ML.NET, o alternativă la framework-urile populare din Python

Iordăchescu Anca-Mihaela, grupa 407

Este binecunoscut faptul că domeniul Inteligența Artificială (IA) și, cu precădere, Învățarea Automată sunt foarte costisitoare din punct de vedere al puterii de procesare, astfel că principalele blocuri de cod rămân cât mai apropiate de "low level", acestea fiind scrise în general în C sau C++. În limbajul Python, care este fără îndoială limbajul de neînlocuit pentru Inteligență Artificială, sunt create diverse API-uri către aceste blocuri de cod pentru a putea fi preluate și utilizate în scopul creării sistemelor de lucru, astfel cu ajutorul acestora, Python câștigându-și reputația de cel mai popular limbaj printre programatorii din acest domeniu.

În momentul de față, Python reprezintă limbajul fundamental pentru implementarea sistemelor bazate pe Învățare Automată (Machine Learning). Cunoscut prin simplitatea sintaxei sale, Python este un limbaj accesibil, fiind ușor de învățat și utilizat în domeniul Inteligenței Artificiale prin multitudinea de librării de cod care au fost create special pentru acest domeniu. Acest limbaj oferă un ecosistem de lucru eficient, pentru a implementa fluxuri de lucruri integrale, și flexibil, programatorii putând să-și aleagă paradigma / stilul de lucru cel mai confortabil lor (imperativ, procedural, funcțional, orientat pe obiecte). De asemenea, acest limbaj dispune de diverse framework-uri de analiză de date (Pandas), vizualizare (Matplotlib), preprocesare (Sklearn) și modelare a soluțiilor (TensorFlow, Keras, PyTorch) care îmbunătățesc și simplifică considerabil fluxul de lucru față de un alt limbaj. Totuși, când vine vorba despre punerea în producție a unui serviciu, un număr foarte mic de proiecte de ML ajung în producție deoarece un proiect de IA poate eșua din diverse motive în timpul unui deployment. Un important motiv este implementarea unei aplicații care folosește mai multe limbaje de programare ale căror elemente nu interacționează bine unele cu altele. De exemplu, dacă vorbim despre un proiect complex realizat in ASP.NET/Java, infrastructura acestuia poate fi implementată printr-un microserviciu in Python, însă acest lucru implică foarte mult efort din cauza problemelor care presupun portabilitate, scalabilitate, putere de calcul etc.

În același timp, lumea se află într-o continuă căutare de alternative, Microsoft propunând un nou framework, ML.NET pentru limbajele de programare C# și F#, care poate fi folosit pentru dezvoltarea propriilor modele de Machine Learning. [1] Această abordare e diferită de cea prin care programatorii își dezvoltă modelele folosind servicii de Inteligență Artificială predefinite, precum sunt cele oferite de Azure Cognitive Services. Acest framework este open-source și cross-platform, putând fi folosit în orice aplicație .NET. De asemenea, modelele dezvoltate cu ML.NET pot fi utilizate și on-premises sau în orice cloud, cât și în scenarii offline. În general, acesta poate fi folosit pentru aproape orice task de Machine Learning, precum clasificare, regresie și prezicerea variabilelor continue, detecție de anomalii, recomandări sau clasificări de imagini.

Un prim avantaj pe care îl aduce ML.NET e reprezentat de extensibilitatea sa. [2] Majoritatea librăriilor și framework-urilor folosite în Machine Learning și Inteligență Artificială sunt scrise în Python. ML.NET reprezintă un ajutor substanțial pentru programatorii care nu cunosc acest acest limbaj sau care nu îl pot utiliza în aplicațiile pe care le dezvoltă, deoarece framework-ul oferă tool-uri care îi ajută pe aceștia să construiască ușor, să antreneze și să dea deploy la modele de Machine Learning fără a fi nevoie de experiență în domeniul respectiv. De asemenea, ML.NET suportă și utilizarea de modele deja scrise în alte limbaje sau cu alte tehnologii. De exemplu, se poate scrie un model folosind Azure Custom Vision, PyTorch, Scikit Learn, după care acesta să fie exportat sau convertit folosind ONNX, iar apoi acesta să fie utilizat folosind ML.NET. Totodată, framework-ul pune la dispoziția programatorilor care au experiență cu Python un modul care face binding cu limbajul menționat anterior, și anume NimbusML, pentru ca aceștia să poată profita de funcționalitățile și performanțele ML.NET.

