Textanalys och maskininlärning:

En jämförelse av maskininlärningsalgoritmer för klassificering av fakturor och kvitton i e-postmeddelanden.

LUNDS UNIVERSITET
Campus Helsingborg

LTH Ingenjörshögskolan vid Campus Helsingborg
Institutionen för datavetenskap

Examensarbete:
Maurits Johansson
Nimer Shadida Johansson

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Lunds universitet
Lund 2018

Sammanfattning

Enligt Skatteverket (Skatteverket, 2018-05-29) skall varje händelse som påverkar ekonomin i ett företag bokföras. Bokföringen skall grunda sig på skriftlig handling, också kallat verifikation. Bland typer av verifikationer kan en sådan vara ett kvitto eller en faktura.

I ett företag där anställda gör köp åt företagets vägnar är det alltid nödvändigt att sammanställa verifikationer. Processen av att samla och sammanställa fakturor och kvitton sker oftast manuellt. Från detta kom idén att utveckla en mobilapplikation vars syfte är att samla fakturor och kvitton från en användares e-post och spara tid för användaren. För att idén för mobilapplikationen skall vara möjlig måste det finnas ett sätt att effektivt klassificera rätt dokument i ett e-postmeddelande. Ur detta kom idén till detta examensarbete. En möjlig användare till applikationen kan vara en anställd på ett företag som måste rapportera till företagets ekonomiavdelning.

Syftet med examensarbetet är att hitta en lämplig lösning med hjälp av maskininlärning och språkbehandlingstekniker, som ordstam och Bag of Words, för att automatiskt kunna identifiera vad som är ett kvitto eller faktura från ett godtyckligt textdokument.

Arbetet utfördes genom att jämföra tre olika maskininlärningsalgoritmer tillsammans med en modul för språkbehandling, Natural Language Toolkit, samt Bag of Words metoder. Två av algoritmerna är baserade på Naive Bayes teorem, *multinomial naive bayes* samt *multivariate bernoulli naive bayes* algoritm. Den tredje maskininlärningsmetoden är ett neuronnät. Resultatet i detta examensarbete visar att neuronnät med minst 80% framgång kan identifiera fakturor och kvitton korrekt. Naive bayes baserade maskininlärningsalgoritmer visar fluktuerande resultat. Klassificeringen av fakturor i multinomial naive bayes visar 100% framgång i klassificering av fakturor och i bästa fall 70% framgång i klassificering av kvitton. Multivariate bernoulli naive bayes algoritm visar i bästa fall 87% framgång i klassificering av fakturor och 13% framgång i klassificering av kvitton. Vid ökning av datamängden för träning av de naive bayes baserade algoritmerna försämrades det sammanställda resultatet för de båda algoritmerna. Neuronnätet visar konsekvent framgång av minst 80% oberoende av datamängden använd för träning i detta examensarbete.

Sammanfattningsvis är metoder som liknar det sätt neuronnät behandlar och tolkar information bättre lämpat för denna typ av klassificeringsproblem. Anledningen till varför naive bayes algoritmerna presterar sämre är på grund av svårigheter att konsekvent formatera data av olika ursprung. Med ursprung menas olika filformat av de dokument som utgör data samt huruvida dokumentet ursprungligen har skapats genom användning av ett program, såsom Microsoft Word, eller ifall det är ett fotografi. Då data, orden i ett textdokument, inte konsekvent formateras med samma resultat är användningen av ett neuronnät fördelaktigt. Anledningen till detta är att neuronnätet inte har samma strikta riktlinjer att förhålla sig till. Nätet kommer genom många och återigen många iterationer försöka anpassa sig så att felen blir så minimala som möjligt. Ger vi nätet inkonsekvent data kommer det därav anpassa sig till det och producera bästa möjlig gissning givet den inkonsekventa träningsdatan. Naive Bayes har inte samma anpassning och måste förhålla sig till Bayes sats.

Nyckelord: Textanalys, maskininlärning, applikation, faktura, kvitto.

Abstract

According to the Swedish tax authorities (2018-05-29), every event that affects the economy of a company must be recorded. The accounting must be based on written action, also known as a verification. Among the types of verifications one such verification can be a receipt or invoice.

In a company where employees make purchases on behalf of the company, it is always necessary to compile verifications. The process of collecting and compiling invoices and receipts is usually done manually. From this came the idea of developing a mobile application whose purpose is to collect invoices and receipts from a user's email and save time for the user. In order for the mobile application idea to be possible, there must be a way to efficiently classify the correct document in an email. From this came the idea of this thesis project. A potential user of the application may be an employee of a company that has to report to the company's finance department.

The purpose of this thesis project is to find an appropriate solution using machine learning and natural language processing techniques, such as word-stemming and Bag of Words methods, to automatically identify a receipt or invoice from any given text document.

The work was done by comparing three different machine learning algorithms together with a natural language processing module, Natural Language Toolkit, and Bag of Words methods. Two of the algorithms are based on Naive Bayes theorem, multinomial naive bayes and multivariate bernoulli naive bayes algorithm. The third machine learning method is a neural network. The result of this thesis project shows that neural networks with at least 80% successrate can correctly identify invoices and receipts. Naive Bayes based machine learning algorithms show fluctuating results. The classification of invoices in multinomial naive bayes shows 100% success and at best 70% success in the classification of receipts. The multivariate bernoulli naive bayes algorithm shows at best 87% success in the classification of invoices and 13% success in the classification of receipts. By increasing the amount of training data for the naive bayes based algorithms, the aggregate result of the two algorithms deteriorated. The neural network shows consistent success of at least 80% regardless of the amount of data used for training in this thesis project.

In summary, methods similar to the way neuron networks treat and interpret information are better suited to this type of classification problem. The reason why naive bayes algorithms perform worse are due to the difficulty of consistently formatting data of different origins. Originally, different file formats are meant by the documents constituting data as well as whether the document was originally created using a program such as Microsoft Word, or if it is a photograph. Since data, the words in a text document, are not consistently formatted with the same result, the use of a neural netowrk is advantageous. The reason for this is that the Neural Network does not have the same strict guidelines to relate to. The network will, through many and again many iterations, try to adapt so that the errors are minimized. If we give the network inconsistent data, it will adapt to it and produce the best possible guess given the same inconsistent training data. Naive Bayes does not have the same adaptation and must follow Bayes theorem.

Keywords: Text analysis, machine learning, application, invoice, receipt.

Innehållsförteckning

Förord

Arbetet har varit väldigt insiktsfullt för oss båda. För potentiella lösningar för problemet beskrivet i denna rapport men även möjligheterna som metoder inom maskininlärning öppnar upp för. Vi vill tacka Mats Lilja för den hjälp han har bidragit med i rapporten samt Malmö Stad och Omvida AB för det underlag av data som de har bidragit med till examensarbetet. Vi är båda glada för chansen att spendera vårt examensarbete med en idé av eget ursprung.

1. Inledning

I ett företag där anställda gör köp åt företaget är det alltid nödvändigt att sammanställa verifikationer. Processen av att samla och sammanställa fakturor och kvitton sker oftast manuellt. Från detta kom idén att utveckla en mobilapplikation vars syfte är att samla fakturor och kvitton från en användares e-post och spara tid för användaren. En användare kan vara en anställd på ett företag som måste rapportera till företagets ekonomiavdelning.

1.1. Bakgrund

Vid utvecklingen av denna mobilapplikation upptäcktes det ett problem att skilja ett e-postmeddelande som är relevant från ett som inte är det. I detta fall anses ett e-postmeddelande vara relevant om det innehåller eller består av ett kvitto eller en faktura. Ett e- postmeddelande består av olika beståndsdelar s.k. MIME-typer, Multipurpose Internet Mail Extensions, som tar tillvara på olika typer av information, i form av bilagor, som ett e-post innehåller. Dessa MIME-typer kan vara allt från pdf filer till bilder och text med mera.

Genom att extrahera nyckelord och HTML-strukturer ur text, är det möjligt att uppskatta graden av relevans hos e-post. Detta är dock inte tillräckligt. Målet med den slutgiltiga applikationen är att individer skall använda detta både privat och i företagssammanhang. På grund av att ett e-postmeddelande ofta innehåller information av känslig typ är det viktigt att lösningen, och slutligen applikationen, korrekt inhämtar relevanta dokument. För att lösningen skall vara tillräcklig måste träffsäkerheten på att identifiera relevanta e-postmeddelanden vara så hög som möjligt.

En ofta använd metod vid klassificeringsproblem är maskininlärning. Därför har maskininlärning valts som metod för detta klassificeringsproblem. Ordet träning förekommer flertalet gånger genom denna rapport och avser de dataset av fakturadokument och kvitton som används för inlärning av de program som presenteras i kapitel 2.

1.2 Syfte

Syftet med detta examensarbete är att ta fram en lämplig lösning med hjälp av maskininlärning och språkbehandlingstekniker, som ordstam och Bag of Words, för att automatiskt kunna identifiera vad som är ett kvitto eller faktura från ett godtyckligt textdokument. Lösningen ska vara effektiv, med avseende på noggrannhet av klassificering. Lösningen skall i sin tur användas i en mobilapplikation för att automatiskt inhämta fakturor och kvitton som befinner sig i en användares e-postmeddelande. En användare kan vara en anställd på ett företag som måste rapportera till företagets ekonomiavdelning.

Ett användningsscenario skulle kunna vara att en anställd som har fakturor liggandes på sin epost enkelt vill få ut, sortera, och exportera dom vidare till nästa person utan att behöva leta igenom sin inkorg. Varje ny faktura som inkommer till en epost kommer direkt till applikationen.

1.3 Målformulering

Målet med detta examensarbete är att undersöka olika maskininlärningsmetoder för att klassificera textdokument samt bidra med förslag på lösning för detta klassificeringsproblem. Klassificeringen av textdokumenten kan vara kvitto, faktura eller intetdera.

1.4 Problemformulering

Nedan presenteras problemformuleringen för det mål och syfte examensarbetet ämnar att uppfylla.

- Hur ska mängden träningsdata för två klasser vara procentuellt uppdelade för att nå bästa möjliga klassificeringsförmåga?

- Vilken datamängd för två klasser anses vara tillräckligt för träning och testning?
- Hur bör datan eller texten behandlas för att bäst kunna klassificeras?
- Hur mycket av datamängden bör användas till test respektive träning?

1.5 Motivering av examensarbete

Nyttan av att kunna inhämta fakturor och kvitton automatiskt via e-post och sammanställa dem på en och samma plattform är olika beroende på om användningen är privat eller i företag. För privata användare ger inhämtningen och sammanställningen en överblick över de kvitton och fakturor som en användare har på sin e-post. I vanliga fall krävs det av användaren att veta vad eller var ett kvitto eller faktura befinner sig i dess e-post för att kunna hitta dem. För företagsanvändare är den automatiska sammanställningen vidare fördelaktig då användaren också har möjligheten att exportera en eller flera dokument samtidigt, åt exempelvis företagets ekonomiavdelning.

För att det ska vara möjligt att automatiskt sammanställa fakturor och kvitton från en användares e-post måste det vara möjligt att kunna urskilja e-post som innehåller fakturor och kvitton mot de som inte gör det. Motiveringen till examensarbetet är därför att prova maskininlärningsmetoder för att komma med förslag till en potentiell lösning som möjliggör att automatiskt kunna inhämta fakturor och kvitton från en användares e-post.

1.6 Avgränsningar

Det eller de program som tas fram i detta examensarbete avgränsas till att enbart exekvera i integrerade utvecklingsmiljöer som kan kompilera programmeringsspråket Python. Vidare avgränsas programmen till att enbart kunna klassificera upp till två klasser, kvitto och/eller faktura.

2. Teknisk bakgrund

För att det skall vara möjligt att utveckla, träna och testa olika maskininlärningsalgoritmer och metoder måste det först existera någon form av data att bearbeta. För att kunna sammanställa data för algoritmer måste det finnas någon parameter som representerar vad som utgör relevant data. I detta kapitel presenteras förbehandling av data samt olika metoder för att klassificera dokument till de två klasserna kvitto och faktura.

2.1 Data

I detta delkapitel presenteras hur data extraheras för användning vid träning av algoritmer.

2.1.1 E-postmeddelande

Ett e-postmeddelande kan innehålla en mängd olika information och se ut på en mängd olika sätt. För att kunna bearbeta ett e-postmeddelande på ett sätt så att ett godtyckligt e-postmeddelande kan klassificeras måste det finnas en typ av information som är gemensam för alla e-postmeddelande.

Ett e-postmeddelande kan bestå av, förutom subject header och message body, ett flertal olika Multipurpose Internet Mail Extensions, också kallat MIME. MIME-typer, en bilaga till body delen av ett e-postmeddelande, sträcker sig från audio-filer och bild-filer till olika former av textdokument, såsom text-filer och PDF-filer. (IETF, 2013-03-02). I detta examensarbete behandlas textinformation från e-post och MIME-typer, i form av bilagor till e-post som innehåller text.

2.1.2 Utvinna text

Som nämnts i Kap. 2.1.1 består ett e-postmeddelande ofta av en mängd olika datatyper. Ett e-postmeddelande kan även vara indelat i olika MIME-typer, ett standardiserat sätt att ange formatet på ett dokument. På grund av att fakturor och kvitton kan vara en godtycklig del av ett e-postmeddelande, det vill säga att det kan befinna sig i olika eller flera delar av ett e-postmeddelande är det viktigt att behandla samtliga delar av e-postmeddelandet för att sammanställa ett dokument av text för behandling till eventuell maskininlärningsalgoritm. Beskrivet härunder är processen som används i detta examensarbete för utvinning och formatering av text för en del av eller ett fullständigt e-postmeddelande.

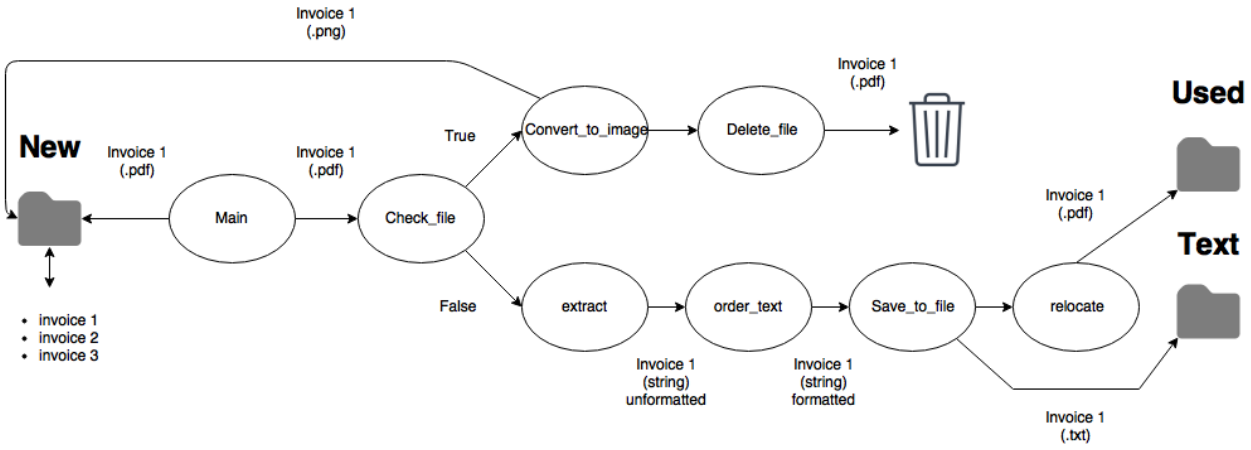


Fig. 1: En illustration av hur programmet extract_text.py körs.

I Fig. 1 illustreras processen av programmet extract_text.py, ett program skrivet i Python, som traverserar en given katalog med fakturor i .pdf format och behandlar en faktura i taget med hjälp av modulen Textract, se Appendix 9.1.1, som med hjälp av OCR (Optical Character Recognition) omvandlar text i en bild till klartext. Klartexten lagras i sin tur i en sträng. På grund av att modulen Textract använder olika metoder att extrahera text från olika typer av dokument är det inte alltid så att det är möjligt att behandla ett dokument i .pdf format. Om modulen kan identifiera att dokumentet är formaterat som en .pdf fil kommer en förutbestämd metod för att extrahera text ur bilagan göras, om .pdf filen inte innehåller text utan är en bild som är sparad i .pdf format är det inte möjligt för modulen att extrahera texten från dokumentet.

En kontroll görs preliminärt, av metoden *check_file(path)*, för att kontrollera att den aktuella filen går att utvinna text från. Om strängen, som returneras av funktionen, är tom kallas funktionen *convert_to_image(filename, path)*, som med hjälp av modulen Wand, se Appendix 9.1.2, konverterar .pdf filen till en .png fil och sparar denna nya fil i samma katalog. Därpå tas den ursprungliga filen, som inte var läslig, bort av metoden *delete_file(filename)*.

Då *check_file(path)* kontrollerar att modulen Textract kan extrahera text från det aktuella dokumentet kommer metoden *extract(path)* att köras. Texten från dokumentet sparas temporärt i en sträng som sedan skickas vidare till metoden *order_text(text)* för att delas upp ord för ord med ett kommatecken som separerar varje ord. När processen är klar sparas klartexten, som nu är separerad med kommatecken, i en .txt fil, i en underkatalog. Varpå metoden *relocate(path, destination_path, filename)* tar det dokument som använts för att extrahera texten och placerar den i en annan underkatalog för att indikera att denna fil nu har använts.

