BDSE Machine Learning

Sentiment analyse met behulp van positieve en negatieve recensies

Quirijn Langedijk
500782499
quirijn.langedijk@hva.nl
email@quirijnlangedijk.nl

Inhoud

1 Samenvatting

In dit verslag worden drie verschillende machine learning algoritmen beschreven die ik gebruikt heb om sentiment analyse uit te voeren op zelfgemaakte reviews.

Als eerst train ik Naive Bayes, deze een gemiddelde accuracy van 88.69%.

Vervolgens train ik een Support Vector Machine met een accuracy van 93.60%.

Als laatste train ik een Logistic Regression model, met een accuracy van 94.06%.

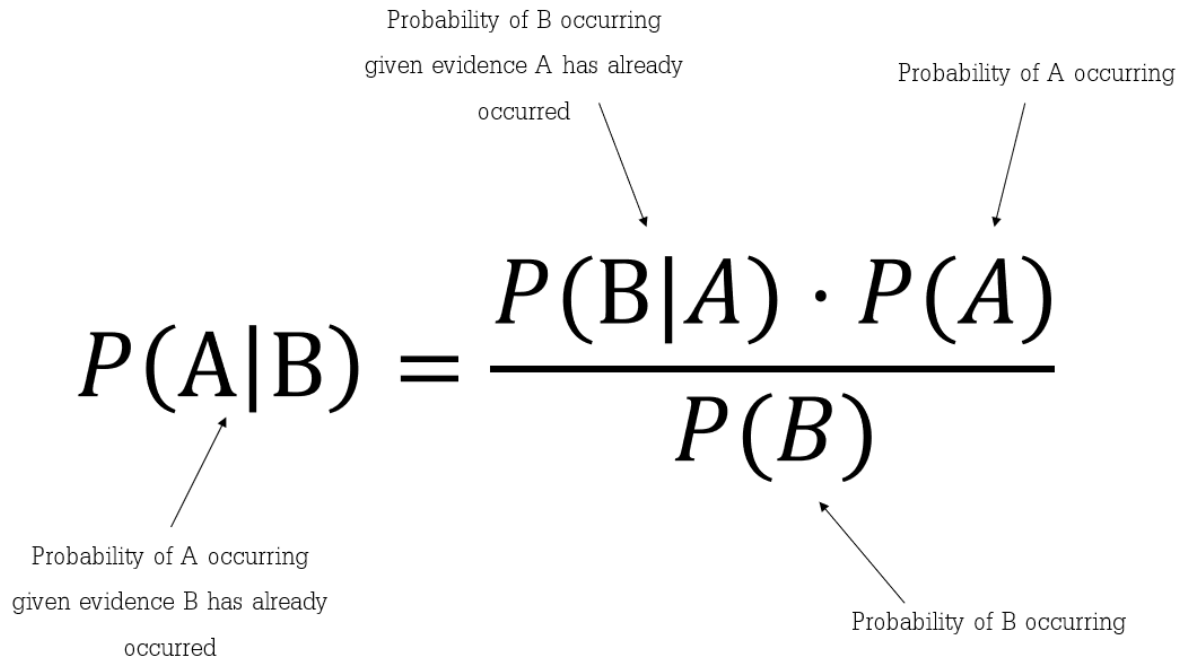
Omdat Logistic Regression en Support Vector Machine bijna dezelfde accuracy hebben, maar Logistic Regression een stuk sneller is om te trainen, vind ik Logistic Regression de beste van de drie models voor sentiment analyse.

2 Achtergrondinformatie

In dit hoofdstuk wordt de techniek achter de drie gebruikte machine learning algoritmen uitgelegd. Ook worden er een aantal positieve en negatieve punten per algoritme gegeven.

2.1 Naive Bayes

Met Naive Bayes kan berekend worden wat de kans is dat gebeurtenis A optreed onder de omstandigheden B.



Figuur 1 Theorema van Bayes

De formule in Figuur 1 (Malhotra, 2017) kan worden toegepast in sentiment analyse, doordat de kans berekend kan worden dat een bepaald woord 'negatief' of 'positief' is. Door dit model genoeg te trainen is het daarna mogelijk om voor individuele gevallen te berekenen wat de kans is dat het woord positief of negatief is. Hiermee kunnen zinnen dus geclassificeerd worden als positief of negatief.

