Osmý domácí úkol z Úvodu do AI - decision tree classifier

Vojtěch Šára

Nejprve vložím výchozí kód z ReCodExu a rozepíšu si parametry funkcí, které budu optimalizovat.

In [32]:
```
import pandas
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
import numpy as np

column_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
feature_columns = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age']
df = pandas.read_csv("diabetes.csv")
df.columns = column_names

X = df[feature_columns]
y = df.label

# Zajímají nás následující dva řádky. Zde si pouze rozepíšu volání s default hodnotami argumentů,
# abychom lépe viděli, co budeme měnit.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
dec_tree = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None,
random_state=None, min_impurity_decrease=0.0, class_weight=None)

dec_tree= dec_tree.fit(X_train, y_train)

y_pred = dec_tree.predict(X_test)
print("Accuracy:", metrics.accuracy_score(y_test, y_pred))
print("Precision:", metrics.precision_score(y_test, y_pred))
```

```
Accuracy: 0.6614583333333334
Precision: 0.44776119402985076
```

Půjdu postupně po všech parametrech a ke každému uvedu krátké zamyšlení a závěr. První je test_size ve funkci train_test_split.

test_size

Nastavení tohoto parametru je balanc dvou požadavků - přesnost natrénování a přesnost změření toho, jak dobře natrénování proběhlo. Zmenšováním tohoto parametru do extrému vede k viditelnému navýšení rozptylu měření výkonu klasifikátoru. S nastavením tohoto parametru na nízké hodnoty (< 0.07) jsem sice dosáhl nejlepších výsledků, ale očividně to byl šum měření s miniaturní testovací množinou, ve které jen náhodou skončily příhodné prvky. Cesta opačným směrem naopak zbytečně plýtvá daty, ze kterých by se klasifikátor mohl učit.

Nejdřív si vytvořím další parametr, number_of_samples, abych lépe změřil efektivitu daného nastavení

In [8]:
```
number_of_samples = 500
```

Následující funkce vygeneruje graf, ze kterého jsem čerpal pro rozhodnutí o parametru test_size. Mohu si dovolit takto jednodimenzionálně optimalizovat, protože by teoreticky tento hyperparametr měl být nezávislý na ostatních.

In [9]:
```
def optimise_test_size(t_size):
    avg = []
    for i in range(number_of_samples):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=t_size)
        dec_tree = DecisionTreeClassifier(criterion='entropy', max_depth=6,
        splitter='best', min_samples_split=25, min_samples_leaf=14)
        dec_tree= dec_tree.fit(X_train, y_train)

        y_pred = dec_tree.predict(X_test)
        avg.append(metrics.accuracy_score(y_test, y_pred))
```

```
    return sum(avg)/len(avg)

test_sizes = list(np.arange(0.01,0.5,0.01))
classifier_performance = list(map(optimise_test_size, test_sizes))

import matplotlib.pyplot as plt
plt.plot(test_sizes, classifier_performance)
plt.title('Výkon klasifikátoru v závislosti na test_size')
plt.show()
```

```
No handles with labels found to put in legend.
```

Jak jsem říkal, v malých hodnotách máme dobré výsledky, ale nejsou příliš průkazné, proto volím test_size = 0.08, abych měl nějakou jistotu, že výsledek má nějakou váhu.

Criterion

Máme na výběr z Entropy a Gini. Teoretický rozdíl těchto dvou způsobů měření "namíchanosti" dat je relativně malý - pro srovnání přikládám vzorečky:

Gini: $\mathrm{Gini}(E) = 1 - \sum_{j=1}^{c} p_j^2$

Entropy : $H(E) = -\sum_{j=1}^{c} p_j \log p_j$

Z toho, co jsem si o těchto dvou funkcích našel na internetu je rozdíl skutečně nepatrný, Gini by měl být trochu šetrnější na výpočet, jelikož neobsahuje logaritmus, ale obecně by rozdíl měl být v rámci dvou procent, což sedí i s experimentálními výsledky, ať už jsem ostatní parametry volil jakkoli, tak tato volba příliš důležitá nebyla, mikroskopicky lepší byl entropy, tak jsem zvolil ten.