2.2 Maskininlärningsalgoritmer

I detta delkapitel redovisas de maskininlärningsalgoritmer som har implementerats för detta examensarbete. I samband med att förklara tillvägagångssättet för implementering beskrivs också hur algoritmerna fungerar.

2.2.1 Multinomial Naïve Baye's Classification

För att förstå hur Naive Bayes klassificering fungerar så behöver man förstå Bayes' sats. Ekv.1 beskriver satsen (Dan Morris, 2016).

$$Posteriorisannolikhet = \frac{Betingad\ Sannolikhet \bullet Priorisannolikhet}{Beviset}$$

$$<=> P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Ekv. 1: Bayes sats i två beskrivningar

2.2.1.1 Posteriorisannolikhet

I kontext av klassificering kan posteriorisannolikhet beskrivas som sannolikheten att ett objekt tillhör en viss klass givet värdet som har observerats. I exemplet i Ekv. 2 görs ett försök att förklara posteriorisannolikheten.

$$P(Sjukdom \mid x_i), x_i = [\text{värde}_1, \text{värde}_2, \ldots, \text{värde}_n]$$
$$x_i = i,\ i \in \{1,2,\ldots,n\}$$

Ekv. 2: Exempel på hur flera datapunkter uttrycks som ett mönster i en vektor

Exemplet lyder: "*vad är sannolikheten att en person har en sjukdom, givet ett eller flera värden?* "

Låt x_i vara vektorn med data, ω_j vara klassnotationen för varje klass och $P(x_i \mid \omega_j)$ vara sannolikheten att observera värdet x_i givet att det tillhör klass ω_j. På så vis är det möjligt att skriva om posteriorisannolikheten till:

$$P(\omega_j \mid x_i) = \frac{P(x_i \mid \omega_j) \cdot P(\omega_j)}{P(x_i)}$$

Ekv. 3: Posteriorisannolikheten uttryckt med ett mönster x_i

Posteriorisannolikheten i Ekv. 3 kommer ge olika värden beroende på indatan x_i och beroende på vilken klass ω_j som man väljer att jämföra med.
Konkret kan ett kriterium för huruvida en person har en viss sjukdom eller inte skivas som:

$$P(Sjukdom \mid x_i) \geq P(Icke - sjukdom \mid x_i)$$

Ekv. 4: Kriterium för hur två klassers värden ska jämföras

2.2.1.2 (Klass)Betingad sannolikhet

För att vidare kunna bestämma posteriorisannolikheten behövs en kännedom om den betingade sannolikheten, $P(x_i \mid \omega_j)$, att observera värde x_i givet att det tillhör klassen ω_j.
Ett antagande som görs vid multinomial naive Bays klassifikationer är att värdena x är villkorligt oberoende (*eng.* Conditional independence) (Harry Zhang, 2004). Villkorligt oberoende, inom sannolikhetsteori, kan beskrivas som två fenomens oberoende givet ett tredje fenomen, det vill säga att de två fenomen är villkorligt oberoende , om de är oberoende på grund av ett tredje fenomens inverkan på dem (A. P. Dawid, a. 1979). För att vidare förklara, två kast av tärning är oberoende av varandra på så vis att man inte kan förutse det andra kastets resultat från det första kastets resultat. Dock, om ett tredje fenomen antyder att summan av de två första kasten är jämnt kommer de resultat kast två kan anta begränsas, därav är de villkorligt oberoende. I detta fall utnyttjas villkorligt oberoende på så vis att de olika klass-betingade sannolikheterna direkt kan läsas från indatan, istället för att jämföra alla möjligheter.

$$P(x \mid \omega_j) = \prod_{k=1}^{d} P(x_k \mid \omega_j)$$

Ekv. 5: Den betingade sannolikheten går att uttrycka som en produktsumma av innehållet i x

En vektor x, där x_k refererar till varje element k i vektorn x, med data kan således användas som illustrerat i Ekv.5 (Harry Zhang, 2004). Här betyder $P(x \mid \omega_j)$ sannolikheten att observera ett mönster x, en vektor av datapunkter, givet att det tillhör en klass ωj.

$$P(x_k \mid \omega_j) = \frac{N_{x_k,\omega_j}}{N_{\omega_j}}$$

Ekv. 6: Den betingade sannolikheten beskrivet som en relativ frekvens

Den betingade sannolikheten, illustrerat i Ekv. 6 som $P(x_k \mid \omega_j)$, räknas fram som en relativ frekvens av hur ofta ett värde x_k förekommer i en klass (Rennie, J. D., Shih, L., Teevan, J., & Karger, D. R 2003). N_{x_i,ω_j} är antalet gånger värdet x_k förekommer i datan från en klass ω_j och N_{ω_j} är totalt antal värden som klass ω_j innehåller.

För att bättre ge en inblick i hur detta används vid klassificering av dokument för kvitton och fakturor, som i detta examensarbete, beräknas ett exempel av att ett e-postmeddelande som innehåller texten ”Stort grattis” skulle vara ett spam-meddelande.

$$P(x = [Stort, Grattis] \mid \omega = spam) = P(Stort \mid spam) \cdot P(Grattis \mid spam)$$

$$P(x = [Stort, Grattis] \mid \omega = spam) = \frac{5}{50} \cdot \frac{40}{50} = 0.08$$

Ekv. 7: Exempel på beräkning av den betingade sannolikheten givet en vektor x

I Ekv.7 används 200 e-postmeddelanden, varav 50 av dessa är s.k. spam, till beräkningens dataset. Med ett dataset menas en mängd information taget från respektive textdokument eller e-postmeddelande. Från resultatet av beräkningen framgår det att e-postmeddelandet har en betingad sannolikhet på 0.08.

En metod som används vid implementeringen av Naive Baye’s är Laplace Smoothing (Jurafsky, D., & Martin, J. H. 2009). Laplace Smoothing är en enkelt knep för att klara av nya förekommande objekt som programmet inte är tränat för. Hanteras inte detta så blir den betingade sannolikheten noll eftersom N_{x_k,ω_j} blir noll och som en konsekvens av detta blir hela posteriorisannolikheten noll. Appliceringen av Laplace Smoothing illustreras i Ekv. 8.

$$P(x_k \mid \omega_j) = \frac{N_{x_k,\omega_j}}{N_{\omega_j}} \quad => \quad P(x_k \mid \omega_j) = \frac{N_{x_k,\omega_j} + 1}{N_{\omega_j} + |V| + 1}$$

Ekv. 8: Applicerad Laplace smoothing där |V| refererar till antalet unika ord i vår träningsdata.

Laplace Smoothing ger alltså okända ord ett litet värde och förhindrar att posteriorisannolikheten blir noll då ett okänt ord förekommer i testdatan.

2.2.1.3 Priorisannolikhet

Priorisannolikheten eller ”förkunskapen” är en viktig del i naive Bayes beräkningen. Det priorisannolikheten beskriver kan variera men när det kommer till klassificering så är den enda egentliga förkunskapen den träningsdata som finns tillgänglig. I klassificeringskontext kallas detta klass-priori och beräknas enkelt genom ekvationen illustrerad i Ekv.9, där N_{ω_j} = Antalet data från en viss klass ω_j och N_c = Antalet data från alla klasser.

$$P(\omega_j) = \frac{N_{\omega_j}}{N_c}$$

Ekv. 9: Priorisannolikheten för en klass beskrivs som ett förhållande mellan en klass träningsdata och den totala träningsdatan.

Resultatet ger alltså ett mått på fördelningen av data på olika klasser (Jurafsky, D 2009). Priorisannolikheten är den enda tidigare erfarenhet programmet har av problemet.

2.2.1.4 Beviset

Genom beskrivningen av posteriorisannolikheten kan man utgöra att $P(x_i)$ är konstant och beror endast på x_i. Detta gör att termen inte behövs för att korrekt kunna beräkna ett maxvärde mellan två jämförelser av $P(\omega_j | x_i)$. Exempel vid beräkning av huruvida något är spam eller inte gäller:

$$P(Spam | x_i) = \frac{P(x_i | Spam) \cdot P(Spam)}{P(x_i)} \quad > \quad P(!Spam | x_i) = \frac{P(x_i | !Spam) \cdot P(!Spam)}{P(x_i)}$$

Ekv. 10: Bestämmelse för huruvida något ska klassificeras som spam eller inte.

Vilket det framkommer tydligt att $P(x_i)$ inte är något annat är en multiplikationsfaktor och således dess värde sättas till 1 utan att maximeringen påverkas. Sannolikheten kan skrivas om.

$$P(\omega_j | x_i) = \frac{P(x_i | \omega_j) \cdot P(\omega_j)}{P(x_i)} \quad => \quad P(\omega_j | x_i) = P(x_i | \omega_j) \cdot P(\omega_j)$$

Ekv. 11: Omskriven posteriorisannolikhet utan beviset

$P(x_i)$ kan för övrigt beskrivas i kontext som sannolikheten att stöta på x_i oberoende av klass.

2.2.1.5 Implementering

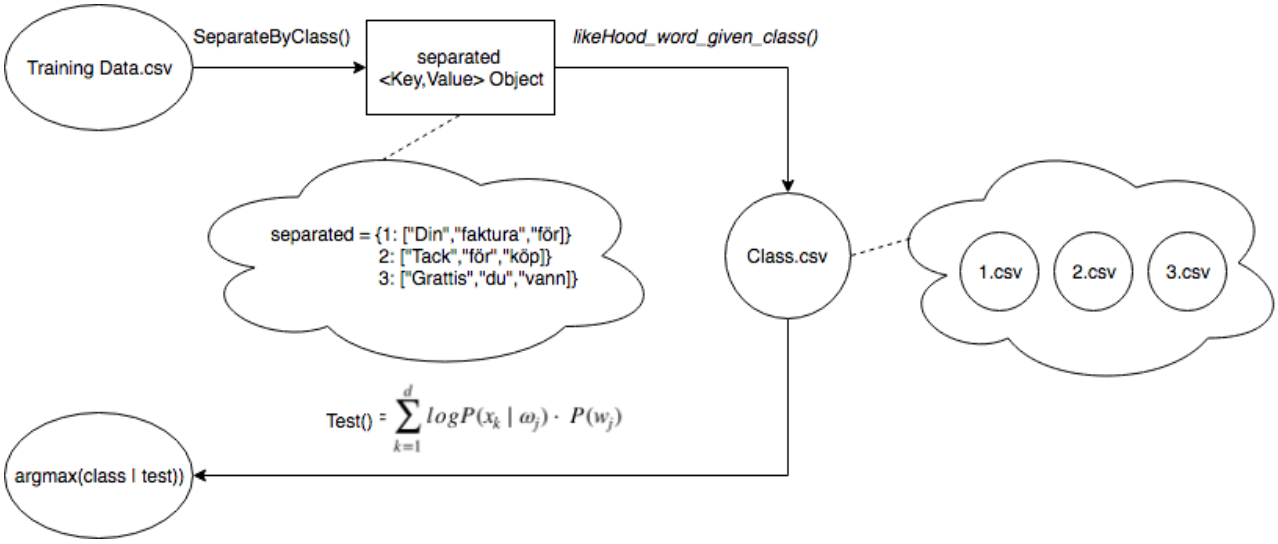


Fig.2: Illustration av Multinomial Naive Bayes implementation

Implementeringen för beräkning av posteriorisannolikheten, se Fig.2, gjordes med Python (Python Software Foundation, 2018-05-31) och utan några externa bibliotek. För lagring av data användes Comma Separated Values filer (IETF, 2018-05-31). I detta examensarbete behandlas frågan om att klassificera ett dokument huruvida det är ett kvitto eller en faktura.

$$P(Faktura|x_i) \geq P(Kvitto|x_i)$$

Ekv. 12: Bestämmelse för huruvida programmet ska klassificera något som en faktura eller ett kvitto

Programmet ska alltså utföra en beräkning av varje specifikt ord x_i posteriorisannolikhet i vektorn x. För att det skall vara möjligt att kunna utföra Multinomial Naive Bayes behövs tillgång till data för träning av algoritmen.

Tab. 1: Illustration av datastruktur i .CSV fil.

	Ord 1	Ord 2	Ord 3	…	Ord n	klass
	Tack	för	köp		Tack	2
	Din	faktura	för		tack	1
	Grattis	du	vann!		Grattis	3

Tab.2: Aktuella klasser samt respektive nummer till varje klass.

Klass	1	2	3
Beskrivning	Faktura	Kvitto	Övrigt

En metod *SeparateByClass( ),* se Appendix 9.3.1, kan nu använda CSV filen i tabell 1, där varje rad representerar ett eget dokument. För att separera dess innehåll och lagra alla ord som tillhör en viss klass, se Tab. 2, i en egen vektor används *SeparateByClass( )* som returnerar ett key-value objekt vars nyckel är den aktuella klassens id. En vektorn med ord tillhörande respektive klass returneras även.

Tab.3: Klass specifik .CSV fil.

Ord	# antal förekomster av ordet i klassen	Totalt antal ord i Klassen	Klassbetingade sannolikheten för ordet givet klassen	Antal unika ord i alla dokument
Faktura	133	10555	x	2444
Hej	…	…	…	..
Totalt	…	…	…	…
Moms	…	…	..	..

I tabell 3 visas hur en .csv fil för en specifik klass ser ut. Där antalet förekomster av ett specifikt ord förekommer för denna klass, totalt antal ord som finns i klassen, samt den beräknade betingade sannolikheten för denna klass och hur stort vokabulär, antal unika ord, klassen har. För att ta fram informationen om respektive fält måste en metod *likeHood_word_given_class(Dictionary dict, String s,_class),* se Appendix 9.3.2, användas.

När metoden väl har börjat behandla informationen i kolumn 1, 2 och 4 i tabell 3 för respektive ord kan metoden börja beräkna den klassbetingade sannolikheten för ordet givet klassen, se Ekv. 13. Där x_k är ordet, ω_j är klassen, N_{x_k,ω_j} är antal förekomster av ordet i klassen, N_{ω_j} är totalt antal ord i klassen och $|V|$ är alla unika ord i alla dokument.

$$data[3] = P(x_k \mid \omega_j) = \frac{N_{x_k,\omega_j} + 1}{N_{\omega_j} + |V| + 1}$$

Ekv. 13: Data[3] innehåller den betingade sannolikheten för x_k

För varje ord maskininlärningsalgoritmen skall tränas på, används denna metod. Resultatet för varje ord blir en rad i klassens .CSV fil. Varje ord i träningsdatan genomgår samma process för varje klass. Skillnaden i .CSV filerna blir således att ord som är starkt knutna för en viss klass får en högre posteriorisannolikhet till den klassfilen än resterande klassfiler. Fig. 3 visar en skärmdump av några rader ur fakturaklassens .CSV fil. Fig. 4 visar samma rader för kvittoklassens .CSV fil, notera skillnaderna i de betingade sannolikheterna i Fig. 3 jämfört Fig. 4.

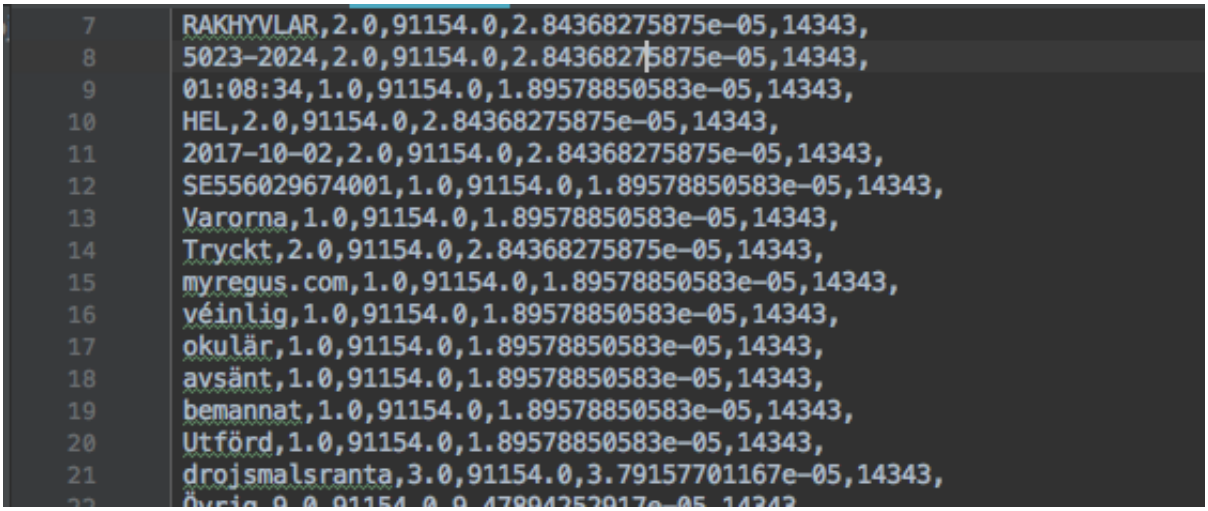


Fig.3: en skärmdump på faktura.csv. Från vänster i denna figur, separerat med komma: Ord, Antal förekomster av ordet i klassen faktura, totalt antal ord i klassen, klassbetingad sannolikhet för ordet, Antal unika ord i alla dokument.

