Strojno učenje u NET-u

Ćurčija, Ivica

SVEUČILIŠTE U DUBROVNIKU
ODJEL ZA ELEKTROTEHNIKU I RAČUNARSTVO

IVICA ĆURČIJA

STROJNO UČENJE U .NET-u

ZAVRŠNI RAD

Dubrovnik, rujan, 2019.

SVEUČILIŠTE U DUBROVNIKU
ODJEL ZA ELEKTROTEHNIKU I RAČUNARSTVO

STROJNO UČENJE U .NET-u

ZAVRŠNI RAD

Studij: Primjenjeno/Poslovno računarstvo
Kolegij: Strojno učenje
Mentor: izv.dr.sc Mario Miličević
Student: Ivica Ćurčija

Dubrovnik, rujan, 2019.

Sažetak

Ovim radom se analizira ML.NET framework unutar .NET platforme. Razmatra se njegova upotreba unutar desktop i web aplikacija koje se mogu razvijati unutar .NET-a. Strojno učenje postaje sve raširenija grana tehnologije i stvaraju se alati koji uvelike pojednostavljuju stvaranje kompleksnih modela i rješenja kroz strojno učenje koja se mogu koristiti bez opširnog prethodnog znanja o istome.

Dakle, ovaj rad cilja izdvojiti jedan osnovan programerski problem i pojednostavniti ga i riješiti koristeći ML.NET framework.

Rad je podijeljen u nekoliko cjelina. U uvodu je kratko opisano područje u kojemu se koristi Strojno učenje i što je ML.NET. U drugoj cjelini objašnjavaju se tehnologije i alati koji će se koristiti od same .NET platforme do algoritma koji je odabran za model unutar aplikacije. Treća cjelina se bazira na praktični dio rada odnosno stvaranje modela koji bi vršio predikciju cijene taxi vožnja na temelju određenih podataka.

Ključne riječi: ML.NET, Strojno učenje, .NET

Abstract

With this thesis my goal is to analyze the ML.NET framework for the .NET platform. The thesis will be looking into it's use inside desktop and web applications that can be developed under .NET.

Machine Learning is seeing extensive growth and tools that simplify making complex models and solutions through machine learning are being made for people without a wide spectrum of konwledge of the subject.

This thesis aims to seperate a single basic programming problem, simplify and solve it using ML.NET framework.

The thesis is split into several main units. In the introduction it go over the area in which machine learning is used and what is ML.NET. In the second unit the main vocal points are the technologies and tools that are used in this thesis from .NET platform through to the algorithm used for the model inside the application. Third unit is based on the practical part of the thesis which implies making a model that would make predictions of taxi rides using a particular data set.

Key words: ML.NET, Machine Learning, .NET

Sadržaj

1. Uvod

Strojno učenje je metoda analiziranja podataka koja automatizira gradnju analitičkih modela. To je grana umjetne inteligencije koja se zasniva na ideji da računala mogu „učiti“ od podataka, identificirati uzorke i donositi odluke s minimalnim djelovanjem čovjeka. Do velikog interesa u strojno učenje dovele su velike količine dostupnih podataka iz raznih sektora kao što su telekomunikacije i financije kao i energetike, osiguranja i maloprodaje, te jeftinija i snažnija procesorska moć računala i jeftiniji prostor za spremanje podataka. U zadnjih nekoliko godina strojno učenje raširilo je svoju upotrebu u kompleksnim sustavima kao što su roboti, samovozeći automobili i slično, te u jednostavnijim aplikacijama koje koristimo u svakodnevnom životu. U ovom završnom radu primjenjivati će se alati dostupni u ML.NET open source frameworku za strojno učenje koji je dostupan unutar .NET razvojne platforme koju je stvorio Microsoft. Korištenjem ovih alata analizirat će se jednostavan model koji će na temelju određenih parametara odrediti cijenu taxi vožnje. Sličan model se već koristi u mnogim taxi aplikacijama kao što su Uber i lokalni taxi servisi.

1.1 Zadatak završnog rada

U ovom završnom radu potrebno je objasniti i pokazati primjenu strojnog učenja unutar .NET platforme korištenjem ML.NET frameworka. Kao osnova za razumijevanje projektnog zadatka potrebno je dati pozadinu strojnog učenja.

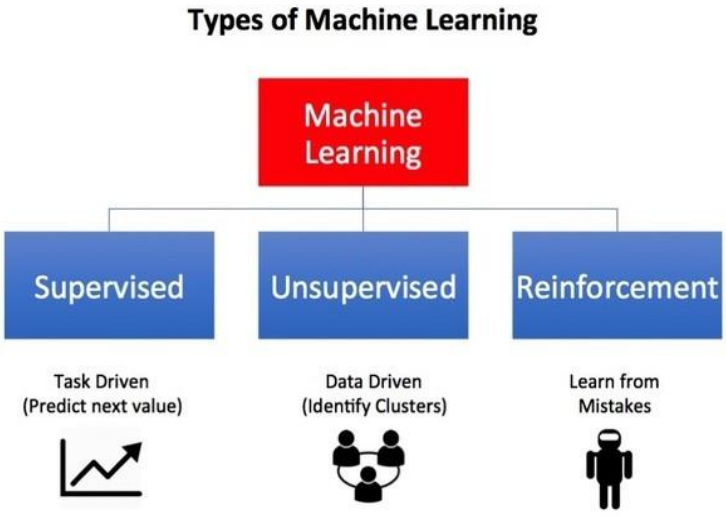
2. Tehnologije i alati

2.1 Strojno učenje

Strojno učenje je grana umjetne inteligencije koja se bavi oblikovanjem algoritama koji svoju učinkovitost poboljšavaju kroz iskustvo. Ulazni podatci koje koristimo uz neki algoritam učenja nazivaju se trening podaci, oni utječu na iskustvo modela strojnog učenja, a izlaz je neko znanje koje se koristi za klasifikaciju podataka, predviđanje određenih numeričkih rezultata, adaptacija varijabli programa i slično. Strojno učenje se koristi kada programi trebaju biti adaptivni ili su previše kompleksni za programiranje.
Neke od najzastupljenijih uporaba strojnog učenja:

- Prilagodljivi programski sustavi
- Bioinformatika
- Obrada prirodnog jezika
- Raspoznavanje govora
- Inteligentno upravljanje
- Predviđanje trendova

Postoji više vrsta strojnog učenja kao što je to vidljivo na slici 1.



Slika 1 Vrste strojnog učenja [1]

2.1.1 Nadzirano učenje

Nadzirano učenje (Supervised learning) odnosi se na učenja s točnim ciljevima učenja kao što je predviđanje novih još ne analiziranih primjera. Najbolji primjeri ove vrste učenja su regresija i klasifikacija. Regresija se odnosi na učenje funkcija, tražena informacija o objektu je numerička npr. cijena, temperatura, tlak, broj cipela i slično. Klasifikacija je učenje o raspoznavanju uzoraka, tražena informacija je kategorijska i može imati jednu ili više klasa. Cilj je izgraditi model koji će za svaki objekt kojoj klasi pripada npr. boja, zaražena/nezaražena osoba i slično. Za rješavanje projektnog zadatka koristit će se ova metoda strojnog učenja.

2.1.2 Nenadzirano učenje

Nenadzirano učenje (Unsupervised learning) odnosi se na učenje bez ciljne vrijednosti, podaci su neoznačeni. Bitna je samoorganizacija, cilj je grupirati primjere, otkriti neku strukturnu pravilnost u podacima, odrediti što podaci znače . Tipične primjene nenadziranog učenja su :

- marketing: segmentacija korisnika
- biologija: grupiranje biljaka ili životinja prema njihovim značajkama
- grupiranje sličnih dokumenata
- pretraživanje informacija: grupiranje sličnih rezultata
- obrada slike : sažimanje slike grupiranjem sl. elemenata sličnih boja

2.1.3 Učenje s podrškom

Učenje s podrškom (Reinforcement learning) je učenje kroz interakciju s okolinom (ili barem simulaciju interakcije). Proizlazi se serija stanja i akcija i tek na kraju se dobiva (ili ne dobiva) nagrada. Cilj je istražiti stanja i akcije koje vode do cilja te maksimizirati ukupnu sumu nagrada na kraju. [1]
Tipične primjene ovakvog modela su :

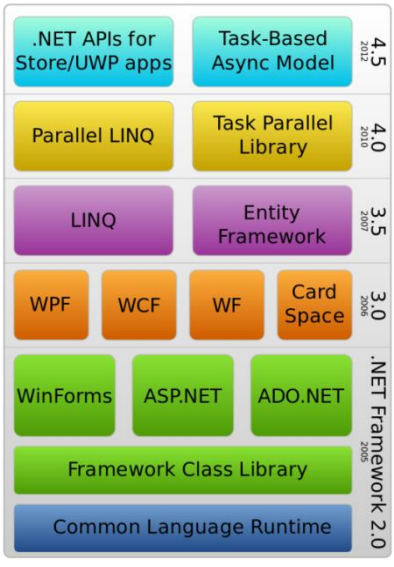
- igranje igara
- robotsko kretanje

- autonomna navigacija
- učenje kontrolnih strategija

2.2 .NET framework

.NET framework je softverski okvir od strane Microsofta koji primarno radi na Microsoft Windows operacijskim sustavima ali je kompatibilan i s IOS i Android OS operacijskim sustavima. Windows operacijski sustavi su najzastupljeniji u okviru desktop računala i laptopa s preko 70% tržišta. .NET Sadrži veliku kolekciju klasa pod nazivom Framework Class Library (FCL) i pruža korištenje 44 programskih jezika od kojih su najpopularniji C#, C++ i VisualBasic. Programi napisani za .NET izvode se u software okruženju zvanom Common Language Runtime (CLR). CLR je aplikacijska virtualna mašina, što znači da se pokreće unutar operacijskog sustava i podupire rad jednog procesa. Na slici 2. prikazan je protocol stack .NET-a i kako se razvijao kroz nadogradnje, te vidimo da su FCL i CLR osnova .NET-a. Prva verzija .NET 1.0 je službeno izdana 2002. godine te je od tada dobila 9 nadogradnji i trenutno najnovija verzija je .NET 4.8.