Pros:

- Snel om te trainen
- Snel met predictions
- Makkelijk te implementeren
- Gemakkelijk te trainen, ook met een kleine dataset

Cons:

- Er mogen geen verbanden zijn tussen de categorieën in de data

2.2 Support Vector Machine

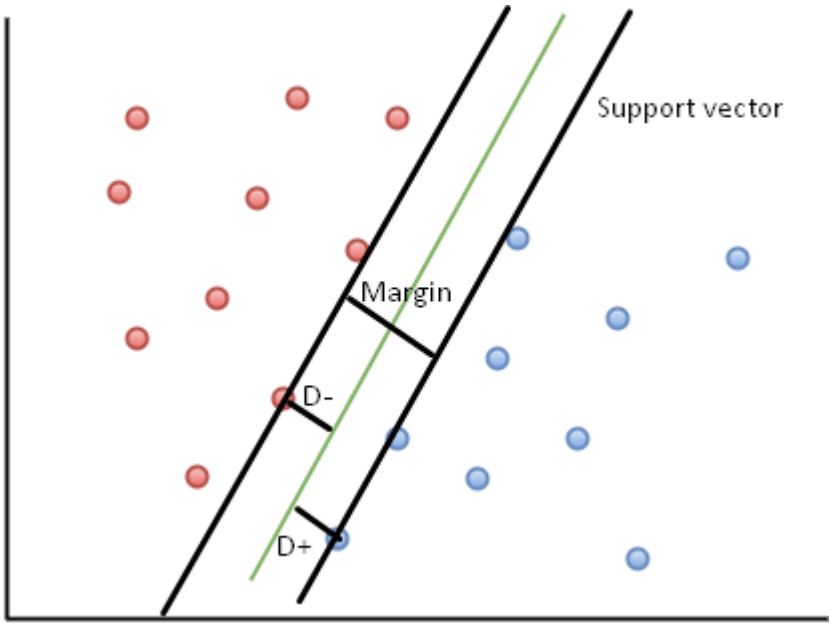
Een Support Vector Machine is een algoritme dat kenmerken indeelt in twee verschillende klassen. Voordat een kenmerk ingedeeld kan worden, moet er eerst een vector van gemaakt worden. Dit kan gedaan worden met bijvoorbeeld een tf-idf vectorizer. Deze vectorizer geeft een woord een weging aan de hand van hoe vaak het woord gebruikt wordt in een document vergeleken met andere documenten. Als een woord vaak gebruikt wordt in een document, maar niet vaak in andere documenten, zegt dat woord vaak wat over de betekenis van het document.

Als het woord 'slecht' 5 keer voorkomt in een review van 100 woorden, is de tf (5/100) = 0.05. Als we dit vergelijken met 10 miljoen andere documenten, waar het woord 'slecht' 10.000 keer in voor komt, is de idf log(10.000.000 / 10.000) = 3. De weging van dit woord is dan (tf * idf) = (0.05*3) = 0.15.

