Förbättringar av några metoder inom maskininlärning

Benjamin Åberg

Kandidatavhandling i datateknik
Handledare: Andreas Lundell
Fakulteten för naturvetenskap och teknik
Åbo Akademi
2022

Abstrakt

När det kommer till problemlösning är det optimala resultatet så gott som alltid eftersträvansvärt. Detta gäller förutom konkreta problem också problem inom mjukvaruvärlden. Just den här aspekten av mjukvaruutveckling har alltid funnits i fokus, oberoende av vilken sorts mjukvara det gäller. Maskininlärning, som till sin natur kan vara väldigt tidskrävande, är heller inget undantag. Optimering har sedan början utgjort en stor del av maskininlärning och kan ses som ett av de viktigaste verktygen vid utveckling och användning av maskininlärningsalgoritmer. Förutom att spara resurser och tid, har optimering med tiden också möjliggjort utvecklingen av alltmer komplexa algoritmer.

Syftet med denna avhandling är att jämföra och diskutera några förbättringsförslag för ett antal maskininlärningsalgoritmer samt lyfta fram vikten av optimering när det kommer till maskininlärning.

Sökord: maskininlärning, optimering, optimeringsmetoder, bayes, svm, kluster

Innehållsförteckning

1. Introduktion

Maskininlärning och optimering har redan från början gått hand i hand och påverkat varandra på olika sätt. Redan i de första maskininlärningsalgoritmerna utnyttjades optimeringsteorier och -metoder och samtidigt har maskininlärningsproblem i sin tur gett upphov till nya optimeringsmetoder vartefter behov uppstått [1].

Maskininlärning delas i allmänhet in i tre huvudsakliga kategorier: vägledd inlärning (eng. *supervised learning*), icke-vägledd inlärning (eng. *unsupervised learning*) och förstärkningsinlärning (eng. *reinforcement learning*) [2]. Vägledd inlärning innebär att en modell tränas med extern hjälp genom att algoritmens output för en specifik input jämförs med det eftersträvansvärda resultatet. Modellen justeras sedan baserat på avvikelsen i resultatet [2]. Icke-vägledd inlärning innebär i motsats till vägledd inlärning att modellen tränas utan extern hjälp, då optimalt resultat är okänt. Modellen analyserar istället underliggande mönster och egenskaper och approximerar baserat på dessa det optimala resultatet [2]. Den tredje huvudsakliga kategorin av maskininlärning är förstärkningsinlärning. Det innebär att modellen tränar sig själv baserat på dess omgivning. Modellen har en startpunkt och en slutpunkt och beroende på hur modellen väljer att ta sig mellan dessa lär den sig vilka handlingar som leder till framsteg och vilka ej [2].

Denna avhandling kommer i synnerhet att behandla förbättring inom datavetenskapens följande tre tillämpningsområden av maskininlärning: (i) datautvinning, (ii) svårprogrammerade applikationer samt (iii) självanpassande mjukvara.

1.1 Datautvinning

Med datautvinning (eng. *data mining*) avses inom maskininlärning användningen av historiska data för att förbättra nuvarande modeller och beslutsfattning [3]. Exempelvis

kan man inom sjukvården analysera tidigare patientdata och få en uppfattning om samband mellan symptom och sjukdomar. I praktiken tilldelas varje patient en lista attribut såsom symptom, ålder, kön och så vidare. Algoritmen bygger sedan upp en modell baserat på denna data där samband mellan attribut framkommer. Med hjälp av tidsbeaktande data kan man också förutspå när i sjukdomsförloppet patienter har en förhöjd risk för att utveckla följdsjukdomar eller dylikt [3].

1.2 Svårprogrammerade applikationer

Med svårprogrammerade applikationer menas i detta sammanhang applikationer som generellt sett klassas som svåra att programmera manuellt, det vill säga med traditionella metoder. Exempel på dessa är applikationer inom röst-, bild- och ansiktsigenkänning. Det har visat sig vara så gott som omöjligt att med traditionella medel uppnå samma noggrannhet och resultat som med maskininlärningsalgoritmer och därför används maskininlärning i någon form i så gott som alla bild-, röst- och ansiktsigenkänningsapplikationer idag [3].