Framework je namijenjen za korištenje od strane većine novih aplikacija stvorenih za Windows platformu. Microsoft je također stvorio integrirano razvojno okruženje pod imenom Visual studio namijenjenom najvećim dijelom za razvoj .NET aplikacija.



Slika 2 .NET component stack [2]

.NET platforma sastoji se od 4 grupe proizvoda. Prva grupa su programski jezici u okviru Visual Studio razvojne okoline (C#, Visual Basic, .NET, C++), drugu grupu tvore .NET Enterprise serveri (SQL Server, Exchange Server, BizTalk), treća grupa su komercijalne web usluge(Project Hailstorm) i četvrta grupa su .NET mobilni uređaji kao npr. Mobilni telefoni, uređaji za igre i slično.
Osnovna skupina klasa zove se Base Class Library (BCL) te sadrži osnovne funkcionalnosti koje se koriste u programiranju (funkcije za transformaciju teksta, hvatanje unosa s tipkovnice, provjera sigurnosnih prava)
Web forms je dio skupa klasa nazvanog ASP.NET i zadužen je za razvoj web aplikacija. ASP.NET predstavlja budućnost programiranja jer web forme u sebi sadrže sve objekte potrebne za generiranje HTML sadržaja objektno orijentiranim pristupom.

Windows Forme predstavljaju standardne klase za rad s Windows okruženjem. Sadrži sve objekte standardnog Windows sučelja te pojednostavljuje stvaranje Windows aplikacija.

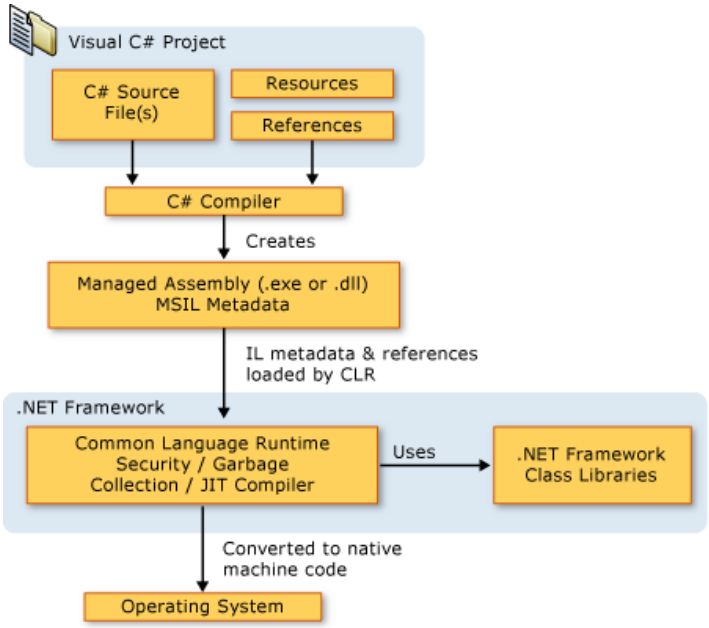
2.3 C#

C# (C sharp) je objektno orijentirani programski jezik razvijen od strane Microsofta. Nastao je s ciljem da .NET platforma dobije vlastiti programski jezik koji će maksimalno iskoristiti njezine mogućnosti, pa je tako došao na tržište zajedno s .NET platformom 2000. godine. Temelji se na ostalim objektno orijentiranim jezicima kao što su Java, C++ i Visual Basic. Semantika korištena u C# jeziku velikim dijelom je preuzeta iz Jave. C# omogućuje kreiranje vizualnih aplikacija čak i onim korisnicima koji nemaju programerskog iskustva te daje dobar uvid u rad i stvaranje vizualnih i objektnih aplikacija. [3]

Veliki dio svojih glavnih svojstava preuzima od prije spomenutih sličnih objektno orijentiranih programskih jezika pa tako ima velike mogućnosti u definiranju klasa, novih metoda koje upravljaju tim klasama te korištenje enkapsulacije, nasljeđivanja i polimorfizma. Također podržava rad s XML stilom unutar dokumenata, sučelja i svojstava te podržava rad s pokazivačem i „garbage collection“.
Garbage collection je postupak oslobađanja memorije koja je bespotrebno zauzimaju objekti koji se više ne koriste u programu. Programer je oslobođen brige o tome koji objekti više nisu u upotrebi i kada ih je potrebno terminirati jer to sustav radi umjesto njega.

Izvorni kod napisan u C# sastavljen je u intermediate language(IL) koji je u skladu s .NET CLI specifikacijama. IL kod i resursi pohranjeni su na disku u izvršnoj datoteci zvanoj assembly s ekstenzijom .dll ili .exe . Datoteka sadrži manifest koji pruža informacije o vrsti,verziji i sigurnosnim zahtjevima assembly-a.

Kada se izvršava C# program, assembly datoteka se učitava u CLR koji poduzima različite radnje na temelju podataka iz manifesta. Ako su zadovoljeni sigurnosni uvjeti, CLR izvršava just in time (JIT compiler) kompilaciju koja pretvara IL kodove u izvorne strojne instrukcije. CLR pruža i druge funkcije vezane za upravljanje iznimkama (exceptions), upravljanje resursima i garbage collection . Na slici 4 je prikaz procesa .NET arhitekture tj. proces prevođenja C# koda u strojne instrukcije.



Slika 3 .NET arhitektura [4]

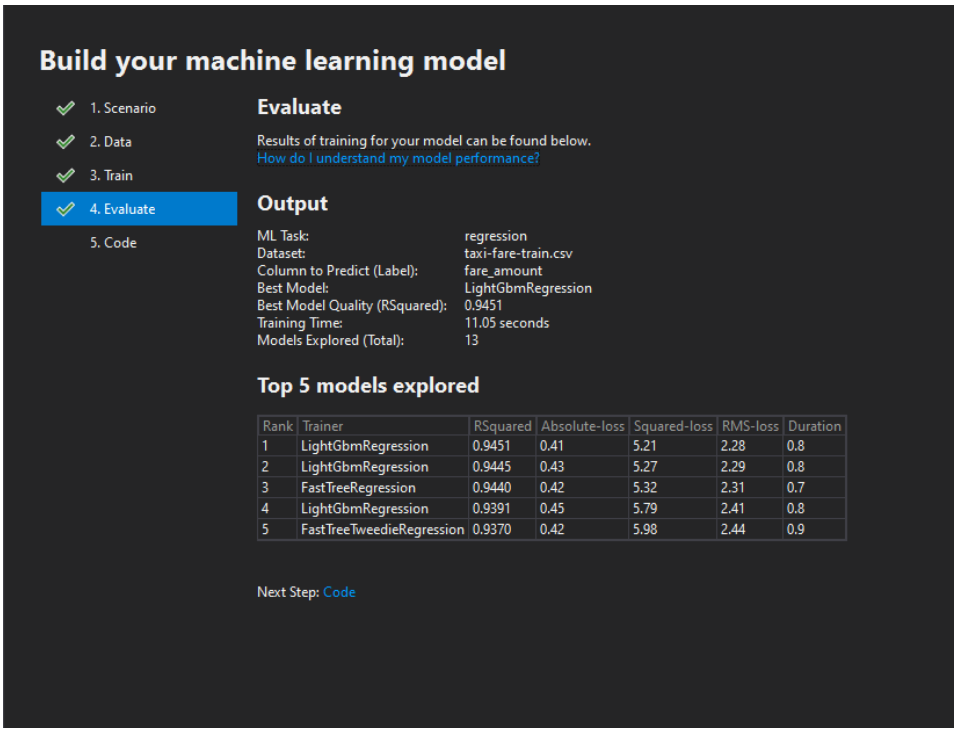
Višejezičnost je ključna značajka .NET okvira. S obzirom na to da je IL kod koji se proizvede iz C# kompajlera u skladu sa .NET CTS, taj kod može komunicirati s bilo kojim kodom generiranim iz .NET verzija Visual C++, Visual Basic-a i bilo kojeg drugog programskog jezika dostupnog unutar .NET platforme.

2.4 ML.NET

ML.NET je besplatan software dodatak za strojno učenje u .NET okruženju. Nadopunjava funkcije .NET sučelja dodajući analitičke analize i mogućnosti predviđanja rezultata na temelju velikih brojeva podataka. ML.NET je stvoren na .NET core i .NET Standardu te nasljeđuje mogućnost pokretanja preko više platformi kao što su Windows, Linux i MacOS. Prva stabilna inačica ML.NET-a puštena je 6.8.2019. Iako je relativno nov ML.NET framework je nastao u 2002 kao Microsoftov istraživački projekt zvan TMSN(text mining search and navigation) za korištenje unutar Microsoftovih proizvoda.

Programeri mogu sami stvarati modele za strojno učenje ili koristiti postojeće modele koje su preuzeli te ih pokretati na bilo kojem okruženju bez potrebe spajanja na internet. Ovo znači da programeri ne moraju imati opširno znanje znanosti o podacima da bih koristili prednosti strojnog učenja u svojim aplikacijama.
ML.NET CLI je Command-line sučelje koje koristi ML.NET AutoML za treniranje modela i biranje najboljeg algoritma za dane podatke.

ML.NET Model Builder je dodatak za Visual Studio koji koristeći GUI (Graphical user interface) pojednostavljuje proces biranja podataka, ishoda učenja te biranja najboljeg algoritma koristeći AutoML i CLI.Na slici 5 se vidi završni korak u Model Builderu evaluacija modela. Model Builder služi kao jednostavniji način generiranja koda i samog modela, ali ne limitira programera u modificiranju modela, generirani kod se može mijenjati kroz korištenje drugih algoritama, mijenjanje parametara algoritma, rada s podacima i slično.



