**Adatbányászat beadandó**

**A feladat rövid összefoglalása**

Adathalmazunk a Free Music Archive (FMA)-ról származik. Különböző zeneszámok tulajdonságait foglalja magában 4 csv fájlon keresztül(tracks.csv, features.csv, genres.csv, echonest.csv). A fájlokat összeköti a *trackid* nevű attribútum.

Feladatunk a zeneszámok műfajának(*genre*) megjósolása volt az adatok alapján. Összesen 163 alműfaj található a fájlokban, amik 16 gyökérműfajba sorolhatók. Végső megoldásunkban mindkét besorolásra építettünk modelleket. Ehhez az alábbi adatbányászati módszereket hívtuk segítségül: Döntési fa, kNN, Random Forest, DBSCAN.

**Kezdeti nehézségek**

Több nehézségbe is ütköztünk a munkánk során, amikre eleinte egyáltalán nem gondoltunk. Ugyanis a legtöbb időt a megfelelő adathalmaz alkalmazható létrehozása, illetve betöltése vett el.

Először a tracks.csv fájllal kezdtünk el dolgozni, ami kezdetben jónak tűnt, mivel egy-egy szám albumáról, előadójáról és különböző tulajdonságairól is többféle adatot tartalmaz. Azonban nagyon sok benne a karakter típusú attribútum, aminek feldolgozásáról nem volt szó a félév során, időnk pedig nem volt megtanulni. Így ezt az utat elvetettük. (De eddig sok idő eltelt.)

A features.csv fájl viszont nagyon sok számadatot tartalmaz a zeneszámok frekvenciáiról, rezgéséről stb., így ezzel már tudtunk dolgozni. A fájl nagy mérete miatt úgy döntöttünk, hogy a könnyebb kezelhetőség miatt egy rövid python kóddal kb. leharmadoljuk az adatokat, így kaptuk meg az adatb.csv-t:

f = open(**r"c:\users\dell\Desktop\a\features.csv"**, **"r"**)

outf = open(**r"c:\users\dell\Desktop\a\adatb.csv"**, **"w+"**)

**import** random

*#az első sort beolvassuk, mivel az attribútum nevekre szükségünk lesz*

first\_line = f.readline

*#ki is írjuk, a "trackid"-t hozzávéve, mivel az a második sorban szerepelt*

outf.write(**"trackid;"**+ first\_line)

f.readline()

*#végigmegyünk az adatokon, és ha a random generált (0,3) közé eső számunk 1, akkor kiírjuk az adott sort*

**for** line **in** f:

**if** random.randint(0, 3) == 1:

outf.write(line.strip() + **"\n"**)

outf.close()

Mivel ez a fájl nem tartalmazott *genre* oszlopot, így nekünk kellett azt létrehozni, ügyelve, hogy minden *trackid*-hoz jól párosítsuk a műfajokat. A tracks.csv-ben szerepeltek a műfajok, így ezt segítségül tudtuk hívni. Egy „szótárt” készítettünk, melyben a *trackid-genre* párokat létrehoztuk (a műfajnak a kódját megadva), majd ezután FKERES függvénnyel hozzárendeltük az adatb.csv-ben minden sorhoz a megfelelő műfajt.

Miután a szótárt elkészítettük, észrevettük, hogy sok sor betűket, furcsa karaktereket tartalmaz. Valószínűleg a baj abból adódott, hogy a csv fájl beolvasásakor tördeléskénk pontosvesszőt használtunk, ami olyan helyeken is szerepelt, ahol nem szerettünk volna tördelni. Így sok, eredetileg egy oszlopba való karaktersorozatot is eltördeltünk külön oszlopokba. Végül a műfaj hozzárendelés után ezt úgy oldottuk meg, hogy az Excel szűrőjét használva ki tudtuk szelektálni a „rossz” sorokat, amikből összesen csak kb. 800 volt. Így ezeket mind töröltük, maradt 25400 adatsorunk.

Volt egy genres.csv fájlunk is, amiben minden *genreid*-hoz meg volt adva a gyökérműfaja(*genre\_top*). Mi ezeket is hozzárendeltük az adatb.csv adathalmazunkhoz, így kétféle célváltozóra tudtunk „jósolni”. A továbbiakban jelöljük ezeket Y, illetve Z esetként (a kódjainkban is ezeket a betűket használtuk):

* Y eset: amikor a gyökérműfaj(*genre\_top*) a célváltozó; ez 16 műfaj.
* Z eset: amikor az összes lehetséges műfaj(*genre*) a célváltozó; ez 163 műfaj.