Splitter

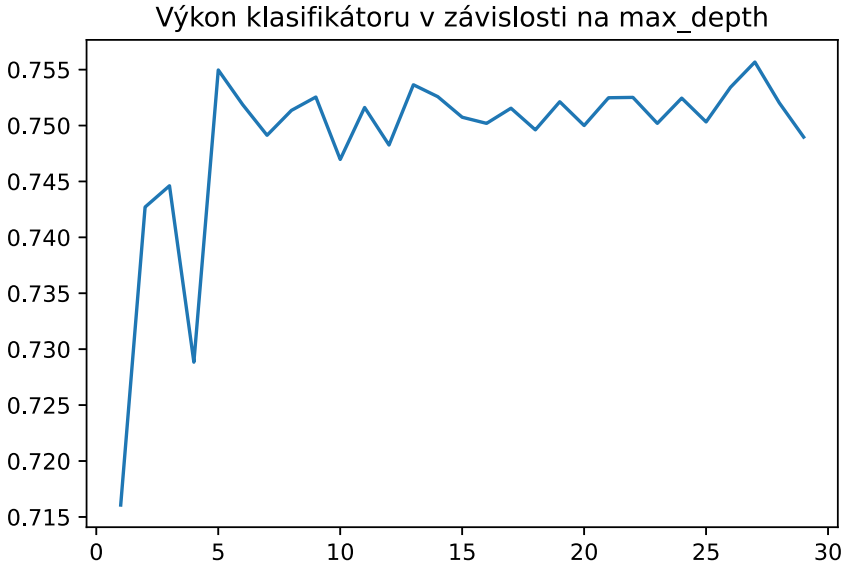
Zde je volba jasná - random je pouze výpočetně snadnější aproximace best, takže určitě použijeme best, jelikož jdeme za maximální přesností a zatím nehledíme na výpočetní čas. Je důležité podotknout, že situace není ve skutečnosti úplně tak černobílá, random bychom mohli využít k tomu, abychom zmírnili overfitting, protože náhodností, kterou do výpočtu přidává se přesně dá proti overfittingu bojovat, ale jelikož ke stejnému účelu máme i jiné parametry, tak ho nechám nastavený na best.

max_depth

Tento parametr se ukázal jako relativně důležitý - příliš hluboké stromy (by default není hloubka omezená) jsou náchylné na overfitting. Tj. větve se prodlužují a rozrůstají se do příliš specifických oblastí stavového prostoru a zhoršují schopnost generalizovat. Proto jsem tuto hodnotu nastavil nízkou - optimum z experimentů vyšlo 5. Následující graf je ale potřeba brát trochu s rezervou - max_depth hodně závisí i na dalších dvou parametrech.

In [12]:

```
def optimise_max_depth(max_depth):
    avg = []
    for i in range(number_of_samples):
        X_train, X_test, y_train, y_test = train_test_split(X, y,   test_size=0.08)
        dec_tree = DecisionTreeClassifier(criterion='entropy', max_depth=max_depth,
        splitter='best', min_samples_split=25, min_samples_leaf=14)
        dec_tree= dec_tree.fit(X_train, y_train)

        y_pred = dec_tree.predict(X_test)
        avg.append(metrics.accuracy_score(y_test, y_pred))
    return sum(avg)/len(avg)

max_depths = list(range(1,30))
classifier_performance = list(map(optimise_max_depth, max_depths))

import matplotlib.pyplot as plt
plt.plot(max_depths, classifier_performance)
plt.title('Výkon klasifikátoru v závislosti na max_depth')
plt.show()
```



