**ML.NET**

A mesterséges intelligencia egy olyan ága a számítástechnikának, amely azzal foglalkozik, hogy a számítógépeket olyan feladatok elvégzésére tanítja be, amelyekhez általában valamilyen emberi beavatkozás, emberi intelligencia szükséges. Napjainkban megkérdőjelezhetetlen ütemben fejlődik és terjed, az élet egyre több területén jelenik meg a használata, és folyamatosan beépítésre kerül a különböző szolgáltatásokba, rendszerekbe. A mesterséges intelligenciának több ágazata is van, ezek közül egyik meghatározó ága a gépi tanulás.

A gépi tanulás alatt egy olyan folyamatot értünk, amely automatikusan képessé tesz egy gépet vagy rendszert arra, hogy tanuljon és javuljon a feldolgozott adatokból vett tapasztalatokból, ezáltal előrejelzéseket, becsléseket tud készíteni az új adatokra vonatkozóan. Tehát az explicit programozás helyett a gépi tanulás algoritmusokat használ nagy mennyiségű adatok elemzésére, a felismerésekből való tanulásra, majd ezek alapján a döntések meghozatalára.

A gépi tanulási algoritmusok teljesítménye idővel javulni tud azáltal, hogy egyre több adattal találkoznak amelyeket elemeznek, majd mintákat ismernek fel belőle amiknek a megtanulásával egyre pontosabb eredményeket tudnak elérni.

Maga a gépi tanulási modell alatt azt értjük, amit a program az algoritmus a tanító adatokon történő futtatásából tanul. Minél több adatot használunk, annál jobb, pontosabb lesz a modell.

A térhódításnak megfelelően a különböző programozási nyelvekhez is megjelentek az utóbbi években a különböző kiegészítők, függvénykönyvtárak, amelyek lehetővé teszik a már előre implementált gépi tanulást használó programrészek, algoritmusok beépítését a programjainkba.

A legelterjedtebb programozási nyelvek közül párat említve a c++-hoz az mlpack, a pythonhoz a scikit-learn függvénykönyvtárak használhatóak a gépi tanulás algoritmusainak eléréséhez, az általam is használt C# nyelv esetén pedig az ML.NET függvénykönyvtár tartalmazza ezeket az algoritmusokat.

Mi az az ML.NET?

Az ML.NET egy ingyenes, nyílt forráskódú és cross-platform, azaz Windows, Linux és macOS operációs rendszerekre is egyaránt elérhető gépi tanulási keretrendszer a .NET fejlesztői platformhoz. A keretrendszer nem csak C#, hanem a kevésbé ismert F# nyelvvel is kompatibilis. Az ML.NET segítségével betanítható egy egyéni modell, vagy akár importálhatóak előre betanított TensorFlow- és ONNX-modellek.

Az ML.NET tökéletesen használható számos kategorizálási, előrejelzési, összehasonlítási feladatra, többek között például az ügyfelek visszajelzéseinek kategorizálására (jó vagy rossz), folytonos jellemzők, tehát például termékek árának megbecslésére, dolgok hasonlóságának összehasonlítására, vagy akár a felhasználók korábbi tevékenységei, érdeklődési körei alapján történő ajánlások készítésére, ami az én szakdolgozatomnak is a feladata.

A függvénykönyvtár első verziójának kiadására 2018 május 7-én került sor, a stabil kiadás legelső, 1.0-ás verzióját pedig 1 évvel később, 2019-ben a Microsoft Build nevű konferenciáján jelentették be. A szakdolgozatom írásakor a legfrissebb kiadott stabil verzió az a 2024 január 18-án megjelent 3.0.1-es verzió, így a program megírásához is ezt a verziót használtam.

Hogyan néz ki egy ML.NET alkalmazás?

Egy ML.NET program felépítése általánosságban a következőképpen néz ki:

* MLContext létrehozása

Egy ML.NET alkalmazás legelső lépése az MLContext létrehozása, hiszen ezek után kezdődhet csak el az adatok betöltése, átalakítása, a modell betanítása, az algoritmusok kiválasztása. Tehát az MLContext az egész ML.NET alkalmazás kiindulópontja.

var mlContext = new MLContext();

* Adatok betöltése

Mint már korábban is említésre került, a gépi tanulás már ismert adatokat dolgoz fel, ezekben keres mintákat, majd az ezekből nyert ismeretek segítségével becsül meg ismeretlen adatokat, információkat. Tehát az ML.NET-nek is szüksége van ezekre az úgynevezett betanító adatokra, amiket fel tud használni a későbbi becslések során.