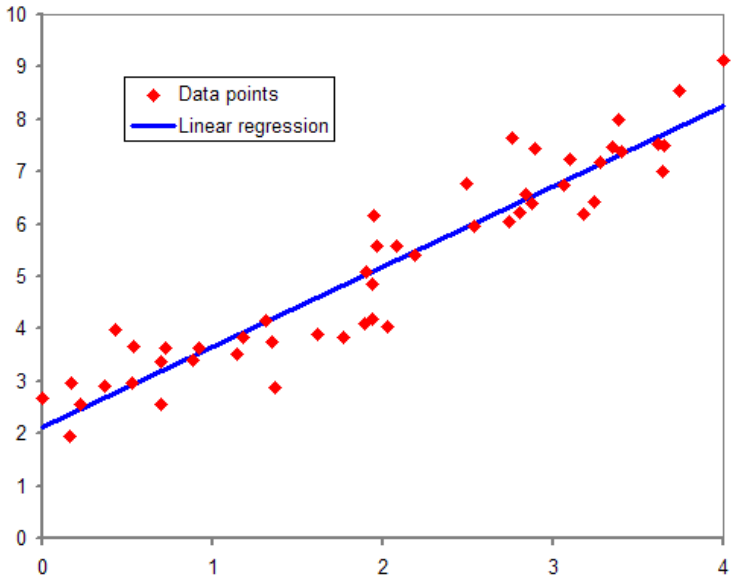
Slika 4 Model Builder 4. korak „Evaluacija“

Na temelju prikazanih informacija vidimo da je Model Builder odabrao LightGBM Regression algoritam na temelju kvalitete modela koja iznosi 0.9451 te je najveća od 13 testiranih modela.

Za ML.NET probleme postoji nekoliko trenirajućih algoritama od kojih se može izabrati najefikasniji i najbolji za slučaj koji se pokušava riješiti. Algoritam je konačan slijed dobro definiranih naredbi za ostvarenje zadatka. Različiti algoritmi stvaraju modele s različitim karakteristikama. Sa ML.NET-om isti algoritam može ze primijeniti za različite slučajeve npr. Stohastic Dual Coordinated Ascent (SDCA) može se koristiti za binarnu klasifikaciju, višeklasnu klasifikaciju i regresiju. Razlika je u tome kako se interpretira rezultat algoritma u odnosu na slučaj. Za svaku kombinaciju slučaj/algoritam ML.NET sadrži komponentu koja odrađuje trening algoritam i radi interpretaciju. Ove komponente se zovu treneri. Primjer je SdcaRegressionTrainer koristi SDCA algoritam primijenjen na problem regresije.

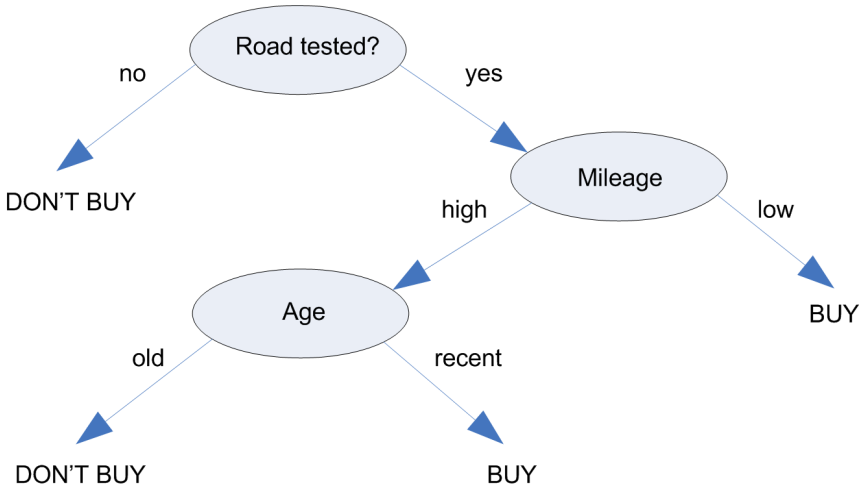
Najveći dio algoritama koji su unutar ML.NET frameworka su linearna regresija i stabla odluke.

Linearna regresija je osnovna i najčešće korištena vrsta analize predviđanja. Glavna ideja regresije je promatrati dva ponašanja, da li set varijabli za predviđanje dobro odrađuje posao biranja izlazne varijable, kakve varijable su važne za proces predviđanja izlazne varijable i koliko one utječu na završni rezultat. Dakle linearna regresija se odnosi na svaki pristup modeliranju relacije između jedne ili više varijabli označenih s Y, te jedne ili više varijabli označenih s X, tako da takav model linearno ovisi o parametrima određenih iz skupa podataka (slika 6.). [7]



Slika 5 Linearna regresija [7]

Stabla odlučivanja su prediktivni modeli koji na temelju podataka izvode njihove veze u cilju dobivanja izlaznih vrijednosti. Stabla odlučivanja su intuitivna i laka za razumijevanje. Postoji velika količina algoritama koji stvaraju stabla odlučivanja. Stabla imaju prednosti jer su brza,precizna i laka za razumjeti. Listovi stabla se odnose na klasifikacije ili stvorene kategorije a grane se dijele po karakteristikama koje dovode do razvrstavanja u kategorije ili klase kao primjer toga prikazano je jednostavno stablo odlučivanja za kupnju automobila na slici 6.



Slika 6 Stablo odlučivanja [6]

Slika 7. prikazuje model u kojem je konačni rezultat kupiti ili ne kupiti automobil, cilj stabla je proći kroz stablo odgovarajućim putem koji predstavlja naš slučaj dok se ne dobije jedan od dva moguća ishoda. Algoritam stabla odlučivanja je osnova algoritma Light GBM koji je korišten u izradi projektnog zadatka.

2.5 LightGBM

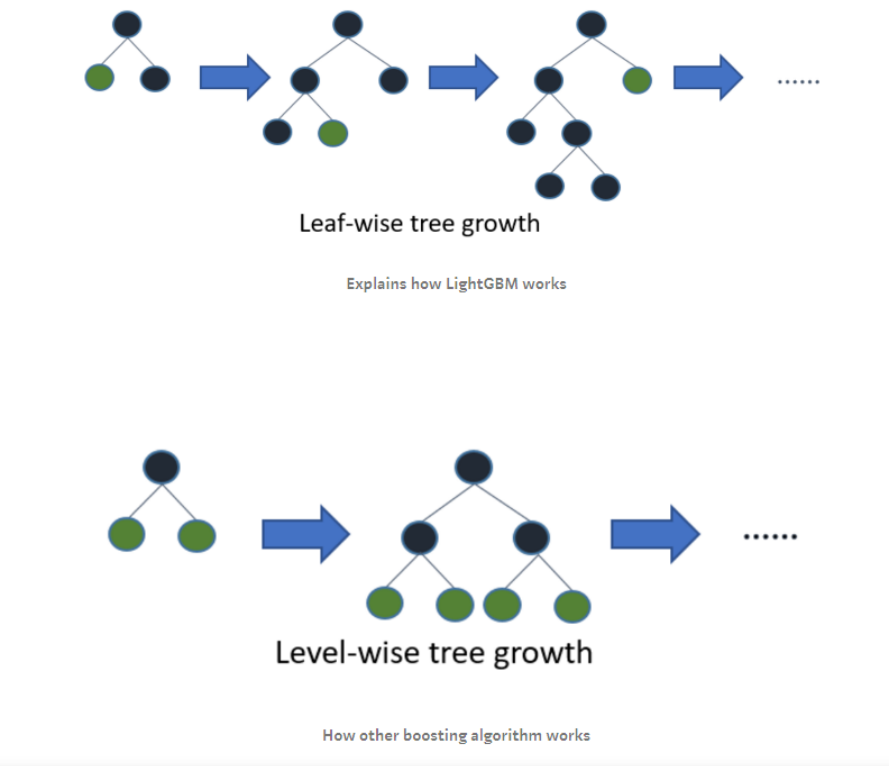
Light GBM je gradient boosting framework koji koristi algoritam stabla odlučivanja. Light GBM razvija svoje stablo vertikalno za razliku od drugih algoritama koji to rade horizontalno. Prednosti korištenja ovog frameworka su :

- Brža brzina treniranja i veća efikasnost
- Manje korištenje memorije
- Bolja preciznost
- Podupire paralelno i GPU učenje
- Sposoban za rukovanjem s velikim skupovima podataka

Boosting spada pod tehnike za izgradnju ansambla odluke pod koje još spadaju bagging i stacking. Boosting daje najbolje rezultate od ove tri tehnike jer koristi neke iste tehnike kao i poznata statistička tehnika aditivnih modela, što je rezultiralo daljnjom optimizacijom algoritma. Metoda se temelji na ideji da u slučaju više eksperta jedan pokriva područje u kojemu drugi nije toliko precizan, što znači da u skupu više eksperta zajedno pokrivaju široku domenu problema. Izgradnja ansambla se temelji na iteraciji osnovnih modela u

kojoj se svaki novi model gradi u odnosu na performanse prijašnjih iteracija.Sve nove iteracije stvaraju modele koji se fokusiraju na ispravljanje grešaka instanci koje je prijašnja iteracija krivo klasificirala, što znači da svi osnovni modeli nisu jednaki, već imaju težinu koja uvjetuje koliko udjela doprinose u glasanju ukupnog ansambla.

Kao što je već spomenuto Light GBM razvija stablo vertikalno sto znači da bira listove s maksimalnom delta loss vrijednosti za grananje. Kod razvijanja istog lista, algoritam koji se bazira na gradnju listova može smanjiti gubitke više nego algoritmi bazirani na gradnju nivoa. Na slici 8. je prikazana razlika između algoritama gradnje listova i algoritma gradnje nivoa .