**Adathalmaz betöltése**

Először a Google Drive-ba töltöttük fel az adathalmazt, így az onnan már bármikor gyorsan elérhetővé vált.

import pandas as pd

import numpy as np

from google.colab import drive

drive.mount('/content/drive/')

Beolvastuk az adathalmazt adf néven, ellenőrzésként kiírattuk az első 10 sorát:

import io

adf = pd.read\_csv('/content/drive/My Drive/adatb.csv', error\_bad\_lines = False, encoding='iso-8859-1', delimiter=";")

adf.head(10)

**Tanító-, és teszthalmaz**

A következőkben felbontottuk az adatainkat tanító-, illetve teszthalmazra. Ez a felosztás az összes modellünkre érvényes lesz.

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

y = adf.genre\_top *#Y eset célváltozója*

z = adf.genre *#Z eset célváltozója*

x = adf.drop(['genre\_top','genre','zcr.6'], axis=1) *#Magyarázóváltozó*

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=2)

x\_train, x\_test, z\_train, z\_test = train\_test\_split(x, z, test\_size=0.3, random\_state=2)

np.asarray(x)

np.asarray(y)

np.asarray(z)

***Megjegyzések:***

* Hibajelzés után vettük észre, hogy a *zcr.6* oszlopunk hibás adatokat tartalmaz, így azt kidobtuk a magyarázóváltozókból.
* Y és Z esetnél is a teszthalmazunk 30%, így a tanulóhalmaz 70%.

**Döntési fa**

Elsőként döntési fát illesztettünk az adathalmazunkra a következő módon:

from sklearn import tree

model = tree.DecisionTreeClassifier()

model = model.fit(x\_train, y\_train)

y\_test\_model = model.predict(x\_test)

print(accuracy\_score(y\_test, y\_test\_model))

model = model.fit(x\_train, z\_train)

z\_test\_model = model.predict(x\_test)

print(accuracy\_score(z\_test, z\_test\_model))

***Eredmények:***

Lefuttatva a kódot, Y esetre 30%-ot, Z esetre 15%-ot kaptunk. Tehát a modell 30%-ban mondja meg jól, hogy milyen gyökérműfajú a zeneszám és 15%-ban mondja meg a pontos műfaját. Azt vártuk mi is, hogy az Y esetre jóval nagyobb %-ot kapunk, viszont összességében jobb eredményre számítottunk. Valószínűleg 16, illetve 163 db műfajba való helyes (vagy legalább 95%-os) besoroláshoz lényegesen nagyobb adathalmaz szükséges.

***Mivel tudunk még javítani esetleg?***

Először a *min\_samples\_leaf* tulajdonság értékének megadásával próbálkoztunk.

Ez a tulajdonság a levélcsomópontoknák szükséges minimális számú mintát adja meg.(Tehát csak akkor veszünk figyelembe egy pontot, ha az ágaiban minimum *min\_samples\_leaf* számú minta van.)

Megnéztük a következő értékekre: 20, 50, 100, 150, 170. Az ezekre kapott pontosságokból úgy tűnt, hogy 100 és 150 között lesznek valahol a legjobb eredmények, így készítettünk egy maximum kereső algoritmust, hogy megtudjuk melyik *min\_samples\_leaf*-nél kapjuk a legjobb eredményeket:

accy = 0

accz = 0

yi = 0

zi = 0

for i in range (100, 200, 10):

model = tree.DecisionTreeClassifier(min\_samples\_leaf=i)

model = model.fit(x\_train, y\_train)

y\_test\_model = model.predict(x\_test)

if accuracy\_score(y\_test, y\_test\_model) > accy:

accy = accuracy\_score(y\_test, y\_test\_model)

yi = i

model = model.fit(x\_train, z\_train)

z\_test\_model = model.predict(x\_test)

if accuracy\_score(z\_test, z\_test\_model) > accz:

accz =accuracy\_score(z\_test, z\_test\_model)

zi = i

print(accy)

print(accz)

print(yi)

print(zi)

0.38438320209973753

0.23083989501312335

130

100

Tehát Y esetnél 38,44%-osan mondja meg jól a műfaját a számnak, 130-as minimális mintánál. A Z esetben 100 mintánál és 23,1%-os a pontosság.

További javítások érdekében a max\_depth tulajdonságot is megadtuk többféle értéket kipróbálva, de ez nem hozott pozitív változást.

