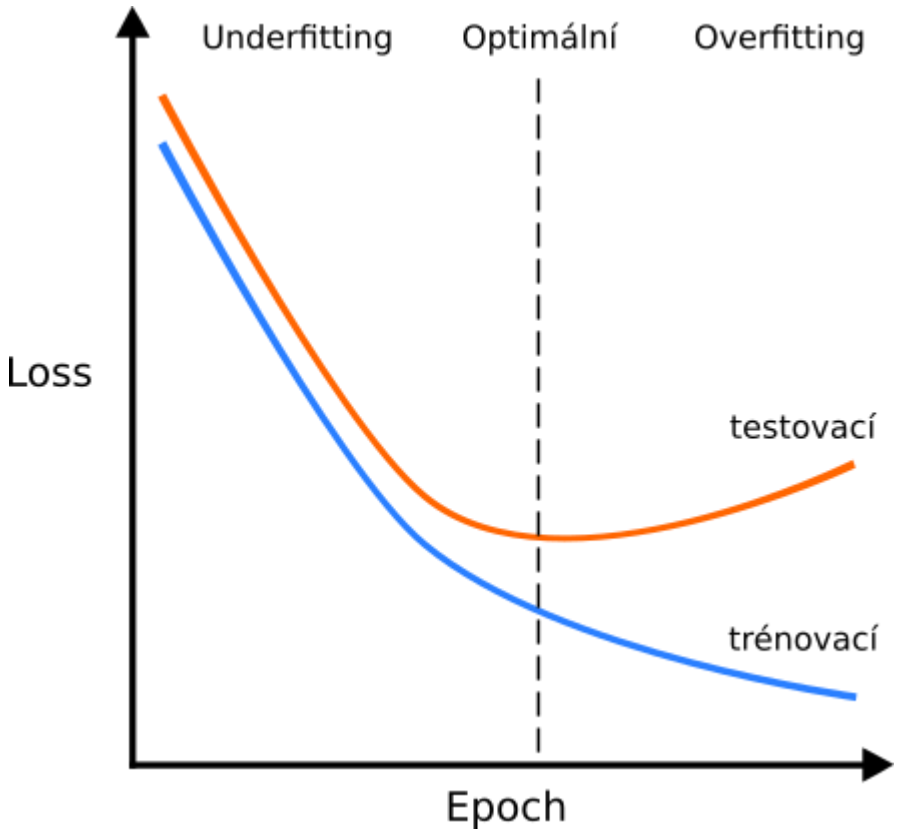
Pyton syntax pro spuštění kompilování a trénování modelu:

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=100, batch_size=70, validation_data=(x_test, y_test))
```

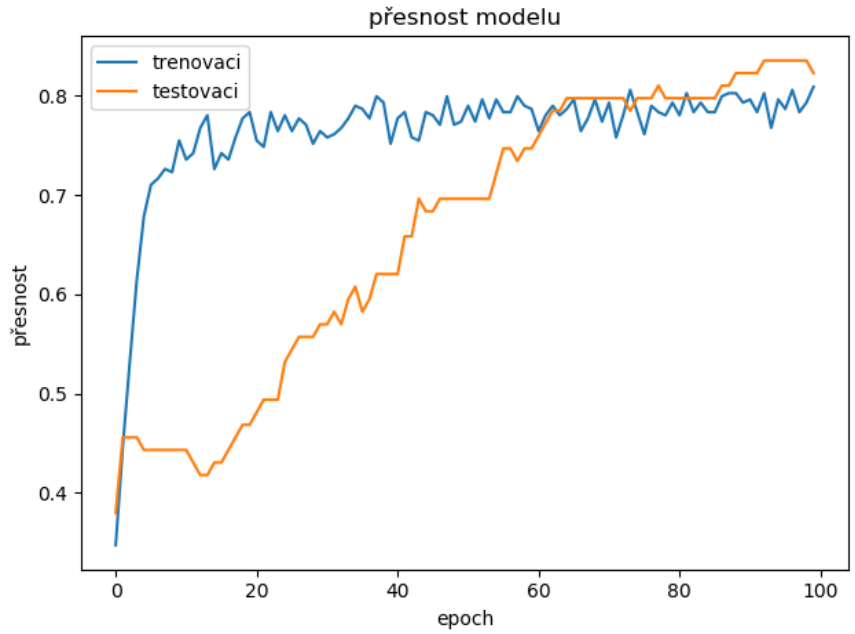
4. VYHODNOCOVÁNÍ EFEKTIVNOSTI MODELU

4.1. Underfitting a overfitting

Při trénování modelu více propagací nutně neznamená lepší výkon modelu a proto je třeba najít optimální počet epochů. V případě že bylo trénováno s málo epochy, model bude data příliš generalizovat, což lze vidět na první polovině grafu. V případě že budeme trénovat s mnoha epochy, model bude mít vysokou varianci, začne kopírovat trénovací data a dále se bude oddalovat od testovacích dat. Námi vytvořený model dosahoval přesnosti přes 80% na testovacích datech. Detail o průběhu trénování modelu lze vidět v sekci 4.2.

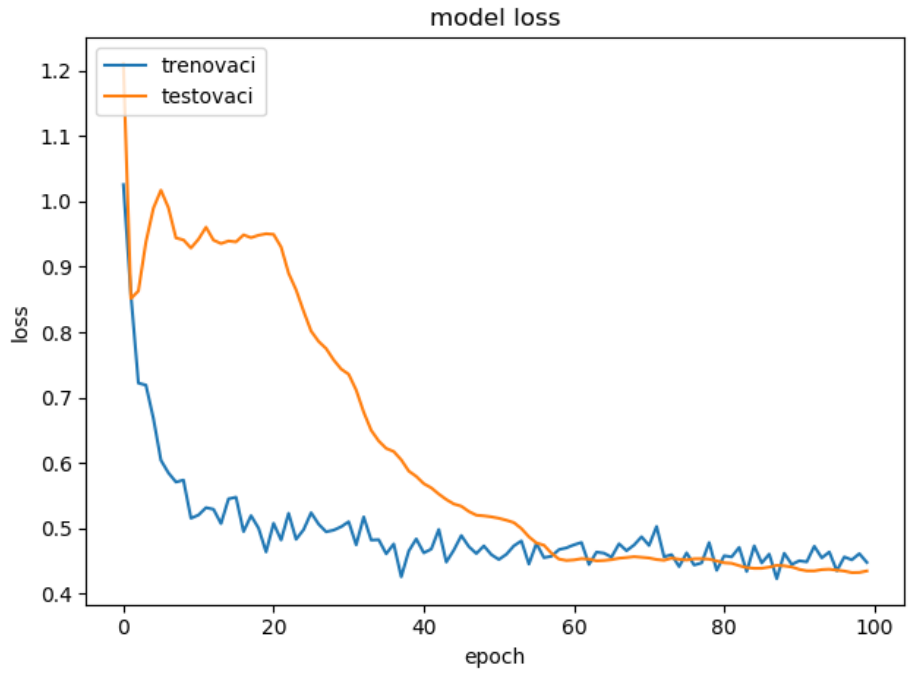


4.2. Vizualizace výkonnosti modelu



Python syntax pro vizualizaci přesnosti:

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('přesnost modelu')     # model accuracy
plt.ylabel('přesnost')           # accuracy
plt.xlabel('epoch')
plt.legend(['trenovaci', 'testovaci'], loc='upper left')
plt.show()
```



Python syntax pro vizualizaci loss:

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['trenovaci', 'testovaci'], loc='upper left')
plt.show()
```

5. DEPLOYMENT

5.1. Exportování modelu

Po vytrénování modelu je model potřeba vyexportovat aby se dal dále používat. V případě že se model vytrénuje a nevyexportuje se, všechna data o modelu budou smazána po dokončení programu.
Základní knihovny Pythonu nepodporují exportování NN a je třeba nainstalovat knihovny jako pickle nebo joblib. Po nainstalování potřebných knihoven lze model vyexportovat, je doporučovaný formát .pkl nebo alternativně .h5.

Python syntax pro exportování modelu do pickle formátu:

```
joblib.dump(model, 'diabetes_nn_model.pkl')
```

Python syntax pro exportování modelu do h5 formátu:

```
model.save("diabetes_nn_model.h5")
```

5.2. Importování modelu

Pokud chceme s model dále trénovat, nebo provést predikce na vlastních datech je třeba model importovat zpět do IDE (Integrated Development Environment).

Python syntax pro importování modelu:

```
model = joblib.load('diabetes_nn_model.pkl')
```

Python syntax pro predikci na vlastních datech:

```
prediction = model.predict(np.array([[0, 90, 60, 15, 140, 22.2, 0.451, 21]]))
```

Reference:

[1] Korkinof D., Heindl A., Rijken T., Harvey H., Glocker B. (2019), '*MammoGAN:High-Resolution Synthesis of Realistic Mammograms*', [online]: https://openreview.net/pdf?id=SJeichaN5E

[2] UCI Machine Learning (2016), '*Pima Indians Diabetes Database*', [online]: https://www.kaggle.com/uciml/pima-indians-diabetes-database