Klassificering av e-post

En undersökning av handledda klassificeringsmetoder

J O H A N L I T S F E L D T

KTH Datavetenskap
och kommunikation

Examensarbete
Stockholm, Sverige 2011

Klassificering av e-post

En undersökning av handledda klassificeringsmetoder

J O H A N L I T S F E L D T

Examensarbete i medieteknik om 15 högskolepoäng
vid Programmet för medieteknik
Kungliga Tekniska Högskolan år 2011
Handledare på CSC var Mikael Goldmann
Examinator var Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2011/litsfeldt_johan_K11086.pdf

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Referat

Denna rapport behandlar metoder för automatisk klassificering av e-post d.v.s. kategorisering av brev med avseende på deras innehåll. Metoder för klassificering beskrivs i rapporten för ett godtyckligt antal kategorier men speciellt för det binära fallet. Algoritmerna analyseras även genom implementation och utvärdering av ett spamfilter baserat på dessa metoder. Utöver klassificeringsalgoritmer innehåller även rapporten språkanalys av e-postbrev, viktningsprinciper och en genomgång av moderna spamtekniker.

Abstract

E-mail classification

This report discusses methods for automatic classification of e-mail i.e. categorization of mails with respect to their content. Methods used for classification are described for an arbitrary number of categories but especially for the binary case. The algorithms are also analyzed through implementation and evaluation of a spam filter based on these methods. In addition to classification algorithms, the report also includes language analysis of e-mails, weighting principles and a review of modern spam techniques.

Innehåll

Kapitel 1

Inledning

1.1 Motivering

Sedan e-post introducerades som kommunikationsmedel under mitten av sjuttiotalet har tekniken växt sig till oanade höjder. Idag skickas varje år omkring 60 biljoner e-postmeddelanden världen över. Detta motsvarar kring två miljoner meddelanden varje sekund. Men var tar alla dessa brev vägen? Och läses verkligen alla dessa brev? En undersökning gjord av *the Radicati Group* visar att 78% av alla e-postbrev som skickas idag är spambrev, d.v.s. brev med oönskad reklam, bedrägeriförsök, datavirus eller liknande. Däremot är det endast 56% av spambreven som faktiskt levereras till mottagarens e-postklient, en siffra som stadigt minskar. [1]

Då brev passerar genom portar och nätverk på vägen till mottagaren passerar de även spamfilter som effektivt rensar bort illasinnade brev. Efter att brevet kommit fram till klienten (*Microsoft Outlook*, *Mozilla Thunderbird*, eller liknande) används oftast ytterligare ett filter implementerat i själva applikationen. E-postklienternas filter placerar sannolika spambrev i en särskild kategori baserat på brevets innehåll. Detta är ett exempel på en sorts *klassificering* av e-post och är precis vad som kommer att undersökas i denna rapport.

Klassificering av dokument i allmänhet förekommer inom en mängd områden utöver just e-post. Ett område som på senare tid tillkommit är något som kallas *narrowcasting* d.v.s. företeelsen att man genom information baserad på kunden endast skickar nyheter eller reklam som är relevant för denne, något som kan automatiseras med dokumentklassificering. Andra områden där dokumentklassificering används är exempelvis funktionalitet för rekommendationer på webbsidor, marknadsundersökningar och liknande.

1.2 Klustring och klassificering

Först och främst skall nämnas att ordet dokument används regelbundet i denna rapport istället för element som kan tänkas vara ett mer allmänt begrepp. Anledningen till detta är att rapporten fokuserar vid klassificering av just texter varför ordet dokument förtydligar vad som menas på ett bättre sätt. Självklart är det inte enbart texter som kan klassificeras med metoderna som beskrivs i denna rapport.

Klustring innebär indelningen av en samling dokument i delmängder så att samtliga dokument inom en viss delmängd liknar varandra i någon mening. Då kategorierna extraheras automatiskt ur textmängden innebär detta att klustringsmetoden är av typen *ickehandledd inlärning*. Klustringsalgoritmer bildar därmed kategorier genom de kluster som uppstår. Kluster upptäcks genom att göra avstånds- och densitetsbedömningar av de dokument man har förfogande över. Inga klustringsalgoritmer kommer att undersökas närmare i denna rapport men begreppet kan ändå vara bra att känna till. [2]

Med klassificering av dokument menas att tilldela ett elektroniskt dokument en eller flera kategorier baserat på dokumentets innehåll givet ett antal redan specifierade kategorier. Klassificering är en sorts *handledd inlärning* vilket betyder att systemet är i behov av extern data för själva klassificeringen, exempelvis manuell klassificering av träningsmängden. En nackdel med klassificering är att oförutsägbara situationer som inte förekommer i träningsmängden måste

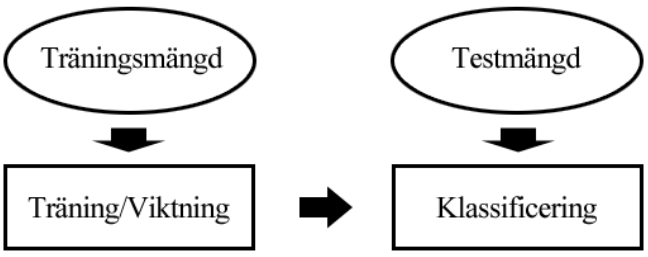
tas hänsyn till vilket inte är ett problem vid klustring. I denna rapport studeras klassificeringsmetoderna *Naive Bayes classifier*, *K-Nearest neighbor* och stödvektormaskiner.

Om kategorierna redan är kända är det bästa tillvägagångssättet förmodligen att använda kategorisering istället för klustring. Klustring lämpar sig när man vill finna strukturer man sedan tidigare inte kände till i testmängden. [2]

1.3 Tränings- och testmängden

Klassificerare används ofta inom maskininlärningsområdet som handlar om hur man lär system att agera enligt statistiska metoder. För att detta överhuvudtaget skall kunna utföras måste systemet först tränas. Under träningsfasen matar man systemet med en *träningsmängd* av redan klassificerade dokument. Denna mängd data måste struktureras och behandlas på ett sådant sätt att de viktigaste särdragen kan urskiljas och användas under klassificeringsfasen.

Systemet skall efter att träningsfasen genomförts ha tillräckligt med information för att klassificera nya dokument baserat på träningsmängden. För att testa systemet använder man då en *testmängd* som innehåller dokument där svaret är givet för användaren men inte för systemet, detta kallas klassificeringsfasen. Efter testning av systemet kan resultatet jämföras med de riktiga svaren och klassificeraren bedömas utifrån detta.



Figur 1.1. Figuren visar en övergripande bild av en klassificerare.

1.4 Uppgift

Denna rapport behandlar metoder för automatisk klassificering av e-post. Ett antal klassificeringsalgoritmer undersöks, implementeras och utvärderas i praktiken. Metoderna kommer att analyseras med hänsyn till hur bra de presterar, framförallt hur precisa de är på att klassificera brev, hur mycket träningsdata som behövs men även till viss del m.a.p. körtid och minnesåtgång. Utvärderingen sker i form av implementation samt testande av spamfilter baserade på olika algoritmer.

För att få en så bra representation som möjligt av varje e-postbrev måste även en betydande vikt läggas vid hur orden väljs ut, representeras och viktas. En stor del av rapporten ägnas därför åt detta område.