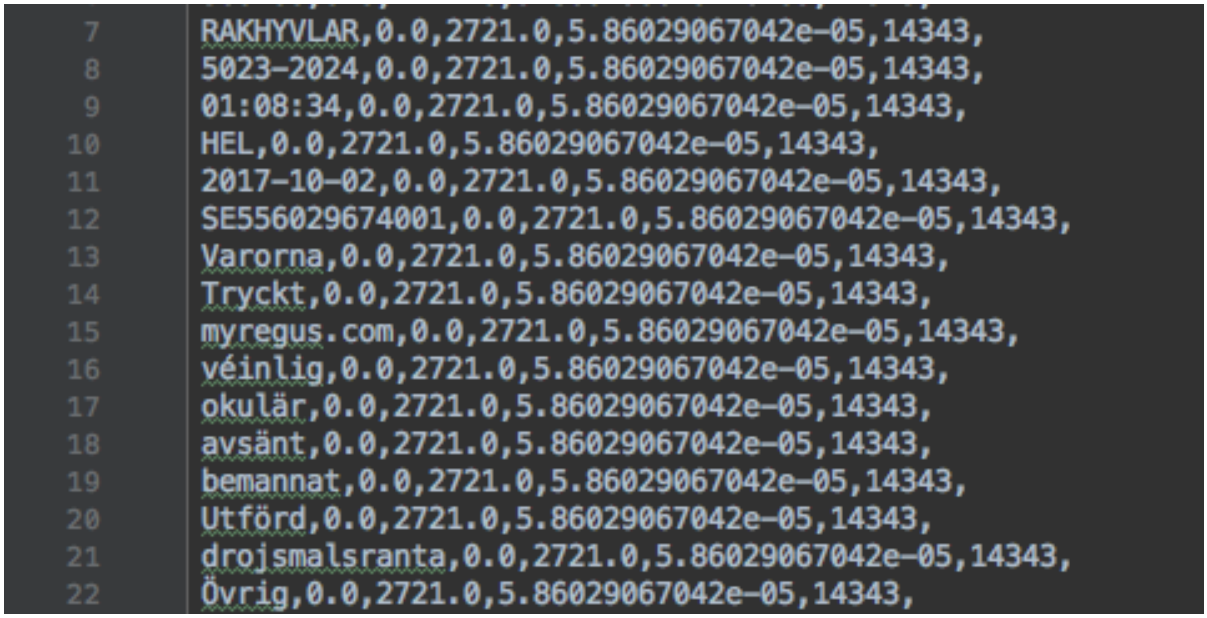


Fig.4: en skärmdump på kvitto.csv. Från vänster i denna figur, separerat med komma: Ord, Antal förekomster av ordet i klassen faktura, totalt antal ord i klassen, klassbetingad sannolikhet för ordet, Antal unika ord i alla dokument.

När programmet gått igenom all träningsdata och producerat de två .csv filerna för respektive klass som visas i Fig. 4 är programmet är färdigt med träning baserad på den tillgängliga datan och testdata kan appliceras för att testa programmet. Med anledning av att programmet testas med en vektor ord så formuleras den betingade sannolikheten om enligt Ekv. 5.
Posteriorisannolikheten ska då i teorin beräknas som:

$$P(\omega_j|x) = P(x|\omega_j) \cdot P(\omega_j) \quad => \quad P(\omega_j|x) = \prod_{k=1}^{d} P(x_k \mid \omega_j) \cdot P(\omega_j)$$

Ekv. 14: Teoretisk beräkning av Posteriorisannolikheten

På grund av mängden ord tenderar värdet av ekvationen att gå mot noll, med förklaring att den betingade sannolikheten är en produktsumma av många små tal, se Ekv. 14. Ett sätt att undgå detta är att beräkna logaritmsummor istället för produktsummor, se Ekv. 15.

$$P(\omega_j|x) = \prod_{k=1}^{d} P(x_k \mid \omega_j) \cdot P(\omega_j) \quad => \quad logP(x|\omega_j) = \sum_{k=1}^{d} logP(x_k \mid \omega_j)$$

Hela beräkningen blir då:

$$logP(\omega_j|x) = \sum_{k=1}^{d} logP(x_k \mid \omega_j) \cdot logP(\omega_j)$$

Ekv. 15: Posteriorisannolikheten beskrivet som en logaritmsumma istället för en produktsumma.

2.2.2 Multivariate Bernoulli Naïve Baye's Classification

En variant till Multinomial naive Bayes är Multivariate Bernoulli naive Bayes. Den direkta skillnaden på dem två klassificeringsmetoderna är hur $P(x \mid \omega_j)$ beskrivs. Tidigare modellering av $P(x \mid \omega_j)$ har gjorts som en frekvens av termer x_k, se ekvation 6.

Genom att istället modellera $P(x \mid \omega_j)$ som en Bernoulli-distribution och använda binära variabler(0,1) för att beskriva förekomster av x_k, istället för frekvenser av x_k, så kan metoden ta i beaktande ett ords närvaro samt ett ords frånvaro i ett dokument. För att beskriva förekomster binärt så använder man sig av en så kallas Bag of Words, se terminologi. $P(x \mid \omega_j)$ kan uttryckas som distributionen i Ekv. 16 (Metsis, Androutsopoulos och Paliouras, 2006).

$$P(x \mid \omega_j) = \prod_{i=1}^{m} P(t_i \mid \omega_j)^b \cdot (1 - P(t_i \mid \omega_j))^{(1-b)}$$

Ekv. 16: Ett dokument x betingade sannolikheten beskriven som en Bernoulli distribution.

I distributionen så utgår man från vektorn Bag of Words, se Tab. 4, där m beskriver storleken på vektorn och där t_i är ordet t på plats i. b är den binära variabeln och beskriver endast om ordet t_i förekommer någonstans i vektorn x. Eftersom den första produkten i produktsumma har exponenten b och den andra produkten har exponenten $(1 - b)$, kommer således varje summa genererar antingen $P(t_i \mid \omega_j)$ eller $1 - P(t_i \mid \omega_j)$ beroende på om t_i finns i x eller inte. Varje $P(t_i \mid \omega_j)$ beskrivs i Ekv. 17 (Metsis 2006):

$$P(t \mid \omega_j) = \frac{N_{t,\omega} + 1}{N_{\omega} + 2}$$

Ekv. 17: Ordet t betingade sannolikhet. Adderingen i täljaren och nämnaren med 1 respektive 2 är ytterligare en Laplace smoothing.

Som beskrivet i detta kapitel är den stora skillnaden mellan Multivariate Bernoulli och Multinomial Naive Bayes att denna metod beskriver förekomsten av ord istället för frekvenser av ord. Därför beskriver $N_{t,\omega}$ antalet dokument i vår träningsdata som tillhör klass ω och innehåller ordet t och N_{ω} beskriver antalet dokument i vår träningsdata som tillhör klass ω. Ett exempel på denna metod kan representeras som följande:

$$P(x \mid \omega_j) = \prod_{i=1}^{m} P(t_i \mid \omega_j)^b \cdot (1 - P(t_i \mid \omega_j))^{(1-b)}$$

Ekv. 18: se Ekv. 16. x = [grattis, till, vinsten]

Tab. 4: Bag of words, en vektor innehållande orden i ett dokument, illustrerad som en tabell.

0	1	2	3	4	5	6
grattis	till	vinsten	hej	tack	till	tillbaka

För att beräkna ett maxvärde för $P(x \mid spam)$ ges följande:

$$P(x \mid spam) = P(grattis \mid spam)P(till \mid spam)P(vinsten \mid spam) \cdot (1 - P(hej \mid spam))(1 - P(tack \mid spam))(1 - P(till \mid spam))(1 - P(tillbaka \mid spam))$$

Ekv. 19: Exempel på beräkning av den betingade sannolikheten

Där varje $P(x_k \mid \omega_j)$ tex $P(grattis \mid spam) = \frac{N_{x,\omega} + 1}{N_\omega + 2}$

Även vid denna beskrivning av $P(x \mid \omega_j)$ så kan värdet närma sig noll vid större beräkningar och det är möjligt att applicera beräkning med summa av logaritmer istället för produktsummor, precis som vid beräkningen av multinomial naive Bayes.

2.2.3 NLP, BoW och Neuronnätverk.

Neuronnätverk (*eng. Neural Network)* är en metod för maskininlärning som vagt efterliknar det biologiska sättet neuroner, en typ av cell i hjärnan, kommunicerar för att lära sig att klassificera eller förutse olika händelser. En koppling mellan två neuroner är en riktad förbindelse, vilket överför sändarens aktivitet till mottagaren. Kopplingen kännetecknas av en fördröjning samt en vikt vars värde bestäms under träningen av nätverket.(Nationalencyklopedin, 2018-04-24)

Nedan redogörs en lösning som använder sig av en metod för att implementera neuronnätverk, hur neuronnätverk fungerar och vilka metoder som valts för att förbehandla datan för träning.

2.2.3.1 Bag of Words & Natural Language Processing

BoW (Bag of Words) är en förenklad representation av innehållet i ett eller flera dokument och används i NLP (natural language processing). Orden i ett dokument sparas, oberoende av ordning eller grammatik, som en multimängd. Till skillnad från hur BoW vanligtvis används, där mångfalden av ord behålls i multimängden och används för att indikera hur många gånger ett ord från ett dokument förekommer i multimängden, sparas bara unika ord i denna BoW, som har beskurits på sådant vis att ordens ursprungsmening är det som sparas, också kallat ordstam, se Fig. 5.

$$Wait : waits, waiting, waited, waiting.$$

Fig. 5: ”waits, waiting, waited, waiting” har alla samma ordstam ”wait”.

Vid användning av textdokument som data inför textklassificering av olika dokumenttyper uppkommer det ofta dupletter av ord eller ord som är morfologiskt lika. Anledningen till att använda sig av NLTK, se Appendix 9.1.3, för att få ordstammen av ett eller flera ord, är för att reducera antalet ord i datan som i sin tur minskar behandlingstiden för programmet. Det i sin tur leder till mer effektiv informationsinhämtning för programmet. (Biba M., Gjati E. 2014, 186).

2.2.3.2 Neuronnät

På sin högsta abstraktionsnivå är ett neuronnät ett nätverk av neuroner som tar en eller flera inparametrar och ger ett eller flera värden som utmatning, beroende på vad det är neuronnätverket skall göra. Neuronerna är lagrade i vektorer som ska representera lager, där det första lagret av neuroner skall representera neuronerna för inparameter och det sista lagret neuroner skall representera resultatet av nätverkets beräkningar. Lagren emellan det första och det sista lagret kallas gömda lager.

I det gömda lagret har varje neuron en koppling till neuronerna i de intilliggande lagerna, före och efter. Beroende på konstruktionen av nätverket har dessa kopplingar ett beroende på kommande resultat, från noderna i lagret efter, och/eller resultatet i de föregående noderna samt en partisk nod. Den partiska noden tillåter nätverket att bättre efterlikna datan som nätverket skall träna på.

Träningen av ett neuronnät går ut på att justera vikterna för kopplingarna mellan neuronerna för att bättre få det resultat som förväntas med hänsyn till ett korrekt svar. Hur bra estimerat ett neuronnätverket är, förutsatt att nätverket har träningsdata som anses korrekt, är beroende av kostnaden från kostnadsfunktionen som beräknar fram ett nytt värde till kopplingarna mellan neuronerna.

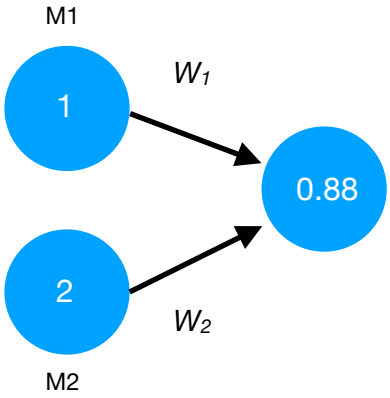


Fig. 6: Resultatet av två mätningar, M1 och M2 , multipliceras med respektive koppling, W1 och W2. Resultatet adderas samman med värdet från en partisk nod, ej i bild, och stoppas in i en sigmoid funktion för att returnera ett värde mellan 0 och 1.

I Fig. 6 illustreras principen av hur ett enkelt neuronnät fungerar. Viktigt att poängtera är att till en början, innan träning av nätet har genomförts, får kopplingarna och värdet från den partiska neuronen ett startvärde som har valt slumpmässigt. För att korrigera vikterna för kopplingarna mellan neuronerna jämför en funktion värdet från dess beräkning, mot ett svar som anses korrekt. Funktionen bestämmer då hur långt ifrån neuronnätet är från svaret och illustrerar det genom att benämna avståndet från svaret genom en kostnad, varav funktionen får namnet kostnadsfunktion.

Vidare, för att minska kostnaden måste funktionen utföra någon form av beräkning. En metod för att göra det kallas MSE (Mean Squared Error), där metoden beräknar fram differensen mellan alla datapunkter och deras respektive korrekta svar, beräknar fram längden emellan de två punkterna genom absolutbeloppet av differensen och beräknar fram medeltalet genom att dividera med antalet datapunkter behandlade, se Ekv. 20.

$$MSE = \frac{1}{n}\sum_{i}(y_i - (m\,x_i + b))^2$$

Ekv. 20: Mean Squared Error funktion, y = korrekt svar, x = datapunkt, m = synapsvikt, b = partiskt värde.

För att kunna optimera denna kostnadsfunktion över alla datapunkter inför man en generell funktion som kallas gradientmetoden (*eng. Gradient descent)*. Anledningen till att använda gradient descent är med att ett neuronnät inte är ett linjärt system, det vill säga att det blir väldigt svårt att hitta en minsta punkt i ett icke-linjärt system genom att använda sig av en utgångspunkt och beräkna den minsta kostnaden iterativt enbart med en kostnadsfunktion.

Gradientmetoden fungerar på så vis att funktionen börjar med en slumpmässigt vald gissning på parametrarna för systemet och börjar med vald gissning. Vidare beräknar funktionen åt vilket håll systemet sluttar nedåt mest och väljer att anta en ny position, ett nytt värde, en bråkdel av nämnt håll och iterativt uppdatera positionen med avseende på lägsta nästa punkt och fortsätter tills en absoluta lägsta punkt av systemet anses ha nåtts. För att beräkna fram varje nytt värde funktionen skall anta måste funktionen beräkna fram gradienten av kostnadsfunktionen. Detta gör den genom att beräkna fram den partiella derivatan av respektive värde och spara varje partiell derivata i en vektor. På så vis håller vektorn varje sluttning för varje värde i systemet.

Enbart den partiella derivatan, som beskriver sluttningen för ett specifikt värde i systemet, är inte tillräckligt för gradientmetoden att avgöra sin nedstigning åt det håll den anser är bäst att gå. Eftersom i gradientmetoden tas en bråkdel av avståndet nedåt, i den sluttning som anses bäst passande, måste en ny parameter bestämma hur stor bråkdelen skall vara. Denna parameter kallas ofta inlärningshastighet (*eng. Learning rate)* och är en hyperparameter, som bestäms manuellt, till systemet. Denna hyperparameter är väldigt känslig, om denna är satt till ett värde som är för litet så kommer det att ta oerhört lång tid att nå systemets lägsta punkt och ifall det är satt för stort finns det risk att passera eller stiga uppåt i systemet.

Utöver gradientmetoden och inlärningshastighet finns det olika metoder som ämnar sig bättre för olika former av situationer, exempelvis mini-batch gradient descent. Där neuronnät som ska behandla stora mängder data inte effektivt kan tränas med hjälp av gradient descent om denna skall gå igenom varje datapunkt iterativt för att lära neuronerna och vikta synapserna.

2.2.3.3 Implementering

I Fig. 7 visas en översiktlig representation av implementeringen av neuronnätet för textklassificering. I detta kapitel beskrivs implementeringen av 2.2.3. Vilka hjälpmoduler som har använts, miljö som har utvecklats i och programmets metoder och hur de samverkar.

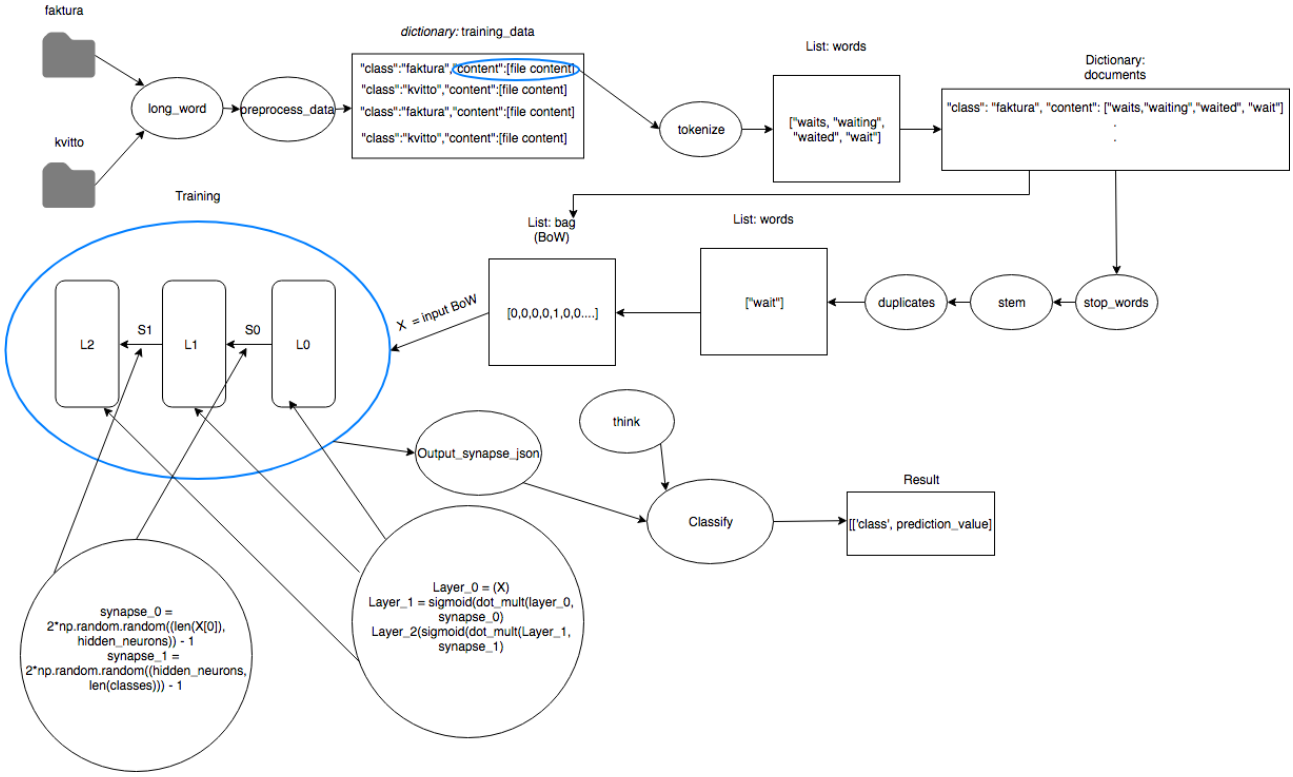


Fig. 7: En översiktlig representation av implementeringen av neuronnät för textklassificering.