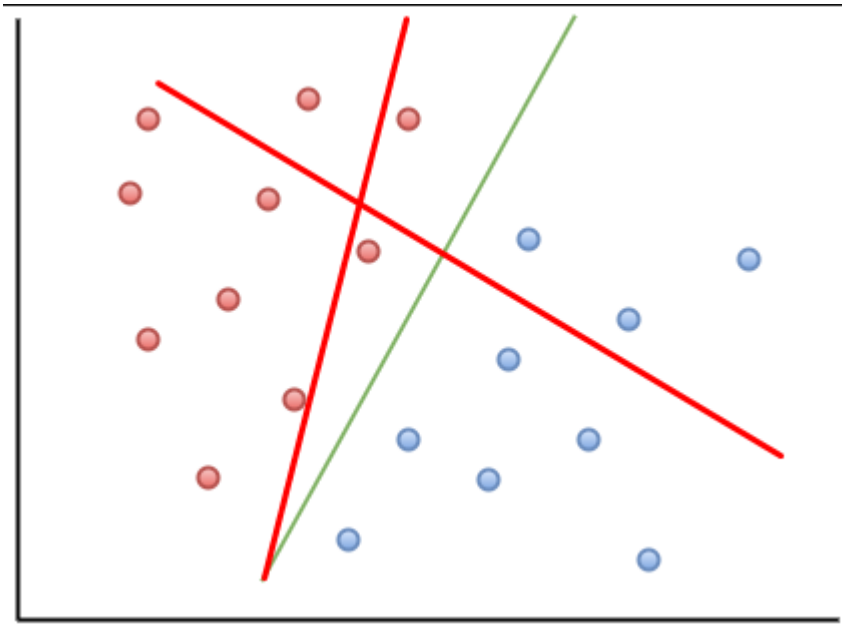
Als we dit vergelijken met het woord 'de', wat 10 keer voorkomt in de review van 100 woorden, is de tf (10/100) = 0.1. Het woord komt 1.000.000 keer voor in de 10 miljoen documenten, dus de idf is log(10.000.000 / 1.000.000) = 1. De tf-idf is dus 0.1 * 1 = 0.1.

In dit document zegt 'slecht' dus meer over de inhoud dan 'de'.

Nadat de woorden gevectorized zijn, kunnen ze ingedeeld worden in een 'grafiek'. Vervolgens zoekt het algoritme de lijn waar de afstand tussen het dichtstbijzijnde 'positieve' woord en het dichstbijzijnde 'negatieve' woorden het verst zijn. D- + D+ moet dus het hoogst zijn.



In het onderstaande voorbeeld is de groene lijn de beste 'hyperplane' (Totta datalab, 2018). De rode lijnen zijn duidelijk niet de best mogelijke lijnen, omdat de D- en D+ daar niet gelijk zijn. Wanneer er een verkeerde hyperplane gekozen wordt, zorgt dit voor veel miskwalificaties.



Pros:

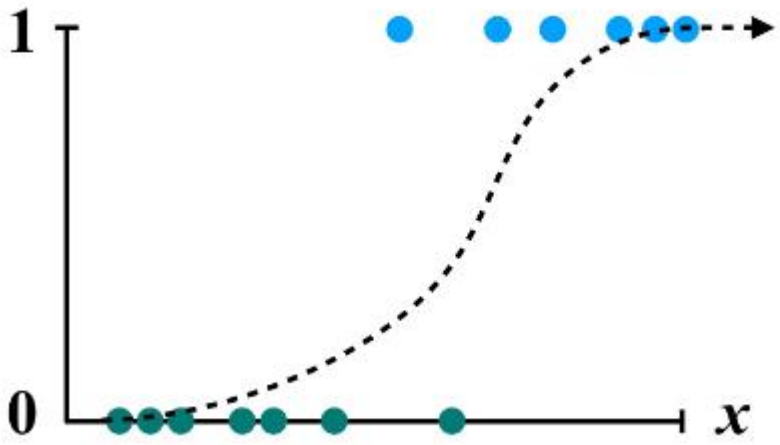
- Werkt goed als er een duidelijke scheiding is tussen data
- Erg precies
- Goed te gebruiken bij classificatie (image recognition, digit recognition, detecting spam, sentiment analysis etc)

Cons:

- Werkt slecht als de data door elkaar loopt.
- Niet goed te gebruiken voor grote datasets doordat trainen veel tijd kost.
- Om een een model goed te trainen heb je veel geheugen nodig, doordat alle support vectors opgeslagen moeten worden.

2.3 Logistic Regression

Met Logistische Regressie kunnen twee uitkomstvariabelen (bijvoorbeeld positief en negatief) gekoppelt worden aan één of meer andere variabelen, in het geval van sentiment analyse de vectoren van woorden. Het algoritme doet dit door het logaritme van een kansberekening te nemen, zodat deze tussen 0 en 1 komt. Nadat alle kansberekeningen met elkaar vermenigvuldigd zijn, krijg je de onderstaande lijn. Vervolgens kan je bij het testen van nieuwe data de kans berekenen, om vervolgens te kijken of deze kans boven of onder de lijn komt, hiermee kan je een zin classificeren als positief/negatief. (Starmer, 2018)



Pros:

- Efficient.