Kapitel 2

Förbehandling

2.1 Vektormodellen

En *vektormodell* (eng. *vector space model*) är en modell som ofta används för att representera dokument som vektorer av identifierare, exempelvis för hur många gånger ord förekommer i texten. [3]

Samlingen av de dokument som tillhör träningsmängden betecknas D. Om ett ord med vikt w_{ij} förekommer i dokumentet $d_j \in D$ är ordets vikt skiljt från noll. Varje unikt ord tillför även en dimension till vektorn d_j.

$$d_j = (w_{1j}, w_{2j}, \ldots, w_{tj})$$

där $w_{1j}, w_{2j}, \ldots, w_{tj}$ är ord som förekommer i dokument j.

Varje dokument i träningsmängden är tilldelad en kategori $c_j \in C$ där C är mängden av möjligt förekommande kategorier. Låt säga att klassificeraren skall bestämma språket av en viss text. Mängden C ges då exempelvis av $C = \{Svenska, Engelska, Tyska\}$.

Detta gör att det lätt kan bli väldigt många dimensioner beroende på hur

många unika ord/attribut som identifieras (inte ovanligt med 10,000+). Av den anledningen vill man hålla denna rymd så liten som möjligt. En stor fördel med vektormodellen är att rummet nu kan betraktas på geometriskt vis, d.v.s. avstånd mellan orvektorer kan enkelt beräknas geometriskt.

För att klassificera ett dokument ställer man en fråga q som även den representeras enligt vektormodellen:

$$q_k = (w_{1k}, w_{2k}, \ldots, w_{tk})$$

där $w_{1k}, w_{2k}, \ldots, w_{tk}$ är vikter för ord som förekommer i frågan.

Vikten w_{ij} för ett specifikt ord kan beräknas enligt olika viktningsmodeller (se 2.4).

Att skapa vektormodellen kallas att indexera dokumenten och görs normalt på träningsmängden innan någon av klassificeringsalgoritmerna körs. Indexeringen handlar inte bara om att läsa in ord för ord utan stor vikt måste även läggas på vilka av dessa ord som skall tas med och inte. Anledningen till detta är att man vill ha en så bra avvägning som möjligt mellan hur orden representerar texten samt hur mycket minne som används.

2.2 Obetydelsefulla ord

Först och främst finns det ett stort antal ord som överhuvudtaget inte ens behöver tas med i representationen. Samtliga ord som är neutrala bör tas bort. Exempel på dessa är: och", "är", bredvidö.s.v. Det man gör åt detta är att bilda en stopplista över så många neutrala ord som möjligt. Vid indexeringen plockar man helt enkelt bort dessa ord och sparar på så vis minnesutrymme utan att förlora representationsvärde. [4]

På samma sätt som att ord som förekommer i många texter inte är betydelsefulla är även ord som förekommer i väldigt få texter inte särskilt betydelsefulla heller. Av den anledningen kan det vara en bra idé att plocka bort ord som

endast förekommer i någon liten procentandel av texterna. Detta gör även att felstavade ord och felaktigt inlästa ord inte tas med vilket är positivt. Ett annat sätt är att enbart ta med de x mest förekommande orden utöver stopplistan ur texten.

Det kan vara en bra idé att begränsa sig till ord mellan ungefär 3 och 24 tecken vid inläsningen då ord med färre tecken än 3 i regel inte har någon större betydelse och ord med fler än 24 tecken oftast inte står för något ord utan oftast är ett lösenord, en kod eller liknande som inte är relevant för uppgiften. Övriga ord som bör väljas bort är sådana som innehåller icke-alfabetiska tecken. Detta eftersom ord med andra tecken precis som med för långa ord inte säger något.

Var i texten ordet förekommer är även det viktigt. Ord som förekommer i titeln av ett e-postbrev kan anses vara av större betydelse varför man exempelvis kan ge dessa en vikt 2-3 gånger större än ett vanligt ord.

2.3 Böjning och ordled

Ord kan förekomma i många olika former men med samma syfte, exempelvis *simma* och *simmade*. Utan att ta hänsyn till böjningar av ord tolkas ord som dessa som helt oberoende. Att reducera orden till grundform innebär problem med ord som inte har samma grundform men ändå är nära besläktade (exempelvis *simma* och *simningen*) som då kommer att tolkas olika. En bättre lösning är att använda något som kallas *stemming*. Metoden försöker ta vara på ord som är nära besläktade och överför dessa till en s.k. *stem*. [4]

Ordled är ofta ett problem i svenska texter där det är betydligt vanligare med sammansatta ord än i engelskan. Ett exempel är ordet *videokamera* som på engelska blir *video camera*. Detta behöver inte vara något negativt men ofta vill man bryta upp ordet i fler delar. Det finns sätt att göra detta på men då denna rapport i större grad behandlar engelska texter är sådana metoder tämligen irrelevanta. [4]

2.4 Viktning

2.4.1 Term frequency (TF)

Det naiva sättet att skapa ordvektorn på är helt enkelt att ta med samtliga ord som förekommer i dokument j och lägga till dessa till ordvektorn d_j. Detta sätt att utföra viktningen på innebär ett antal problem, exempelvis att olika ordvektorer blir olika stora beroende på hur många ord breven innehåller. Med viktningsprincipen TF (term frequency) menas helt enkelt att ett ord som förekommer fler gånger i en text har ett större värde än ett ord som förekommer färre gånger i samma dokument.

För att få samma längd på samtliga ordvektorer kan normalisering av ordvektorerna utföras (se 2.4.3). Ett annat sätt är att välja ut de exempelvis 10 mest förekommande orden i texten och låta dessa representera dokumentet. Den senare lösningen gör att programmet tar upp betydligt mindre minne.

2.4.2 Term frequency-Inverse document frequency (TFIDF)

Viktningsmodellen TFIDF [4][5] består av två allmänna viktningsprinciper. Den ena grundas på termfrekvensen (se 2.4.1). Den andra principen baseras på träningsmängden D. Om ett ord har låg frekvens i D anser man att det har stor urskiljningsförmåga. På samma sätt har ett ord som förekommer i många dokument en låg betydelse och därmed en lägre vikt. Denna viktningsprincip kallas IDF (inverse document frequency) och tillsammans bildar de TFIDF. Vikten för ett ord beräknas enligt produkten av TF och IDF samt lagras i vektormodellen enligt

$$w_{ij} = tf_{ij} \cdot idf_i$$

där tf_{ij} är ordfrekvensen för det i:te ordet i den j:te texten samt idf_i är den omvända textfrekvensen för i:te ordet i j:te texten.

Ordfrekvensen kan beräknas på ett antal olika sätt, nedan följer en möjlig variant:

$$tf_{ij} = \log\left(f_{ij} + 1.0\right)$$

där $f_{i,j}$ är antalet förekomster av ord i ur text j. För den omvända textfrekvensen kan följande variant användas.

$$idf_i = \log\left(\frac{n}{n_i}\right)$$

där n är antalet dokument och n_i är antalet dokument ord i förekommer i.

Att beräkna TFIDF är aningen krångligare än att beräkna enbart TF. Samtliga brev måste läsas in två gånger, först för att ta fram orden ur varje brev och sedan för att undersöka i hur många av breven ordet förekommer.