För att underlätta utvecklingen och testningen av programmet användes modulen Jupyter Notebook, se Appendix 9.3.4. Underlag för träning och testning laddades upp till modulen i separata mappar, underlaget bestod av klartext-filer, se Kap. 2.1.2 för mer information om konstruktionen av filerna.

Vidare, innan dess att klartext-filernas innehåll blev till användning som träningsdata och testdata var det nödvändigt att rensa innehållet från eventuella bitar av information som inte ansågs tillföra värde. Metoden *long_word* filtrerade ut ord som bestod av fler än 20 karaktärer, då vid processen av OCR i Kap. 2.1.2 i fåtal fall medförde långa strängar text, som inte var ord, till klartexten. I metoden *preprocess_data* lästes klartext-filerna in, med möjlighet att bestämma antal filer, och strukturerades sedan i en dictionary, se terminologi, variabel *training_data* se Fig. 8.

”Class”:”faktura”, ”content”: [file content]

Fig. 8: Ett exempel på innehållet av variabeln training_data, [file content] är innehållet av klartext-filen och faktura är dokumentets klassificering.

Textinnehållet från *training_data,* kopplat till nyckeln ”content”, separerades sedan ord för ord med hjälp av modulen NLTK tokenize, se Fig. 9. Metoden tokenize separerar en sträng och returnerar en lista, en vektor i Python, med respektive ord i strängen separat i listan. Varje samling karaktärer, i strängen, som separerats med white-space tecken tolkas som ett ord och returneras på följande vis.

String = ”This is a sentence”
List = [”This”, ”is”, ”a”, ”sentence”]

Fig. 9: Representation av resultatet som returneras till följd av att nltk.tokenize(string) har exekverats.

Ordlistorna lades sedan till i en dictionary variabel *documents* med tillhörande klass. Variabeln används sedan i skedet att skapa den inparameter till neuronnätet, se Kap. 2.2.3.1. Vektorn som skall representera inparametern till neuronnätet i senare skede skapas genom att gå igenom innehållet av varje, så kallat, dokument i variabeln *documents.* Varvid innehållet, som består av en lista med ord, behandlas med hjälp av metoden *stop_words* och *stem*, också från modulen NLTK. För vidare förklaring av stem, se Kap. 2.2.3.1. Metoden *stop_words* filtrerar ut betydelsefattiga ord med hjälp av en lista över svenska ord, med anledning att enbart behålla ord vars innebörd skall ha en påverkan på beslutet av neuronnätet. När väl ordlistan har filtrerats på betydelsefattiga ord och beskurits med metoden *stem* återstår enbart ordstammarna av de ord som har en innebörd som kan påverka dokumentets betydelse. Ordlistan görs om från en vektor av ord till en mängd av ord, vilket i sin tur tar bort eventuella dupletter. Vid detta skede traverseras respektive plats i ordlistan och listan som representerar Bag of Words och lägger till en nolla för varje ord. Senare, då ett nytt dokument tillkommer för jämförelse, kommer ordlistan att kollas igenom och vid en matchning kommer ordets plats i Bag of Words listan ersättas med en etta, se Fig. 10.

Bow = [0,0,0,1,…,0]

Fig. 10: En lista, vid namn bow, vars innehåll är antingen nollor eller ettor. Varje siffra representerar ett ord i en ordlista, en etta visar att förekomsten av ett ord i ett inkommande dokument existerar i ordlistan och en nolla dess avsaknad. Siffervektorn används som inparameter till neuronnätet.

När klartexten väl har behandlats och sammanställningen av Bag of Words vektorn är färdig, är det möjligt att påbörja konstrueringen av neuronnätet och dess träning. Metoden *train* tar som inparameter listan för Bag of Words, en vektor för respektive dokuments korrekta klassificering,

```
train(X, y, hidden_neurons=10, alpha=0.1, epochs=50000, dropout=False, dropout_percent=0.2)
```

Fig. 11: Metodspecifikationen för metoden train.

antalet neuroner i det gömda lagret, hyperparameter för inlärningshastighet, antalet epoker (se terminologi), samt ifall en dropout (se terminologi) skall användas och en hyperparameter för procentuell dropout.
I metoden *train,* se Fig. 11, skapas också variabler som ska representera synapser, de viktade kopplingarna mellan neuronerna. Med hjälp av metoden *random.random*, som returnerar ett decimalvärde mellan i intervallet [0.0, 1.0)*,* i modulen numpy, se Appendix 9.1.5, kan variablerna initialiseras slumpmässigt. Lagerna av neuroner initialiseras i en loop, vars längd bestäms av inparametern för antalet epoker som ska köras. Det första lagret tar inparametern, listan från Bag of Words och omvandlar det till en numpy array. Det andra lagret tar en skalärprodukt av det första lagret och dess synapser, produkten stoppas i en sigmoid funktion, en logistik funktion som modellerar en S-kurva mellan 0 och 1, och returneras till det andra lagret. Likaså gör det tredje lagret, med det andra lagret och dess synapser. Resultatet som returneras till det tredje lagret är den slutgiltiga klassificeringen av aktuellt dokument.

Själva träningen, justeringen av de viktade synapserna, går till genom att jämföra resultatet i det sista lagret mot den korrekta klassificeringen och spara värdet i en variabel *layer_2_error.* Resultatet som är sparat i *layer_2_error* indikerar ifall systemets nuvarande viktning av synapser och lager är väldigt mycket fel från det korrekta svaret eller ej. För att undvika att inte ta för stort avsteg från eventuellt korrekt svar sparas produkten av värdet i *layer_2_error* och derivatan av sigmoidfunktionen med resultatet från det sista lagret som inparameter till funktionen i en variabel *layer_2_delta*.

$$layer\_2\_delta = layer\_2\_error \cdot sigmoid\_derivative(layer\_2)$$

```
def sigmoid_derivative(output):
    return output*(1-output)
```

Fig. 12: Första raden illustrerar hur det nya värdet för synapserna i neuronnätet mellan det gömda lagret och det sista lagret tas fram. Andra raden visar metoden för att beräkna derivatan av sigmoidfunktionen med avseende på det sista lagrets resultat.

Vidare måste samma korrigering ske bakåt genom neuronnätet, så att de viktade synapser i tidigare lager som påverkat resultatet i kommande lager korrigeras följaktligen. För att ta fram hur pass mycket fel det tidigare lagret bidrog till klassificeringen, används resultatet av *layer_2_delta* för att ta fram en skalärprodukt mellan resultatet och värdet från synapserna mellan det gömda lagret och det sista lagret, produkten sparas i en variabel *layer_1_error*. Processen av att beräkna fram delta för lager 1 görs på samma sätt som i Fig. 12.

Uppdateringen av synapserna, korrigeringen av deras viktade värden, görs i två steg. Först tas skalärprodukten mellan det föregående lagrets värde och värdet av delta för det nuvarande lagret som beräknades fram ovan, resultatet sparas i en variabel *synapse_1_weight_update.* Variabeln används sedan till att multipliceras med hyperparametern alpha, inlärningshastigheten, produkten adderas till värdet synapserna redan besitter, se Fig. 13.

```
synapse_1_weight_update = (layer_1.T.dot(layer_2_delta))
...
...
...
synapse_1 += alpha * synapse_1_weight_update
```

Fig. 13: En illustration av uppdateringen av synapsernas vikter, ett annat ord för värdet.

I detta skede har neuronnätet iterativt gått igenom alla dokument, deras innehåll, behandlat och uppdaterat neuronnätet för att bättre passa den information den har behandlat och de klassificeringar det väntas göra. Den återstående processen består av att spara värdet av de nu uppdaterade synapserna av neuronnätet. Detta görs genom att spara värdena till en .json fil, formatet illustreras i Fig. 14.

```
synapse = {'synapse0': synapse_0.tolist(), 'synapse1': synapse_1.tolist(),
           'datetime': now.strftime("%Y-%m-%d %H:%M"),
           'words': words,
           'classes': classes
          }
```

Fig. 14: Synapsernas värden sparas i listor i en .json fil enligt formatet ovan. Datum, tid, alla ord använda samt klasserna vilket neuronnätet kan klassificera är alla även sparade i synapserna.

Neuronnätet är nu färdigställt, dess träning fullbordad och dess vikter sparade på så vis att det tillåter att använda neuronnätet så att det inte behövs tränas på nytt vid klassificering. För klassificeringen använd en metod *classify* och en hjälpmetod *think*. Hjälpmetoden skapar en Bag of Words av det dokument som avses klassificering och matar detta genom neuronnätet genom att ta det omvandlade dokumentet in i det första lagret och successivt ta skalärprodukten igenom samma sigmoidfunktion som tidigare använts genom respektive lager. Metoden *classify* sorterar ut och presenterar resultatet som *think* returnerade, se Fig. 15.

```
classification: [['faktura', 0.989…64062381]]
```

Fig. 15: Formatet på vilket metoden classify presenterar efter en klassificering av ett dokument.

3. Metod

Detta kapitel behandlar de faser examensarbetet har genomgått samt den arbetsmetod författarna för detta examensarbete har valt att följa genom arbetet. Kommunikation mellan författarna samt motivering av valen gjorda av författarna i deras arbete är också en del av detta kapitel.

3.1 Faser

Faser avser de olika perioder författarna har arbetat i för att genomföra detta examensarbete. Detta examensarbete består av 3 faser, varav varje fas består av 3 delfaser. Varje fas representeras av en av de tre maskininlärningsalgoritmerna i detta examensarbetet, se fig. 16. Delfaserna är: Studier och inhämtning av information, Implementering och testning samt Dokumentation. Efter att dokumentation har skett, till följd av lyckad implementering och testning i den tidigare delfasen, påbörjades en ny fas. Förklarat i Kap. 3.2 och som illustrerat i Fig. 17 var arbetet genomfört i iterationer, där författarna valt att se varje iteration som en övergripande fas.

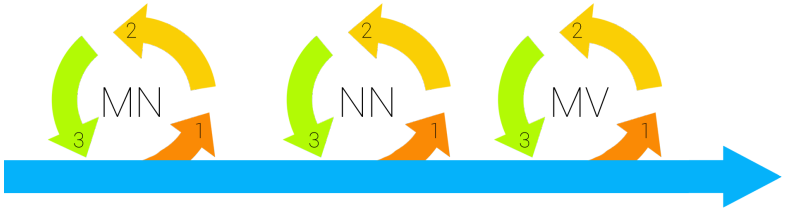


Fig. 16: Illustration av faser och delfaser. Faserna från vänster: Multinomial Naive Bayes, Neural Network, Multivariate Bernoulli Naive Bayes. Representerat av MN, NN och MV respektive. Delfaserna numrerade 1 - 3 respektive: Studier och inhämtning av information, Implementering och testning samt Dokumentation. Den blå pilen representerar examensarbetets riktning, att faserna har genomgåtts i den riktning som pilen visar i denna figur.

3.1.1 Studier och inhämtning av information

De delfaser som arbetet har genomgått presenteras under denna punkt, Kap. 3.1.2 samt Kap 3.1.3. Delfaserna som tillhör Kap. 3.1.1 är studier och inlärning och källor.

3.1.1.1 Studier och inlärning

Processen som de olika faserna genomgick var iterativa, se Kap. 3.2.1. I olika iterationer genomfördes studier om maskininlärning för att först skapa en generell uppfattning om på bästa sätt hantera och bemöta examensarbetets problemställningar samt det mål och syfte examensarbetet har. Studierna inleddes med att skapa en uppfattning om vad maskininlärning innebär och vikten vid formatering av data, med hjälp av boken ”Building machine learning systems with Python” av Richert och Coelho (2013).

Vid varje ny iteration behövdes studier och inlärning av olika maskininlärningsmetoder preliminärt för att avgöra om en möjlig metod var till användning samt ifall det var möjligt att sätta sig in i och implementera inom ramen för examensarbetet. Studierna gav också förutsättningen för diskussion mellan författarna för att kunna avgöra vad som är en möjlig lösning och planera arbetet i sin helhet.

3.1.1.2 Källor

Informationen som användes för inlärning har direkt påverkan på de val som togs i arbetet, gällande möjliga algoritmer och tillvägagångssätt kring implementering. Därav var enbart källor som har genomgått någon form av prövning, som exempelvis en publicerad akademisk artikel eller bok, en del av den information som användes vid inlärning.

Förklaringar av termer samt inspiration för potentiella lösningar har även tagits del av genom forum, såsom Stackoverflow, och artiklar, såsom Medium, på Internet. Den information som inte har haft någon väl beprövad källa, har uteslutits från denna rapport.

3.1.2 Implementering och testning

I denna fas presenteras delfaserna implementering, test och resultat.

3.1.2.1 Implementering

Fasen för implementering överlappar delvis med studier och inhämtning av information med det avseende att implementering påbörjades i takt med att information inhämtades, som en del av inlärningsprocessen.

Maskininlärningsmetoderna baserade på Naive Bayes teorem utgick från den matematiska teori som hade inhämtats. Implementeringen skedde utan några externa bibliotek eller moduler som inte tillhörde Pythons standardbibliotek. Genom att inte använda många av de färdiga bibliotek som finns tillgängliga, krävdes det en djupare förståelse för hur algoritmerna fungerar. De metoder som skapades för programmet baserades på de regler och ekvationer som studierna av Naive Bayes visade. Generellt delades implementationen av Naive Bayes algoritmerna upp i tre delar, bestående av:

I. Inhämtning av data , sortering och beräkning av priorisannolikheten, skapandet av ett set innehållande alla unika ord, samt skapandet av alla hjälpmetoder för programmet.
II. Beräkning av alla ords betingade sannolikheter med hjälp av informationen tidigare inhämtad. Dessa sannolikheter skrevs sedan till CSV filer beskrivet i Kap. 2.2.1.5 för användning vid test.
III. Skapande av metoder för testning och resultat framtagning som följer beskrivningen i ekvation 15.

Innan en riktigt implementering gjordes med en verklig datamängd så användes en mindre, beprövad, datamängd. Detta gav fördelen att det gick att manuellt kontrollräkna så att programmet gav ett matematiskt korrekt resultat.

För implementeringen av neuronnätet se 2.2.3.3. Utöver själva implementeringen av neuronnätet var inte tillvägagångssättet helt självklart för att definiera om neuronnätet var färdigt med sin träning. Processen av att träna neuronnätet var utdragen och efter en mängd tester, med avseende på felfrekvensen och hur väl neuronnätet presterade, så fattades beslutet att en felfrekvens >0.002 var acceptabel för att kunna vidare testa mängden gömda neuroner och storleken av datamängden som användes för träning.

3.1.2.2 Test och resultat

Testerna utfördes enligt kolumnbeskrivningen i Tab. 5-9 i Kap. 5.1 för neuronnätet samt Tab.10 -11 i Kap. 5.2, 5.3 för de Naive Bayes baserade algoritmerna. Testerna byggdes upp med parametrar så att resultatet skulle bli överskådligt och rättvist. När det kommer till Naive Bayes var testerna enkla att genomföra då det enda som krävdes var att byta ut den hämtade och beprövade datamängden mot den verkliga datamängden.

Utformandet av fasen för test och resultat påverkades till stor del av problemformuleringen. Då det inte var möjligt att införskaffa mer av den data tillgänglig än de 600 dokument som användes i Kap. 5.1 var det intressant att se hur resultaten skulle bli påverkade då det fanns en betydligt större datamängd för fakturor än det fanns för kvitton. Testerna för neuronnätet sattes upp på så vis att mängden data, som visas i tabell 5-9 i Kap. 5.1, för fakturor och kvitton var jämlika till en början

och sedan successivt utökade faktura datamängden för träning för neuronnätet. Då det inte fanns något officiellt krav på mängden data för träningen av ett neuronnät, förutom det att det måste finnas data, utgick författarna från att neuronnätets kapacitet att klassificera testdokumenten var det som avgjorde hur mycket data som skulle vara tillräcklig.

