Verslag Distributed Databases

Dylan Cluyse, Laura Renders, Liam Goethals

14 december 2022

Inhoudsopgave

Inleiding

Tijdens het opleidingsonderdeel 'Distributed Databases' maakten wij kennis met Apache Spark. Spark biedt data-analyse gericht grootschalige gegevensverwerking. Deze technologie wordt aangeboden voor Java, Python, R en Scala. Voor dit opleidingsonderdeel werd Java gekozen als programmeertaal. Spark bevat enkele zijtakken dat zich richt op andere aspecten binnen dataverwerking, één daarvan is MLLib. MLLib is een pakket gericht op machine learning met Spark. Om meer kennis te vergaren kregen wij de groepsopdracht om via het platform Kaggle deel te nemen aan machine learning (ML) gerichte competities.

In dit verslag nemen wij u mee in de wereld van ML met gebruik van Spark. Wij willen u met volle plezier enkele concrete casussen tonen die gebruik maken van de verschillende regressie- en classificatiemethoden.

Allereerst willen wij de tijdsduratie van een taxi-rit in downtown New York berekenen met regressie. Vervolgens detecteren wij kredietkaartfraude met behulp van binaire classificatie. Als derde oefening willen wij op basis van tekstinhoud achterhalen of een Tweet gerelateerd is aan een ramp. Tot slot geven wij u onze bevindingen mee van het MLLib pakket. Hier leggen we de aanpak van Spark en SKLearn, een pakket dat we in het opleidingsonderdeel Machine Learning zagen, parallel tegenover elkaar.

1. Tijd van een taxi-verplaatsing voorspellen met regressie.

Probleemstelling

Als eerste opdracht deden wij mee aan de "New York City Taxi Trip Duration"competition. Dit als late inzending. Bij deze competitie krijgen wij twee datasets: een training- en een testset. Er wordt gevraagd om de totale tidjsduur van een taxirit te berekenen op basis van de gekregen data. Volgende kolommen werden gegeven: het ID van de verkoper, het aantal passagiers in een taxi, het longitude en latitude van de plaats waar de passagier(s) werden opgehaald, de longitude en latitude van de plaats waar de passagiers werden gedropt en de pick-up en drop-off tijd. De data bevat echter foute datatypes en outliers. Deze moet eerst worden weggefilterd.

Bron: https://www.kaggle.com/competitions/nyc-taxi-trip-duration

Gevolgde notebook: https://www.kaggle.com/code/ashishc17/from-exploration-to-linear-regression-prediction

Aanpak

Onze data is nog te ruw en daarom maken wij een clean-functie aan. De pick-up en drop-off datetime wordt meegegeven als datetime-object. Dit moeten we veranderen naar een bruikbaar formaat voor het regressiemodel. Eerst gaan wij de datetime kolommen opsplitsen in twee kolommen: 'hour' en 'day'. We willen numerieke waarden, dus we behouden enkel de dag van de week (bijvoorbeeld 1) en het uur van de dag (bijvoorbeeld 12). Vervolgens gaan wij de outliers uit onze dataset halen. Voor de outliers willen wij kijken naar de rijen die een veel te hoge longitude of latitude hebben. De outliers kunnen wij bepalen door te kijken naar rijen die buiten het bereik van (3 x standaardafwijking) vallen. Als laatst verwijderen we alle rijen die geen

passagiers meenmenen.

Vervolgens gaan wij twee kolommen toevoegen: distance en speed. In de dataset krijgen wij de coordinaten mee (longitude en latitude) van het vertrek- en aankomstpunt. Deze bieden weinig waarde aan ons model. Met de afgelegde afstand heeft het model meer grip om de afgelegde tijd te kunnen berekenen. Hiervoor gebruiken wij de haversine-formule zoals hieronder afgebeeld. Met deze formule berekenen we de kortst mogelijke afstand tussen twee punten op het oppervlak van een bolvormig object. De 'great-circle distance' berekenen we met behulp van een user-defined function. Hierbij geven we vier double-variabelene mee met een double als return-waarde. De vier input-variabelen zijn: de longitude van het vertrekpunt, de latitude van het vertrekpunt, de longitude van het aankomstpunt en de latitude van het aankomstpunt. Wiskundige tussenstappen, zoals de vierkantswortel of het omzetten naar een radiaan, doen we met de Math-library van Java.