2.4.3 Normalisering

Efter att en viktningsmetod utförts på ordvektorerna så är de i regel av olika längd. Detta har nackdelen att längre texter får en större vikt än kortare. Av denna anledning vill man normalisera dokumentens ordvektorer enligt

$$d_j = \frac{d_j}{\|d_j\|}$$

där d_j är dokument j och $\|d_j\|$ betecknar den euklidiska normen.

På så vis får varje ordvektor samma totala vikt vilket innebär att varje dokument har samma inverkan på resultatet. [4]

Kapitel 3

Algoritmer

3.1 Mer om klassificerare

En klassificerare för kategori $c_i \in C$ är en funktion f_i' som utför en mappning av dokumenten som tillhör D huruvida de är av kategori c_i eller inte.

$$f_i' : D \rightarrow \{falskt, sant\}$$

Klassificeraren approximerar den okända funktionen f_i som uttrycker relevansen av dokumenten för c_i.

$$f_i : D \rightarrow \{falskt, sant\}$$

För att en klassificerare skall kunna appliceras på dokument av okänd kategori måste först en träningsfas utföras där vektormodellen skapas. Antalet träningsdokument måste vara av tillräcklig storlek beroende på val av algoritm. Viktigt är även att varje kategori $c_i \in C$ som är tänkt att användas bör förekomma minst en gång bland träningsdokumenten för att dokument i testmängden överhuvudtaget skall kunna tilldelas denna kategori. [6]

Texter kan kategoriseras efter innehåll, genre, prioritet eller egentligen vad som helst. I regel vill man dock inte blanda kategorier med olika kontext, exempelvis genre med prioritet då det kan ge oönskade bieffekter.

Klassificerare kan delas upp i *mono classification* och *multi classification*. Vid mono classification tilldelas nya dokument exakt en utav de möjliga kategorierna. Multi classification innebär att ett nytt dokument kan tilldelas noll eller fler utav de möjliga kategorierna. Skillnaden mellan dessa ligger i klassificeringsfasen. Vid mono classification tilldelas dokumentet den mest sannolika kategorin till skillnad mot multi classification där dokumentet tilldelas de kategorier som får ett sannolikhetsvärde över en viss gräns. [6]

3.2 Klassificeringsalgoritmer

3.2.1 Naive Bayes classifier

Naive Bayes classifier (NBC) är en enkel probabilistisk klassificerare som likt namnet antyder baseras på Bayes sats. Att klassificeraren är naiv syftar på att klassificeraren är starkt oberoende d.v.s. förekomsten av en händelse gör varken en annan händelse mer eller mindre sannolik. Som exempel kan en frukt anses vara en apelsin om frukten är orange, rund och c:a 1 dm i diameter. Naive Bayes classifier avväger samtliga av dessa egenskaper helt oberoende även om flera av dessa attribut hänger ihop. Många moderna e-postklienter tillämpar NBC för filtrering av spam. [7]

Vid indexering av vektormodellen kallas de olika kategorierna hinkar. En hink består av en samling ord samt deras vikter. Hinkarna är även namngivna efter de kategorier som förekommer i träningsmängden. Utifrån en samling hinkar kan sannolikheten att ett specifikt ord existerar i en viss hink samt sannolikheten att ett brev finns i en viss hink beräknas.

Hinkar betecknas $d_1, d_2, \ldots, d_n$. Orden betecknas för hink j som $w_{1j}, w_{2j}, \ldots, w_{sj}$

där s är antalet ord tillhörandes hinken.

För ett nytt dokument q med t ord $w_{1k}, w_{2k}, \ldots, w_{tk}$ vill man bestämma sannolikheten $P(d_i|q)$ att ett dokument q tillhör hinken d_i. $P(d_i|q)$ är alltså sannolikheten att $w_{1k}, w_{2k}, \ldots, w_{tk}$ existerar i hinken d_i.

Detta kan beräknas med Bayes sats vilken säger att

$$P(d_i|q) = P(q|d_i) \cdot \frac{P(d_i)}{P(q)}$$

Vi har att $P(d_i|q)$ är sannolikheten att för en given hink d_i så existerar orden tillhörandes q i just den hinken.

$P(d_i)$ är sannolikheten att något brev finns i hink d_i.

$P(q)$ är sannolikheten av att brev q påträffas.

För att beräkna vilken hink brev q skall placeras i måste $P(d_i|q)$ beräknas för varje hink samt en bedömning $P(d_i)$ huruvida brevet borde tillhöra denna hink eller inte göras. För att beräkna $P(d_i|q)$ för en viss hink d_i måste sannolikheten för varje ord beräknas och multipliceras samman.

$$P(d_i|q) = P(w_{1k}|d_i) \cdot P(w_{2k}|d_i) \cdot \ldots \cdot P(w_{tk}|d_i) \cdot P(d_i)$$

Sannolikheten $P(w_{ik}|d_j)$ för ett av orden ges av hur många gånger ordet förekommer bland alla texterna i hinken delat på summan av längden av dokumenten. $P(d_i)$ ges av totala antalet ord i hink i delat på totala antalet ord i samtliga hinkar.

Ett fall som måste tas i beaktande är när en term $P(w_{ik}|d_j) = 0$. Görs inget åt detta fall så kommer $P(d_i|q)$ tilldelas värdet noll oavsett övriga attribut. En lösning på detta problem är att för samtliga sådana fall tilldela

$$P(w_{ik}|d_j) = \frac{0.1}{\|d_j\|}$$

Efter att $P(d_i|q)$ beräknats för samtliga hinkar väljs den hink vars sannolikhetsvärde blev högst som den mest sannolika hinken att placera dokumentet i. Om problemet är av typ multi-classification kan de hinkar med ett sannolikhetsvärde över en viss tröskel tilldelas dokumentet.

3.2.2 K-Nearest neighbor

Med metoden K-Nearest neighbor (KNN) klassificeras dokument baserat på de k närmst intilliggande dokumenten distansmässigt. Fallet då $k = 1$ kallas *närmsta granne* och representerar fallet då ett testdokument helt enkelt klassas som kategorin av det dokument som skiljer sig så lite som möjligt från dokumentet i fråga. [8]

Under träningsfasen används dokument definierade enligt vektormodellen. Vanligtvis används euklidiskt avstånd för att beräkna var dokumenten skall placeras i den n-dimensionella rymden (n står för antalet unika attribut). Låt x_i vara ett dokument med p vikter $(x_{i1}, x_{i2}, \ldots, x_{it})$ där $i = 1, 2, \ldots, n$ och n är antalet dokument. Avståndet mellan två dokument beräknas enligt

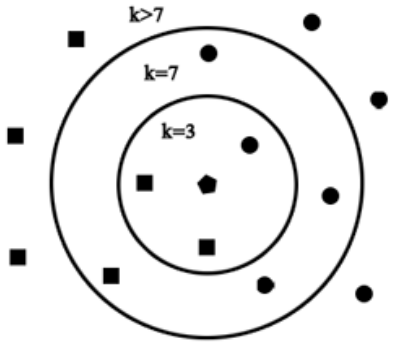
$$dist(x_i, x_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{it} - x_{jt})^2}$$

där även x_j likt x_i är ett godtyckligt dokument ($j = 1, 2, \ldots, n$).