3.1.3 Dokumentation

Delfaserna som har varit del av dokumentationsfasen är initial rapportbeskrivning och rapportskrivning.

3.1.3.1 Initial rapportbeskrivning

Den första fasen av dokumentering bestod av den initiala rapportbeskrivningen. Efter ett första inskickat utkast och ett möte med både handledare och examinator så uppdaterades den initiala rapportbeskrivningen och lade grunden för det fortsatta arbetet. Mycket av planeringen av hur arbetet skulle ske gjordes i samband med den initiala rapportbeskrivningen, vilket skapade en bättre säkerhet att rätt spår följdes. De flesta delar i arbetet blev på så vis genomtänkta från start och underlättade eventuella problem som skulle kunna ha uppstått.
Rapportbeskrivningens skapande följde samma arbetssätt som beskrivet i Kap. 3.2.

3.1.3.2 Rapportskrivning

Som tidsplanen i Appendix 9.2.2 visar har rapportskrivningen utförts löpande genom arbetets gång. Efter att en initial beskrivning slutförts påbörjades studier och inlärning beskrivet i 3.2 för att sedan övergå till en implementeringen av de olika metoderna för klassificering. Efterhand som en implementering slutfört så påbörjades rapportens tekniska bakgrund för respektive metod. Detta var ett effektivt sätt att skriva rapporten då all fakta och inlärning nyligen var inhämtad och kunde beskrivas korrekt i rapporten utan att ägna tid att gå tillbaka till ett tidigare stadie. Efterhand som den tekniska implementationen blev klar kunde resultat skrivas för att sedan övergå till skrivandet av slutsatsen. Under rapportens gång har appendix, terminologi och referenser fyllts på samt eventuella formateringar ändrats för att skapa en enhetlighet och en bättre läsbarhet i rapporten. Eftersom kapitlet metod sträcker sig över hela arbetet så avslutades rapporten med det kapitlet. Valet motiveras av att ju närmare slutet arbetet närmar sig, desto starkare bild har skapats för att i slutändan korrekt kunna återge hur arbetet har utförts.

3.2 Arbetsmetod

Arbetsmetoden är det val av planering, arbete och fullföljande av examensarbete författarna har valt att förhålla sig till.

3.2.1 Iterativa självstudier och implementering

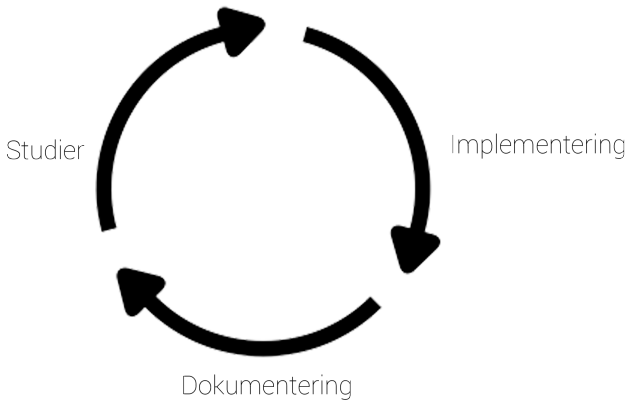


Fig. 17: Illustration av arbetssättet använt i detta examensarbete.

Det arbetssätt som har valts för detta examensarbete är en form av iterativ arbetsprocess (Larman, C., 2004), se Fig. 17, tillsammans med parprogrammering, se Kap. 3.2.2. Motiveringen till detta val är bland annat att författarna för examensarbetet har arbetat i par och på samma plats. I sin guide till scrum skriver grundarna, Ken Schwaber och Jeff Sutherland (2017), att mindre än tre medlemmar i en scrum grupp leder till mindre interaktion och sämre produktivitet, därav valet av en egen modifierad arbetsprocess. En anledning att inte använda Kanban är att även det var för formellt (Anderson, 2010), ifall en tidig indikation hade antytt att den valda arbetsmetoden inte hade fungerat hade det varit mer relevant att överväga Kanban som arbetsmetod.

Den iterativa arbetsmetoden bestod av att författarna självständigt studerade och skapade en uppfattning för att sedan tillsammans diskutera möjlig implementering och genom parprogrammering implementera den valda lösningen. Arbetet som implementerades delades upp emellan författarna till examensarbetet och dokumenterades var för sig.

3.2.2 Parprogrammering och kommunikation

I det iterativa arbetet, vid implementering och inhämtning av information, har parprogrammering varit den främsta arbetsmetoden. Termen parprogrammering i detta sammanhang innebär bland annat att båda författarna har arbetat vid en och samma dator. En författare har skrivit kod och den andra granskat den förstes arbete, medan denna utförde det. Utöver granskningen spenderade också den andra författaren tid på inhämtning av information, för eventuellt bättre lösningar eller lösningar på problem som har uppstått. Valet av parprogrammering som metod, kontra individuellt arbete, kommer från att författarna båda tidigare har erfarenhet av denna arbetsmetod tillsammans. Vidare finns det även kvantitativ data som underlag till att parprogrammering ofta producerar bättre resultat. (Williams et al., 2000)

Fördelarna med denna arbetsmetod är att kommunikationen emellan de båda författarna ofta är frekvent, där bland annat granskning av den kod som då skrevs var en av de mest utstående sakerna som samtalades. Dock var inte alltid denna arbetsmetod passande. Vid tillfällen där eventuella fel i

kod uppstod eller dokumentering i rapport inte gick att formulera tillsammans var en betydligt bättre lösning att inhämta information var för sig respektive. Informationen diskuterades sedan i efterhand tillsammans, men behövde inte nödvändigtvis implementeras tillsammans.

3.3 Källkritik

Källorna använda i detta examensarbete kommer att utvärderas i detta kapitel. Källorna är uppdelade härunder efter:

- Böcker och E-böcker,
- Artiklar, journaler och Publiceringar.
- Hemsidor

I Kap. 3.3.1, Kap. 3.3.2 och Kap. 3.3.3 respektive. Källorna är uppdelade i detta omfång med motivering att varje omfång har liknande publiceringskrav.

3.3.1 Böcker och E-böcker

I listan nedan refereras alla källor använda som är en bok eller en elektronisk bok, också kallat E-bok. Anledning till att lita på dessa källor är att de har publicerats av ett förlag och har därav genomgått granskning före publicering.

- Anderson, D. (2010)
- Jurafsky, D., & Martin, J. H. (2009)
- Harrington, P. (2012).
- Larman, C. (2004)
- Richert, W. and Coelho, L. (2013)
- Schwaber, K. and Sutherland, J. (2017)
- Richert, W. and Coelho, L. (2013)
- Dan Morris (2016)

3.3.2 Artiklar, journaler och publiceringar

I listan nedan presenteras de källor använda som är en artikel, del i en journal eller en annan form av publicering.

- Biba M., Gjati E. (2014)
- Bird, S., & Loper, E. (2004)
- Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002)
- Lewis, D. D. (1998).
- A. P. Dawid, a. (1979)
- Metsis, V., Androutsopoulos I. and Paliouras G.(2006)
- Rennie, J. D., Shih, L., Teevan, J., & Karger, D. R. (2003).
- Shibata, K., & Ikeda, Y. (2009, August).
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014)
- Williams, L., Kessler, R., Cunningham, W. and Jeffries, R. (2000)
- Zhang, H. (2004).
- Zhang, Y., Jin, R., & Zhou, Z. (2010)

Följande källor, utav de listade ovan, är publicerade av Springer: *Biba M., Gjati E., Lewis, D. D., Jin, R., & Zhou, Z.* Springer är en samlingsplats med mer än 3400 olika vetenskapliga tidskrifter.

Artiklarna skrivna här tillhör var för sig en vetenskaplig tidskrift. Dessa kan anses trovärdiga då vetenskapliga tidskrifter granskas innan publicering.

Källan, av författare Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P., är publicerad i *Journal of Artificial Intelligence Research* (JAIR) en vetenskaplig tidskrift. JAIR granskar de artiklar som ansöker om att publiceras. Med samma anledning som artiklarna publicerade av Spring kan denna källa också anses trovärdig.

Av de källor som har publicerats av Universitet finns Zhang, H från *University of New Brunswick* samt författarna Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. från *University of Toronto*. Publicerad av *Massachusetts Institute of Technology* (MIT) är artikeln av författarna Rennie, J. D., Shih, L., Teevan, J., & Karger, D. R. Informationen om olika Naive Bayes baserade maskininlärningsalgoritmer presenterade av författarna Metsis, V., Androutsopoulos I. & Paliouras G är publicerad av *Institute of Informatics and Telecommunications* samt *Department of Informatics, Athens University of Economics and Business.* Informationen om riktlinjer kring bestämmandet av antalet neuroner i ett neuronnät och hur det påverkar inlärningsförmågan av detta neuronnät som presenterades i artikeln av författarna Shibata, K., & Ikeda, Y har publicerats av *Oita University.* Dessa källor kan anses trovärdiga då de har granskats före publicering av respektive universitet.

Informationen inhämtad om NLTK, av författarna Bird, S., & Loper, E., beskriver sitt arbete i framtagandet av modulen som ska använda språkbehandlingstekniker i Python. Modulen togs fram av författarna och utöver denna artikel har också författarna skrivet en bok inom samma ämne. Utöver den granskning som görs inför publicering är också modulen och arbetet refererat till i detta examensarbete information om författarnas egen mjukvara. Med denna anledning kan informationen anses trovärdig.

3.3.3 Hemsidor

I listan nedan presenteras de källor använda som är hämtade från Internet.

Python Software Foundation, Jupiter Team, The Scipy community är alla källor som presenterar information om deras dokumentation. Dessa kan anses trovärdiga då dokumentationen är ägnad deras egen skapelse, felaktig dokumentation skulle inte nyttja någon.

Nationalencyklopedin kan anses trovärdig då den kom till på initiativ av Riksdagen, granskas internt av egen redaktion och har hundra tusentals användare i Sverige, såväl privat som i skola.

Informationen använd från Riksdagen och Skatteverket är båda lagstiftning och kan därför anses trovärdiga.

IETF (Internet Engineering Task Force) är den enhet som definierar standard Internet operativ-protokoll som TCP / IP. De granskas av Internet Society som är en internationell ideell organisation som fungerar som vägledning för Internets arbete.

- Nationalencyklopedin
- Riksdagen
- Skatteverket
- IETF

- Python Software Foundation
- Jupyter Team
- The Scipy community

4. Analys

De beslut som har tagits vad gäller programmeringsspråk, maskininlärningsmetoder, val för dataformatering moduler och hjälpmedel presenteras i detta kapitel.

4.1 Python

Valet av programmeringsspråk gjordes utifrån tillgänglighet, både med avseende på möjligheterna att utföra maskininlärningsalgoritmer utan att behöva återskapa vartenda del av respektive algoritm samt den uppsjö av dokumentation och litteratur det finns för både maskininlärning med Python och alla dess moduler. Utöver detta är mängden av moduler baserade på C eller C++, som i många fall resulterar i snabb och robust kod även i praktisk användning. (Coelho och Richert, 2013)

Förutom dess moduler är programmet väldigt enkelt att överskåda, då programmets syntax är enkelt att sätta sig in i, samt för att göra enkla matrismultiplikationer och andra matrisoperander kräver Python inte många rader kod, till skillnad från Java eller C. (Harrington, 2012)

4.2 Maskininlärningsmetoder

För att möjligen kunna separera ett dokument från ett annat och bestämma ifall det tillhör klassen kvitto eller klassen faktura måste någon form av identifierbar aspekt av innehållet i texten av dokumenten finnas. Specifika ord som enbart förekommer i samband med en typ av dokument är en av dessa klassidentifierare. En annan är kompositionen av ett dokument och vilka ord som förekommer i dokumentet.

Det finns en mängd maskininlärningsalgoritmer för att lösa klassificeringsproblem. I detta fall har valet av algoritm varit med hänsyn till möjligheten att implementera och grundligt testa olika algoritmers lämplighet för att förstå innebörden av dokumenten i den tidsram möjlig för detta examensarbete. Frågan om vilken algoritm som hade passat bäst är en väldigt bred fråga då algoritmers förmåga att klassificera kan vara olika från ett dataset till ett annat. Därav kommer inte algoritmer, förutom de som valts för detta examensarbete, att beskrivas.

Naive Bayes är en familj av algoritmer som alla är baserade på att tillämpa Bayes teorem för att förutse kategorin för ett givet prov. De olika algoritmerna för denna familj gör ett naivt antagande att varje funktion, i detta fall ord, är oberoende av de andra i sin klassificering av ett dokument. Dock har Naive Bayes framgångsrikt tillämpats på flertalet domäner, däribland textklassificering. (Lewis, 1998)

I fallet om att klassificera dokument efter dess innehåll i text, eller innehåll som saknas, passar detta antagande som Multinomial och Multivariate Bernoulli Naive Bayes gör, då vissa ord utmärker sig mer än andra i fakturor och kvitton. Ord som utmärker sig mer än andra, exempelvis OCR i en faktura eller ordet Kvitto i ett kvitto, är en stark indikation på vad för typ av dokument programmet behandlar.

Utöver ord med en indikation av betydelsen av dokumentet och slutligen vad dokumentet är för något är också sammanhanget en aspekt för klassificering. Sammanhanget kan vara kompositionen av ord som förekommer i dokumentet och hur de hänger ihop. Förutom det antagande och sannolikhetsberäkning som Naive Bayes algoritmerna gör kan ett neuronnät lära sig ifrån kompositionen av texten i ett dokument. Neuronnätet har också en fördel med att inte behöva en datamängd för träning som är balanserad mellan de olika klasserna. Resultatet av hur de olika metoderna presterar berättar mer om hur obalanserade datamängder kan påverka algoritmernas prestation.

4.3 Formatering av data

Datan, det underlag till vilket träning och test för de olika modellerna använde, kommer från diverse källor. Eftersom att syftet att klassificera ett godtyckligt dokument som kvitto, faktura eller icke tillhörande de två förstnämnda klasserna, med hjälp av de lösningar presenterade i kapitel 2, behövs det data att träna modellerna på. Som förklarat i Kap. 2.1.2 består datan av textdokument vars innehåll ursprungligen kommer från fakturor och kvitton.

Fakturorna består av 500 dokument, alla vars ursprungliga format har varit digitala, kommer från bland annat Malmö Stad och företaget Omvida AB. Kvittona består av 143 dokument, vars format dels ursprungligen har varit digitala och dels har varit i pappersformat. Med anledning av att det inte fanns någon möjlighet att få åtkomst till fler kvitton i digitalt format, än tillgänglig från privat innehav, fick detta underlag till träning och test av respektive modell presenterad i kapitel 2 bestå.

Gemensamt för alla textklassificerings metoder har datan blivit behandlad före användning för träning eller test. Bag of words metoden valdes eftersom det är en av de mest populära modellerna för objektklassificering, enligt Zhang, Jin och Zhou (2010). Samt användningen, i vissa av fallen i kapitel 2, av NLTK för att få textens ordstammar för att öka effektiviteten av inhämtningen av information. (Biba M., Gjati E. 2014, 186)

4.4 Moduler och hjälpmedel

Nedan presenteras moduler och hjälpmedel som användes för att genomföra detta examensarbete.

4.4.1 CSV(Comma separated values)

Användningen av CSV format, vars värden är separerade med kommatecken, var passande på grund av det format värdena lästes in i (främst listor).Pythons inbyggda CSV läsare möjliggjorde sedan att konvertera dessa till objekt på ett enkelt sätt.

4.4.2 NumPy

NumPy, se Appendix 9.1.5, ger framförallt större effektivitet och mer bekvämlighet än vad pythons egna listobjekt ger. Genom många färdiga inbyggda matrisoperationen så slipper man göra mycket av jobbet själv. Effektiviteten i biblioteket gör framförallt beräkningarna snabbare vilket har stor betydelse om man hanterar större mängd data. Många gånger kan beräkningar med NumPy prestera 25x snabbare i jämförelse med om man inte hade använt biblioteket. (Richert & Coelho)

5. Resultat

I detta kapitel presenteras resultaten av de tester gjorda på de olika potentiella lösningarna presenterade i kapitel 2.

5.1 NLP, BoW och Neuronnätverk

Nedan presenteras resultaten för träning och test för neuronnätet, se 2.2.3. För varje tabell visas parametrarna för inlärning, antalet neuroner i det gömda lagret, systemets felfrekvens, samt resultaten av klassificering på testsviterna för faktura respektive kvitto. Kolumnen *”Sentence”* är en uppdiktad mening text, inte menad att klassificeras till något av de två klasserna där FALSE står för misslyckat test och TRUE står för lyckat test.

För att en klassificering skall räknas som giltig, i resultaten presenterade i detta delkapitel, måste den procentuella förutsägelsen av neuronnätet överstiga 80 procent. Alla träningsförsök genomgår 4000 epoker och måste understiga 0.002 i felfrekvens. Testdatan består av 86 dokument, varav hälften är kvitton och andra hälften är fakturor. Tab. 5 - 9 samt Fig. 18-19, används i Kap. 6.3.