Naast een kolom 'distance' voegen wij ook een kolom 'speed' toe. Deze zal de snelheid in miles per hour bijhouden. Hiervoor gebruiken wij opnieuw een user-defined-function met twee doubles al inputparameters en één double als outputparameter.

Na het maken van de kolommen tonen wij, in de terminal, de gemiddelde snelheid en gemiddelde afstand per dag en per huur. Vervolgens starten we met een pipeline te maken. Voor we de features assembleren naar één kolom moeten wij eerst de categorische waarde "store and forward flag"gaan omzetten naar een numerieke waarde. Een vlag met waarde 'true' zal dan waarde 1 krijgen en omgekeerd. Dit doen we met een StringIndexer object. Vervolgens gaan wij de features omzetten naar één feature-kolom. Dit doen we met een VectorAssembler-object. Hierbij geven we de vendor id, aantal passagiers, het uur, de dag, de vlag en de afstand mee als features. De features-kolom zal een kolom zijn met alle features in één vector. Het probleem is dat alle features verschillende numerieke waarden en bereiken heeft. Dit zal een effect hebben op de uitkomst van ons model. Wij moeten onze feature-kolom op een gelijke schaal gaan brengen. Dit doen we door te werken met een MinMaxScaler-object. Een alternatief voor deze opdracht was een StandardScaler. Als laatste object gaan we ons regressiemodel meegeven. Voor dit regressieprobleem hebben wij gekozen voor een lineair model, een randomforestmodel, een gradientboostmodel en een generalised lineair model.

Bij alle modellen, op het lineaire model na, moeten we de parameters finetunen. Als we dit niet doen zal ons model niet optimaal benut worden en tegenvallende resultaten teruggeven. Dit probleem pakken wij aan door te werken met een ParamGridBuilder. Hiermee kunnen wij meerdere waarden voor verschillende parameters meegeven. Iedere combinatie zal aan bod

komen. Hoe meer parameters, hoe langer de uitvoertijd van de applicatie.

Evaluatie en controle

Bij dit regressieprobleem voeren wij twee algemene testen uit: de correlatiematrix en de vier metrieken. Als tussentijdse controle toonden wij de inhoud van de dataframe. Dit om zeker te zijn dat onze transformaties of clean-opdrachten juist werden uitgevoerd. Deze metrieken zijn de Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE) en de correlatiewaarde (R^2).

2. Credit Card fraude achterhalen met binaire classificatie.

Bron: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

Gevolgde notebook: https://www.kaggle.com/code/janiobachmann/credit-fraud-dealing-with-imbalanced-datasets

Probleemstelling

Deze dataset werd eerder voor een andere competitie gebruikt, maar wij konden geen toegang verkrijgen tot deze competitie. Deze dataset bestaat uit ongeveer 25.000 rijen. We hebben hier twee klassen: niet-fraudulente en fraudulente transacties. We pakken dit aan met een binair classificatiemodel. Bij het bestuderen van de dataset merken we op dat 99.83 percent van alle transacties niet fraudulent zijn, terwijl enkel 0.71 percent wél fraudulent is.

Aanpak

De oorspronkelijke dataset is skewed. Ons model zal een grotere kans hebben op overfitten aangezien het er van uit zal gaan dat de meeste transacties niet fraudulent zal zijn. Daarom gaan we een 'sub-sample' aanmaken. Bij deze kleinere dataset richten we ons op een 50:50 ratio.