Slika 7 Usporedba algoritma baziranog na gradnju listova i algoritma za gradnju nivoa [5]

Algoritmi bazirani na gradnji listova podložni su overfittingu kad je količina podataka mala, te Light GBM uvodi max_depth parametar koji ograničava dubinu stabla. Iako se stabla još uvijek razvijaju kroz gradnju listova i kada je max_depth definiran.

Light GBM određuju parametri koji se mogu podešavati kako bi se ostvario sto bolji rezultat. Najvažniji kontrolni parametri su :

- max_depth – opisuje maksimalnu dubljinu stabla. Koristi se za rješavanje problema overfittinga.
- min_data_in_leaf – minimalni broj zapisa koje jedan list može sadržavati.
- early_stopping_round – ovaj parametar pomaže ubrzati analizu. Model će prestati trenirati ako se mjera jednog validacijskog podatka ne poboljša u posljednjoj early_stopping_round rundi. Ovo će smanjiti pretjerani broj iteracija.

Jezgreni parametri određuju varijacije tipova modela te specifične vrste problema koji se rješavaju. Najvažniji jezgreni parametri su:

- task – određuje operaciju koju želimo obavljati nad podacima, može biti treniranje ili predviđanje
- application – određuje primjenu modela , bio to regresijski ili klasifikacijski problem, uobičajeno će biti odabran regresijski model pored kojeg su dostupni binarni i višeklasni klasifikacijski modeli
- boosting – označava tip algoritma koji želimo koristiti, uobičajeno Gradient Boosting Decision Tree (gbdt) , dok su još dostupni Random Forest,DART i GOSS algoritmi
- num_leaves – broj listova u punom stablu, uobičajena vrijednost je 31
- device – procesor koji se koristi za razvijanje modela može biti cpu ili gpu. [5]

3. Izrada modela

Kao primjer rada ML.NET frameworka i njegovih sposobnosti primijenit ćemo njegove funkcije na stvaran slučaj te razmotriti korake koji su potrebni za postizanje željenih rezultata. Primarni cilj eksperimenta je prikazivanje osnovnih funkcija i parametara koji se koriste unutar ML.NET frameworka te analiziranje rezultata i metrika modela koji će se koristiti unutar primjera.

3.1 Upoznavanje problema

Problem koji se razmatra je određivanje cijene taxi vožnje u New Yorku. Podaci su dobiveni iz data seta koji nudi NYC Taxi & Limousine Commision. Na prvi pogled mogli bi zaključiti da se cijena određuje samo po udaljenosti koja se prelazi. Ali taksisti naplaćuju na temelju više faktora kao što su dodatni putnici, kartično plaćanje i slično.
Stoga cilj ovog modela je predvidjeti cijenu vožnje prije nego što se naruči sa što većom preciznošću. Data set koji koristimo sadrži podatke iz 100 000 različitih taxi vožnji u New Yorku, te bi trebao biti dovoljno opširan za treniranje algoritma i precizne rezultate. Ova vrsta problema naziva se regresija tj. predviđanje neke stalne vrijednosti za dane parametre, slični primjeri ovome bi bili predviđanje cijene kuće na temelju lokacije, broju soba, godine izgradnje, ili potrošnja goriva na temelju vrste goriva i parametrima automobila. Cijena koja se dobiva kao rezultat modela može biti okvirna tj. treba okvirno prikazivati koliko će otprilike cijena vožnje iznositi na odredištu.
Ovakav model se može koristiti i već se koristi u većini postojećih taxi i prijevoznih aplikacija.

Za rješavanje ovog problema stvorit ćemo ML.NET model koji kao ulazne podatke prima:

Vendor ID – oznaku određene vrste taxija, postoje samo dvije moguće vrste podataka CMT ili VST

Rate code – oznaka cijene po prijeđenoj milji

Passenger count – broj putnika u taxiju

trip_time_in _seconds – vrijeme putovanja u sekundama
payment type – CRD za card tj. kartično plaćanje i CSH za cash tj. plaćanje gotovinom

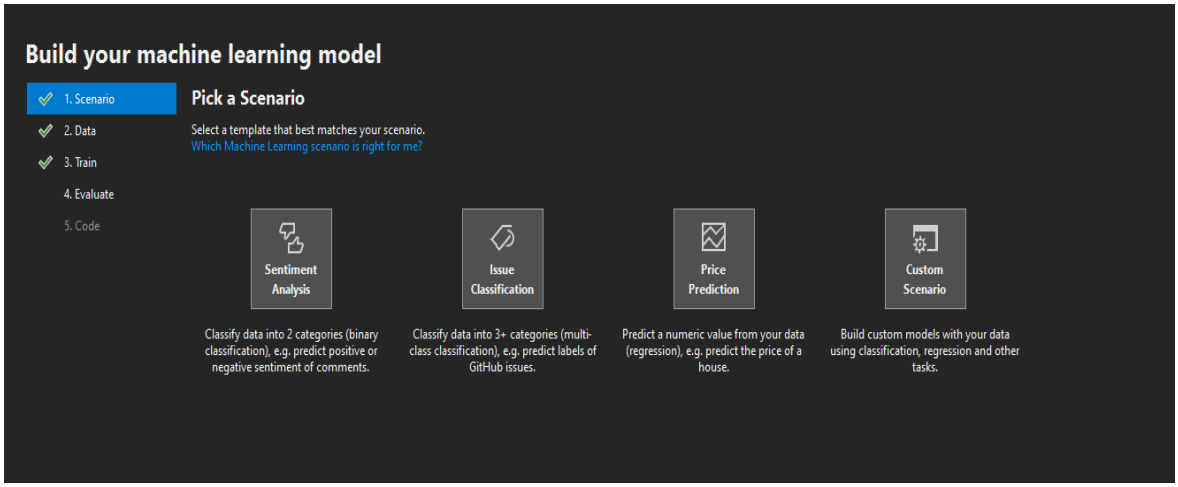
Zadnji podatak je fare amount i označava cijenu vožnje u dolarima, to će ujedno i biti ciljani podatak koji se treba predvidjeti.

vendor_id	rate_code	passenger_count	trip_time_in_secs	trip_distance	payment_type	fare_amount (Label)
CMT	1	1	1271	3.8	CRD	17.5

Slika 8 Primjer jednog retka iz skupa podataka

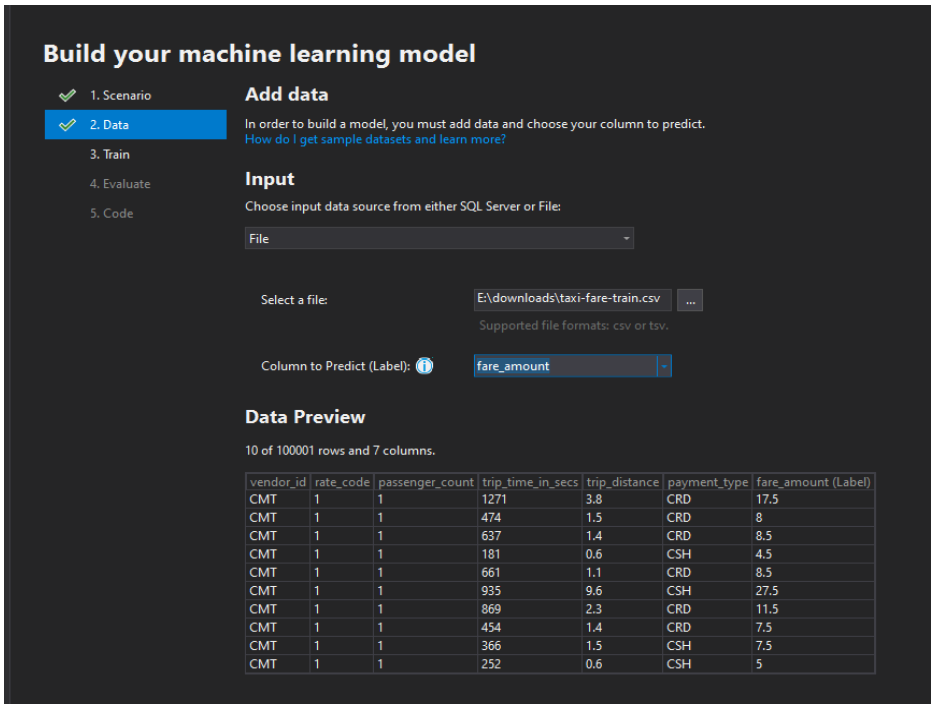
3.2 Stvaranje modela

Za stvaranje modela koji će se koristiti za rješavanje problema koristiti Auto ML Model Builder koji je dio ML.NET frameworka i uvelike olakšava rješavanje jednostavnih problema koji se rješavaju uz pomoć strojnog učenja. Ovakav pristup stvaranju modela izgleda uvelike automatiziran, ali rezultati korištenja Model Buildera davaju podlogu koja se može mijenjati i adaptirati prema željama programera kroz promjenu algoritama, parametra algoritma ili samog procesa stvaranja modela. Prvi korak u gradnji modela prikazan je na slici 10. zove se Scenario te nam nudi 4 opcije različitih vrsta klasifikacija i regresija. Kako se u ovom primjeru pokušava predvidjeti cijenu vožnje koristit će se Price Prediction Scenario.