*Tab. 5: Resultat för neuronnät, vid en träningsmängd om **200 dokument**, varav 100 är kvitton. Neuronnätets fullständiga specifikation vid träning samt den procentuella framgången, betecknad ”%Noggrannhet”, presenteras i denna tabell.*

# test	Alpha	# Neuroner i gömt lager	Delta	# Korrekt klassific. Faktura	# Korrekt klassific. Kvitto	# Felaktig Klassific. Faktura	# Felaktig Klassific. Kvitto	% Noggrannhet	Sentence
1	0,095	30	0.0013	37	39	6	4	88,37209	FALSE
2	0,095	60	0.0011	37	43	6	0	93,02325	FALSE
3	0,095	100	0.00076	38	40	5	3	90,69767	FALSE
4	0,095	120	0.00064	37	41	6	2	90,69767	FALSE
5	0,035	240	0.00078	34	40	9	3	86,04651	FALSE
6	0,019	480	0.00093	30	40	13	3	81,39534	TRUE

*Tab. 6: Resultat för neuronnät, vid en träningsmängd om **300 dokument**, varav 100 är kvitton. Neuronnätets fullständiga specifikation vid träning samt den procentuella framgången, betecknad ”%Noggrannhet”, presenteras i denna tabell.*

# test	Alpha	# Neuroner i gömt lager	Delta	# Korrekt klassific. Faktura	# Korrekt klassific. Kvitto	# Felaktig Klassific. Faktura	# Felaktig Klassific. Kvitto	% Noggrannhet	Sentence
1	0,095	30	0.0011	41	41	2	2	95,34883	FALSE
2	0,035	60	0.0018	39	39	4	4	90,69767	FALSE
3	0,035	100	0.0014	40	36	3	7	88,37209	FALSE
4	0,035	120	0.0013	39	39	4	4	90,69767	FALSE
5	0,019	240	0.0015	38	41	5	2	91,86046	FALSE
6	0,019	480	0.0042	40	38	3	5	90,69767	TRUE

*Tab. 7: Resultat för neuronnät, vid en träningsmängd om **400 dokument**, varav 100 är kvitton. Neuronnätets fullständiga specifikation vid träning samt den procentuella framgången, betecknad ”%Noggrannhet”, presenteras i denna tabell.*

# test	Alpha	# Neuroner i gömt lager	Delta	# Korrekt klassific. Faktura	# Korrekt klassific. Kvitto	# Felaktig Klassific. Faktura	# Felaktig Klassific. Kvitto	% Noggrannhet	Sentence
1	0,095	30	0.00086	41	39	2	4	93,02325	FALSE
2	0,065	60	0.00085	42	38	1	5	93,02325	TRUE
3	0,035	100	0.0012	39	39	4	4	90,69767	TRUE
4	0,035	120	0.0010	40	41	3	2	94,18604	TRUE
5	0,019	240	0.0011	41	39	1	4	93,02325	FALSE
6	0,0135	480	0.0019	35	42	8	1	89,53488	TRUE

*Tab. 8: Resultat för neuronnät, vid en träningsmängd om **500 dokument**, varav 100 är kvitton. Neuronnätets fullständiga specifikation vid träning samt den procentuella framgången, betecknad ”%Noggrannhet”, presenteras i denna tabell.*

# test	Alpha	# Neuroner i gömt lager	Delta	# Korrekt klassific. Faktura	# Korrekt klassific. Kvitto	# Felaktig Klassific. Faktura	# Felaktig Klassific. Kvitto	% Noggrannhet	Sentence
1	0,065	30	0.00056	43	38	0	5	94,18604	FALSE
2	0,035	60	0.0011	39	41	4	2	93,02325	TRUE
3	0,015	100	0.0011	39	35	4	8	86,04651	FALSE
4	0,013	120	0.0018	38	40	5	3	90,69767	FALSE
5	0,007	240	0.0017	39	40	4	3	91,86046	FALSE
6	0,0035	480	0.0018	36	39	7	4	87,20930	FALSE

*Tab. 9: Resultat för neuronnät, vid en träningsmängd om **600 dokument**, varav 100 är kvitton. Neuronnätets fullständiga specifikation vid träning samt den procentuella framgången, betecknad ”%Noggrannhet”, presenteras i denna tabell.*

# test	Alpha	# Neuroner i gömt lager	Delta	# Korrekt klassific. Faktura	# Korrekt klassific. Kvitto	# Felaktig Klassific. Faktura	# Felaktig Klassific. Kvitto	% Noggrannhet	Sentence
1	0,025	30	0.0014	42	37	1	6	91,86046	TRUE
2	0,0127	60	0.0019	42	37	1	6	91,86046	FALSE
3	0,0095	100	0.0019	40	39	3	4	91,86046	FALSE
4	0,0095	120	0.0019	41	40	2	3	94,18604	FALSE
5	0,0045	240	0.0018	42	33	1	10	87,20930	FALSE
6	0,00897	480	0,0016	36	39	4	7	87,20930	TRUE

Resultat presenterade i stapeldiagram.

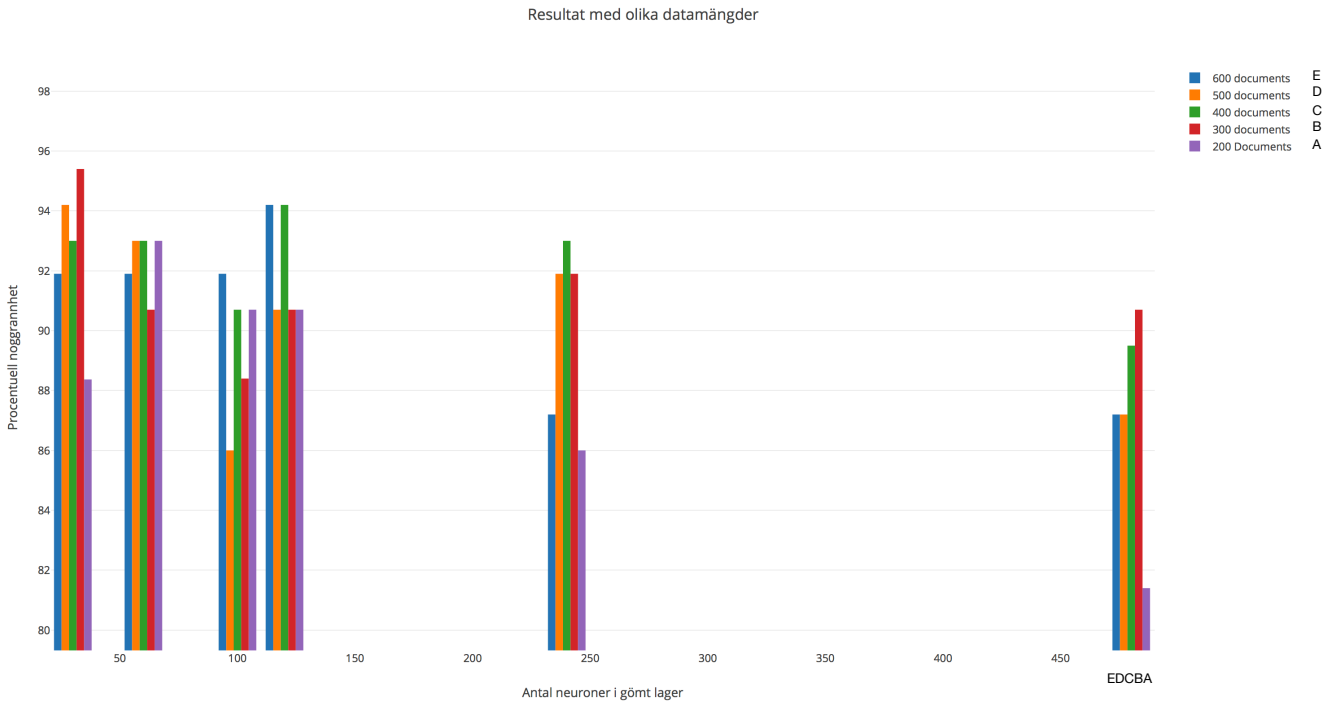


Fig. 18: Stapeldiagram över resultaten presenterade i tabell 5-9 i kapitel 5.1. X-axeln visar antalet neuroner. Y axeln dess procentuella noggrannhet i klassificering.

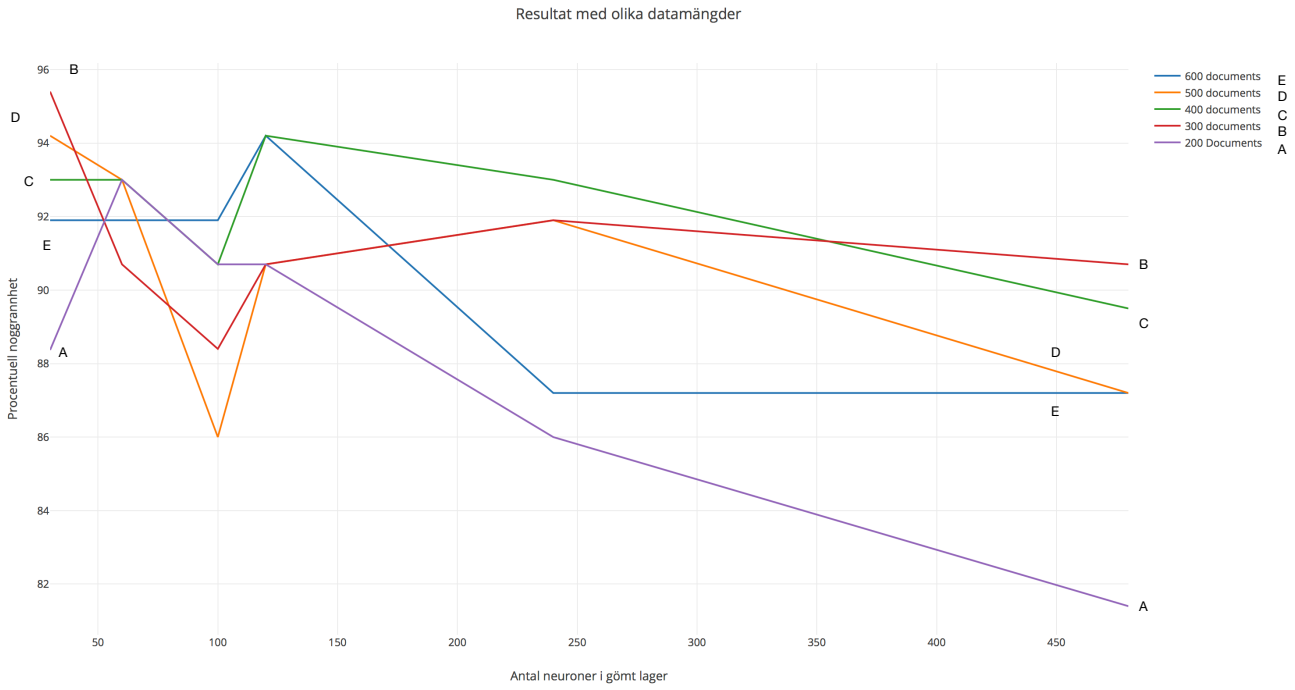


Fig. 19: Linjediagram över resultaten presenterade i tabell 5-9 för kapitel 5.1. X-axeln visar antalet neuroner. Y axeln dess procentuella noggrannhet i klassificering.

5.2 Multinomial Naive Bayes

I tabell 10 presenteras resultaten för träning och test för Multinomial Naive Bayes, se 2.2.2. Tab. 10 visar parametrar för Antalet träningsdokument, antalet testdokument , Andel korrekt klassificering av fakturor, samt kvitton, och den procentuella framgången för klassificeringen av fakturor respektive kvitton.

Tab. 10: Resultat för Multinomail Naive Bayes med olika antal träningsdokument.

Antalet Träningsdokument (Fakturor/ Kvitton)	Antalet Testdokument (Fakturor/ Kvitton)	Andel korrekt klassificering av Fakturor	Andel korrekt klassificering av Kvitton	Procentuell framgång Faktuor	Procentuell framgång Kvitton
100/30	30/30	30	21	100 %	70 %
100/100	30/30	30	16	100 %	53 %
300/100	90/30	90	5	100 %	17 %
500/100	180/30	180	3	100 %	10 %

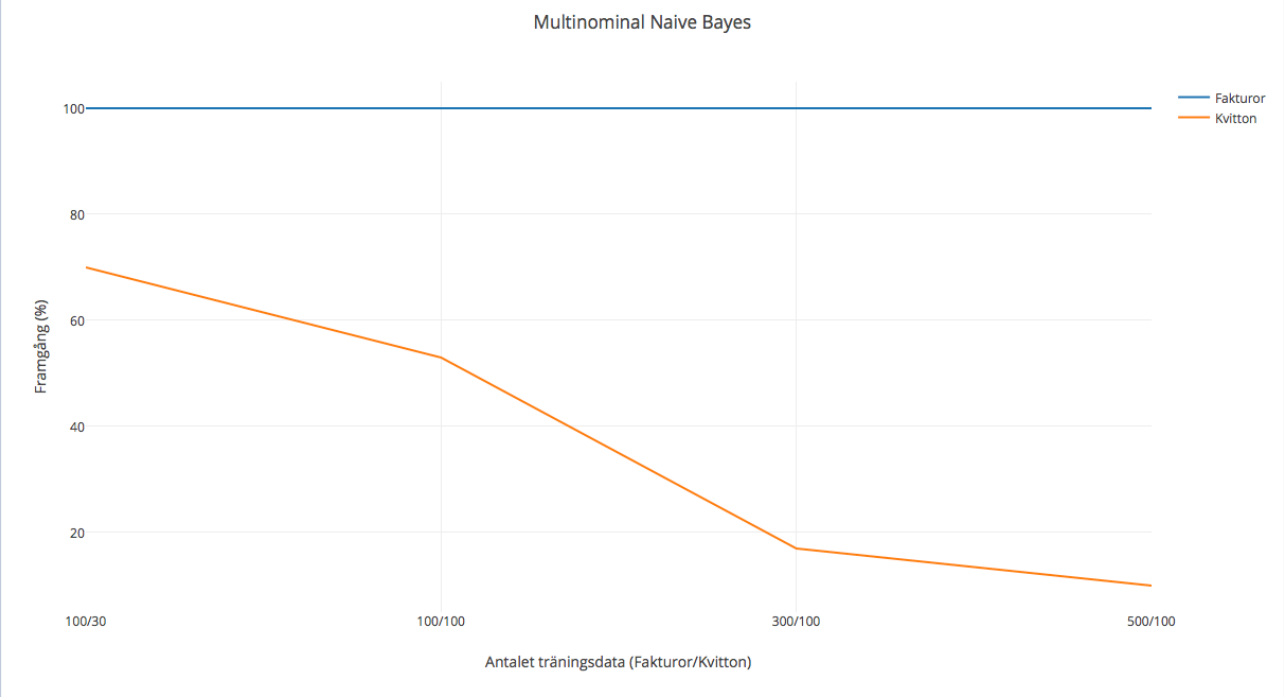


Fig. 20: Illustration av resultatet i Tab. 10.

5.3 Multivariate Bernoulli Naive Bayes.

I tabell 11 presenteras resultaten för träning och test för Multivariate Bernoulli Naive Bayes, se 2.2.2. Tabellen visar parametrar för Antalet träningsdokument, antalet testdokument , Andel korrekt klassificering av fakturor, samt kvitton, och den procentuella framgången för klassificeringen av fakturor respektive kvitton.

Tab. 11: Resultat för Bernoulli Naive Bayes med olika antal träningsdokument.

Antalet Träningsdokument (Fakturor/Kvitton)	Antalet Testdokument (Fakturor/ Kvitton)	Andel korrekt klassificering av Fakturor	Andel korrekt klassificering av Kvitton	Procentuell framgång Faktuor	Procentuell framgång Kvitton
100/30	30/30	25	3	83 %	10 %
100/100	30/30	26	5	87 %	17 %
300/100	90/30	75	4	83 %	13 %
500/100	180/30	159	4	88 %	13 %

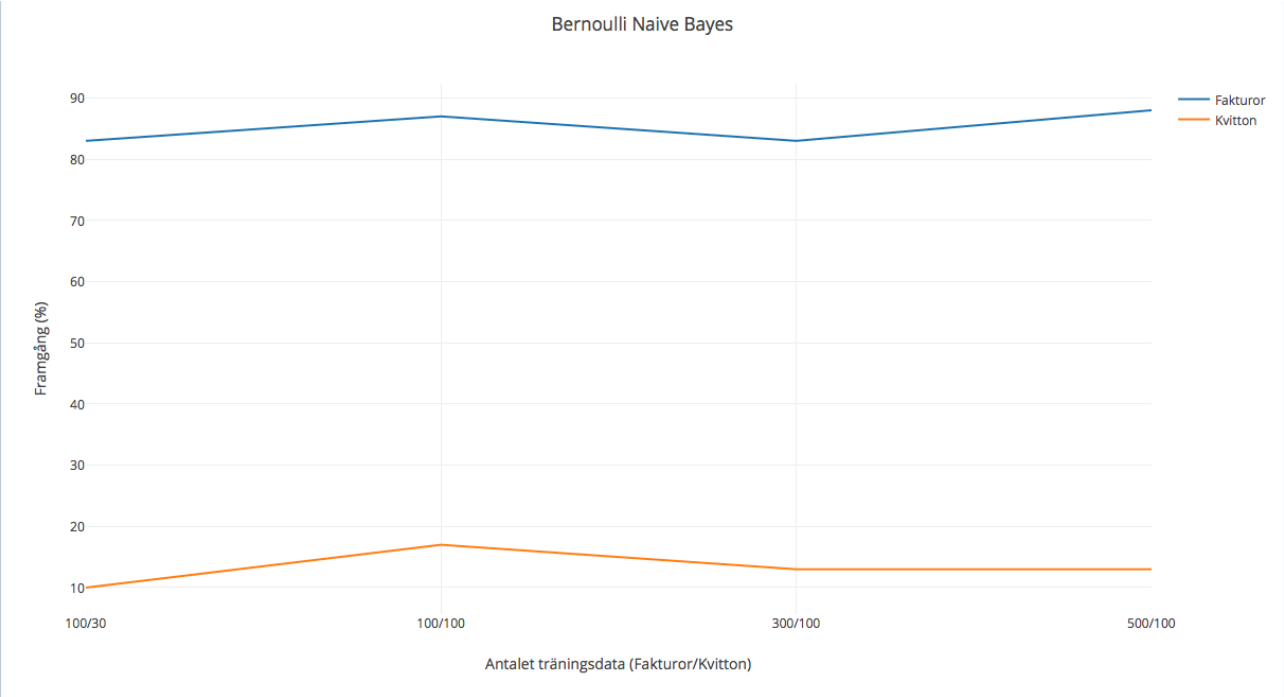


Fig. 21: Illustration av resultat i tab. 11

6. Slutsats

Detta kapitel behandlar de resultat presenterade för respektive maskininläringsalgoritm i Kap. 5.1, 5.2 och 5.3 med avseende på examensarbetets problemformulering samt även formateringen av underlaget för data vid träning av samma maskininlärningsalgoritmer i Kap. 2.1. Följande är problemformuleringen som skall besvaras i detta kapitel:

- Vilken datamängd för två klasser anses vara tillräckligt för träning och testning?
- Hur bör datan eller texten behandlas för att bäst kunna klassificeras?
- Hur mycket av datamängden bör användas till test respektive träning?

