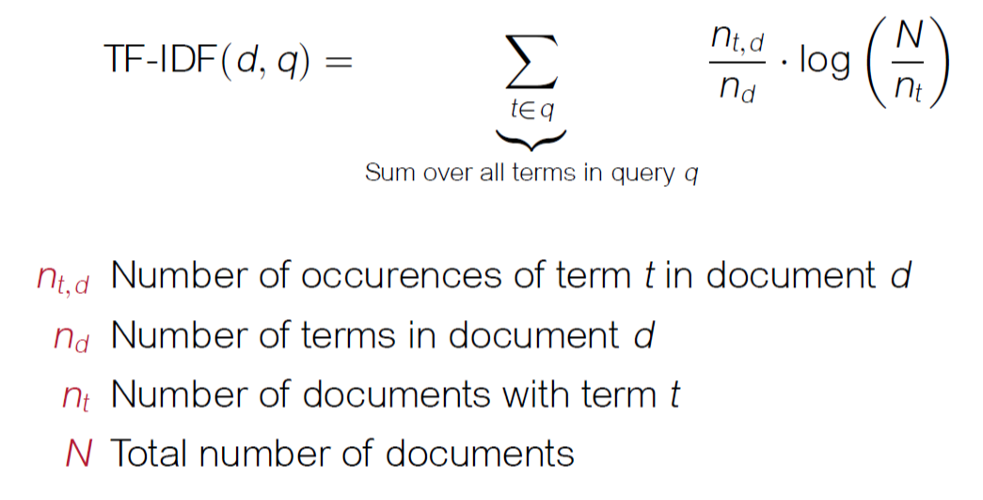
Guide til Naive Bayes (og lidt datatransformation)

**Datatransformation:**

Vi har valgt selv at lave datatransformation frem for at lade SciKit’s TfidfVectorizer klare alt arbejdet. Dette var bl.a. for at opnå en større forståelse for implementeringen af data transformaton ifb. natural language processing i Python.

TfidfVectorizer-delen har derfor primært følgende funktion:

1. Først tages træningsættets features ind (dette er alle vores emails), og der laves en dictionary over alle de ord der indgår (vocabulary) hvor hvert ord tildeles et index (fra 0 og op). Bemærk at dette intet med bag-of-words har at gøre.
2. Derefter beregnes der en TF-IDF-score for hvert dokument (d) i træningsættet ift. hvert ord i vocabulary som søgestreng (q). Formlen er givet ved:



Det beregnes dog ikke helt på samme måde i SciKit (dette vendes der tilbage til). Træningsættets korpus (det som der itereres igennem med alle dokumenterne) består af samtlige trænigsmails. Således vægtes hvert ord i vocabulary ift. hvor meget det indgår i dokumentet der fokuseres på ift. længden af dette dokument (TF), og hvert ord vægtes ift. hvor typisk et ord det er i de andre dokumenter (IDF). Resultaterne samles i en matrix:

|  |  |  |  |
| --- | --- | --- | --- |
| Dokument (observation) / Ord | 0 (f.eks. ”hey”) | 1 (f.eks. ”you”) | Osv. |
| 0 (email 1) | 0.35 | 0 | ---- |
| 1 (email 2) | 0.21 | 0.86 | ---- |
| Osv. | ---- | ---- | ---- |

Matricen (selv om andet vises) indeholder primært 0’er. Dette skyldes at træningsættet måske indeholder 5.000 unikke ord og hver email i gennemsnit måske indeholder 50 unikke ord. Dette betyder altså at for alle ud over de 50 unikke ord vil ntd-variablen være lig 0 dvs. at TFIDF-scoren for dette ord i dokumentet bliver 0. For at spare plads outputes resultatet som en sparse matrix – en type matrix der ikke gemmer 0’erne og dermed sparer plads. Vi omdanner dog denne sparse matrix til en numpy-array (så dette gør ingen forskel).