Låt $d_i \in D$ beteckna ett träningsdokument och c_i förbestämda kategorier för träningsdokumenten. När en ny fråga q skall klassificeras undersöker man hur många dokument från varje kategori de k närmsta dokumenten från frågan representerar. Med andra ord beräknas $dist(q, d_i)$ för $i = 1, 2, \ldots, t$ samtidigt som en lista över de k värden av i som gav lägst avstånd hålls uppdaterad. Dokumentet q tilldelas sedan den kategori som förekommer flest gånger i listan.

Valet av konstanten k beror i regel på vilken data som används. Ett större värde på k reducerar ofta bruseffekten av klassificeringen men gör gränserna mellan olika kategorier mindre tydliga.

I de fall då enbart två kategorier förekommer väljs k med fördel till ett udda tal då detta förhindrar att dokumentet klassificeras med samma avstånd mellan två kategorier.



Figur 3.1. Figuren visar hur KNN väljer vilken klass ett brev tillhör beroende på valet av k. Om cirkeln betecknar spam och fyrkanten icke-spam så klassificeras dokumentet som icke-spam om $k = 3$. Om däremot k väljs till 7 klassificeras brevet som spam.

3.2.3 Stödvektormaskiner

Stödvektormaskiner (SVM) är ett samlingsnamn på handledda klassificeringsmetoder och regression som tillhör generaliserade linjära klassificerare. Algoritmerna är relativt nya och är på många sätt motsatsen till NBC. Det som karaktäriserar SVMs är att de är väldigt kraftfulla men även mycket komplexa, långsamma och minneskrävande. [9]

Givet ett plan i en $(n+1)$-dimensionell rymd $\mathbb{R}^{n+1}$ definieras punkter i planet på formen $(\mathbf{x}_1, c_1), (\mathbf{x}_2, c_2), \ldots, (\mathbf{x}_p, c_p)$ där konstanten c tilldelas 1 eller -1 och specificerar vilken av två kategorier punkten tillhör. Variabeln $\mathbf{x}$ är en n-dimensionell vektor av vikter (se 2.1).

Vid tillämpandet av en SVM vill man genom att dela upp den $(n + 1)$-dimensionella rymden urskilja de olika dokumenten genom att finna ett hyperplan som delar upp punkterna i den samling punkter som ligger på ena

sidan av planet respektive den andra sidan. På detta sätt vill man finna en så bra uppdelning som möjligt.

Hyperplanet antar formen $\mathbf{w} \cdot \mathbf{x} - b = 0$ där vektorn $\mathbf{w}$ är rätvinklig till hyperplanet. Variabeln b ger hyperplanet en viss marginal, denna variabel gör bl.a. att hyperplanet inte forceras att gå genom origo. Målet vid beräkningen av hyperplanet är att maximera marginalen mellan de två klasserna så mycket som möjligt.

För att ta fram den maximala marginalen måste parallella hyperplan samt stödvektorer tas fram. De parallella hyperplanen ges enligt nedan.

$$\mathbf{w} \cdot \mathbf{x} - b = 1$$

$$\mathbf{w} \cdot \mathbf{x} - b = -1$$

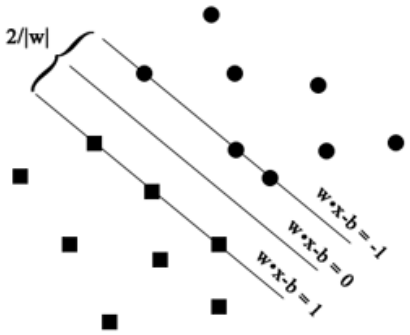
Om rymden är linjärt separabel kan dessa plan väljas så att inga punkter existerar mellan dem och på så sätt maximera avståndet mellan dessa till $d = \frac{2}{|\mathbf{w}|}$. Således måste $|\mathbf{w}|$ minimeras. Stödvektorerna ges av punkterna längst hyperplanen. För exkludering av datapunkter måste följande förutsättning uppfyllas.

$$c_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1, 1 \leq i \leq n$$

Att minimera $|\mathbf{w}|$ givet ovanstående ekvation blir ett kvadratiskt optimiseringsproblem d.v.s. minimera $\frac{2}{\|\mathbf{w}\|^2}$ givet $c_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1$, $1 \leq i \leq n$.

3.3 Sammanställning

Naive Bayes classifier är enkel, effektiv algoritm som fungerar för de allra flesta fall då vanliga texter skall klassificeras. Metoden lär sig väldigt snabbt d.v.s. kräver inte en så stor mängd träningsdata för att ge bra resultat. För



Figur 3.2. Figuren visar hyperplan med maximal marginal i $\mathbb{R}^2$.

mer avancerad indata är metoden dock otillräcklig. En fördel med probabilistiska metoder är att det går att säga hur pass säkert systemet är på att klassificeringen är korrekt. Om värdet är väldigt lågt kan det övervägas om ett dokumentet skall förbli oklassificerat eller rent av klassificeras manuellt.

Support Vector Machines är komplexa och långsamma men betydligt mer kraftfulla än NBC. En fördel med SVMs är att de kan hantera väldigt många attribut (vilket leder till hög dimensionalitet). Nackdelar är att de kräver mycket minne och processorkraft. När träningsmängden blir alltför stor blir algoritmen ineffektiv men för små mängder träningsdata kan SVMs vara ett bra alternativ.

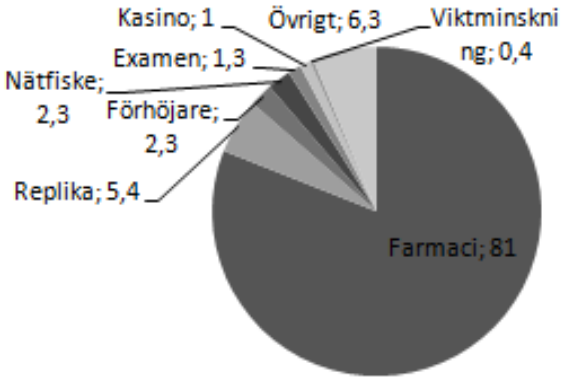
KNN är en enkel klassificerare som är billig processorkraftsmässigt under träningsfasen men väldigt dyr under klassificeringsfasen. Hela träningsmängden måste itereras igenom för varje testdata, detta är långsamt men leder även till att man får tillgång till avstånden från frågan till samtliga kategorier varför metoden är bra när det är väldigt många kategorier inblandade. KNN tar även upp väldigt mycket minne då samtliga dokument måste sparas i minnet. De kan inte slås ihop som vid utförandet av NBC utan varje dokument i träningsmängden måste finnas tillgängligt under klassificeringsfasen.

Kapitel 4

Vad är spam?

4.1 Allmänt

En vanlig definition på spambrev är att de skickas till ett stort antal mottagare som i regel inte efterfrågat dessa brev. De som skickar spambrev samlar ofta på sig e-postadresser (vilka de sedan skickar breven till) från offentliga webbsidor, nyhetsgrupper och liknande. Innehållet i spambrev är oftast av likartad karaktär vilket figur 1 nedan visar. [10]



Figur 4.1. Figuren visar uppdelningen av ämnen som spambrev vanligtvis innefattar.