6.1 Multinomial

Från resultaten som tabellerna och diagrammet visar så sticker framförallt resultatet från faktura-testerna ut. En horisontell linje som representerar en korrekt klassificering på 100%. Detta resultat sträcker sig oberoende av procentuell fördelning eller mängd av träningsdata för varje klass.

Resultaten för kvitto-testerna såg annorlunda ut och resultaten visade en trend nedåt som även här sträckte sig oberoende av fördelning och mängd. Bäst presterade testerna vid en liten mängd träningsdata och ett mängdförhållande på 3:1, fakturor respektive kvitton, där 70 procent korrekt klassificering uppnåddes. Vid en ökning till ett förhållande på 1:1 så minskade den korrekta klassificeringen till 53 procent och fortsatte därefter ner till 10 procent, då mer träningsdata användes.

Resultaten för kvittoklassificeringen speglar ett oväntat resultat vid första anblicken och skiljer sig väsentligt från ett resultat som kan beaktas som användbart i ett sammanhang beskrivet i kapitel 1. Att klassen fakturor presterade så bra som resultaten visar beror dels på hur fakturorna som används för att träna programmet med består av, samt hur själva algoritmen fungerade när den betingade sannolikheten $P(x \mid \omega_j)$ modellerades.

Multinomial Naive Bayes beräknade, som beskrivet i ekvation 6, frekvensen av ett ords förekomst i ett dokument. När innehållet hanterades i fakturor så finns det alltid nyckelord som ofta förekommer, exempelvis ordet ”faktura”, samt innehåll som måste förekomma i en faktura enligt dom riktlinjer som existerar för hur en faktura faktiskt måste se ut (riksdagen, 2018-05-30). Eftersom dessa förekommer i princip varje dokument så får dessa en betingad sannolikhet som är relativt hög gentemot andra ord. Enligt CSV filerna från varje klass var det möjligt att se hur ordet ”fakturaspecifikation” har en betingad sannolikhet på ca 0.0016 i klassen faktura och en betingad sannolikhet på 0.000005 i klassen kvitto. Dessa frekventa förekommande ord som finns i fakturor gör det alltså väldigt enkelt för programmet att klassificera fakturor eftersom klassen har ord som bara förekommer i fakturor men aldrig hos ett kvitto.

När det kommer till klassen kvitto så finns även här ord som frekvent förekommer, men till skillnad från fakturor så förekommer dessa inte exklusivt hos kvitton. Vanligt förekommande ord i kvitton är exempelvis orden ”totalt” eller ”summa”, men dessa förekommer även hos många fakturor.

Genom denna slutsats går det att skapa en förståelse för varför fakturaklassificeringens framgång, men förklaringen är inte tillräcklig för att förklara varför kvittoklassificeringen presterade avsevärt sämre. En förklaring kan helt enkelt vara att givet ett testkvittos ordinnehåll så gav fakturaklassens betingade sannolikheter för dom orden helt enkelt ett högre värde än kvittoklassens, som i sin tur kan bero på storleken av datamängden hos fakturaklassen. Detta beror isåfall på att fakturor och

kvitton delar många liknade ord men att kvittoklassen saknar explicita ord och träningsdatan för fakturor ofta var större än träningsdata för kvitton.

Vid en obalanserad datafördelning där den ena klassen har en större mängd data än den andra så finns det lösningar för att skapa balans. En metod är genom syntetisering av data för den klassen som är mindre (Chawla, Bowyer, Hall, Kegelmeyer, 2002). Lösningen ligger inte inom ramen för examensarbetet. Det går även att läsa i tabell 10 på rad 2 att även vid jämn fördelning så presterade kvittoklassificeringen fortfarande dåligt, trots att datan går från obalans till balans, vilket pekar på att problemet ligger i något författarna till examensarbetet inte kan fastställa.

Eftersom problemet verkar ligga på annat håll är det svårt att med säkerhet säga hur balansen i datamängd bör se ut men kollar man på resultaten i tabell 10 på rad 3 och rad 4 så ser man hur resultatet försämras när klassernas datamängd blir mer obalanserad. Matematiskt följer det rätt mönster då priorisannolikheten, se ekvation 9, blir en större faktor när obalansen ökar. En jämn fördelning bör möjligtvis eftersträvas i fallet med fakturor och kvitton men värt att nämna är att det inte alltid behöver vara så. En obalans kan vara eftersträvat i fall där man vill att priorisannolikheten ska spegla verkligheten. Ett exempel på detta är spam-filtrering. Samlar man alla inkommande e-postmeddelanden så kommer det naturligt inte skapas en jämn fördelning av spam och icke-spam. Här hade det varit en bra ide att låta obalansen vara, alternativt att manuellt ändra priorisannolikheten.

En förklaring till resultatet i tabell 10, rad 2 kan vara att kvaliteten på kvittodata har varit sämre än kvaliteten för fakturor. Fakturor finns tillgänglig i elektroniska format och är mycket enklare att få ut korrekta ord från med hjälp av OCR. Kvittodatan kommer till viss del i elektronisk form men även från fotograferade bilder som inte alls ger samma konsekventa resultat när ord ska identifieras. Detta resulterar i sin tur att klassificeringen blir inkonsekvent. I kvittoklassen kan det förekomma flera stavelser av samma ord som OCR programmet helt enkelt tolkat fel. Hade det identifierats korrekt hade ordet fått en högre betingad sannolikt, eftersom sannolikheten beräknas med hjälp av ordets förekommande frekvens i ett dokument.

En lösning, som dock inte blev implementerad, är att låta datan gå genom stavningkontroller och regex, se Appendix 9.1.6, filtrering för att korrigera de ord som tolkats fel i processen av OCR behandling. Anledningen till att detta inte blev en del av programmet från början var att effekten helt enkelt inte förutsågs, varken att OCR programmet tolkade orden fel eller att den inkonsekventa datan skulle bli ett problem.

En motivering till varför inte regex korrigering av ord tillämpades redan vid första observation av datamängden var att eftersom algoritmen inte bryr sig om innehållet av orden så skulle det fortfarande fungera som tänkt. Vad författarna misslyckades med att förutse, förutom att orden feltolkades, var inkonsekvensen i den felaktiga ordtolkningen.

Om rad 3 och 4, i tabell 10, ignoreras och man endast kollar på resultatet på rad 1 och 2 så sker en minskning i procentuell framgång trots att förhållandet går från 3:1 till 1:1, faktura respektive kvitto. Trots att en jämn fördelning existerar så har mängden data av sämre kvalitet ökat vilket försämrar resultatet.
För att algoritmen skall fungera som tänkt är vikten av konsekvent data stor. Mer så i Naive Bayes baserade algoritmer än andra. Som tidigare nämnt räknar Naive Bayes algoritmer frekvensen av ord förekommande i dokument, om datan är inkonsekvent formaterad har inte algoritmerna förmågan

att komma upp i en rimlig frekvens. Detta på verkar i sin tur algoritmens förmågan av att klassificera dokument.

6.2 Bernoulli

Liknande resultatet för Multinomial så presterar algoritmen i fakturaklassificeringen mycket bättre än kvittoklassificeringen. Fakturaklassificeringen presterar över 80 procent oberoende av procentuell fördelning av dokumenten. Kvittoklassificeringen presterar bäst med 17 procent framgång vid fördelningen av träningsdata vid ett förhållande på 1:1, faktura respektive kvitto. Dock är prestationen långt under ett användbart resultat för scenariot beskrivet i kap 1.

Undersöker man närmare på hur Bernoulli-distributionen ser ut så modelleras förekomsten av ett testdokuments ord i BoW. Här spelar alltså inte antalet förekomster roll, istället blir systemet binärt där ett ord antingen förekommer i BoW eller inte.

Det enda algoritmen beräknar är frekvensen om ett ord förekommer i ett dokument, till skillnad från multinomial som beräknar frekvensen av antalet samma ord förekommer i samma dokument. För vidare förklaring se ekvation 17. Algoritmen ger inga fördelar i att ett ord frekvent förekommer i samma dokument utan en fördel ges där ordets förekomst är konsekvent upprepande i träningsdatan, vilket i bästa fall skulle göra $N_{t,\omega}$ lika stor som antalet träningsdata. Här har fakturor en mycket stark fördel på grund av dom tidigare nämnda konsekventa upprepningarna av vissa ord, se Kap. 6.1.

Intressanta iakttagelser från tabell 11 , rad 3 och 4 är hur framgången för kvittoklassificeringen går ned trots att ett mängdförhållande på 6:1, då nämnaren växer i proportion med antalet dokument i ekvation 17. Trots detta så fortsätter programmet att endast klassificera 4 kvitton rätt. Ännu en gång är det troligt att det är den sämre kvittodatan som ger fakturaklassen ytterligare fördelar. Som tidigare nämnts i Kap. 6.1 så ger detta effekten att kvittoklassen får avsevärt mindre korrekta ord knuta till sig och således kommer programmet nästan alltid att välja fakturor som det bästa valet.

Om man närmare undersöker kvittona som programmet lyckades klassificera rätt så står dom ut i den aspekten att orden som dokumentet i fråga innehåller är fullständiga samt korrekta samt att alla innehåller ordet ”kvitto” eller ”kvittot” ett flertal gånger samt andra ord som är starkt förknippat med kvittoklassen.

För att ytterligare styrka svaret till frågan: ”*Hur bör datan eller texten behandlas för att bäst kunna klassificeras?”* så visar resultaten att det måste finnas en jämn kvalitet på datan för båda klasserna. Det absolut bästa är att från en början försöka få ut så bra data som det går för att i efterhand utföra kontroller som upprätthåller kvaliteten.

6.3 Neuronnät

Syftet med detta examensarbete ämnar att hitta en potentiell lösning för klassificeringsproblem av e-postmeddelande. Utifrån frågorna formulerade i problemformuleringen i Kap. 1.4 kommer denna del sammanfatta och dra en slutsats för de resultat presenterade i Kap. 5.1.

Problemformuleringen ämnar att ta reda på mängden data nödvändig för träning och testning av maskininlärningsalgoritmerna, hur datan skall behandlas för ändamålet av klassificering samt även fördelningen av datan tillgänglig åt träning respektive test av maskininlärningsalgoritmerna. Frågornas natur behandlar till stor del huruvida anpassningen av olika data-relaterade fenomen påverkar resultatet av de olika metoderna prövade för att lösa klassificeringsproblemet och vilken

anpassning av dessa fenomen som resulterar i ett lämpligt slutresultat. Vad gäller neuronnätet, samt formateringen av data med hjälp av NLTK och BoW metoder, är det möjligt att se att dessa fenomen, för den datamängd tillgänglig för detta arbete, har liten påverkan på resultatet för precisionen av klassificering som neuronnätet kan tillföra. Med förhållning att datamängden kan vara otillräcklig för att ge ett definitivt svar, samt att testerna utförda inte kan garantera en rättvis representation av neuronnätets totala prestation, presenteras varierande resultat, se Tab. 5-9 samt Fig. 18-19, på klassificeringsförmågan för neuronnätet som sträcker sig från ~80% till 95%.

Neuronnätet har i vissa fall har presterat bättre med avseende på antalet neuroner tillgängliga i det gömda lagret samt att datamängden tillgänglig för träning för neuronnätet båda har varit mindre, se Tab. 5. Värt att poängtera här är att relationen mellan den datamängd tillgänglig för träning och antalet neuroner i det gömda lagret är svårt bestämma. Trots att det finns vissa riktlinjer kring antalet neuroner och datamängden måste användaren slutligen bestämma förhållandet manuellt för sitt specifika ändamål.(Shibata, K., & Ikeda, Y., 2009)

Utöver de testsviterna som användes för det resultat presenterat i Kap. 5.1, infördes också ett mindre test för att ge en indikation på hur väl neuronnätet presterade att klassificera viss text som inte tillhörde något av de två klasserna kvitto och faktura, se Tab. 5-9 kolumn *Sentence*. Indikationen gav viss perspektivskifte från hur passande den redovisade konfigurationen av neuronnätet är för ändamålet menat att uppfyllas i detta examensarbete. Även om storleken av testet inte kan ge ett definitivt svar huruvida neuronnätet klassificerar icke tillhörande textdokument som något av de två klasserna kvitto eller faktura är det en indikation på hur väl tränat neuronnätet är för de två klasserna med avseende på datamängd och antal gömda neuroner. För neuronnätet är det möjligt att konstatera att en mängd gömda neuroner som understiger 30 i det gömda lagret inte är tillräckligt, trotts bra prestation vad gäller testsviterna för kvitton och fakturor. Anledningen till detta är att det med största sannolikhet inte klarar av att avgöra ifall ett icke tillhörande dokument tillhör något av klasserna faktura eller kvitto. För att ge ett definitivt svar behövs mer testdata. Detta gäller även prestationen för neuronnätet med avseende på dess klassificeringsförmåga av dokument som tillhör något av de två klasserna faktura eller kvitto, även om det inte är i samma utsträckning som testerna som behövs för dokument som inte tillhör något av de två klasserna.

En annan intressant punkt att lyfta fram är att trots obalansen av datamängden mellan de två olika klasserna klarar neuronnätet, i de fall testade i Kap. 5.1, alltid att klassificera kvittodokument med 75% framgång och i många fall med samma framgång som neuronnätet gör med klassificeringen för fakturadokument. Fördelen med neuronnätet är att den använder sig likväl av datan extraherad från fakturadokument samt den data extraherad från kvittodokument för klassificeringen av ett godtyckligt dokument.

6.4 Sammanfattning

Av att döma från resultaten presenterade i kapitel 5, samt den slutsats dragen för respektive metod använd, för att bemöta frågeställning, syfte och målsättning för detta examensarbete är det möjligt att sammanfatta huruvida vilken metod som uppfyller examensarbetets syfte och målsättning. Vad gäller den metod och resultat presenterade i kapitel 2.2.3 respektive kapitel 5.1 är neuronnätet i sin helhet överlägsen i prestation och är därför en potentiell lösning för den målsättning satt för detta examensarbete. Detta med avseende på att den både kan klassificera faktura och kvitton på ett korrekt sätt, i en godtycklig grad. Samt att framtida utvecklingsmöjligheter, 6.6, visar potentiella möjligheter att vidare öka prestationen av neuronnätet i detta sammanhang.

6.5 Reflektion över etiska aspekter

Då både kvitton och fakturor kan innehålla konfidentiell och känslig information så kan det vara viktigt att se över hur programmet behandlar innehållet i dokumenten samt se över på vilka sätt det skulle kunna gå fel. För att slippa göra många beräkningar om och om igen sparar programmet mycket av datan i CSV filer som beskrivet i Kap. 2.2.1.5. Utan behandling av datan kommer alla ord som förekommer i alla fakturor och kvitton att hamna här.
Enligt personuppgiftslagen (PuL) som gäller fram till 25e maj 2018 görs en skillnad på ostrukturerad och strukturerad data. I fallet med programmets CSV filer skulle ett sådan data klassas som ostrukturerad data och hade varit tillåten så länge inga uppgifter kränks.
Med det nya dataskyddsförordningen (GDPR) som träder i kraft 25e maj 2018 ändras detta och gör ingen skillnad på strukturerad och ostrukturerad data. Här skulle saker enkelt kunna gå fel men samtidigt existerar enkla åtgärder för att följa GDPR.

I fallet med neuronnätet beskrivet i 2.2.3 skulle dessa känsliga uppgifter, som GDPR har som uppgift att skydda, endast användas vid träning för att sedan raderas då endast vikterna mellan synapserna är av värde för programmet.