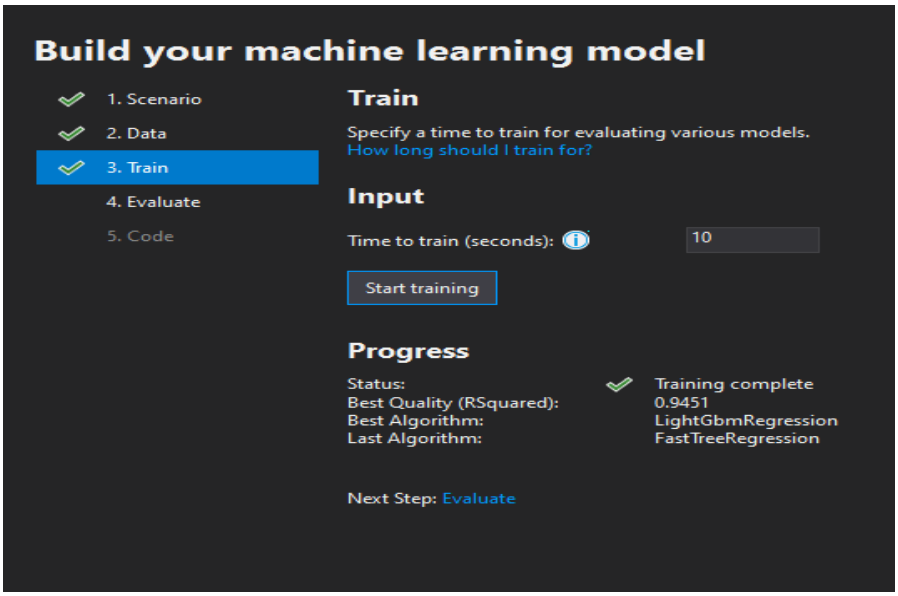
Slika 9 Izbornik slučajeva

Drugi korak u stvaranju modela je dodavanje data seta tj. podataka koje ćemo analizirati i na koje ćemo primijeniti algoritam te ga istrenirati da može raditi predikciju. Podaci se mogu pribavljati iz SQL servera ili datoteka (.csv ili .tsv formata). Te se bira label odnosno vrijednost koja se želi predvidjeti i primiti nazad kao rezultat kao što je vidljivo na slici 11.



Slika 10 Odabir podataka

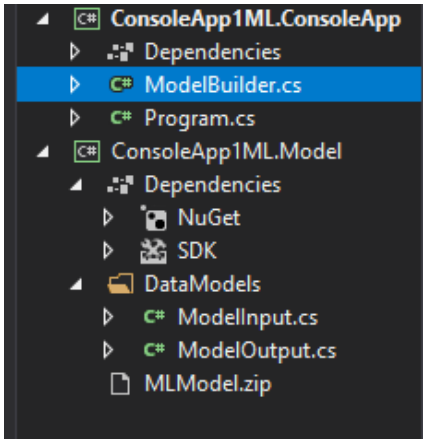
Sljedeći korak je Treniranje imamo opciju biranja vremena koje je potrebno treniranje našeg modela i određivanje najboljeg algoritma za korištenje. Preporučeno vrijeme treniranja za data setove ispod 10 MB je 10 sekundi pa sam i tu vrijednost koristio. Treniranje nam prezentira najbolji algoritam za naš slučaj te podatak Rsquared.R-squared je statistička mjera koja označava koeficijent korelacije između promatrane vrijednosti i promatrane predikcijske vrijednosti, dakle u savršenom slučaju on bi iznosio 100%. Po našem modelu Rsquared iznosi 94.5% što je poprilično dobar koeficijent za ovakvu vrstu predviđanja i proizvest će jako dobre rezultate za ovaj primjer (slika 12.).



Slika 11 Trening modela

Zadnji korak Model Buildera je Evaluacija koja nam pokazuje koliko različitih modela je testirano koji su imali najbolje rezultate, koliko je trajalo vrijeme treniranja, koji algoritam je izabran i slično. Ovaj korak je prikazan na slici 4 kao primjer Model Builder sučelja. Evaluacija daje okvirnu sliku usporedbe algoritama na modelu ali ne daje dovoljno informacija za zaključivanje kvalitete modela jer za testne podatke koristi iste podatke kao i za treniranje modela.

Nakon evaluacije Model Builder stvara model datoteku, i kod koji se koristi za dodavanje modela na postojeću aplikaciju. ML.NET modeli se spremaju kao zip datoteke. Model Builder također dodaje primjer korištenja stvorenog modela kroz aplikaciju koja radi u konzoli i kod koji služi za stvaranje modela algoritma kao što je to vidljivo na slici 13.



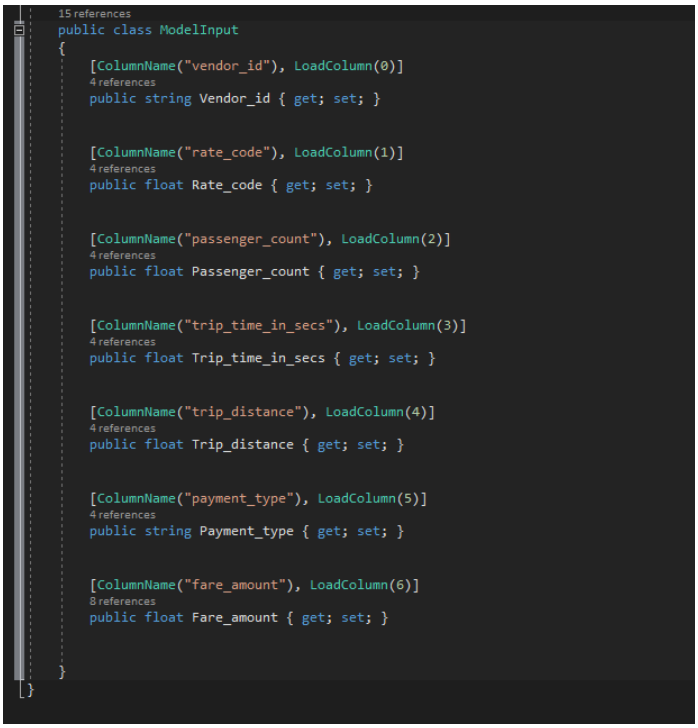
Slika 12 Generirane datoteke

Generirana aplikacija koja se pokreće preko konzole sadrži pozivanje modela iz zip datoteke te pokretanje jednog ispita za testiranje točnosti rezultata algoritma (slika 13). Uobičajeno se unutar programa sam generira jedan ispitni podatak iz data seta koji se koristio za treniranje algoritma , ali za svrhe ovog zadatka dodana su četiri primjera iz test datoteke koja se nije koristila za treniranje kao hard coded objekti klase model input tj. oblika našeg data seta (slika 14). Koristeći različitih testnih podataka od onih korištenih za trening dodatno provjerava rad modela i način na koji će se on raditi s nepoznatim podatcima. Jedna od najčešćih grešaka je over-fitting, testiranjem podataka korištenjem modelu nepoznatih primjera osigurava realistične izlazne podatke.

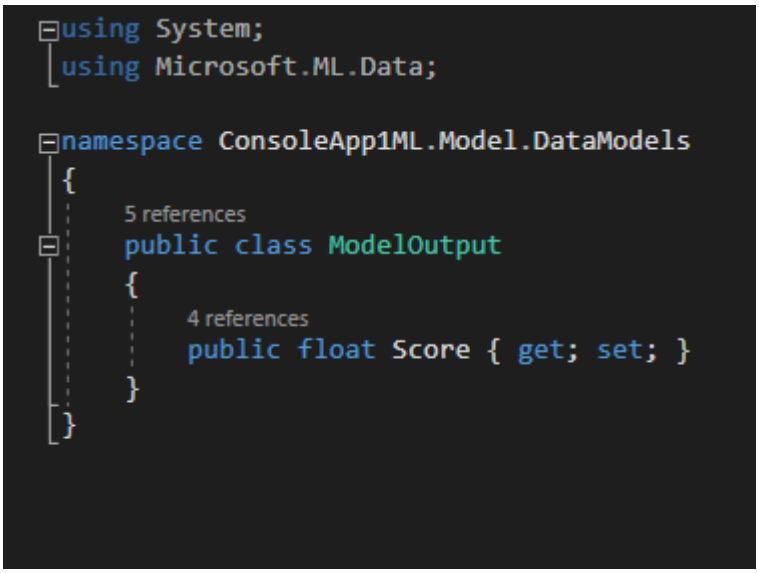
```
private static ModelInput taxiSample4 = new ModelInput
{
    Vendor_id = "VTS",
    Rate_code = 1,
    Passenger_count = 1,
    Trip_time_in_secs = 600,
    Trip_distance = 1.59f,
    Payment_type = "CSH",
    Fare_amount = 8.5f
                                    };
```

Slika 13 Stvaranje testnih objekata iz ModelInput klase

Izlazni i ulazni podaci samog modela definiraju se kao dvije klase unutar paketa modela pod nazivom DataModels. Ove dvije klase definiraju podatke koji se koriste za treniranje modela te njegove izlazne podatke. Klase nisu kompleksne i sadrže jednostavne objekte svih ulaznih podataka te get i set metode za dohvaćanje istih. Na slikama 15 I 16 Prikazani su primjeri ModelInput(ulazni podatci) i ModelOutput(izlazni podatci) klasa.