Cons:

- Werkt niet goed wanneer er veel verschillende categorieën data gebruikt worden.
- Overfitten van het model is mogelijk.

3 Methoden

In de volgende paragrafen wordt de code die ik heb gebruikt om de 3 modellen te trainen, testen en nieuwe predictions te maken. Ook wordt uitgelegd hoe ik aan de data ben gekomen, en wat ik doe om de data te cleanen.

3.1 Webscrapen

Ik heb ongeveer 1100 reviews gewebscraped van twee verschillende websites, met twee verschillende technieken.
Als eerst heb ik BeautifulSoup gebruikt om ongeveer 1000 reviews van TripAdvisor te scrapen. Hoewel de reviews op verschillende pagina's staan kan ik, doordat het paginanummer in de url staat, toch met BeautifulSoup alle reviews ophalen. Nadat ik de pagina opgehaald heb, zoek ik met een regex naar de divs/q's waar de ratings en reviews in staan.
Ik gebruik het gegeven cijfer dat de hotelgast als beoordeling heeft gegeven om de review in te delen als positief of negatief.

Vervolgens gebruik ik selenium om Trivago te openen. Hierna moeten er een aantal clicks uitgevoerd worden om de reviews ingeladen te krijgen. Als dit eenmaal is gebeurd kan ik net als bij TripAdvisor BeautifulSoup gebruiken om de gewenste spans/p's op te halen, en in te delen als positief/negatief.

```
with closing(
        Chrome(executable_path='../../Downloads/chromedriver.exe')) as driver:

driver.get("https://www.trivago.co.uk/?cpt2=47362%2F100&sharedcid=47362&tab=rating"
)

    # Sleep to load pages.
    button = driver.find_elements_by_class_name("tabs__label")
    button[3].click()
    see_more = driver.find_elements_by_class_name("td-underline--hover")[1].click()
    page_source = driver.page_source

    soup = BeautifulSoup(page_source, 'html.parser')
    # Find all scores and reviews.
    reviews = soup.find_all('p', {'class': 'sl-review__summary'})
    scores = soup.find_all('span', {'class': 'item-components__pillValue--4748f
item-components__value-med--a26b7 item-components__pillValue--4748f'})
    clean_reviews = [x for x in reviews if x.text[len(x.text)-4:len(x.text)] ==
"...." or not x.text[len(x.text)-3:len(x.text)] == '...']

    # Loop over reviews, discard reviews with no score (/).
    for k in range(len(clean_reviews)):
        if scores[k].text != "/":
            if float(scores[k].text) > 5.5:
                positive_list.append(str(reviews[k].text))
            elif float(scores[k].text) < 5.5:
                negative_list.append(str(reviews[k].text))
```

De lists die hieruit onstaan voeg ik toe aan een dataframe en upload ik naar de database.

3.2 Uploaden naar database

Voordat ik de data upload naar de database, clean ik de dataframes. Dit doe ik door de kolommen die ik niet gebruik te droppen zodat ik geen nutteloze data opsla. Vervolgens check ik met een regex of de reviews 'no negative', 'nothing', 'none' of 'null' zijn, om te zorgen dat 'lege' reviews het model niet beinvloeden. Ook zorgt de regex ervoor dat reviews als 'all positive' in de Negative_Review kolom eruit gehaald worden.

```
REGEX =
"^(((\d|\w)*?\s?nothing\s?(\d|\w)*\s?)|none|null|((\d|\w)+\s?(negative|posi
tive)))$"

df = df.drop(df[(df['Negative_Review'].str.contains(REGEX)) |
                (df['Positive_Review'].str.contains(REGEX))
                ].index)
df.reset_index(drop=True, inplace=True)
df.dropna(inplace=True)
return df
```

Nadat het dataframe gecleaned is, upload ik het naar de database met df.to_sql

3.3 Ophalen uit database

Om de dataframes weer op te halen uit de database gebruik ik een drietal stored procedures, één voor elke data source:

```
CREATE DEFINER=`u49825p47019_bdse`@`%` PROCEDURE `GetWebScrapedSet`()
BEGIN
    SELECT Negative_Review, Positive_Review FROM WebScraped;
END
```

Deze kan ik dan met behulp van pd.read_sql_query als dataframe uit de database halen. Doordat elke table in de database er hetzelfde uit ziet, kan ik de datasets aan elkaar koppelen met behulp van de .append() methode, hierdoor kan ik trainen met een dataset uit alle drie de data sources.