Cel mai mare avantaj pe care îl aduce însă ML.NET e faptul că face domeniul de Machine Learning accesibil pentru oricine, indiferent de nivelul de cunoștințe în domeniu. [3] În primul rând, se poate utiliza Model Builder, o extensie a Visual Studio, care oferă o interfață grafică ce ghidează programatorul, pas cu pas, prin etapele ce trebuie desfășurate pentru crearea și antrenarea unui model de Machine Learning. Pentru persoanele care preferă să lucreze din linia de comandă se poate folosi ML.NET CLI, o abordare prin care se poate atât antrena modelul, cât și genera codul care definește modelul. ML.NET CLI reprezintă o alternativă foarte bună în cazul scenariilor de tip MLOPs (Machine Learning Operațional),

unde antrenarea și dezvoltarea modelului sunt parte a unui pipeline care realizează integrare continuă (CI) sau livrare continuă (CD).

Pentru cei care doresc mai mult control și preferă să scrie codul aferent sau au o gamă mai largă de cunoștințe cu privire la domeniu, există alte alternative. În primul, pot folosi AutoML API care ajută programatorul să identifice cel mai bun model. În mod normal, dezvoltarea celui mai bun model depinde de mai mulți factori, precum cantitatea și modul de distribuire al datelor, precum și de timpul de antrenare. Acest API ține cont de aceste impedimente și testează mai mulți algoritmi, folosind hiperparametrizare, pentru a genera la final cel mai bun model.

Pentru persoanele care doresc control total asupra programului, se poate utiliza ML.NET API, care furnizează acces la componentele care realizează operațiunile de încărcare și transformare a datelor, de antrenare a modelelor și de prezicere, ele putând fi configurate în conformitate cu cerințele problemei.

```
// 1. Initalize ML.NET environment
MLContext mlContext = new MLContext();

// 2. Load training data
IDataView trainData = mlContext.Data.LoadFromTextFile<ModelInput>("taxi-fare-train.csv", separatorChar:',');

// 3. Add data transformations
var dataProcessPipeline = mlContext.Transforms.Categorical.OneHotEncoding(
    outputColumnName:"PaymentTypeEncoded", "PaymentType")
    .Append(mlContext.Transforms.Concatenate(outputColumnName:"Features",
    "PaymentTypeEncoded","PassengerCount","TripTime","TripDistance"));

// 4. Add algorithm
var trainer = mlContext.Regression.Trainers.Sdca(labelColumnName: "FareAmount", featureColumnName: "Features");
var trainer2 = mlContext.Regression.Trainers.FastTree();

var trainingPipeline = dataProcessPipeline.Append(trainer);
var trainingPipeline2 = dataProcessPipeline.Append(trainer2);

// 5. Train model
var model = trainingPipeline.Fit(trainData);
var model2 = trainingPipeline.Fit(trainData2);

// 6. Evaluate model on test data
IDataView testData = mlContext.Data.LoadFromTextFile<ModelInput>("taxi-fare-test.csv");
IDataView predictions = model.Transform(testData);
var metrics = mlContext.Regression.Evaluate(predictions,"FareAmount");

IDataView predictions2 = model2.Transform(testData);
var metrics2 = mlContext.Regression.Evaluate(predictions2,"FareAmount");

// 7. Predict on sample data and print results
var input = new ModelInput
{
    PassengerCount = 1,
    TripTime = 1150,
    TripDistance = 4,
    PaymentType = "CRD"
};

var result = mlContext.Model.CreatePredictionEngine<ModelInput,ModelOutput>(model).Predict(input);

Console.WriteLine($"Predicted fare: {result.FareAmount}\nModel Quality (RSquared): {metrics.RSquared}");
```

În imaginea atașată se poate observa codul prin care am simulat etapele de creare, antrenare, testare și prezicere a unui model de Machine Learning, scris în C#, folosind ML.NET API. Se poate observa că există o schemă standard după care s-a lucrat. Mai întâi am încărcat datele din fișierul .csv folosind IDataView, obiectele de acest tip putând încorpora tipuri de date multiple (numere, texte, vectori). De asemenea, datele pot fi preluate din orice tip de sursă de date. Se poate folosi File Loader pentru fișiere binare sau imagini, sau Database Loader pentru preluarea datelor din orice tip de bază de date relațională. După aceea am făcut preprocesarea datelor folosind One Hot Encoding, una dintre metodele de transformare puse la dispoziție de ML.NET. Apoi am folosit regresie pentru a îmi antrena modelul. ML.NET pune la dispoziție aproximativ 40 de algoritmi de antrenare care rezolvă diferite task-uri precum clasificare, regresie, clustering sau time series. Recent, s-au dezvoltat și algoritmi de antrenare pentru task-uri de Computer Vision, precum clasificarea imaginilor sau recunoaștere. Iar, în final am evaluat și testat modelul creat pe setul de date de test pentru a îi observa performanțele și a îl compara cu alte modele încercate.