A bemeneti adatokat, amikben a minták keresése, felismerése történik, Feature-öknek nevezzük, így jelennek meg a kódban. A kimeneti adatok, amelyeknek az előrejelzése gyakorlatilag a programnak a célja, Label-ként szerepelnek a kódban.

Az adatok IDataView objektumokba kerülnek betöltésre, amelyek tartalmazhatnak különböző típusú számokat, stringeket, bool értékeket, vagy akár vektorokat és számokat tartalmazó tömbőket is tölthetünk bele, listákat viszont nem támogat. Az adatok betöltése, amennyiben fájlból történik, lehetséges többek között txt, csv, tsv és egyéb formátumokból is, viszont élő, valós idejű adatok betöltésére is lehetőség van.

IDataView trainingData = mlContext.Data.LoadFromTextFile<SentimentInput>(dataPath, separatorChar: ',', hasHeader: true);

Amennyiben már memóriában lévő, változókból, tömbökből származó adatokat szeretnénk betölteni, akkor arra is lehetőségünk van.

IDataView trainingData = mlContext.Data.LoadFromEnumerable<SentimentInput>(inMemoryCollection);

* Adatok átalakítása

Sok esetben az adatok hiába tartalmaznak rengeteg hasznos és értékes információt, eredeti formájukban, amikben betöltésre kerültek sajnos nem használhatóak, így átalakításokat kell végeznünk rajtuk annak érdekében, hogy a modellünk betanítására fel tudjuk őket használni. A string értékeket például számokká kell átalakítanunk, a különböző hosszúságú tömböket pedig ahhoz, hogy összehasonlíthatóak legyenek, elő kell készítenünk olyan módon, hogy egységes elemszámúak legyenek.

Számos lehetőséget kínál a függvénykönyvtár az adatok átalakítására. Az átalakításokat végző metódusok egy részének feltétele az, hogy csak a betanítási adatokra lehet őket meghívni, egyes adatátalakításokhoz viszont nem szükségesek a betanítási adatok, így használhatóak azok nélül is. A teljesség igénye nélkül néhány, amelyek gyakoribb problémákat oldanak meg:

* ConvertType: A bemeneti oszlop típusát lehet vele átalakítani
* MapValueToKey: A művelet során a bemeneti adatok értékeit egy adott kategóriákhoz (kulcsokhoz) rendeljük, tehát minden kategória egyedi kulcsértékkel fog rendelkezni, amelyek alapján azonosítani tudjuk majd őket.
* TokenIntoWords: Egy vagy több darab, szöveget tartalmazó oszlop osztható fel vele szavakra
* OneHotEncoding: segítségével a kategóriákat különálló, bináris vektorokba kódoljuk, ami lehetővé teszi a szöveges adatok hatékony feldolgozását és elemzését
* Concatenate: Segítségével megoldhatjuk azt a problémát, hogy ha több oszlop adatait egyetlen oszlopba szeretnénk egyesíteni, például szöveges adatok esetén.
* Algoritmus kiválasztása

Miután az adatokat megfelelő formába alakítottuk, kiválaszthatjuk a használni kívánt algoritmust attól függően, hogy mi a programunk célja. Az ML.NET függvénykönyvtárból több mint 30 algoritmus közül választhatunk. Ugyanazon probléma megoldására több algoritmus is alkalmas lehet, így ahhoz, hogy megtaláljuk az adott helyzetben a legjobban működőt, érdemes kipróbálnunk minél többet.

Az algoritmusok között megtalálhatóak a következő típusú metódusok:

* Lineáris algoritmusok
* Döntési fa algoritmusok
* Mátrix-faktorizációk
* Meta-algoritmusok
* K-közép
* Modell tanítása
* Modell kiértékelése
* A modell kimentése, használata

Források:

<https://cloud.google.com/learn/artificial-intelligence-vs-machine-learning>

<https://en.wikipedia.org/wiki/ML.NET>

<https://learn.microsoft.com/hu-hu/dotnet/machine-learning/how-does-mldotnet-work>

<https://dotnet.microsoft.com/en-us/learn/ml-dotnet/what-is-mldotnet>

<https://dotnet.microsoft.com/en-us/apps/machinelearning-ai/ml-dotnet>

<https://medium.com/@audaciatech/an-introduction-to-ml-net-and-the-functions-it-performs-8eeae0d94641>