3.4 Naive Bayes

Het eerste model wat ik getrained heb is Naive Bayes (met NLTK). Nadat de data opgehaald/gecleaned is wordt er met behulp van nltk.word_tokenize getokenized. Deze tokens worden samen met een indicatie van positieve/negative review gesplit in een train/test set. Ik heb ervoor gekozen om de train/test in een verhouding van 80/20 te verdelen, omdat dit met deze dataset de hoogste accuracy had.

```
for i in range(df.shape[0]):
    # To lowercase and remove punctuation in order to improve accuracy
    neg.append([format_sentence(df.at[i,
'Negative_Review'].lower().translate(None, string.punctuation)),
'negative'])
    pos.append([format_sentence(df.at[i,
'Positive_Review'].lower().translate(None, string.punctuation)),
'positive'])
```

Hierna maak ik een classifier door de train set te trainen. Deze classifier wordt vervolgens getest door over de testset heen te loopen.
Hierna kunnen we een confusion matrix en accuracy laten printen.

```
classifier = NaiveBayesClassifier.train(X)

prediction_result = []
actual_result = []

for i in range(len(y)):
    prediction_result.append(classifier.classify(y[i][0]))
    actual_result.append(y[i][1])

classifier.show_most_informative_features(n=50)
print('\nAccuracy:\n', accuracy(classifier, y))
print('\nConfusion matrix:\n', confusion_matrix(actual_result, prediction_result))
```

Om de accuracy te verhogen heb ik een aantal dingen geprobeerd, zo heb ik bijvoorbeeld contractions (can't -> can not, won't -> will not) uit woorden geprobeerd te halen, hiermee ging de accuracy van mijn model omhoog met 0.8%.

3.5 Support Vector Machine

Als eerst verdeel ik de dataframe in een test/train set. Ook maak ik twee andere lists, die bijhouden of de review positief of negatief is.
Vervolgens maak ik een tf-idf vectorizer met behulp van de sklearn library. De tf-idf houdt bij welke woorden het meest belangrijk zijn. Om te zorgen dat 'nutteloze' woorden niet in de vectorizer terecht komen, zorg ik dat woorden die in meer dan 80% van de reviews voorkomen, of minder dan 10 keer voorkomen niet in de vectorizer terecht komen. Ook staat sublinear_tf op True, zodat woorden die heel vaak gebruikt worden minder belangrijk zijn.

Om de accuracy te verhogen heb ik geprobeerd stopwoorden weg te halen door de parameter stop_words toe te passen en door de vectorizer een bigram te laten maken. Helaas hielp geen van beide settings met het krijgen van een hogere accuracy.

Nadat ik train en test text gefit hebben in de vectorizer, train ik de Support Vector Machine met de train vectors.

```
training_text, training_labels, test_text, test_labels =
utils.divide_train_test(df, .8)

vectorizer = TfidfVectorizer(min_df=5,
                             max_df=0.8,
                             sublinear_tf=True,
                             use_idf=True)

train_vectors = vectorizer.fit_transform(training_text)
test_vectors = vectorizer.transform(test_text)

classifier_linear = svm.SVC(kernel='linear')
classifier_linear.fit(train_vectors, training_labels)
prediction_linear = classifier_linear.predict(test_vectors)
```

Vervolgens kan ik met de getrainde Support Vector Machine de test vectors testen.