I de allra flesta fall är anledningen till att spambrev skickas att skaparna vill göra reklam för olika produkter och tjänster, ofta illegala. I andra fall försöker avsändaren utföra bedrägeriförsök (även kallat nätfiske eller phising) där mottagaren är tillfrågad att fylla i personuppgifter, bankkontonummer eller liknande.

4.2 Spamtekniker

Medan spamfiltreringsteknikerna blir allt mer avancerade blir även skaparna av spambrev allt bättre på att kringgå filtren. Här beskrivs de vanligaste metoderna för att kringgå spamfilter som används idag. [11]

Bayesiansk förgiftning är en teknik som används för att få spambrev att kringgå spamfilter byggda med bayesianska metoder. Metoden går ut på att lägga till antingen slumpmässiga eller utvalda ord till brevet för att filtret skall tolka brevet som något annat än just spam under viktningsfasen.

Ett sätt att gömma spammeddelanden är att göra bilder av texten man vill presentera. Många spamfilter undersöker inte bilder varför dessa spambrev kan släppas igenom.

Ytterligare ett sätt att försöka kringgå spamfilter är att rubba inläsningsdelen. Exempel på detta är att lägga till tecken i de ord man vill framföra till läsaren av brevet. Ett exempel på detta skulle kunna vara M-O-N-E-Y istället för ordet MONEY. Vid vanlig inläsning väljs karaktärerna M, O, N, E och Y ut och tolkas var för sig. Andra exempel är att man med avsikt stavar ord fel eller byter ut bokstäver som liknar varandra (exempelvis S mot $ eller A mot @).

Att förhindra spamteknikerna är ett väldigt komplicerat ämne i sig men krävs för att skapa de riktigt effektiva spamfiltren som används av de mest framstående e-postklienterna idag.

Kapitel 5

Utförande

5.1 Specifikation

Uppgiften som beskrivs i detta avsnitt är att skapa en klassificerare för spam respektive icke-spam. Klassificeringsprogrammen är beroende av externa källor och skulle kunna utvidgas till fler än två kategorier. Det är alltså textklassificering genom handledd inlärning som tillämpas. Vidare är problemet av typ mono classification.

Klassificerarna är skrivna i Java Standard Edition 6 och utvärderade på ett system med 3.00 GHz AMD Phenom II X4 945 Processor med operativsystem Windows 7 64-bit.

Nedan följer en överblick över problemet:

1. Bilda en träningsmängd och klassificera dessa manuellt.
2. Applicera träningsalgoritmer på träningsmängden (träna systemet).
3. Applicera klassificeringsalgortmerna på en testmängd och utvärdera.
4. Upprepa 2 och 3 med nya parameterval tills ett tillräckligt bra resultat erhållits.

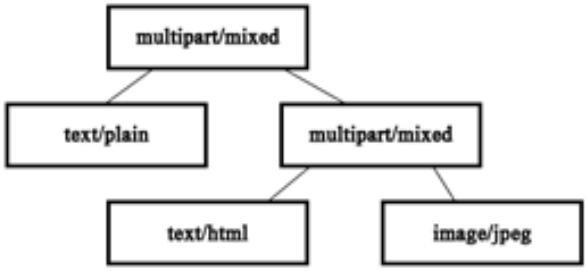
5.2 MIME-formatet

Då e-postmeddelanden hämtas av e-postklienter sparas breven i ett format specificerat av klienten. Det allra vanligaste formatet är *eml* vilket används av kända klienter som Microsoft Outlook Express, Windows Mail och Mozilla Thunderbird. Övriga format är *emlx* (Apple Mail), *msg* (Microsoft Office Outlook) och *mbx* (Opera Mail m.fl.). Detta projekt baseras på användandet av filer med ändelse eml. [12]

E-post med filändelsen eml är kodade i vanlig text skrivet på MIME-format (*Multipurpose Internet Mail Extensions*). MIME tillåter bl.a. att brevet att skickas dels i ren text, dels i HTML. Formatet tillåter även bilder och bilagor. MIME är även bakåtkompatibelt så att det fungerar med den äldre RFC/822-standarden. Att notera är att HTML inte kan läsas in som vanlig text för att sedan indexeras utan måste behandlas först för att ta bort taggar (se 5.3.2).

Ett brev i MIME-format har alltid en header som är av en viss typ (*Content-Type*). Denna typ kan vara av ett antal olika sorter men de som behandlas i denna rapport är i huvudsak **multipart/x** samt **text/y** där **x** kan ersättas med **mixed**, **alternative** m.m. beroende på innehållet och **y** kan ersättas med **html** eller **plain**. Brevets struktur kan beskrivas som ett träd där **text/y** är ett löv och **mutipart/x** ett inre hörn. Med andra ord måste hela trädet traverseras för att all data skall extraheras ur brevet.

Övriga huvudtyper utöver **multipart** och **text** är **image** och **application** som representerar en bild respektive ett datorprograms egna format (exempelvis **application/pdf**). Dessa kan, om de inte skall analyseras bortses från. Slutligen måste även noteras att **multipart/alternative** innefattar ett antal typer som alla är av samma information fast av olika typer. Exempelvis skulle **multipart/alternative** kunna ha två subtyper **text/plain** och **text/html** som egentligen förmedlar samma information och där egentligen endast en av dessa subtyper behöver undersökas.



Figur 5.1. Figuren visar en struktur av ett e-postmeddelande med formatet MIME.

5.3 Paket

5.3.1 Java Mail

För att kunna läsa e-post i MIME-format effektivt används paketet *JavaMail*. JavaMail tillhandahåller ett platformsoberoende och protokolloberoende ramverk för applikationer som använder e-post och liknande meddelanden. JavaMail API finns tillgängligt som ett frivilligt paket till Java SE plattformen.

Klassen *MimeMessage* används för att representera ett MIME-formaterat e-postbrev. Då man instansierar objekt av denna typ kan man efter att ett brev lästs in utvinna titel, avsändare samt innehåll. Innehållet fås som antingen ett Multipart-objekt eller ett BodyPart-objekt varav Multipart-objektet måste traverseras som tidigare beskrivits i avsnitt 5.2. [13]

5.3.2 HTML-Parser

HTML Parser är ett paket öppen källkod till Java som bl.a. används för att hantera HTML-formaterade dokument. När ett brev av typ **text/html** läses in så innehåller texten ett antal html-taggar som behöver tas bort. Detta kan enkelt göras med paketet HTML Parser som kan användas för att läsa in en HTML-fil samt extrahera de faktiska orden. [14]

5.4 Korpus

En korpus är en stor samling språkbaserad data för användning vid språkforskning (exempelvis en stor samling dokument). För att få en så bra representation som möjligt vill man att en korpus har ett så naturligt språkbruk som möjligt för den texttyp, genre, tidsperiod eller liknande som avses. [15]

För att utvärderingen av klassificeringsmetoderna skall bli så noggrann som möjligt krävs att de testdata man använder är bra och rättvist utformade. Dessa är ett par viktiga punkter:

- Språket i samtliga dokument bör vara samma om det inte är själva språket i sig man vill klassificera.
- Samlingen bör vara tillräckligt stor, minst ett par tusen dokument bör användas dels vid träning och dels vid utvärdering.
- Dokumenten skall vara så verklighetstrogna som möjligt. Helst skall dokumenten vara tagna direkt från det sammanhang i vilka metoderna är menade att användas inom.