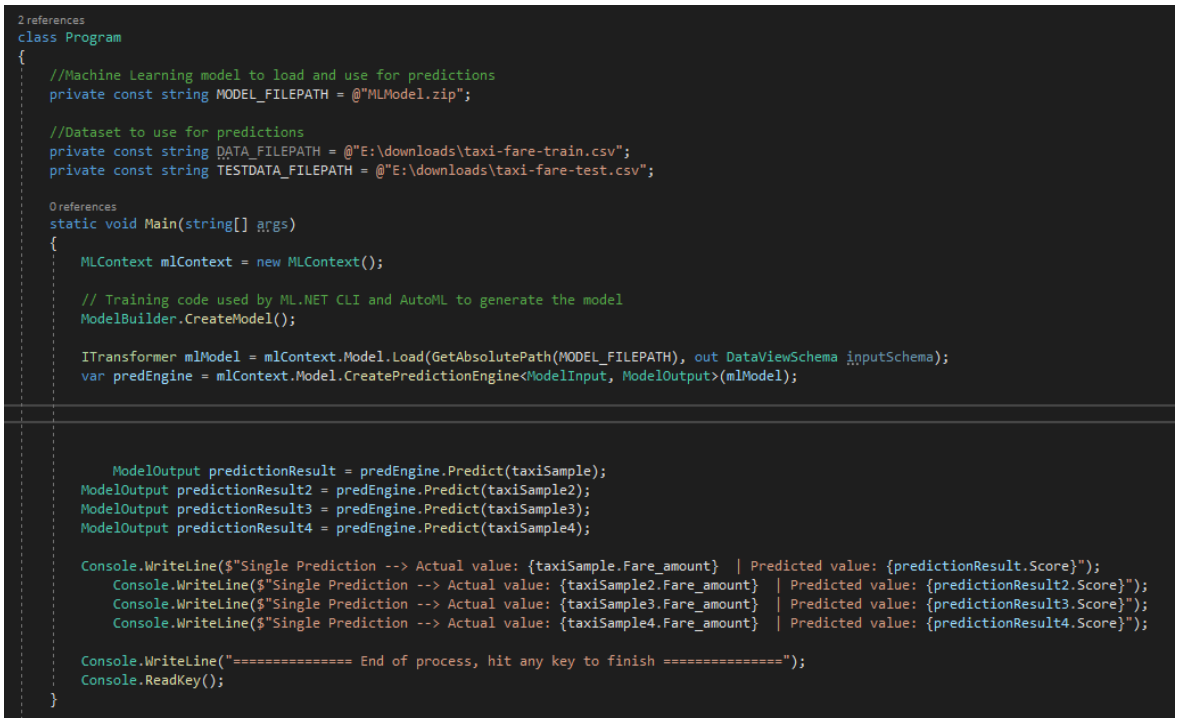


Slika 14 ModelInput klasa



Slika 15 ModelOutput klasa

Na slici 17. je primjer koda koji se koristi za pozivanje modela i funkcije za predikciju. Model se poziva direkt iz zip datoteke što znači da programer može dobiti gotov model od drugih izvora ne mora ga ni sam generirati ako ima pristup modelu.



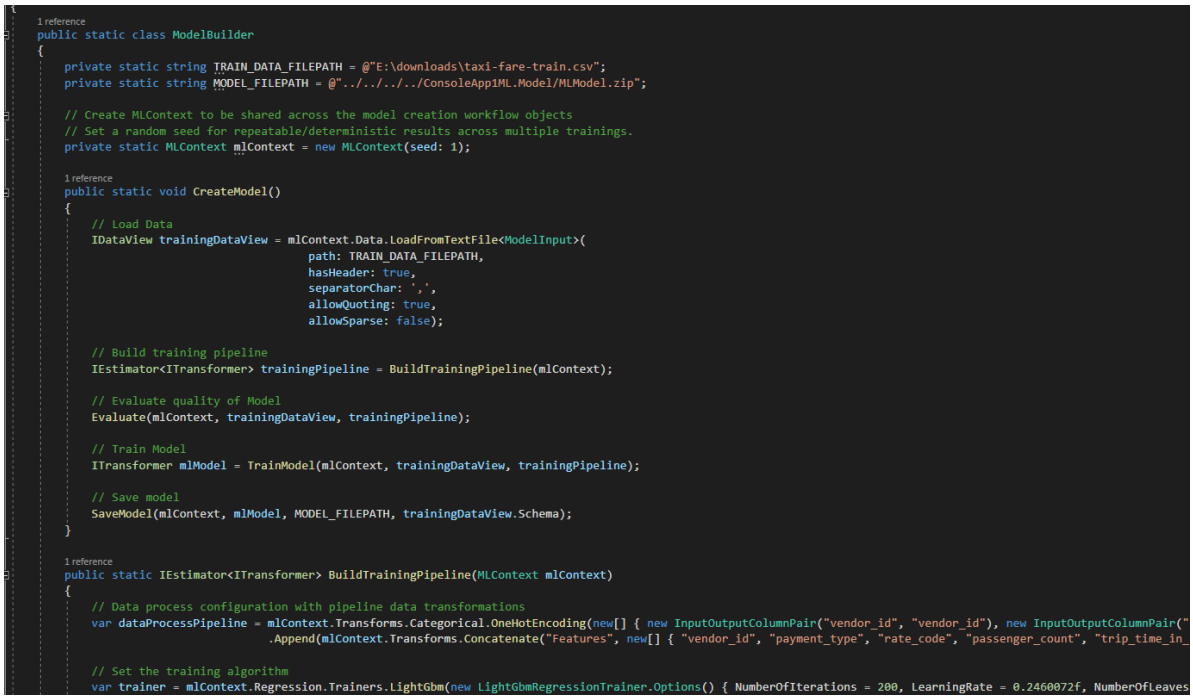
```
2 references
class Program
{
    //Machine Learning model to load and use for predictions
    private const string MODEL_FILEPATH = @"MLModel.zip";

    //Dataset to use for predictions
    private const string DATA_FILEPATH = @"E:\downloads\taxi-fare-train.csv";
    private const string TESTDATA_FILEPATH = @"E:\downloads\taxi-fare-test.csv";

    0 references
    static void Main(string[] args)
    {
        MLContext mlContext = new MLContext();

        // Training code used by ML.NET CLI and AutoML to generate the model
        ModelBuilder.CreateModel();

        ITransformer mlModel = mlContext.Model.Load(GetAbsolutePath(MODEL_FILEPATH), out DataViewSchema inputSchema);
        var predEngine = mlContext.Model.CreatePredictionEngine<ModelInput, ModelOutput>(mlModel);



            ModelOutput predictionResult = predEngine.Predict(taxiSample);
        ModelOutput predictionResult2 = predEngine.Predict(taxiSample2);
        ModelOutput predictionResult3 = predEngine.Predict(taxiSample3);
        ModelOutput predictionResult4 = predEngine.Predict(taxiSample4);

        Console.WriteLine($"Single Prediction --> Actual value: {taxiSample.Fare_amount}  | Predicted value: {predictionResult.Score}");
            Console.WriteLine($"Single Prediction --> Actual value: {taxiSample2.Fare_amount}  | Predicted value: {predictionResult2.Score}");
            Console.WriteLine($"Single Prediction --> Actual value: {taxiSample3.Fare_amount}  | Predicted value: {predictionResult3.Score}");
            Console.WriteLine($"Single Prediction --> Actual value: {taxiSample4.Fare_amount}  | Predicted value: {predictionResult4.Score}");

        Console.WriteLine("=============== End of process, hit any key to finish ==============");
        Console.ReadKey();
    }
```

Slika 16 Primjer poziva modela u aplikaciji

Iz navedenog koda vidimo da se poziv modela vrši kroz dvije naredbe Load i CreatePredictionEngine, gdje prvo učitavamo model te iz njega stvaramo engine koji vrši predikciju događaja. Nakon učitavanja engine-a i modela možemo pozivanjem naredbe Predict na PredictionEngine-u testirati naš model koristeći testne objekte koje smo stvorili u prijašnjem koraku. U ovom kodu još postoji poziv metode CreateModel() koja stvara model te ga sprema u .zip file, ova naredba nije potrebna ako je model preuzet ili već postoji.

Bitni parametri korišteni za algoritam LighGBM definirani unutar koda su NumberOfIterations broj iteracija stabla, Learning Rate, NumberOfLeaves broj listova i MinimumExampleCountPerLeaf najmanji broj primjera po listu.

Osim programske datoteke tijekom stvaranja modela generira se još jedna datoteka ModelBuilder.cs koja sadrži kod koji se koristi za proces pozivanja modela i metode kojima se model sprema, trenira i evaluira (slika 18).



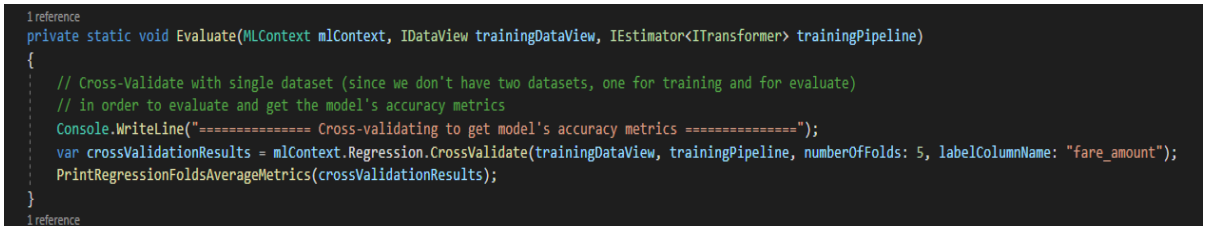
```
1 reference
public static class ModelBuilder
{
    private static string TRAIN_DATA_FILEPATH = @"E:\downloads\taxi-fare-train.csv";
    private static string MODEL_FILEPATH = @"../../../../ConsoleApp1ML.Model/MLModel.zip";

    // Create MLContext to be shared across the model creation workflow objects
    // Set a random seed for repeatable/deterministic results across multiple trainings.
    private static MLContext mlContext = new MLContext(seed: 1);

    1 reference
    public static void CreateModel()
    {
        // Load Data
        IDataView trainingDataView = mlContext.Data.LoadFromTextFile<ModelInput>(
                                        path: TRAIN_DATA_FILEPATH,
                                        hasHeader: true,
                                        separatorChar: ',',
                                        allowQuoting: true,
                                        allowSparse: false);

        // Build training pipeline
        IEstimator<ITransformer> trainingPipeline = BuildTrainingPipeline(mlContext);

        // Evaluate quality of Model
        Evaluate(mlContext, trainingDataView, trainingPipeline);

        // Train Model
        ITransformer mlModel = TrainModel(mlContext, trainingDataView, trainingPipeline);

        // Save model
        SaveModel(mlContext, mlModel, MODEL_FILEPATH, trainingDataView.Schema);
    }

    1 reference
    public static IEstimator<ITransformer> BuildTrainingPipeline(MLContext mlContext)
    {
        // Data process configuration with pipeline data transformations
        var dataProcessPipeline = mlContext.Transforms.Categorical.OneHotEncoding(new[] { new InputOutputColumnPair("vendor_id", "vendor_id"), new InputOutputColumnPair("
                                  .Append(mlContext.Transforms.Concatenate("Features", new[] { "vendor_id", "payment_type", "rate_code", "passenger_count", "trip_time_in_

        // Set the training algorithm
        var trainer = mlContext.Regression.Trainers.LightGbm(new LightGbmRegressionTrainer.Options() { NumberOfIterations = 200, LearningRate = 0.2460072f, NumberOfLeaves
```

