BDSE Machine Learning

Quirijn Langedijk

500782499

quirijn.langedijk@hva.nl

[email@quirijnlangedijk.nl](mailto:email@quirijnlangedijk.nl)

Inhoud

[Samenvatting 2](#_Toc23022834)

[1 Inleiding 2](#_Toc23022835)

[2 Achtergrondinformatie 2](#_Toc23022836)

[2.1 Naive Bayes 2](#_Toc23022837)

[2.2 Support Vector Machine 3](#_Toc23022838)

[2.3 Logistic Regression 3](#_Toc23022839)

[3 Methoden 3](#_Toc23022840)

[3.1 Webscrapen 3](#_Toc23022841)

[3.2 Naive Bayes 3](#_Toc23022842)

[3.3 Support Vector Machine 4](#_Toc23022843)

[3.4 Logistic Regression 5](#_Toc23022844)

[3.5 Algemene ‘utils’ 5](#_Toc23022845)

[4 Resultaten 5](#_Toc23022846)

[5 Conclusie 6](#_Toc23022847)

[6 Bronvermelding 6](#_Toc23022848)

[7 Woordenlijst 6](#_Toc23022849)

# Samenvatting

# Inleiding

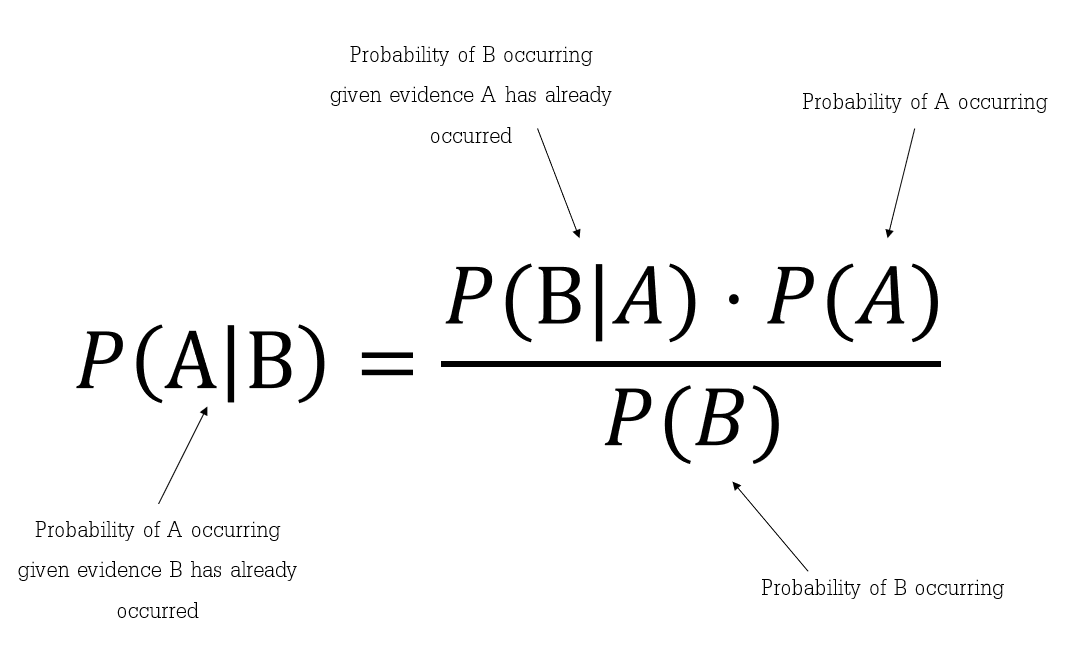
In dit verslag worden drie verschillende machine learning algoritmen beschreven die ik gebruikt heb om

# Achtergrondinformatie

In dit hoofdstuk worden de gebruikte machine learning algoritmen besproken.

Naive Bayes

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



Figuur 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

Support Vector Machine

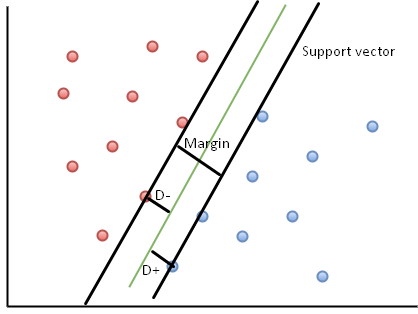
Een Support Vector Machine is een algoritme dat kenmerken indeelt in twee verschillende klassen. Voordat een kenmerk ingedeeld kan worden, moet er eerst een vector van gemaakt worden. Dit kan gedaan worden met bijvoorbeeld een tf-idf vectorizer. Deze vectorizer geeft een woord een weging aan de hand 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.