min_samples_split + min_samples_leaf

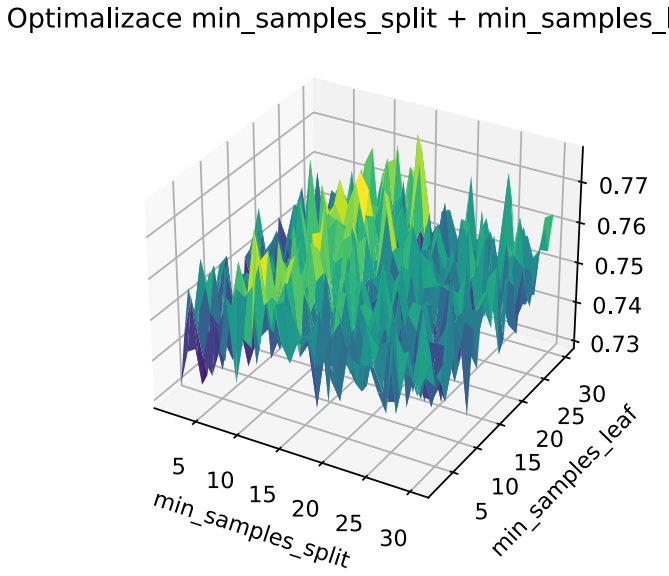
Tyto dvě hodnoty definují pravidla, kterými se musí řídit roustoucí strom. Tedy kolik samplů potřebujeme pro odůvodnění vzniku nové větve a kolik pro vznik nového listu. Ani po delším bádání jsem nepřišel na žádnou hypotézu, která by mi pomohla v nějakém inteligentním nastavení těchto parametrů, takže jsem se uchýlil k použití hrubé síly, ostatní hyperparametry jsem nějak inteligentně zafixoval a hledal jsem maximum této nové binární funkce, což vypadalo zhruba takto:

In [41]:

```
def optimise_max_depth(split, leaf):
    avg = []
    for i in range(number_of_samples//10):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.08)

        dec_tree = DecisionTreeClassifier(criterion='entropy', max_depth=5,
        splitter='best', min_samples_split=int(split), min_samples_leaf=int(leaf))
        dec_tree = dec_tree.fit(X_train, y_train)

        y_pred = dec_tree.predict(X_test)
        avg.append(metrics.accuracy_score(y_test, y_pred))
    return sum(avg)/len(avg)

graph_resolution = 30

x1 = np.linspace(2, 30, graph_resolution)
y1 = np.linspace(2, 30, graph_resolution)

X1, Y1 = np.meshgrid(x1, y1)
#Z1 = optimise_max_depth(X1,Y1)
```

```
Z1 = np.zeros((graph_resolution, graph_resolution))
for i in range(graph_resolution):
    for j in range(graph_resolution):
        Z1[i,j] = optimise_max_depth(x1[i],y1[j])

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X1, Y1, Z1, rstride=1, cstride=1,
                cmap='viridis', edgecolor='none')
ax.set_xlabel('min_samples_split')
ax.set_ylabel('min_samples_leaf')
ax.set_title('Optimalizace min_samples_split + min_samples_leaf')
```

Out[41]: Text(0.5, 0.92, 'Optimalizace min_samples_split + min_samples_leaf')



Dobré hodnoty vychází pro min_samples_split = 25 a min_samples_leaf = 14.

max_features

Tento parametr je v něčem podobný jako Splitter - default nastavení zkouší nejvíce možností, ostatní možnosti slouží k zjednodušení výpočtu - experimentálně jsem si ověřil, že žádná alternativní možnost výsledky nezlepšila.

random_state

Tento parametr jsem občas používal, když jsem si potřeboval zafixovat náhodnost, ale na výsledek tento parametr samozřejmě vliv nemá (respektive má, ale optimalizace jehož odpovídá cherry-pickingu náhodně vytvořených dobrých výsledků).