Vid Multinomial och Bernoulli Naive Bayes beskrivet i Kap. 2.2.1 och Kap. 2.2.2 kommer minst en BoW alltid behöva sparas för att kunna utföra klassifikationerna. Utan behandling kommer självklart denna BoW innehålla mycket information så som personnummer, namn, adresser m.m. Dock skulle man kunna argumentera för att det i ett dataset på många tusen ord så är det endast lagring av personnummer som skulle kunna knytas till en viss person och på så vis bryta mot GDPR.
Då personnummer i sig inte tillför någon egentligt värde till algoritmerna så kan dom enkelt uteslutas med ett filter.

6.6 Framtida utvecklingsmöjligheter

För att föra arbetet vidare finns det olika möjligheter att beakta. Vid implementeringen av Naive Bayes är det möjligt att vidta åtgärder för att förhindra att programmet använder sig av dålig data, ord som har inte formateras konsekvent, vid träning. Det är möjligt, att med hjälp av textfilter, rensa dålig data. Användningen av enbart elektroniska kvitton är också en möjlighet, då datan är mer konsekvent kontra de dokument som fotograferats för hand. Att förbehandla data med hjälp av textfilter, alternativt fokusera på att inhämta data som leder till konsekvent datamängd för träning ger vidare möjligheter att testa algoritmernas förmåga att klassificera fakturor och kvitton.

För att ytterligare öka prestationen är också en möjlighet att införa en klass som ska representera varken kvitton eller fakturor. Datan för denna klass ska baseras på e-postmeddelande vars ursprung är godtyckligt men också specifikt inte ska vara, eller innehålla, ett kvitto eller en faktura. Vid situationer där maskininlärningsmetoderna inte riktigt har kapaciteten att klassificera ett dokument till antingen ett kvitto eller faktura kan det uppstå problem där något av metoderna ändå försöker att klassificera dessa. Den tredje klassen skall då underlätta klassificeringen för metoderna, med att inge mer förståelse för vad som inte är ett kvitto eller en faktura. Möjligheten att införa detta är även god, då tillgången till datakällor av denna typ är lättåtkomlig.

En tredje möjlighet är mer test kring klassificeringströskeln för de olika metoderna. Klassificeringströskeln innebär den gräns till vilket av metoderna har uppskattat ett dokument tillhör en klass ska vara. Ett exempel är i neuronnätet användes en klassificeringströskel på 80 procent. Det är möjligt att en mindre, eller högre, tröskel hade producerat bättre resultat.

En sista utvecklingsmöjlighet för detta ändamål som också hade varit intressant att beakta är prestationen av metoderna i kombination med varandra. I de fall där en metod har gjort en form av bedömning, eller klassificering, av ett dokument är det möjligt att använda en jämförelse av två metoder för maskininlärning och låta en tredje maskininlärningsmetod göra en bedömning om vilket av resultaten presenterade av de två första metoderna, baserat på en historik av respektive metods föregående prestation.

7. Terminologi

- Dictionary variabel - En key-value pair variabel, med följande format: *dict = {'Name': 'Author', 'Age': 24, 'Class': 'Third'}*. Varvid utskrift: *print(dict[Name]),* ger resultatet*: Author.*
- Epok - En cykel av all träningsinformation då den först går igenom och sedan korrigerar tillbaka genom ett neuronnät.
- Dropout - En teknik inom konstrueringen av neuronnät för att reglera beroendet på specifika neuroner. En hyperparameter bestämmer procentuell andel av neuroner som skall avaktiveras, tillsammans med deras synapser. Genom detta tvingas neuronnätet att bli mer balanserat vid träning och minskar risken för överanpassning. (Nitish Srivastava et al 2014)
- Felfrekvens - den punkt i det plan som representerar systemet, neuronnätets vikter justerade efter det faktiska svaret. Felfrekvensen ger en indikation hur väl tränat systemet är.
- Bag of Words (BoW) är ett sätt att lagra ord från texter för användning vid modellering, till exempel med maskininlärningsalgoritmer. En BoW är en representation, ofta en vektor, av text som beskriver förekomsten av ord i ett dokument. Det kallas en "Bag" of words, eftersom information om ordningen eller strukturen i orden ej är relevant. Modellen speglar bara huruvida kända ord förekommer i dokumentet eller inte.

8. Källförteckning

- Anderson, D. (2010). *Kanban*. Sequim, Wash. D.C.: Blue Hole Press. (pp. 61-69)
- A. P. Dawid, a. (1979). Conditional Independence in Statistical Theory. *Journal Of The Royal Statistical Society. Series B (Methodological)*, (1), 1.
- Biba M., Gjati E. (2014) Boosting Text Classification through Stemming of Composite Words. In Thampi S., Abraham A., Pal S., Rodriguez J. (Eds) *Recent Advances in Intelligent Informatics. Advances in Intelligent Systems and Computing*, vol 235. Springer, Cham (pp. 186)
- *IETF | Internet Engineering Task Force*, RFC2046. https://datatracker.ietf.org/doc/rfc2046/ (2018-05-31)
- Bird, S., & Loper, E. (2004). NLTK: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*. Association for Computational Linguistics. (pp. 31)
- Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, *16* (pp. 321-357).
- Dan Morris (2016) *Bayes' Theorem Examples: A Visual Introduction For Beginners*. Blue Windmill Media (pp. 39)
- Harrington, P. (2012). *Machine learning in action*. Shelter Island (N.Y.): Manning Publications Co., (pp. 13)
- Jurafsky, D., & Martin, J. H. (2009). *Speech and language processing : an introduction to natural language processing, computational linguistics and speech recognition.* Upper Saddle River, N.J. : Pearson Education International/Prentice Hall. (pp. 47, 64)
- Larman, C. (2004). *Agile and iterative development : a manager's guide. Boston* : Addison-Wesley, cop. 2004. (pp. 27)
- Lewis, D. D. (1998). Naive (Bayes) at forty: The independence assumption in information retrieval. In *European conference on machine learning*. Springer, Berlin, Heidelberg (pp. 4-15)
- Metsis, V., Androutsopoulos, I., & Paliouras, G. (2006). Spam filtering with naive bayes-which naive bayes?. In *CEAS,* Vol. 17 (pp. 28-69)
- *Nationalencyklopedin,* neuronnätverk. http://www.ne.se/uppslagsverk/encyklopedi/lång/neuronnätverk (hämtad 2018-04-24)
- Rennie, J. D., Shih, L., Teevan, J., & Karger, D. R. (2003). Tackling the poor assumptions of naive bayes text classifiers. In *Proceedings of the 20th international conference on machine learning (ICML-03*) (pp. 616-623).
- Richert, W. and Coelho, L. (2013). *Building machine learning systems with Python*. 1st ed. Birmingham, UK: Packt Publishing. (pp. 3,16)
- *Riksdagen,* Mervärdesskattelag (1994:200) 11 kap. 8 §. https://www.riksdagen.se/sv/dokument-lagar/dokument/svensk-forfattningssamling/mervardesskattelag-1994200_sfs-1994-200 (Hämtad 2018-05-30).
- Shibata, K., & Ikeda, Y. (2009). Effect of number of hidden neurons on learning in large-scale layered neural networks. In *ICCAS-SICE, 2009* . IEEE (pp. 5008-5013).
- *Skatteverket,* Bokföring – vad kräver lagen?. https://www.skatteverket.se/foretagochorganisationer/startaochdrivaforetag/bokforingochbokslut/bokforingvadkraverlagen.4.18e1b10334ebe8bc80005195.html (hämtad 2018-05-29).
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929-1958.
- Schwaber, K. and Sutherland, J. (2017). *The Scrum Guide*. (pp. 7) [ebook]. Från https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf#zoom=100 (2018-05-16).

- Williams, L., Kessler, R., Cunningham, W. and Jeffries, R. (2000). *Strengthening the case for pair programming*.17(4) (pp.19-25)
- Zhang, H. (2004). The optimality of naive Bayes. *AA* (pp. 1-4)
- Zhang, Y., Jin, R., & Zhou, Z. (2010). Understanding bag-of-words model: a statistical framework. *International Journal Of Machine Learning & Cybernetics*, 1(1-4), 43. doi:10.1007/s13042-010-0001-0
- *IETF | Internet Engineering Task Force*, RFC4180. https://tools.ietf.org/html/rfc4180 (2018-05-31)
- *Python Software Foundation*, https://www.python.org/ (2018-05-31).
- *Hong Minhee*, Wand. http://docs.wand-py.org/en/0.4.4/ (2018-05-31).
- *Jupyter Team*, The Jupyter Notebook, https://jupyter.org (2018-05-31)
- *The Scipy community*, NumPy v1.13 Manual, https://docs.scipy.org/doc/numpy-1.13.0/index.html (2018-05-31)

9. Appendix

I appendix presenteras vidare information om det som har benämnts i rapporten, som inte direkt i rapporten kan tillskrivas. Dessa är moduler använda i program, bilder och kod som har implementerats av författarna.

9.1 Moduler

9.1.1 Textract

Textract är en modul för Python, gjord för att extrahera text från alla typer av filer som innehåller klartext utan att behöva använda många olika moduler. Textract stödjer ett antal filformat, däribland .pdf, .png, .csv med mera. För en fullständig lista över filformat som stöds se dokumentationen, http://textract.readthedocs.io

Textract fungerar som ett typ av interface mellan flera andra Python-moduler, varje specifik modul gjord för att extrahera text från en typ av filformat. Modulen används genom att kalla på *textract.process(”Path/to/file”)*, Textract detekterar formatet på filen som ska behandlas och väljer, i sin tur, en lämplig modul för att extrahera texten och returnera det i form av en sträng.

9.1.2 Wand

Wand är ett ctype baserad bindning till modulen ImageMagick®, som möjliggör att använda ImageMagick® genom Python kod (Hong Minhee, 2018-05-31).

ImageMagick® används för att skapa, redigera, komponera eller konvertera bitmap bilder och stödjer över 200 format.

9.1.3 Natural Language Toolkit (NLTK).

Natural Language Toolkit (NLTK) är en svit av programmoduler, data set, övningar och handledningar som täcker statistisk och symbolisk naturlig språkbehandling.

NLTK utvecklades i samband med en kurs i beräkningslingvistik vid University of Pennsylvania år 2001 med tre pedagogiska tillämpningar som syfte: uppgifter, projekt och demonstrationer.

Genom att använda sig av olika språkkroppar, s.k. corpus, är det möjligt för användaren att hitta ett ords ordstam, beteckna vilken typ av ordklass ordet tillhör, ta bort betydelsefattiga ord (s.k. stoppord) och en uppsjö andra metoder för att förbehandla eller behandla och klassificera och hitta innebörden av en text från dess komposition av ord. (Bird, S., & Loper, E. 2004)

9.1.4 Jupyter Notebook

Jupyter Notebook (Jupyter Team, 2018-05-31) är en server-klient applikation som gör det möjligt att redigera och köra Notebook documents via en webbläsare. Jupyter Notebook kan köras lokalt och kräver ingen internetanslutning eller kan installeras på en extern server och nås via internet.

Notebook documents är dokument som produceras av Jupyter Notebook, som innehåller både datorkod (t.ex. Python) och klartext. Notebook-dokument är både läsbara dokument som innehåller analysbeskrivning och resultat (figurer, tabeller osv.) Samt exekverbara dokument som kan köras för att utföra dataanalys.

9.1.5 NumPy

NumPy (The Scipy community 2018-05-31) är det grundläggande paketet för vetenskaplig databehandling i Python. Det är ett Python-bibliotek som tillhandahåller ett flerdimensionellt arrayobjekt, olika härledda objekt (till exempel maskerade arrayer och matriser) och ett sortiment av rutiner för snabba operationer på arrayer, inklusive matematisk, logisk, formmanipulation, sortering, val, I / O , diskreta Fourier-transformer, grundläggande linjär algebra, grundläggande statistiska operationer, slumpmässig simulering och mycket mer.

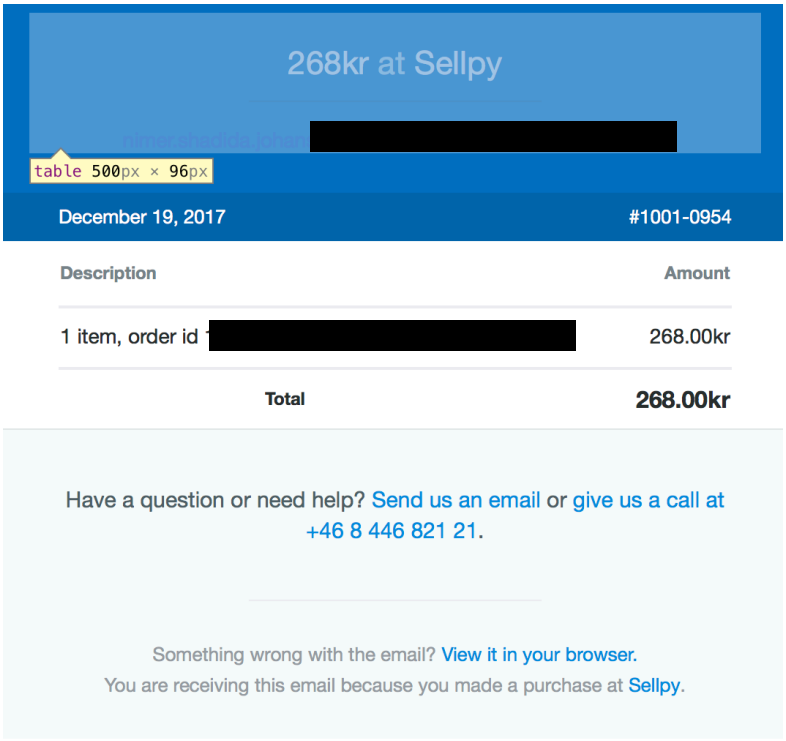
9.1.6 Regex (regular expressions)

Ett reguljärt uttryck (eng. regular expression) är ett uttryck av en sträng som följer vissa specificerade regler. Reguljära uttryck kan användas för att manipulera strängar och skapa filter genom att tvinga indata av text att följa deras syntaxregler.

9.2 Bilder

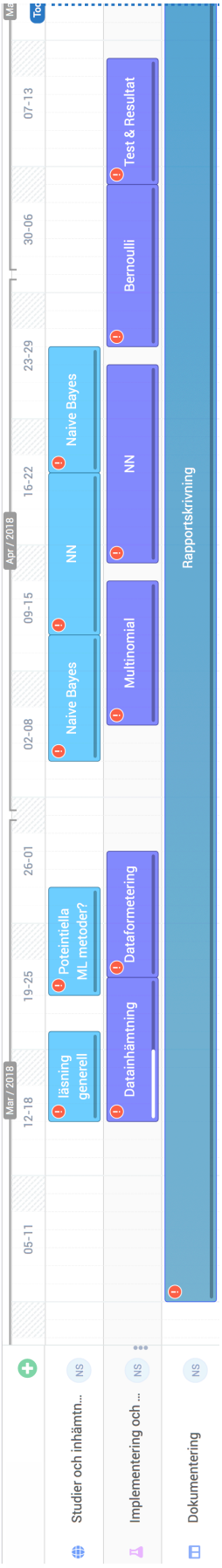
9.2.1 Sellpy kvitto

Ett kvitto på köp online från företaget Sellpy. Bilden visar ett e-postmeddelande där ett kvitto skickas inline i ett e-postmeddelande, markerat är <*table*> strukturen i html koden i e-postmeddelandet.



9.2.2 Faser och arbete

I bilden nedan visas de faser, samt de moment i respektive fas, som författarna i detta examensarbete har genomgått.



9.3 Kod

Denna del av kapitel 9 ämnar användas som förklaring för kod implementerad, eller använd, för kapitel 2 som i annat fall är för stor för att förklaras direkt i kapitel 2.

9.3.1 SerarateByClass(dataset)

Vid implementering av Multinomial Naive Bayes används funktionen, illustrerad nedan, för att separera innehållet av en .csv fil med data (text) efter respektive klass.

```
15      dataset = loadCsv(filename)
16
17
18      def separateByClass(dataset):
19          separated = {}
20          for i in range(len(dataset)):
21              vector = dataset[i]
22              if vector[-1] not in separated:
23                  separated[vector[-1]] = []
24              separated[vector[-1]].append(vector)
25          return separated
26
```

9.3.2 likeHood_word_given_class(Dictionary dict, String s,_class)

***Separated_values*:** Dictionary objektet som returnerades av *SerarateByClass()*.
***get_unique_words_count()*:** Returnerar antalet unika ord i alla dokument.
***word*:** Ordet vars värden ska beräknas enligt tabellen ovan.
***_class*:** Vilken klass ordet ska testas gentemot
data[n]: Vektor med samma index som tabellen ovan

Metoden itererar genom nyckeln ***_class***'s vektor i objektet ***Separated_values*** och fyller först på värdena i data[2] och data[1] där data[2] ökar med 1 för varje ord i vektorn.
data[1] ökar sedan med ett om samma ord påträffas igen.
data[4] sätts i början direkt till ***get_unique_words_count()***.

```
135
136     def likeHood_word_given_class(separated_values,word,_class):
137         data = [word,float(0),float(0),float(0),float(0)]
138         data[4] = get_unique_words_count()
139
140         list = [val for key, val in separated_values.items() if key == _class]
141
142         for i in range(len(list)):
143             for x in range(len(list[i])):
144                 for y in range(len(list[i][x]) - 1):
145                     data[2] += 1
146                     if list[i][x][y] == word:
147                         data[1] += 1
148
149         data[3] = (float(data[1]+1) / (float(data[2]) + float(data[4])))
150         return data
151
152
```