Un alt aspect care avantajează ML.NET este viteza de procesare a seturilor de date. Conform unui studiu realizat în 2019, *Machine Learning at Microsoft with ML.NET* [4], a reieșit faptul ca ML.NET are un timp de procesare mai bun decât al altor framework-uri, obținând totodată valori de acuratețe foarte bune.

Experimentul constă în a compara ML.NET, folosind 2 metode diferite, una cu API și alta Nimbus, cu Sklearn și H2O, pentru a verifica performanțele acestora pe mai multe seturi de date. S-au aplicat aceleași metode de transformare a datelor și s-au utilizat aceiași algoritmi de antrenare, astfel încât să se realizeze comparația numai la nivelul framework-urilor, iar metodele de construire a modelelor să nu influențeze performanțele. Experimentele s-au raportat la timpul de rulare, acuratețe și media pătratică (cu ajutorul acesteia se evidențiază loss-ul), pentru verificarea performanțelor de scalabilitate testându-se pe 4 eșantioane care conțin 0,1%, 1%, 10% și 100% date din setul de antrenare.

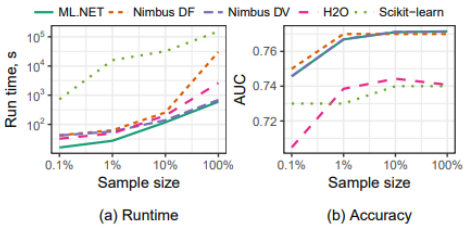

Figure 6: Experimental results for the Criteo dataset.

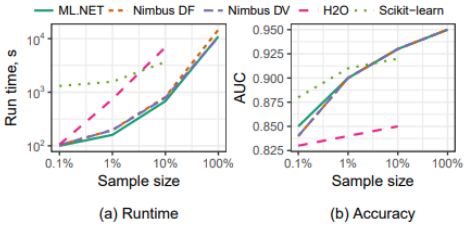

Figure 7: Experimental results for the Amazon dataset.

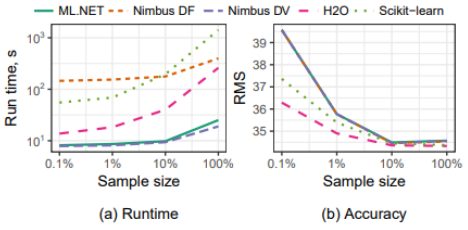

Figure 8: Experimental results for the Flight Delay dataset.

Analizând graficele atașate, se poate observa faptul că ML.NET a obținut performanțe mai bune, având un timp de rulare mult mai mic și o acuratețe mult mai precisă, cum e în cazul setului de date Criteo. În cazul setului de date Amazon se poate observa faptul că folosind Sklearn și H2O, nici măcar nu s-au putut obține rezultate din cauza faptului că acestea nu făceau față volumului mare de date și se făcea overflow. Comparând valoarea loss-ului de pe setul de date Flight Delay, se evidențiază că toate obțin valori mici, relativ apropiate, diferența făcându-se însă la nivelul timpului de rulare, care este de aproape 10 ori mai mic pentru abordarea cu ML.NET.

Concluzionând, se poate afirma faptul că ML.NET reprezintă un framework foarte important adus de Microsoft, prin prisma sustenabilității și extensibilității sale, putând fi folosit de orice programator, indiferent de nivelul său de cunoștințe din domeniul Machine Learning. De asemenea, faptul că reușește să obțină performanțe foarte bune din punct de vedere al acurateții și timpului de procesare reprezintă un alt bonus semnificativ, putându-se recurge, în cadrul dezvoltării aplicațiilor .NET, direct la utilizarea framework-ului descris pentru a se încorpora Machine Learning și a se renunța la abordările de utilizare a microserviciilor în Python.

REFERINȚE

[1] **https://docs.microsoft.com/en-us/dotnet/machine-learning/how-does-mldotnet-work**

[2] **https://www.codemag.com/Article/1911042/ML.NET-Machine-Learning-for-.NET-Developers**

[3] **https://www.daveabrock.com/2020/09/05/dotnet-stacks-15/**

[4] **https://arxiv.org/pdf/1905.05715.pdf**