Ellenőrzés

A tanulóhalmazra tesztelve ellenőriztük, hogy jól működik- e a modell, és ahogy vártuk, 1-et kaptunk mindkét esetben.

model = tree.DecisionTreeClassifier()

model = model.fit(x\_train, y\_train)

y\_train\_model = model.predict(x\_train)

print(accuracy\_score(y\_train, y\_train\_model))

model = model.fit(x\_train, z\_train)

z\_train\_model = model.predict(x\_train)

print(accuracy\_score(z\_train, z\_train\_model))

1.0

1.0

**kNN**

Következő modellünknek a kNN-t választottuk.

kNN lényege: vesszük a teszt vektor k legközelebbi szomszédját a tanító halmazból. Amelyik osztálycímke legtöbbször fordul elő, azt rendeljük a tesztadathoz.

Kirajzoltattuk mindkét esetünkre, hogy a szomszédok számának változtatásával hogy változik a modell pontossága. (weights='distance' paranccsal a távolságot is figyelembe vettük)

from sklearn.neighbors import KNeighborsClassifier

import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsClassifier

accs\_y = []

accs\_z = []

n\_y = []

n\_z = []

for i in range(1,100):

knn\_model = KNeighborsClassifier(n\_neighbors=i, weights='distance')

knn\_model.fit(x\_train, y\_train)

y\_test\_model = knn\_model.predict(x\_test)

accs\_y.append(accuracy\_score(y\_test, y\_test\_model))

n\_y.append(i)

knn\_model.fit(x\_train, z\_train)

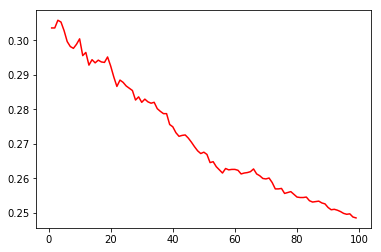
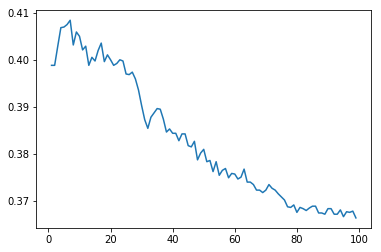
z\_test\_model = knn\_model.predict(x\_test)

accs\_z.append(accuracy\_score(z\_test, z\_test\_model))

n\_z.append(i)

plt.plot(n\_y,accs\_y)

plt.plot(n\_z,accs\_z)



Látszik, hogy mindkét esetben a szomszédok növelésével csökken a teljesítmény. Y esetben kb 20 szomszédig a kNN jobban teljesít,mint a döntési fa, 40-41%-os pontosságot adva. A Z esetben pedig még 90-es szomszédszámnál is jobb a teljesítés, 25-30%-osra emelkedett a korábbi 23%-hoz képest.

**Random Forest**

Random Forestet futtattunk először 50 fával(*n\_estimators = 50*), ettől a modelltől vártuk a legjobb eredményt.

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n\_estimators=50, random\_state=0)

rfc.fit(x\_train, y\_train)

y\_test\_model = rfc.predict(x\_test)

print(accuracy\_score(y\_test, y\_test\_model))

rfc.fit(x\_train, z\_train)

z\_test\_model = rfc.predict(x\_test)

print(accuracy\_score(z\_test, z\_test\_model))

A pontosság növekedett is, Y esetben közel 46%-ra, míg Z-nél 30%-ra.

*n\_estimator* 150-es értékénél már Y 47%-os, Z pedig 31%.

**DBSCAN**

Végül a DBSCAN-t futtatunk az adathalmazunkra. Az eredményeket a

from sklearn import metrics

print("Silhouette Coefficient: %0.3f" % metrics.silhouette\_score(x,labels))

segítségével ellenőriztük, ami 0.022-t adott. Ez nagyon közel van 0-hoz, ami nagy átfedést jelent a klaszterek közt, emiatt ez nem egy jól használható módszer.

**A beadandóval kapcsolatos gondolatok**

*Pozitív élmények:*

Többször találkoztunk hárman (kétszer hétvégén is), hogy közösen dolgozzunk a beadandó feladaton, mindenki lásson minden lépést. Ezek nagyon jó hangulatban teltek, együtt próbálkoztunk és örültünk, ha valami (végre) sikerült.

*Negatív élmények:*

Mire egy használható adathalmazt kaptunk, nagyon sok idő elment. Így a modellek próbálgatására, „kísérletezésre”, vizuális ábrázolására már nagyon kevés maradt, így az elképzeléseinknél egyszerűbb beadandót sikerült csak készítenünk.