1. Der bergnes på lignende måde TDFIDF-scorer for testsættet. For testsættet opfattes korpus alene som den mail der ses på (man opfatter hver mail som alfeveres til algoritmen som værende uafhængige). Dette skyldtes at man f.eks. ved et spam-filter jo ikke vil sortere 1000 mails på en gang (mellem spam/ikke-spam). Man modtager en mail af gangen der skal sorteres til/fra. Dermed bliver N=1, og der gælder at Nt=1 eller Nt=0 (alt efter om ordet indgår i den ene mail man ser på). Hvis Nt=0 ville man få en division by zero error. Derfor laves der idf\_smoothing (det jeg nævnte senere med at SciKit benytter anden formel). Jeg tror at forskellen er at der tillæges 1 til nt-variablen i formlen. Dette svarer altså til at hvert dokument (om man ser på det alene eller har et større korpus) tilføjes at indeholde hvert ord 1 ekstra gang. Dette gælder for resten også ved træningsættet – der sker altså generelt en forskydning til mindre IDF og dermed mindre TF-IDF-score.

(hvilket giver 10.000 TF-IDF-features )

**Naive Bayes:**

Grundlæggende teori (Bayes therom):

Kendetegnet for Naive Bayes metoderne er at de alle bygger på bayes therom:

Dette kan i vores tilfælde specificers til:

Her angiver "data" vores input data dvs. en vektor med en række features (som f.eks. længde og antal af blade på blomst). "class" anigver den klassifikation (target) som vi ønsker at mappe observationen til (angives ift. spam (1) og ikke-spam (0)).

Man kalder modellen naiv da man antager at værdierne af input features'ne er uafhængige af hinanden. I vores eksperiment vil det sige at man antager at der er lige så stor sandsynlighed for at ordet "alzheimers" indgår i en sætning med et ord som "falafel" som med et ord som "sygdom" (da vores features udgøres af TF-IDF-scorer for ord der indgår i en email-besked). Dette er selfølgelig ikke rigtigt, men ved at antage det får man gode resultater. Ved denne opfattelse kan man lave følgende omskrivning:

At denne omskrivning er mulig følger af princippet bag "joint probability" (se forelæsning om kausalitet læsestof). Bemærk at man i praksis yderligere ofte fjerner nævneren. Dermed kan man regne sandsynligheden for at et sæt features (dvs. en observation) skal knyttes til en class med formlen:

Man fjerne ofte nævneren da denne er konstant og har samme værdi for begge classes. Fjerner man nævneren får man ikke længere en sandsynlighed ud af bayes therom (da dette kræver normalisering fra nævneren). Når vi ønsker at set sandsynligheden med SciKit beregnes den efterfølgende på en leler anden måde.

I vores eksperiment vil vi kunne sammenligne sandsynligheden for en observation (email) udgjort af features (TF-IDF-score for ord) tilhører enten klassen "spam" (1) eller klassen "ikke-spam" (0). Den sandsynlighed der er størst bestemmer klassifikationen.

Der findes 3 versioner af Naive Bayes (hvoraf de to sidste er relevante for vores projekt):

1. Gaussian (normalfordeling) - Denne bruger vi
   1. Bruges når input-features er kontinuerte værdier (som f.eks. IF-IDF-scorer)
2. Mutlinomial – Denne er optional (burde være hurtiger og mere præcis)
   1. Bruges når input-features er diskrete værduer (som f.eks. Bag-of-words ord optællinger)
3. Bernoulli (binomialfordeling) – laves ikke
   1. Dette bruges når input-features er bineære (sand/falsk)

Implementering af Gauss NB (unikke dele):

Man skal beregne delsansynlighederne i den korrigerede Bayes therom. Den ene sandsynlighed er let at beregne:

Det svære opstår når man skal beregne sandsynlighederne af typen

dvs. sandsynligheden for at en observation tilknyttet class indeholder feature  f.eks. TF-IDF-scoren 2,3 for et bestemt ord (som jo typisk indgår i en sætning med mange andre ord). Man starter med at separere datasættet i klasser. Derefter er det muligt at beregne følgende 2 statistiker for de variabelværdier assignet til hver klasse (dvs. f.eks. for datasæt bestående af -værdier der hører til en observation klassificeret til class ):