3.6 Logistic Regression

Mijn aanpak voor Logistic Regression is vrijwel hetzelfde als bij SVM. Na het cleanen van mijn reviews verdeel ik ze in train/test en maak ik een tf-idf vectorizer. Na verschillende min_df/max_df's uit te proberen kwam ik op 25 en 0.8 uit.
Ook hier heb ik geprobeerd stop_words op english te zetten, en een bigram te gebruiken, maar hierdoor ging de accuracy van het model met 1.5% omlaag.

Ook hier fit en train ik mijn model met 80% van mijn reviews

```
training_text, training_labels, test_text, test_labels =
utils.divide_train_test(df, TRAIN_SIZE)

vectorizer = TfidfVectorizer(min_df=25,
                             max_df=0.8,
                             sublinear_tf=True,
                             use_idf=True)

train_vectors = vectorizer.fit_transform(training_text)
test_vectors = vectorizer.transform(test_text)

logmodel = LogisticRegression()
logmodel.fit(train_vectors, training_labels)
predictions = logmodel.predict(test_vectors)
```

4 Resultaten

TN	FP
FN	TP

Naïve Bayes

Amount of Reviews	100k		200k		300k	
	19298	702	32462	1720	49094	2830
	3661	16339	7538	38280	10906	57170
Accuracy	89.09%		88.43%		88.55%	

Support Vector Machine

Amount of Reviews	100k		200k		300k	
	18905	1095	37415	2585	66723	3683
	1576	18424	3168	36832	5048	65358
Accuracy	93.32%		92.80%		94.67%	

Logistic Regression

Amount of Reviews	100k		200k		300k	
	19190	810	37973	2027	57175	3178
	1422	18578	2802	37198	4290	56063
Accuracy	94.42%		93.96%		93.81%	

Testzinnen:
Om te testen of mijn modellen werken heb ik 6 testzinnen gebruikt, waarvan 3 positief en 3 negatief

Classified As	Model	Sentence	Correct?
pos	Support Vector Machine	This hotel is really good	✓
pos	Naive Bayes	This hotel is really good	✓
pos	Logistic Regression	This hotel is really good	✓
---	---	---	---
neg	Support Vector Machine	The shower was broken and we werent compensated for it	✓
neg	Naive Bayes	The shower was broken and we werent compensated for it	✓
neg	Logistic Regression	The shower was broken and we werent compensated for it	✓
---	---	---	---
neg	Support Vector Machine	I want my money back	✓
neg	Naive Bayes	I want my money back	✓
neg	Logistic Regression	I want my money back	✓
---	---	---	---
pos	Support Vector Machine	This hotel was great! I would recommend staying here	✓
pos	Naive Bayes	This hotel was great! I would recommend staying here	✓
pos	Logistic Regression	This hotel was great! I would recommend staying here	✓
---	---	---	---
neg	Support Vector Machine	I didn't like this hotel at all, Im not coming back	✓
neg	Naive Bayes	I didn't like this hotel at all, Im not coming back	✓
neg	Logistic Regression	I didn't like this hotel at all, Im not coming back	✓
---	---	---	---
pos	Support Vector Machine	Great hotel	✓
pos	Naive Bayes	Great hotel	✓
pos	Logistic Regression	Great hotel	✓

5 Conclusie

Er zijn twee modellen die een redelijk hoge accuracy hebben, Support Vector Machine en Logistic Regression. Het verschil tussen de SVM met 300.000 regels (600.000 reviews) en Logistic Regression met 100.000 regels (200.000 reviews) is vrij klein.

Hierom denk ik dat het voor deze dataset het beste is om logistic regression te gebruiken met 100.000 rows, omdat de accuracy hoog is, en de training time laag.

6 Bibliografie

Malhotra, A. (2017, Juli 3). *Naive Bayes Theorem.* Opgehaald van Medium: https://medium.com/@akankshamalhotra24/naive-bayes-theorem-79832d506a63

Starmer, J. (2018, maart 5). *StatQuest: Logistic Regression*. Opgehaald van Youtube: https://youtu.be/yIYKR4sgzI8

Totta datalab. (2018, november 28). *Wat zijn support vector machines en hoe werken ze?* Opgehaald van Totta datalab: https://www.tottadatalab.nl/2018/11/28/support-vector-machines/