Slika 17 Kod za kreiranje modela

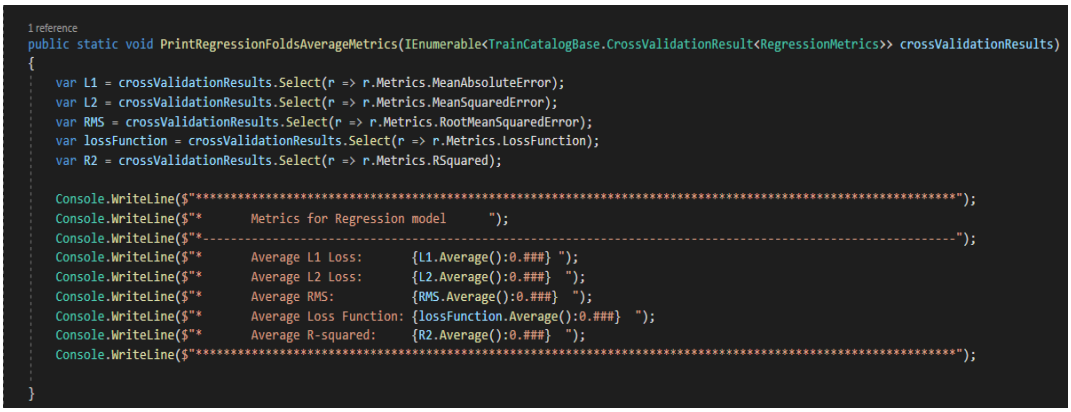
ModelBuilder.cs datoteka predstavlja kod koji stoji iza procesa Model Buildera ML.NET-a kao što je vidljivo na slici 17, sadrži metodu CreateModel() koja stvara model i sprema ga unutar .zip file-a. Unutar CreateModel() metode koriste se standardne metode za učitavanje podataka, stvaranje trening pipeline-a, evaluaciju i treniranje modela te spremanje istog. Ovo je glavna datoteka unutar projekta i sadrži glavne informacije o modelu i načinu treniranja.

Unutar ModelBuilder datoteke je prikazan i način na koji se dohvaćaju i ispisuju metrike za evaluaciju kvalitete modela kao što su Rsquared,MeanAbsoluteError i slične(slika 19.).

Cross Validation je tehnika koja dijeli podatke u nekoliko particija i trenira nekoliko algoritama na tim particijama. Ova tehnika poboljšava robusnost modela izdvajanjem podataka iz trening procesa.



```
1 reference
private static void Evaluate(MLContext mlContext, IDataView trainingDataView, IEstimator<ITransformer> trainingPipeline)
{
    // Cross-Validate with single dataset (since we don't have two datasets, one for training and for evaluate)
    // in order to evaluate and get the model's accuracy metrics
    Console.WriteLine("=============== Cross-validating to get model's accuracy metrics ==============");
    var crossValidationResults = mlContext.Regression.CrossValidate(trainingDataView, trainingPipeline, numberOfFolds: 5, labelColumnName: "fare_amount");
    PrintRegressionFoldsAverageMetrics(crossValidationResults);
}
1 reference
```



```
1 reference
public static void PrintRegressionFoldsAverageMetrics(IEnumerable<TrainCatalogBase.CrossValidationResult<RegressionMetrics>> crossValidationResults)
{
    var L1 = crossValidationResults.Select(r => r.Metrics.MeanAbsoluteError);
    var L2 = crossValidationResults.Select(r => r.Metrics.MeanSquaredError);
    var RMS = crossValidationResults.Select(r => r.Metrics.RootMeanSquaredError);
    var lossFunction = crossValidationResults.Select(r => r.Metrics.LossFunction);
    var R2 = crossValidationResults.Select(r => r.Metrics.RSquared);

    Console.WriteLine($"*************************************************************************************************************");
    Console.WriteLine($"*       Metrics for Regression model      ");
    Console.WriteLine($"*------------------------------------------------------------------------------------------------------------");
    Console.WriteLine($"*       Average L1 Loss:       {L1.Average():0.###} ");
    Console.WriteLine($"*       Average L2 Loss:       {L2.Average():0.###}  ");
    Console.WriteLine($"*       Average RMS:           {RMS.Average():0.###}  ");
    Console.WriteLine($"*       Average Loss Function: {lossFunction.Average():0.###}  ");
    Console.WriteLine($"*       Average R-squared:     {R2.Average():0.###}  ");
    Console.WriteLine($"*************************************************************************************************************");

}
```

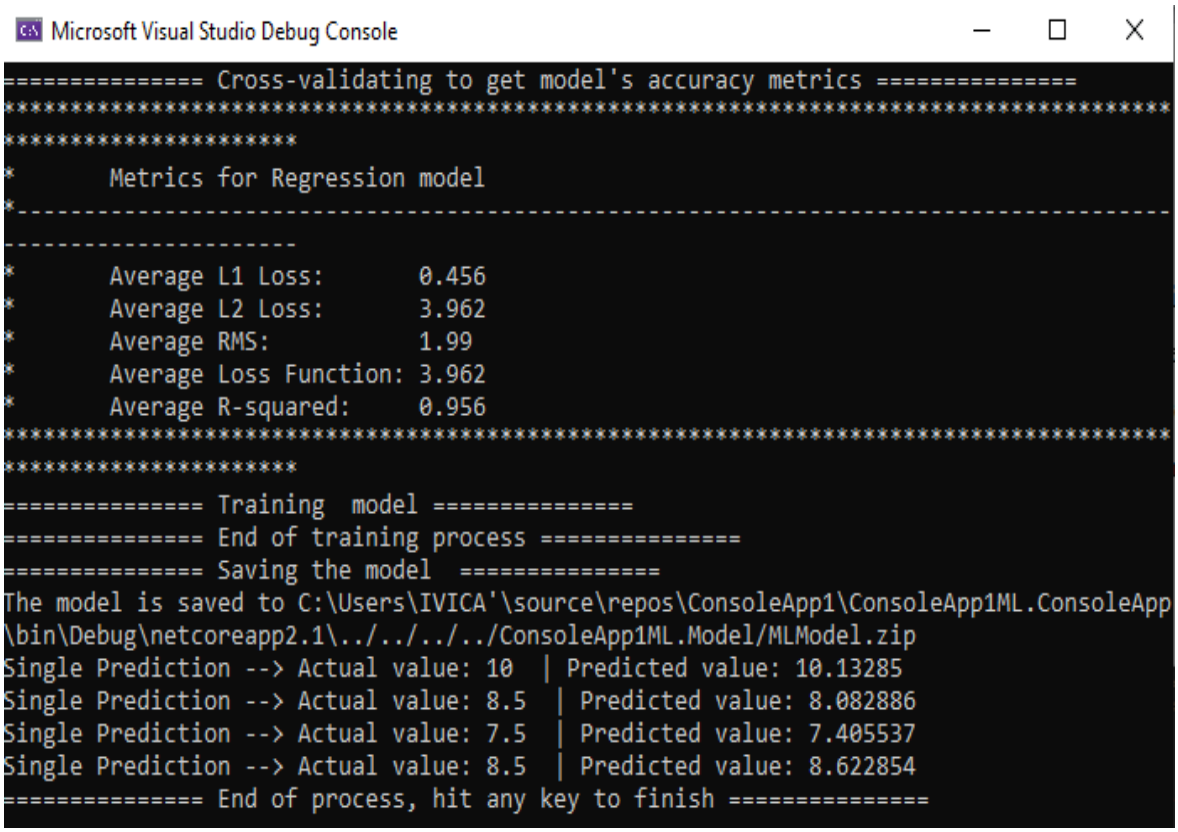
Slika 18 Ispis metrika za Regresijsku analizu

Osim metrike i nekih ostalih funkcija za stvaranje modela koje su implementirane kroz ML.NET framework unutar ModelBuildera moguće je na jednostavan način mijenjati parametre modela koji definiraju proces treniranja te završni model koji će se stvoriti.

3.3 Analiza rezultata

Kroz analizu rezultata potrebno je odrediti točnost modela i mogućnost njegove primjene na nepoznate podatke, treba obratiti pozornost na neke osnovne probleme koji se javljaju kod modela strojnog učenja kao i na moguća poboljšanja koda ili samog algoritma koji se koristi.

Već smo dobili RSquared postotak po kojem bi naš model trebao raditi preciznu predikciju i davati nam vrijednosti koje su približne onim stvarnima. Testirat ćemo model koristeći četiri testna objekta koje smo stvorili te analizirajući metrike koje su nam dohvatljive kroz evaluacijski proces modela, jako je bitno da se testni podaci ne koriste u procesu treniranja jer model može imati pristranost njima. Slika 20 sadrži rezultat pokretanja programa kojeg smo generirali i izmijenili. Metrike koje se koriste su tipične za regresijsku analizu i opisuju model na temelju preciznosti i točnosti rezultata koje daje.



Slika 19 Testni program modela za Regresijsku analizu

Na slici 19 prikazane su metrike definirane kroz ModelBuilder klasu i rezultati testnih predikcija koji su definirani u aplikacijskom dijelu projekta koji koristi zadani model. Metrike koje se koriste za regresijske modele su Mean Absolute Error, Mean Squared Error,Root Mean Squared Error i R Squared. Ove metrike su definirane kao izlazne podaci CreateModel metode za stvaranje modela.