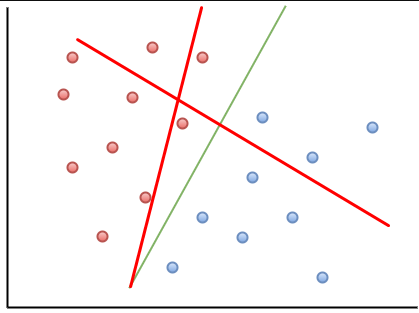
Als we dit vergelijken met het woord ‘de’, wat 10 keer voorkomt in de review van 100 woorden, is de tf (10/100) = 0.1. Het woord komt 1.000.000 keer voor in de 10 miljoen documenten, dus de idf is log(10.000.000 / 1.000.000) = 1. De tf-idf is dus 0.1 \* 1 = 0.1.

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:

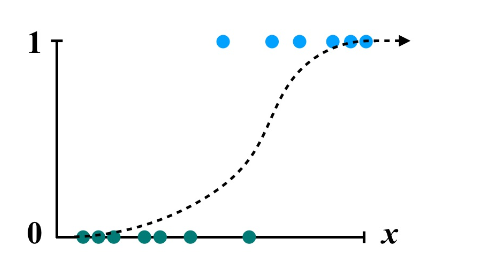
* Werkt goed als er een duidelijke scheiding is tussen data
* Erg precies
* Goed te gebruiken bij classificatie (image recognition, digit recognition, detecting spam, sentiment analysis etc)

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.

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



Pros:

* Efficient

Cons:

* Werkt niet goed wanneer er veel verschillende categorieën data gebruikt worden

# 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.

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.

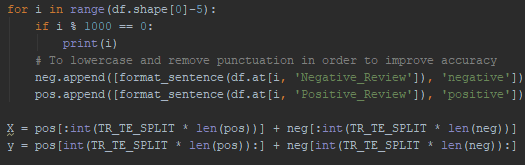
Vervolgens gebruik ik selenium om Trivago te openen. Hierna moeten er een aantal clicks uitgevoerd worden om de reviews ingeladen te krijgen. Als dit eenmaal is gebeurd kan ik net als bij TripAdvisor BeautifulSoup gebruiken om de gewenste spans/p’s op te halen, en in te delen als positief/negatief.

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.

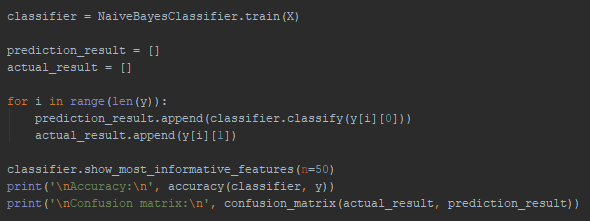
Naive Bayes

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.



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.



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

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. Verder haal ik met stop\_words engelse stopwoorden (a, the, this etc) weg. Ook staat sublinear\_tf op True, zodat woorden die heel vaak gebruikt worden minder belangrijk zijn.

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,  
 stop\_words="english",  
 ngram\_range=(1, 2))  
  
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.

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 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.

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)

Algemene ‘utils’

# 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% | |

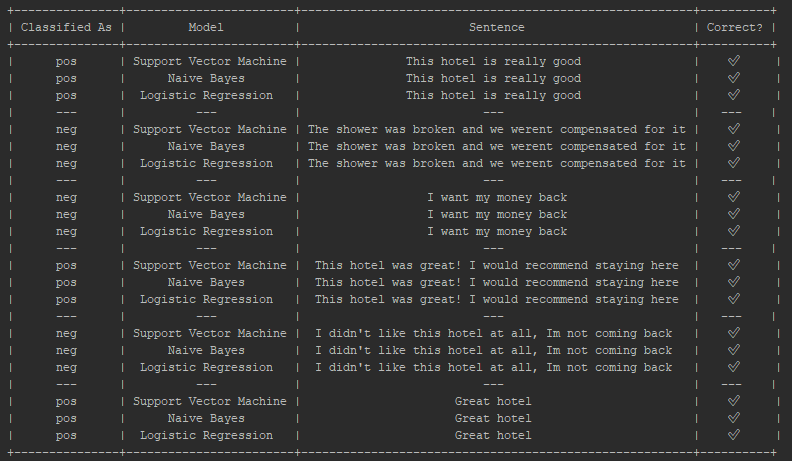
Support Vector Machine

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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:



# 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.

# Bibliografie

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

# Woordenlijst

|  |  |
| --- | --- |
| Tf-(idf) | term frequency–(inverse document frequency) |
|  |  |
|  |  |