För utförandet i denna rapport har en korpus vid namn CSDMC2010 använts. Samlingen har använts under informationsutvinningstävlingar och består av ett urval e-postmeddelanden på engelska, väl anpassade för att utvärdera spamfilter. Korpusen innehåller 4327 märkta e-postmeddelanden varav 1378 är märkta spam och 2949 icke-spam. [16]

5.5 Implementation

5.5.1 Generellt

Programmen delas upp i en inläsningsdel, en träningsdel och en klassificeringsdel. Under inläsningsdelen används stopplistan för att filtrera bort neutrala

ord. Dessa ord kommer således aldrig att tas med i beräkningarna.

Inläsningsfaserna ser tämligen likartade ut för båda algoritmerna. Nedan följer pseudokod över utförandet. m_j representerar ett brevobjekt, M är en lista över samtliga brev och *df* är en hashtabell med nycklar representerandes samtliga förekommande ord och värden som visar hur många dokument ordet förekommer i. Observera att *df* endast behövs vid tillämpning av viktningsprincipen TFIDF.

```
foreach d_j ∈ D do
    foreach w_ij ∈ d_j && !stoplist.contains(w_ij) do
        if m_j.contains(w_ij) then
            m_j.put(w_ij, m_j.get(w_ij) + 1)
        end
        else
            m_j.put(1)
            if df.contains(w_ij) then
                df.put(w_ij, df.get(w_ij) + 1)
            end
            else
                df.add(w_ij)
            end
        end
    end
    M.add(m_j)
end
```

Algorithm 1: Pseudokod för inläsningsfasen till godtycklig klassifierare.

5.5.2 Implementation av Naive Bayes classifier

Ett av de första problemen man stöter på i praktiken vid tillämpning av NBC är att när antalet attribut växer så blir värdena $P(w_j|c_k)$ allt mindre vilket resulterar i överflödesproblem då datorn inte kan hantera hur små tal som

helst vid beräknandet av $P(c_i|Q) = P(w_1|c_i) \cdot P(w_2|c_i) \cdot \ldots \cdot P(w_t|c_i) \cdot P(c_i)$. En lösning på detta är att använda logaritmer.

Istället för att multiplicera ihop produkterna kan logaritmregeln $log(A \cdot B) = \log(A) + \log((B))$ användas och ersätta $P(c_i|Q)$ med $log(P(c_i|Q)) = \log P(w_1|c_i) + \log P(w_2|c_i) + \cdots + \log P(w_t|c_i) + \log P(c_i)$. Den mest sannolika kategorin ges av det största värdet på $\log(P(c_i|Q))$ av de n kategorierna.

Låt som exempel $P(c_i|Q) = \{0.0002, 0.0001, 0.0003\}$. Med den ordinarie metoden erhålls produkten av dessa som 0.000000000006. Vi ser snabbt att detta värde kommer avrundas mot noll om fler attribut hade funnits. Om istället logaritmer används fås $\log(0.0002) - \log(0.0001) - \log(0.0003) = -3.699 - 4 - 3.523 = -11.222$ vilket är ett betydligt mer hanterbart värde.

Vid tränandet av programmet itereras varje brevobjekt igenom för att sedan värdet på varje ord i i brev j skall beräknas enligt TF eller TFIDF. De m ord med högst värde väljs ut och läggs sedan till i den hashtabell $trainingData_0$ eller $trainingData_1$ som brevet tillhör (d.v.s. spam eller icke spam).

Vid utvärderingen av algoritmen läses ord för ord in från testdokumenten för att sedan utvärderas efter hur sannolikt det är ordet är av spam- respektive icke-spamkaraktär. Detta värde adderas till variabler för spam samt icke-spam och om totalvärdet för spam överstiger det för icke-spam klassificeras brevet som spam (eller motsvarande för icke-spam).

foreach $m_j \in M$ **do**
  foreach $w_i \in m_j$ **do**
    $val_i \leftarrow TF$ or $TFIDF$
  end
  Sortera orden w m.a.p. val i fallande ordning.
  for $i = 0$ *to* m **do**
    if $trainingData[c_j].contains(w_i)$ **then**
      $trainingData[c_j].put(trainingData[c_j].get(w_i) + 1)$
    end
    else
      $trainingData[c_j].add(w_i)$
    end
  end
end

Algorithm 2: Pseudokod över träningsfasen för NBC.

foreach $q_j \in Q$ **do**
  $ham \leftarrow spam \leftarrow 0$
  foreach $w_i \in q_j$ **do**
    if $trainingData[0].contains(w_i)$ **then**
      $spam = spam + \log(\frac{trainingData.get(w_i)}{totalWords[0]})$
    end
    else
      $spam = spam + \log(\frac{0.1}{totalWords[0]})$
    end
    if $trainingData[1].contains(w_i)$ **then**
      $ham = ham + \log(\frac{trainingData.get(w_i)}{totalWords[1]})$
    end
    else
      $ham = ham + \log(\frac{0.1}{totalWords[1]})$
    end
  end
  if $ham > spam$ **then**
    Klassificerad som icke-spam.
  end
  else
    Klassificerad som spam.
  end
end

Algorithm 3: Pseudokod över klassificeringsfasen för NBC.

5.5.3 Implementation av K-Närmsta granne

Vid träningen av programmet användes en hashtabell för varje brev istället för endast två stycken (som vid träningen av NBC). Nedan följer pseudokod över träningsfasen.

foreach $m_j \in M$ **do**
  foreach $w_i \in m_j$ **do**
    $val_i \leftarrow TF$ or $TFIDF$
  end
end

Algorithm 4: Pseudokod över träningsfasen för KNN.

Under klassificeringsfasen beräknas för varje ord i det nya brevet deras avstånd till samtliga brev som undersökts i träningsfasen. När samtliga ord undersökts har ett totalt avstånd för varje brev i träningsmängden till brevet som skall undersökas beräknats. De k brev med lägst sådant värde väljs ut och dess kategorier hämtas ut. Den kategori som förekommer flest gånger är den kategori som det nya brevet ges. Detta upprepas för varje brev i testmängden.

foreach $q_j \in Q$ **do**
  foreach $w_i \in q_j$ **do**
    for $t = 0$ *to #training examples - 1* **do**
      if $trainingData[t].contains(w_i)$ **then**
        $score[t] = score[t] + (q_j.get(w_i) - trainingData[t].get(w_i))^2$
      end
      else
        $score[t] = score[t] + (q_j.get(w_i))^2$
      end
    end
  end
  Sortera *score* in stigande ordning.
  Välj den mest frekventa kategorin bland de k första elementen i *score*.
end

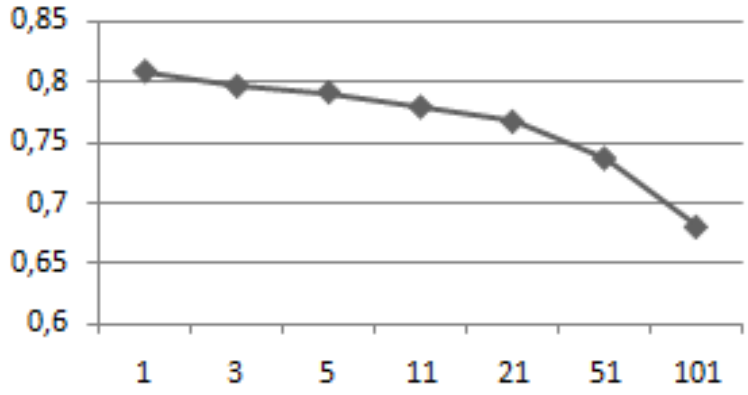
Algorithm 5: Pseudokod över klassificeringsfasen för KNN.