L1 ili Mean Absolute error je mjera razlike između dvije stalne varijable. Ako su X i Y dvije varijable koje predstavljaju iste promatrane objekte, apsolutna pogreška između ta dvije varijable je :

$$|e_i| = |y_i - x_i|$$

A iz toga proizlazi da je Mean Absolute Error prosjek svih apsolutnih pogrešaka:

$$\boldsymbol{MAE} = \frac{\sum_{i=1}^{n} y_i - x_i}{n}$$

Dakle prosječna razlika između predviđene vrijednosti i one stvarne je 0.456 što znači da u prosjeku greška u procjeni cijene iznosi 45 centi što je za ovakav slučaj prihvatljiva vrijednost.

L2 loss funkcija je također poznata pod nazivom Least Squares Error. Ona umanjuje sumu apsolutnih razlika između tražene vrijednosti Y_i i predviđene vrijednosti ($f(x_i)$):

$$S = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

S obzirom na to da L2 kvadrira pogrešku (povećavajući se eksponencijalno kada je pogreška > 1) prikazuje poprilično veću pogrešku od Mean Absolute Error metrike, te je model osjetljiviji na ovu metriku i podešava model da se smanji njegova pogreška.

RSquared je statistička mjera koja predstavlja količinu proporcije varijance za ovisni model koji je objašnjen samostalnom varijablom ili varijablama u regresijskom modelu. RSquared opisuje koliko varijanca jedne varijable opisuje varijancu druge.

$$r = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2}}$$

Ako RSquared iznosi 0.50, onda otprilike pola promatranih varijance mogu biti objašnjeno kroz model. S rezultatom od 0.956 skoro sve varijance mogu biti objašnjene.

Promatrajući rezultate ovih predikcija vidimo da model radi poprilično dobro najveću razliku od 4 slučaja imamo na predikciji broj 2 i iznosi oko 40 centi što je prihvatljivo i dovoljno precizno da se ovaj model koristi i u nekoj stvarnoj aplikaciji kao izbor okvirne cijene vožnje taksi uslugom.

Način na koji se može modificirati kod i pokušati poboljšati rad modela je kroz mijenjanje parametara algoritma i dodatno treniranje modela ili korištenje potpuno drugog algoritma.
U cilju poboljšavanja točnosti eksperimenta i samog modela koristit će se različite parametarske vrijednosti za algoritam LightGBM, iako trenutni rezultati donose dobre rezultate, adaptiranjem algoritma možemo postići još veću točnost.
Glavni problem koji se može dogoditi kod adaptiranja algoritama i stvaranje modela je Over-fitting, Over-fitting se odnosi na sklonost modela da dobiva dobre rezultate za slučajeve koji su mu poznati dakle podatke iz trening data seta, ali kada se taj isti model

primjeni na nove, modelu nepoznate podatke dolazi do velikog pada u točnosti i ne iskoristivosti takvog modela. Modifikacije algoritma se vrše unutar metode BuildTrainingPipeline koja sadrži definiciju trenera tj. algoritma koji se koristi i njegovih parametara. Unutar navedenog primjera na slici 21 modificirat će se parametar NumberOfLeaves.

```
1 reference
public static IEstimator<ITransformer> BuildTrainingPipeline(MLContext mlContext)
{
    // Data process configuration with pipeline data transformations
    var dataProcessPipeline = mlContext.Transforms.Categorical.OneHotEncoding(new[] { new InputOutputColumnPair("vendor_id", "vendor_id"), new InputOutputColumnPair("payment_type", "payment_type") })
                              .Append(mlContext.Transforms.Concatenate("Features", new[] { "vendor_id", "payment_type", "rate_code", "passenger_count", "trip_time_in_secs", "trip_distance" }));

    // Set the training algorithm
    var trainer = mlContext.Regression.Trainers.LightGbm(new LightGbmRegressionTrainer.Options() { NumberOfIterations = 200, LearningRate = 0.460072f, NumberOfLeaves = 8, MinimumExampleCountPerLeaf = 10
    var trainingPipeline = dataProcessPipeline.Append(trainer);

    return trainingPipeline;
}
```

Slika 20 Modifikacija TraningPipeline-a

Number of leaves je glavni parametar koji kontrolira kompleksnost modela stabla odlučivanja. Postavljanje ovog parametra na preveliku vrijednost može dovesti do over-fittinga. Kao test koristit će se veći broj listova. Rezultati su prikazani na slici 22.

```
Microsoft Visual Studio Debug Console
=============== Cross-validating to get model's accuracy metrics ===============
*************************************************************************************************************
**
*       Metrics for Regression model
*------------------------------------------------------------------------------------------------------------
--
*       Average L1 Loss:       0.418
*       Average L2 Loss:       4.044
*       Average RMS:           2.01
*       Average Loss Function: 4.044
*       Average R-squared:     0.955
*************************************************************************************************************
**
=============== Training  model ===============
=============== End of training process ===============
=============== Saving the model  ===============
The model is saved to C:\Users\IVICA'\source\repos\ConsoleApp1\ConsoleApp1ML.ConsoleApp\bin\Debug\netcoreap
p2.1\../../../../ConsoleApp1ML.Model/MLModel.zip
Single Prediction --> Actual value: 10  | Predicted value: 9.87033
Single Prediction --> Actual value: 8.5  | Predicted value: 8.376078
Single Prediction --> Actual value: 7.5  | Predicted value: 7.433597
Single Prediction --> Actual value: 8.5  | Predicted value: 8.372921
=============== End of process, hit any key to finish ===============
```

Slika 21 Rezultati modela sa većim brojem listova

Broj listova smo sa šest povećali na osam što nam je rezultate promijenilo u pozitivnom smislu, Mean Absolute Error tj. L1 je uvelike smanjen sa 0.456 na 0.418 dok su ostale metrike lošije za zanemarive vrijednosti. Na testnim primjerima predviđene vrijednosti su bliže ili jednake onima iz prošlog primjera što pokazuje da veći broj listova povećava točnost rezultata na ovom primjeru.

Promjenom ostalih parametara rezultati ne postaju drastično drukčiji ili bolji već se model sve više približava problemu overfittinga sa svakom promjenom. Razlog tome je što je problem jednostavan u okviru strojnog učenja i kao takav lako se nad njim vrši regresija i dobivaju dobri rezultati.

Nakon promjene koda i nekih parametara unutar LightGBM algoritma rezultati su precizniji dok se sveukupni okvir modela nije smanjio, s obzirom na to da je RSquared 0.955 model je već u blizini svojih maksimalnih mogućnosti s obzirom na broj podataka i vrijeme korišteno za treniranje modela. Najbolji način povećanja preciznosti modela bi bio korištenje većeg data seta za treniranje te kontinuirano treniranje modela koristeći taj data set s kojim bi model stekao iskustvo koje bi rezultiralo u boljim rezultatima. Daljnje modificiranje pridonijelo bi over-fittingu i stvorilo loše rezultate primjene koda na nepoznate podatke i dalo ne realistične metrike za same trening podatke.

4. Zaključak

ML.NET iako u ranim stadijima svojeg razvoja pokazuje se kao koristan alat koji je vrlo lako koristiti i pojednostavljuje stvaranje modela strojnog učenja. Rezultati koji su postignuti koristeći ML.NET framework i njegove funkcije su se dokazale kao korisne posebno za ljude koji nemaju puno znanja i iskustva unutar strojnog učenja i stvaranja modela za rješavanje problema kao što su klasifikacije i regresijske analize.

Model koji je generiran bez previše ljudske interakcije dovoljno je dobar da koristi za prave aplikacije te predstavlja samo dio onoga što se može postići korištenjem ML.NET frameworka i strojnog učenja kao grane umjetne inteligencije.

Glavni razlozi zašto su rezultati takvi je dobar data set koji je dovoljan za stvaranje preciznog stabla odlučivanja te algoritmi koji su dostupni za stvaranje modela unutar samog frameworka.

Način na koji bi se metrike koje smo dobili unutar modela mogle poboljšati je povećanjem količine podataka za treniranjem ili ručnim mijenjanjem nekih parametara unutar LightGBM algoritma, ali u svrhu ovog rada to nije bilo potrebno jer su rezultati modela već upotrebljivi na pravim slučajevima.

5. Literatura

[1]H. Heidenreich, "What are the types of machine learning?", *Medium*, 2018. [Online]. Available: https://towardsdatascience.com/what-are-the-types-of-machine-learning-e2b9e5d1756f. [Accessed: 21- Sep- 2019].

[2]".NET Framework", *En.wikipedia.org*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/.NET_Framework. [Accessed: 21- Sep- 2019].

[3]*Uvod u C#*. Osijek: Elektrotehnički fakultet Osijek, 2019.

[4]"Introduction to the C# Language and the .NET Framework", *Mircrosoft*, 2019. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework. [Accessed: 21- Sep- 2019].

[5]"Features — LightGBM 2.2.4 documentation", *Lightgbm*, 2019. [Online]. Available: https://lightgbm.readthedocs.io/en/latest/Features.html. [Accessed: 21- Sep- 2019].

[6]"IBM Knowledge Center", *IBM*. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SS3RA7_15.0.0/com.ibm.spss.modeler.help/nodes_treebuilding.htm. [Accessed: 21- Sep- 2019].

[7]"Linearna regresija", *Wikipedia*. [Online]. Available: https://hr.wikipedia.org/wiki/Linearna_regresija. [Accessed: 21- Sep- 2019].

6. Prilozi

6.1 Popis slika

IZJAVA

Izjavljujem pod punom moralnom odgovornošću da sam seminarski / završni / diplomski rad izradio/la samostalno, isključivo znanjem stečenim na studijima Sveučilišta u Dubrovniku, služeći se navedenim izvorima podataka i uz stručno vodstvo mentora izv.dr.sc Mario Miličević, kome/kojoj se još jednom srdačno zahvaljujem.

Ivica Ćurčija