1.3 Självanpassande mjukvara

Ett tredje användningsområde inom datavetenskap för maskininlärning är självanpassande mjukvara, det vill säga mjukvara som anpassar sig till användaren. Den här typens mjukvara utnyttjas bland annat av nyhetssajter [3], online-filmtjänster och webbsidor avsedda för videoströmning. I praktiken innebär det att material rekommenderas till användaren baserat på det slags innehåll som användaren tidigare konsumerat, och en modell baserat på användarens intresse byggs upp. På en filmbaserad webbsajt kan olika filmer beskrivas av en mängd attribut såsom kategori, längd, producent, med mera. I bakgrunden analyseras det konsumerade materialet av

en algoritm och på så sätt skapas en modell över användarens intresse för att liknande material ska kunna rekommenderas [4].

2. Maskininlärningsmetoder och optimering

Maskininlärning fick sin början redan på 1950-talet i och med maskinen ’Perceptron’ skapad av psykolog Frank Rosenblatt [5]. Rosenblatt sägs ha baserat maskinen på människans nervsystem och den kunde känna igen bokstäver och bilder [5]. Maskinens sätt att lära sig liknade starkt dåtidens modeller för hur människor och djur lär sig och lade grunden för de moderna artificiella neurala nätverk som används ännu i dagens läge [5].

Under 1960-talet gjordes stora framsteg inom forskningen inom maskininlärning och både deterministiska (exempelvis stödvektormaskin, se kap 2.1) och stokastiska metoder utvecklades [5]. Efter 1960-talet kom vad som ibland brukar kallas för *den första vintern inom AI,* det vill säga en period då forskningen minskade drastiskt inom området [5]. Denna period pågick ända tills 1980-talet då Kunihiko Fukushima presenterade ett nytt slags neuralt nätverk vid namn ”neocognitron Fukushima” som på nytt väckte intresset för forskning inom området [5]. Forskningen pågick i ungefär 10 år innan den period som brukar kallar för *den andra vintern inom AI* inträffade på tidigt 90-tal. År 1995 gjordes dock stora framsteg då Cortes och Vapnik presenterade en ny version av stödvektormaskin-algoritmen vid namn *support-vector networks* som återupplivade forskningen på många håll [5].

Från 2000-talets början och framåt har de största framstegen inom maskininlärning skett och början på millenniet visade sig vara en vändpunkt inom maskininlärningens historia [5]. Exempelvis fenomenet *Big Data,* som refererar till de stora mängder data som nuförtiden behöver lagras och hanteras, gav upphov till forskning inom

datahantering, inte bara på grund av kuriositet, utan på grund av behov av nya datahanteringssätt [5]. Forskning inom djupinlärning (eng. *deep learning*) fick också sitt genombrott på 2000-talet och för tillfället forskas det aktivt inom området [5].

Maskininlärning är till sin ålder fortfarande relativt nytt, med många outforskade områden. Utvecklingen av maskiner som i högre grad fungerar som människor vad beträffar lärande är fortfarande i ett tidigt stadie, dock med aktiv forskning [6]. För tillfället är exempelvis de flesta maskininlärningsalgoritmerna fokuserade på endast ett område till motsats från människan som kan lära sig många olika saker. Utveckling av algoritmer som kan arbeta med andra algoritmer för att lösa problem är ett annat intressant område för forskningen [6].

I och med den snabba utvecklingen av maskininlärning har också optimering inom området fått stor uppmärksamhet av forskare [7]. På senaste tid har också fenomen såsom *Big Data* introducerat nya utmaningar vad gäller lagring och hantering av stora mängder data, vilket har lett till att en hel del nya optimeringsförslag introducerats [7].

