Driving Behaviour

Denisa Andrei

https://www.kaggle.com/competitions/driving-behaviour/overview

Despre dataset

Files

- **train_motion_data.csv** - the training set
- **Test.csv** - the test set
- **Sample.csv** - a sample submission file in the correct format

Columns

- `AccX` \
- `AccY` | - X, Y, Z acceleration axis
- `AccZ` /
- `GyroX` \
- `GyroY` | - X, Y, Z orientation axis
- `GyroZ` /
- `Class` - classification label
- `Timestamp` - time in seconds

Primele mele observatii

- In fisierul de test se afla datele pe care trebuie sa testam prezicerile modelului care a fost antrenat pe fisierul de training, dar in fisierul de training, pe langa coloanala care trebuie prezisa(Class) avem si o coloanal de Timestamp care nu se regaseste in fisierul de test => o eliminam din hiperparametrii. Ceilalti parametrii au sens asa ca i-am pastrat in continuare pe toti 6.
- Se observa ca este o problema clasica de clasificare, precum am facut si la laborator.

Implementare in Python

- Am folosit Google Colab pentru a implementa codul care rezolva aceasta problema de clasificare.
- Am impartit codul meu in 4 sevente de cod:
  - Importurile
  - Citirea datelor din fisierul de train si test
  - Antrenarea modelelor si precizerea datelor
  - Formatarea prezicerilor oferite de modele pentru a corespunde outputului solicitat.

Secventa de cod 1: Importuri

Am ales sa folosesc algoritmii de clasificare despre care am studiat la laborator: DecisionTree si Naive Bayes.

```
[1]  import numpy as np
     import pandas as pd
     from sklearn.naive_bayes import GaussianNB
     from sklearn.tree import DecisionTreeClassifier
```

Arborele de decizie este un model discriminativ, în timp ce Naive Bayes este un model generativ. Arborii de decizie sunt mai flexibili si mai usori. Tăierea arborelui de decizie poate neglija unele valori cheie în datele de antrenament, ceea ce poate duce la o precizie inferioara.

Secventa de cod 2: Citirea datelor de train si test

```
[3]  train = pd.read_csv("train_motion.csv")
     used_columns = ['AccX', 'AccY', 'AccZ', 'GyroX', 'GyroY', 'GyroZ']
     x_train = train[used_columns].dropna()
     y_train = train['Class']
     test = pd.read_csv("Test.csv")
     x_test = test[used_columns].dropna()
```

Pe prima linie de cod am citit datele din fisierul csv care contine datele pentru antrenarea modelului. Pe linia a 2-a am selectat doar parametrii relevanti pentru inregistrarile din fisierul de test, cum am mentionat anterior. Pe linia 3, din toate valorile din fisierul de train, am pus in “x_train” doar coloanele specificate in “used_columns”, iar pe linia 4, am pus in “y_train” doar coloanal care trebuie prezisa. Ne putem gandi la o functie de genul F(X) = Y. Pe linia 5 am citit toate datele de test si pe linia 6 am pus in “x_test” doar datele care se regasesc in “used_columns.”. De mentionat este ca a trebuit sa fac un dropna() la train/test[used_columns] deoarece aveam o eroare din cauza ultimei linii din fisier.

Secventa de cod 3: antrenarea modelelor si prezicerea datelor

Aici am reusit performanta de a antrena un model si a prezice datele intr-o singura linie de cod!!!

```
[6]  dec_tree_pred = DecisionTreeClassifier().fit(x_train, y_train).predict(x_test)
     naive_bayes_pred = GaussianNB().fit(x_train, y_train).predict(x_test)
```

Am apelat constructorul pentru fiecare clasificator, apoi l-am antrenat folosind metoda “fit”, care primeste ca parametrii x_train si y_train. Adica noi ii dam ca parametrii exemplu de genul: pentru aceste valori => clasificarea aceasta. El pe baza lor va invata sa faca astfel de clasificari si pentru date noi care nu sunt surprinse in fisierele de antrenament. In final, apelam metoda “predict” care primeste ca parametru x_test, adica noile valori pentru care trebuie sa precizem din care clasa fac parte. Acesti pasi se repeata atat pentru Decision tree cat si pt GaussianNB, datorita bibliotecilor foarte bune din Python care ofera metode generice de abordare a diversor algoritmi, fara sa fie nevoie de metode diferite pentru fiecare algoritm in parte.

Secventa de cod 4: Formatarea outputului

Ultimul pas din cod este formatarea outputului ca sa fie valid cu cerintele competitiei. Pasii sunt generic pentru cei 2 algoritmi: formam fisierul csv astfel: Initial ii atribuim coloanal cu numele “Class” cu valoarile prezise anterior, dupa care formam o noya coloaa de index care va fi reprezentat de ID-ul acestor inregistrari. Deoarece ID-urile ar incele de la 1, am apelat “Predictions.index += 1” pentru a incrementa cu o unitate toate ID-urile. Ultimul pas este de a exporta in format CSV prezicerile si sa le submitem pe site-ul competitiei.

```
[8]  Predictions= pd.DataFrame(dec_tree_pred, columns=['Class'])
     Predictions.index.name='ID'
     Predictions.index += 1
     Final_predictions= Predictions.to_csv(
         'Predictions1.csv', header= 'Class', index='true')

     Predictions= pd.DataFrame(naive_bayes_pred, columns=['Class'])
     Final_predictions= Predictions.to_csv(
         'Predictions2.csv', header= 'Class', index='true')
```

Concluzii

- Ambii algoritmi au necesitat un timp neobservabil de antrenament pentru aceste date de train;
- Naïve Bayes a avut performante mult mai bune decat Decision Tree din punct de vedere al acuratetii
- Totusi, niciun algoritm nu a reusit sa atinga pragul de 50% preziceri corecte.
- Ma bucur ca am participat la aceasta competitie, pentru ca m-a ajutat sa vad putin din puterea limbajului de programare Python si putin din acest domeniu de invatare automata care este intr-o continua expansiune.

Referinte

- https://www.w3schools.com/python/python_ml_decision_tree.asp
- https://www.analyticsvidhya.com/blog/2021/11/implementation-of-gaussian-naive-bayes-in-python-sklearn/
- Laboratoarele 2 si 3