Alle nodige features starten met een 'V'-teken. Met behulp van een for-lus en de 'startswith' methode kunnen wij de features zo aan een array van Strings toevoegen. Dit omdat we anders alle 28 features manueel moeten ingeven. De array geven we mee aan de assembler. Die zal de waarden in de 28 kolommen omzetten naar één vector van features.

We hebben hier 28 features dat sterk op elkaar gelijken. Om te experimenteren maken wij ook in onze pipeline gebruik van PCA om dimensionality

Tabel 1: Confusion matrix: Random Forest

	0	**1**
0	100	3
1	5	98

Tabel 2: Confusion matrix: Lineaire SVM

	0	**1**
0	100	3
1	5	98

reduction op onze features uit te voeren. Zo behouden wij enkel de essentiële features.

Voor de binaire classificatie keken wij naar drie verschillende classificatiemodellen: Random Forest, Lineaire SVM en logistische regressie. Het tunen van de parameters doen we, net zoals de vorige oefening, met een ParamGridBuilder.

Evaluatie

Bij het evalueren van het classificatiemodel ging onze voorkeur uit naar de confusionmatrix. Zo hebben wij een zicht op hoe goed ons model de klassen kan voorspellen. Uit de confusion matrix kunnen wij ook de precision en recall berekenen. Deze functies zijn ingebouwd in Spark en dit hoeven wij niet manueel te berekenen. Als extra berekenen wij ook de nauwkeurigheid van het model. Wij merken op dat zowel de Random Forest als de Lineaire SVM even goed scoort. Het logistische regressiemodel daarentegen komt net te kort.

Tabel 3: Confusion matrix: Logistische regressie

	0	**1**
0	99	4
1	6	97

3. De inhoud van tweets detecteren met binaire NLP-classificatie.

Bron: https://www.kaggle.com/c/nlp-getting-started

Probleemstelling

Voor deze opdracht moeten we, op basis van gegeven Engelstalige Tweets, achterhalen of een Tweet gaat over een ramp of niet. De dubbelzinnigheid is een grote horde, want zo kan iemand zeggen 'look at the sky it was ablaze' terwijl het woord 'ablaze' een andere context heeft. In deze zin is het metaforisch.

Aanpak

We krijgen rauwe tekstdata binnen. Deze bevat niet-alfanumerieke karakters, overbodige spaties en stopwoorden. Deze woorden willen wij liefst mijden in ons model. Om de tekstdata om te zetten naar bruikbare data voor ons model maken wij gebruik van een pipeline. Het classificatiemodel voeren wij buiten de pipeline uit.

Allereerst gaan we enkel de alfanumerieke karakters behouden. Dit doen we met een RegexTokenizer-object. Het patroon W geeft aan dat we enkel alfanumerieke karakters willen. Vervolgens willen we de stopwoorden verwijderen. Dit doen we met een StopWordsRemover. Als de tweets in een andere taal waren, dan hadden wij eerst de stopwoorden uit die taal moeten opladen en vervolgens moeten toekennen aan het model. Dit hoeven wij niet te doen aangezien alle tweets Engelstalig zijn. Als laatste deel van de NLP-pipeline moeten wij nog de gefilterde woorden gaan omzetten naar features. Dit doen we met een CountVectorizer-object.

TODO

Evaluatie

Net zoals daarnet gaat onze voorkeur hier uit naar de confusionmatrix. Zo hebben wij een zicht op hoe goed ons model de klassen kan voorspellen. Uit de confusion matrix kunnen wij ook de precision en recall berekenen. Deze functies zijn ingebouwd in Spark en dit hoeven wij niet manueel te berekenen. Als extra berekenen wij ook de nauwkeurigheid van het model. Wij merken op dat zowel de Random Forest als de Lineaire SVM even goed scoort. Het logistische regressiemodel daarentegen komt net te kort.

4. Bevindingen ML met Spark.

Sklearn

TODO

Conclusie

TODO