Evaluarea algoritmilor/modelelor de învăţare automată - implementare Python

Cea mai bună modalitate de a evalua performanța unui algoritm de învăţare automată este să facem predicţii pentru date care nu le-am folosit în procesul de antrenare a modelului, dar a căror etichetă-output îl cunoaștem.

Se pune întrebarea: de ce nu putem să antrenăm algoritmul de învățare automată pe un set de date și să utilizăm acest set de date pentru a evalua algoritmii? Răspunsul simplu este: supra-estimarea (en. overfitting).

Să ne imaginăm un algoritm care își amintește fiecare observație care îi este prezentată. Dacă utilizăm algoritmul pe același set de date folosit pentru a antrena algoritmul, atunci un algoritm ca acesta ar avea un scor perfect pe setul de date de antrenament. Dar previziunile făcute pe date noi, care nu apare în setul de antrenament ar fi foarte proaste. În concluzie, trebuie să evaluăm algoritmii de învățare automată pe date care nu sunt folosite pentru a a antrena modelul.

Evaluarea este o estimare pe care o putem folosi pentru a ne da seama cât de bine va merge algoritmul în practică. Nu este o garanție a performanței.

Odată ce estimăm performanța algoritmului, putem re-antrena algoritmul pe întregul set de date și îl putem pregăti pentru utilizare operațională.

În continuare vom examina patru tehnici diferite pe care le putem folosi pentru a împărți setul de date de antrenare și pentru a face unele estimări privind performanța algoritmilor:

1. Seturi diferite de date pentru antrenare și de testare.
2. Validarea încrucișată (en. k-fold cross-validation).
3. Validarea încrucişată cu k egal cu 1 (en. leave-one-out cross-validation).
4. Repetarea aleatorie cu seturi diferite de antrenare şi testare.

1. Seturi diferite de date pentru antrenare şi de testare.

Cea mai simplă metodă pe care o putem folosi pentru a evalua performanța unui algoritm de învățare automată este de a folosi seturi de date diferite seturi pentru antrenare și testare. Se poate lua setul original de date şi împărți în două părți. Algoritmul este antrenat pe prima parte a setului de date, se fac predicţii pe a doua parte și se evaluează aceste predicții în raport cu output-urile reale.

Modul de împărțire poate depinde de mărimea și specificul setului de date, dar ca și o regulă standard se utilizeze 67% din datele pentru antrenament și restul de 33% pentru testare.

Această tehnică de evaluare a algoritmilor este foarte rapidă. Este ideală pentru seturi mari de date (milioane de înregistrări) în cazul cărora ambele seturi de date de antrenare şi testare sunt reprezentative. Această abordare este utilă atunci când algoritmul pe care îl investigați este lent la antrenare.

Un dezavantaj al acestei tehnici este faptul că poate avea o variație mare. Acest lucru înseamnă că diferențele dintre seturile de date de antrenare și testare pot avea ca rezultat diferențe semnificative în estimarea performanţei algoritmului.

În exemplul de mai jos am împărțit setul de date Pima Indians (https://www.kaggle.com/uciml/pima-indians-diabetes-database) în 67% / 33% pentru antrenare şi testare și evaluarea corectitudinii unui model de regresie logistică.

```
import pandas
from sklearn import model_selection
from sklearn.neighbors import KNeighborsClassifier
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg','plas','pres','skin','test','mass','pedi','age','class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y,
test_size=test_size, random_state=seed)
model = KNeighborsClassifier()
model.fit(X_train, Y_train)
result = model.score(X_test, Y_test)
print("Accuracy: %.3f%%" % (result*100.0))
```

Precizia estimată a modelului este de aproximativ 75%. Se poate observa din codul de mai sus că, în afară de specificarea dimensiunii divizării setului de date, specificăm și seed-ul pentru împărţrea aleatoare. Deoarece împărţirea este aleatoare, vrem să ne asigurăm că rezultatele sunt reproductibile. Prin specificarea seed-ului aleator, ne asigurăm că facem aceeaşi împărţire de fiecare dată când executăm codul.

Acest lucru este important dacă vrem să comparăm acest rezultat cu precizia estimată a unui alt algoritm de învățare automata sau cu același algoritm dar cu o configurație diferită. Pentru a avea sens comparaţia între diferiţi algoritmi, trebuie să ne asigurăm că algoritmii sunt antrenați și testați pe aceleași date.

2. Validarea încrucişată (en k-fold cross-validation)

Validarea încrucișată este o metodă care se poate folosi pentru a estima performanța unui algoritm de învățare automată cu o variație mai mică decât folosind seturi disjuncte de antrenare şi testare.

Acestă metodă funcționează prin împărțirea setului de date în k-părți (de ex. k = 5 sau k = 10).. Algoritmul este antrenat pe k-1 părţi şi testat pe o parte dintre cele k. Acest lucru este repetat, astfel încât fiecare parte a setului de date va fi folosită o data pentru testare.

După rularea validării încrucișate, se va calcula media şi deviaţia standard a scorurile de performanță obţinute.

Rezultatul este o estimare mai precisă a performanței algoritmului, deoarece algoritmul este instruit și evaluat de mai multe ori pe date diferite.

Alegerea numărului k trebuie să permită ca dimensiunea fiecărei partiții de testare să fie suficient de mare pentru a fi un eșantion reprezentativ al problemei, permițând în același timp o repetare suficientă a evaluării algoritmului pentru a oferi o estimare corectă a performanței algoritmului pe datele nevăzute. Pentru seturile de date cu dimensiuni mici (adică mii de mii de înregistrări), se considerând de obicei pentru k valorile 3, 5 și 10.

În exemplul de mai jos, folosim validarea încrucișată de 10 ori.

```
from sklearn.linear_model import LogisticRegression
num_instances = len(X)
seed = 7
kfold = model_selection.KFold(n_splits=10, random_state=seed)
model = LogisticRegression()
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0,
results.std()*100.0))
```

3. Validarea încrucişată cu k egal cu 1 (en. leave-one-out cross-validation)

```
loocv = model_selection.LeaveOneOut()
model = LogisticRegression()
results = model_selection.cross_val_score(model, X, Y, cv=loocv)
print("Acc: %.3f%% (%.3f%%)" %(results.mean()*100.0, results.std()*100.0))
```

4. Repetarea aleatorie cu seturi diferite de antrenare şi testare.

O altă variantă a validării încrucișate este de a crea o împărțire aleatorie a datelor prin metoda de împărţire a setului de date în antrenare şi testare descrisă mai sus, dar repetarea procesului de divizare și evaluare a algoritmului de mai multe ori, cum ar fi validarea încrucișată.

Această variantă are viteza metodei de validare folosind seturi disjuncte pentru antrenament şi testare și reduce varianța performanţei estimate la fel ca validarea încrucișată. Procesul se poate repeta de câte ori este nevoie. Un dezavantaj al acestei metode este faptul că repetările pot include de mai multe ori aceleaşi date în setul de antrenament şi testare, introducând redundanță în evaluare.

Exemplul de mai jos împarte datele într-o divizare de 67% / 33% pentru antrenament şi testare și repetă procesul de 10 ori.

```
kfold = model_selection.ShuffleSplit(n_splits=10, test_size=test_size,
random_state=seed)
model = LogisticRegression()
results = model_selection.cross_val_score(model, X, Y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0,
results.std()*100.0))
```