min_impurity_decrease

U tohoto parametru se mi nepodařilo zjistit, co z hlediska svobody optimalizace přináší nového oproti min_samples_split a min_samples_leaf a i kdybych to věděl, tak bych tuto informaci nebyl schopen přeložit do hypotézy o tom, jak parametr změnit, aby se výsledek zlepšil. Experimentálně jsem nenašel lepší nastavení této hodnoty než 0.0, ale jestli existuje nějaká kombinace, ve které nějak souzní s ostatními parametry a skýtá tak nějaké lepší globální minimum, nevím, ale je to určitě možné.

class_weight

Do tohoto parametru by se dal uložit nějaký přednostní odhad toho, jaký bude poměr diabetes / no diabetes v populaci. Po experimentování s informacemi z internetu o reálné hodnotě tohoto poměru jsem ale došel k tomu, že

vyhledané hodoty se nijak neliší od hodnoty, která je jakoby implicitně uložená v našich datech, což je informace, která do učení vstoupí a s nastavením class_weight na None si algoritmus sám najde tento poměr na základě vstupních dat, což mi také přijde optimální.

Výsledek

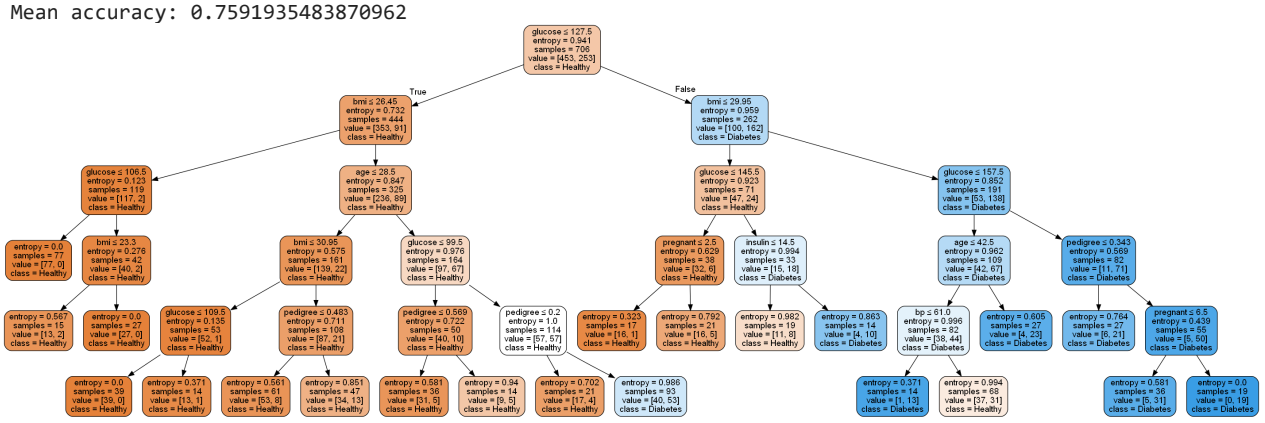
Tím bych uzavřel diskuzi o parametrech, zbývá tedy vykreslit nějaké řešení na základě vybraných parametrů:

In [42]:

```
from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus

avg = []
best_tree = None
for i in range(500):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.08)
    dec_tree = DecisionTreeClassifier(criterion='entropy', max_depth=5,
    splitter='best', min_samples_split=25, min_samples_leaf=14)
    dec_tree= dec_tree.fit(X_train, y_train)

    y_pred = dec_tree.predict(X_test)
    accuracy = metrics.accuracy_score(y_test, y_pred)
    if len(avg) > 0 and accuracy >= max(avg):
        best_tree = dec_tree
    avg.append(metrics.accuracy_score(y_test, y_pred))


print("Mean accuracy: " + str(sum(avg)/len(avg)))


dot_data = StringIO()
export_graphviz(best_tree, out_file=dot_data,
            filled=True, rounded=True,
            special_characters=True, feature_names =
            feature_columns,class_names=['Healthy','Diabetes'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('decision_tree_viz.png')
Image(graph.create_png())
```

```
Mean accuracy: 0.7591935483870962
```

Out[42]:



Přesnost mého řešení na základě vybraných parametrů se pohybuje mezi 75% a 76%.