Kapitel 6

Resultat

Under träningsfasen användes alltid lika många spamklassificerade brev som icke-spam. För själva utvärderingen används testdata bestående av 1000 e-postbrev. Stopplistan består av 322 neutrala, vanligt förekommande ord (se A.4).

Valet av k till klassificeraren KNN undersöktes genom att testa olika värden på konstanten och köra programmet med övriga värden oförändrade (se A.2).

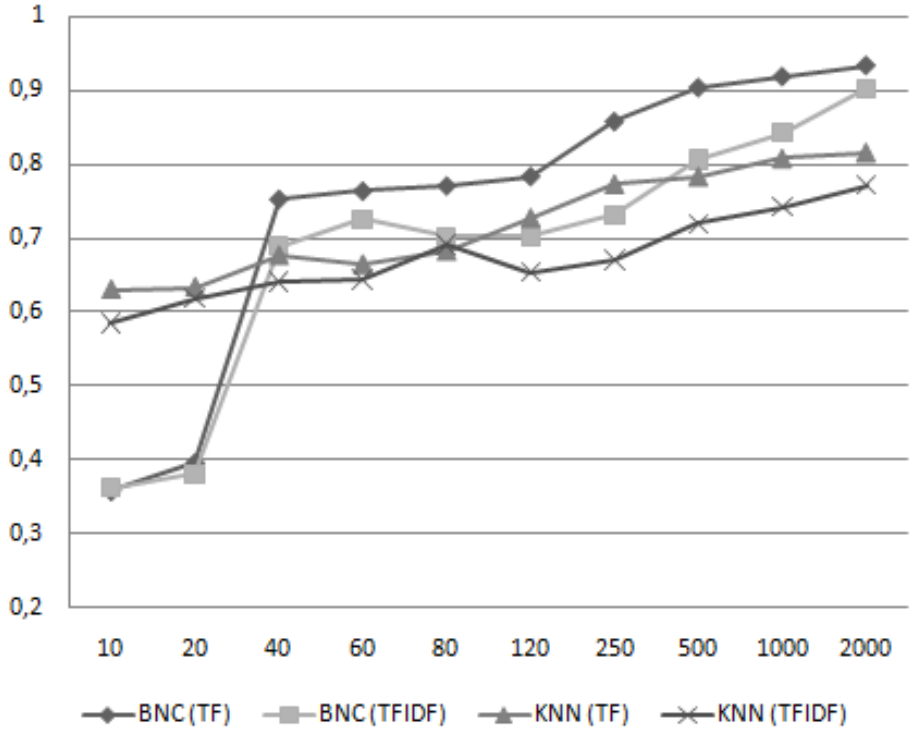


Figur 6.1. Figuren visar hur KNN presterar efter val av konstanten k. Y-axeln visar antal korrekt klassificerade brev procentuellt.

Undersökningen visar att ett lägre värde på k i regel ger bättre resultat. Kon-

stanten k har därmed i fortsättningen valts till 3.

Det slutgiltiga resultatet för klassificerarna ges av att köra bägge algoritmerna med de två presenterade viktningsprinciperna TF samt TFIDF. Olika stora mängder träningsdata används för att se hur algoritmerna presterar vid olika förhållanden. Figur 6.2 nedan visar resultaten.



Figur 6.2. Diagrammet visar hur BNC och KNN presterar efter hur mycket träningsdata de matas med. (se A.1, A.3)

Av diagrammet kan följande konstateras.

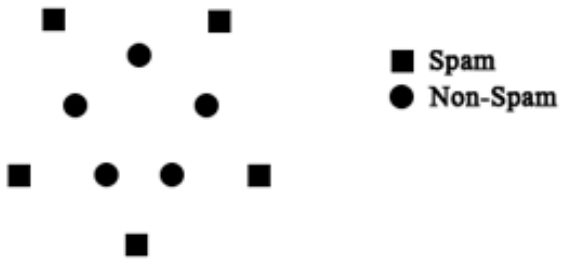
- Viktningsprincipen TFIDF ger en sämre bedömning än TF för bägge klassificerarna.
- Klassificeraren NBC presterar bättre än KNN när träningsmängden överstiger c:a 30 träningsbrev.
- Som bästa resultat för klassificerarna erhölls en precision av 93% för NBC respektive 82% för KNN.

Kapitel 7

Diskussion

De resultat som erhålls är förhållandevis väntade. Att ett lågt värde på k vid tillämpning av KNN gav ett bättre resultat torde tyda på att spambrev i regel liknar vanliga brev men har någon avskiljande faktor som ter sig annorlunda mot andra spambrev medan vanliga brev är ihopklumpande (eller tvärtom).

Illustrationen nedan ger en möjlig bild över hur breven kan vara utplacerade efter träningsfasen. Under testfasen kan brev som borde klassificeras som spam ha placerats i ytterkanterna av figur 7.1 och således svara bra vid $k = 1$ men sämre vid $k > 1$.



Figur 7.1. Figuren visar en möjlig bild över hur KNN kan ha placerat ut breven. Notera att antalet dimensioner egentligen är betydligt fler än två.

Anledningar till att viktningsprincipen TFIDF presterar sämre än TF är inte lätta att sätta fingret på. Det har dock påvisats att även om TFIDF är ett väldigt populärt medel inom informationsutvinningsområdet ger viktningsprincipen inte alltid ett särskilt bra resultat vid just textklassificering viket denna undersökning bekräftar. [17]

Att KNN presterar bättre än NBC för väldigt låga mängder träningsdata bör bero på att NBC har egenskapen hög bias i utbyte mot låg varians vilket är motsatsen till de egenskaper KNN har. Hög bias innebär att algoritmen har för få parametrar relativt storleken av träningsdata och på så sätt gör grova uppskattningar. Låg bias innebär istället att modellen har för många parametrar relativt storleken av träningsdata. KNN kan därför vara en mer flexibel algoritm vid väldigt låga mängder träningsdata. [18]

De bästa resultaten som erhölls (93% resp. 82%) kan för den tämligen låga mängden träningsdata anses vara ett halvbra resultat. Hade större mängder träningsdata använts hade procentandelen rätta klassificeringar förmodligen blivit betydligt bättre.

Det finns även ett antal ytterligare tekniker som hade kunnat förbättra resultatet. Stemming hade gjort att ord i olika former slagits ihop och betraktats som lika vilket de inte gör i denna undersökning. Vidare hade en större vikt kunnat läggas vid att motverka spamteknikerna (se 4.2). Slutligen hade förmodligen stödvektormaskiner gett ett gott resultat även om det är svårt att säga hur bra de hade presterat i praktiken.