1. Estimat på populationsmiddelværdi (almindelig middelværdis formel)
2. Estimat på populationens standardafvigelse.

Man antager derefter at -værdier klassificeret til class er normalfordelte (dvs. trukket fra en normalfordeling ift. sandsynligheden for at variablen antager forskellige værdier). Sandysynligheden for at få en given værdi berenes med gauss-kurven (defineret ud fra middelværdi og standafvigelserne beregnet). Der med kendes alle delsandsynligher og kan derfor beregnes for alle classes (i vores tilfælde 2 classes – spam/ikke-spam). Der vælges den class med den største sandsynlighed. Bemærk at sandsynlighederne ofte bliver tæt på 0 eller 1 (typisk for Naive Bayes formel).

**Scikit parametre:**

Til gauss-versionen NB er der ikke rigtig nogle parametre der skal ændres på. Alle parametre er konstante og den letteste forståelse af disse parametre kommer ved at læse dokumentationen og implementeringskoden selv (her menes den samlede test kode).

**Spørgsmål:**

1. Jeg forstår stadig ikke helt hvordan at det fungerer med TF-IDF
   1. Hvad er korpus ved de forskellige beregninger af TF-IDF? Som jeg har opfattet det beregnes TF-IDF over hele korpus (alle trænings-mails) – dermed vægtes altså vigtigheden af en term (på den måde det normalt er meningen med TF-IDF). Når man så regner TF-IDF for testsættet – hvad er så korpus? I princippet burde vores test af spam-filteret jo egentligt opfatte hver mail som uafhæhængig (man får en mail – den vurderes – den slettes eller lades stå). Derfor burde korpus jo egentlig blot udgøres af en enkelt mail – men hvordan giver IDF mening ift. dette så og hvilken sammenhæng har dette til idf\_smoothing?
   2. Hvilken effekt har idf\_smoothing og hvordan virker den + hvorfor crasher vores program ikke når den fjernes?
2. Få styr på den normalisering som TfidfVectorizer laver – har det noget at gøre med forskyding der sker ved idf\_smoothing.
3. Hvordan regnes probabilities på bagkant af scikit når den ikke har normlsieret.

Bemærk:

I vores eksperiment vil der være ekstremt mange features (ummidelbart en for hvert ord i samtlige 5000 mails). Da der skal beregnes en sandsynlighed for værdien af hver feature knyttet til en class betyder det (i og med eksperimentet har 2 classes) at der skal opretholdes 5000∙2 normalfordelinger (altså 10.000 forskellige normalfordelinger).

Bemærk:

Især for små sample sizes kan en Naive Bayes classifier ofte outpeforme mere powerfulle alternativer (<https://sebastianraschka.com/Articles/2014_naive_bayes_1.html>). Dog kan antagelsen at features er statistisk uafhænige af hinanden og ikke-lineære klassifikationsproblemer lede til rigtig dårlige resultater.

Computergenereret alternativ tekst:
000 
O 

Bemærk:

Delene i Bayes therom har alle sammen navne:

Computergenereret alternativ tekst:
conditional probability • prior probability 
posterior probability 
evidence 

Her gælder at:

* Posterior = Senere
* Prior = tidligere

Bemærk:

If the priors are following a uniform distribution, the posterior probabilities will be entirely determined by the class-conditional probabilities and the evidence term. And since the evidence term is a constant, the decision rule will entirely depend on the class-conditional probabilities (similar to a frequentist’s approach and maximum-likelihood estimate).

*Fra <*[*https://sebastianraschka.com/Articles/2014\_naive\_bayes\_1.html*](https://sebastianraschka.com/Articles/2014_naive_bayes_1.html)*>*

Dette betyder vel at jeg er nød til at sikre at der er en 50%/ 50% fordeling mellem spam og ham classen. Eller hvad?