Kapitel 8

Slutsats

Denna rapport har behandlat metoder för klassificering av e-post i dels det generella fallet och dels i det mer nischade fallet spam respektive icke-spam. Hur man representerar och behandlar e-postbrev har studerats och olika viktningsprinciper har presenterats. Klassificeringsmetoderna Naive Bayes classifier, K-Nearest neighbor och stödvektormaskiner har undersökts varav två implementerats och testats i praktiken med ett lovande resultat.

De slutsatser man kan dra av undersökningen som helhet är att ett enklare angreppssätt ofta ger ett bättre resultat än mer komplicerade metoder. Klassificeraren NBC med den enklaste viktningsprincipen TF gav ett bättre resultat än klassificeraren KNN samt ett bättre resultat än NBC med den mer avancerade viktningsprincipen TFIDF. Något som bör has i åtanke är dock att så länge man har gott om träningsdata så fungerar de flesta klassificerare alldeles utmärkt.

Kapitel 9

Referenser

1. Dan Fletcher (2009) *A Brief History Of Spam* http://www.time.com/time/business/article/0,8599,1933796,00.html, 2009-11-02, 2011-04-14

2. Section for Digital Signal Processing, IMM, DTU (1999) *Document Classification* http://cogsys.imm.dtu.dk/thor/projects/multimedia/textmining/node11.html, 2011-04-14

3. Section for Digital Signal Processing, IMM, DTU (1999) *Vector Space Model* http://cogsys.imm.dtu.dk/thor/projects/multimedia/textmining/node5.html, 2011-04-14

4. Magnus Rosell (2002) *Klustring av svenska tidningsartiklar* http://www.csc.kth.se/~rosell/thesis/rosell_masters_thesis.pdf, 2002-03-08 , 2011-04-14

5. William J. Turkel (2008) *A Naive Bayesian in the Old Bailey* http://digitalhistoryhacks.blogspot.com/2008/06/naive-bayesian-in-old-bailey-part-8.html 2008-06-18, 2011-04-14

6. C. H. A. Koster (2003) *Text classification (lecture notes)* http://www.cs.ru.nl/~kees/ir2/papers/h03.pdf, 2011-04-14

7. John Graham-Cumming (2005) *Naive Bayesian Text Classification* http://drdobbs.com/tools/184406064, 2005-05-01, 2011-04-14

8. Victor Lavrenko, Charles Sutton (2010) *Nearest neighbour method* http://www.inf.ed.ac.uk/teaching/courses/iaml/slides/knn-2x2.pdf, 2011-04-14

9. Institut fur Informationssysteme und Computer Medien *Support Vector Machine* http://www.iicm.tugraz.at/cguetl/courses/isr/opt/classification/Support_Vector_Machine.html, 2011-04-14

10. Commtouch Software Ltd. (2010) *Q1 2010 Internet Threats Trend Report* http://www.commtouch.com/download/1679, 2011-04-14

11. Michael Fong (2008) *Spam or Ham* http://www.shannarasite.org/project/Spam/report.pdf, 2008-12-12, 2011-04-14

12. Mark Grand (1993) *(*MIME Overview) http://mgrand.home.mindspring.com/mime.html, 1993-10-26, 2011-04-14

13. Oracle (2011) *JavaMail* http://www.oracle.com/technetwork/java/javamail/index.html, 2011-04-14

14. Derrick Oswald, Somik Raha m.fl. (2006) *HTML Parser*http://htmlparser.sourceforge.net/, 2011-04-14

15. Viggo Kann (2006) *Lexikon och datasamlingar* http://sprakteknologi.se/vad-aer-sprakteknologi/lexikon/korpusar, 2006-03-20, 2011-04-14

16. CDM (2010) *CSDMC2010 SPAM corpus* http://csmining.org/index.php/spam-email-datasets-.html, 2011-04-14

17. Boley, D., Gini, M., m.fl. (1999) *Partitioning Based Clustering for Web Document Categorization* http://maya.cs.depaul.edu/~mobasher/papers/webace-dss.pdf, 2011-04-14

18. Hedvig Kjellström (2011) *DD2475 Information Retrieval Lecture 9: Vector Space Classification* `http://www.csc.kth.se/utbildning/kth/kurser/DD2475/ir10/forelasningar/Lecture9_4.pdf`, 2011-25-01, 2011-04-14

Bilaga A

Appendix

A.1 NBC

Testdata	**TF**	**TFIDF**
10	0.358	0.362
20	0.397	0.381
40	0.753	0.689
60	0.764	0.726
80	0.771	0.702
120	0.784	0.703
250	0.858	0.731
500	0.904	0.806
1000	0.919	0.843
2000	0.934	0.902

A.2 KNN efter val av k.

k	Precision
1	0.808
3	0.796
5	0.791
11	0.779
21	0.767
51	0.737
101	0.681

A.3 KNN ($k = 3$)

Testdata	TF	TFIDF
10	0.630	0.585
20	0.633	0.618
40	0.676	0.641
60	0.664	0.644
80	0.682	0.691
120	0.727	0.654
250	0.773	0.670
500	0.783	0.720
1000	0.808	0.742
2000	0.816	0.772

A.4 Stopplista

a, about, above, across, after, afterwards, again, against, all, almost, alone, along, already, also, although, always, am, among, amongst, amoungst, amount, an, and, another, any, anyhow, anyone, anything, anyway, anywhere, are, around, as, at, back, be, became, because, become, becomes, becoming,

been, before, beforehand, behind, being, below, beside, besides, between, beyond, bill, both, bottom, but, by, call, can, cannot, cant, co, computer, con, could, couldnt, cry, de, describe, detail, did, do, done, down, due, during, each, eg, eight, either, eleven, else, elsewhere, empty, enough, etc, even, ever, every, everyone, everything, everywhere, except, few, fifteen, fifty, fill, find, fire, first, five, for, former, formerly, forty, found, four, from, front, full, further, get, give, go, had, has, hasnt, have, he, hence, her, here, hereafter, hereby, herein, hereupon, hers, herself, him, himself, his, how, however, hundred, i, ie, if, in, inc, indeed, interest, into, is, it, its, itself, keep, last, latter, latterly, least, less, ltd, made, many, may, me, meanwhile, might, mill, mine, more, moreover, most, mostly, move, much, must, my, myself, name, namely, neither, never, nevertheless, next, nine, no, nobody, none, noone, nor, not, nothing, now, nowhere, of, off, often, on, once, one, only, onto, or, other, others, otherwise, our, ours, ourselves, out, over, own, part, per, perhaps, please, put, rather, re, s, same, see, seem, seemed, seeming, seems, serious, several, she, should, show, side, since, sincere, six, sixty, so, some, somehow, someone, something, sometime, sometimes, somewhere, still, such, system, take, ten, than, that, the, their, them, themselves, then, thence, there, thereafter, thereby, therefore, therein, thereupon, these, they, thick, thin, third, this, those, though, three, three, through, throughout, thru, thus, to, together, too, top, toward, towards, twelve, twenty, two, un, under, until, up, upon, us, very, via, was, we, well, were, what, whatever, when, whence, whenever, where, whereafter, whereas, whereby, wherein, whereupon, wherever, whether, which, while, whither, who, whoever, whole, whom, whose, why, will, with, within, without, would, yet, you, your, yours, yourself, yourselves