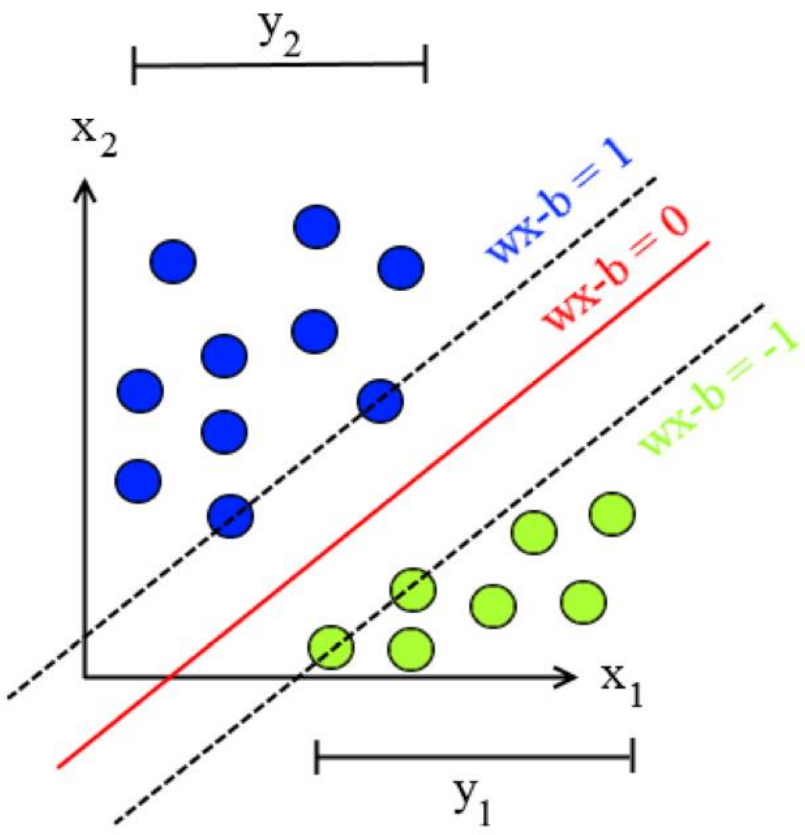
Optimeringsmetoder delas generellt in i tre kategorier: (1) optimering av första graden (eng. *first-order optimization*), (2) optimering av högre grad (eng. *high-order optimization*) samt (3) deriveringsfri optimering (eng. *derivative-free optimization*) [7]. I detta kapitel presenteras några vanliga maskininlärningsmetoder samt en introduktion till ovannämnda optimeringsmetoder.

2.1 Stödvektormaskin

Stödvektormaskinen (eng. *support-vector machine*, *SVM*) var bland de första metoderna som utvecklades då forskning inom området började [1]. Metoden används främst inom linjär klassifikation och regression och i praktiken fungerar det så att man har en mängd $\{(x_i, y_i)\}$ där $x_i \in \mathbb{R}^n$ är attributvektorer och $y_i \in \{-1, +1\}$ är klassvärden [1], [2]. Metoden bygger på att hitta det så kallade hyperplan (exempelvis i två dimensioner en linje och i tre dimensioner ett plan) som med maximalt avstånd separerar klasserna $y_i = -1$ och $y_i = +1$ [8]. Hyperplanet kan anges i formen:

$$\boldsymbol{w}^T\boldsymbol{x} + \boldsymbol{b} = 0$$

där $\boldsymbol{w} \in \mathbb{R}^n$ är en viktvektor och $\boldsymbol{b} \in \mathbb{R}$ är skärningspunkten med y-axeln [1]. Maximering av avstånd mellan hyperplanet och klasserna innebär i sin tur att klassifikationsfelet minimeras. En enkel version av en SVM beskrivs av figur 2.1 nedan. I kapitel 3 presenteras implementation av SVM i Python och i kapitel 4 ges förslag på en förbättrad version med resultat.



Figur 2.1 Linjär stödvektormaskin där mängderna med gröna och blåa punkter representerar klasser och den röda linjen hyperplanet. Bild inspirerad av [2].

2.2 Naive Bayes

Naive Bayes (NB) är en populär maskininlärningsmetod inom klassifikation som bygger på Bayes teorem (se nedan) vilket innebär att den är fullständigt baserad på sannolikhet [9]. Liksom SVM utnyttjar Naive Bayes klasser och attribut för att för att lära sig och kan på basen av dem förutsäga vilken klass y någonting hör till givet attribut x [9]. För att beskriva detta utgår vi från Bayes teorem:

$$P(y \mid \boldsymbol{x}) = \frac{P(y)P(\boldsymbol{x} \mid y)}{P(\boldsymbol{x})}$$

där $\boldsymbol{x}$ är attributvektor och y klass. Antagandet att alla attribut är oberoende av varandra gäller, det vill säga att förekomsten av ett visst attribut inte påverkar förekomsten av andra [9]. För detta antagande gäller:

$$P(\boldsymbol{x} \mid y) = \prod_{i=1}^{n} P(x_i \mid y)$$

där x_i är det i:te attributet i attributvektorn $\boldsymbol{x}$ och n är antalet attribut [9]. Vidare gäller:

$$P(\boldsymbol{x}) = \prod_{i=1}^{k} P(c_i)(P(\boldsymbol{x} \mid c_i)$$

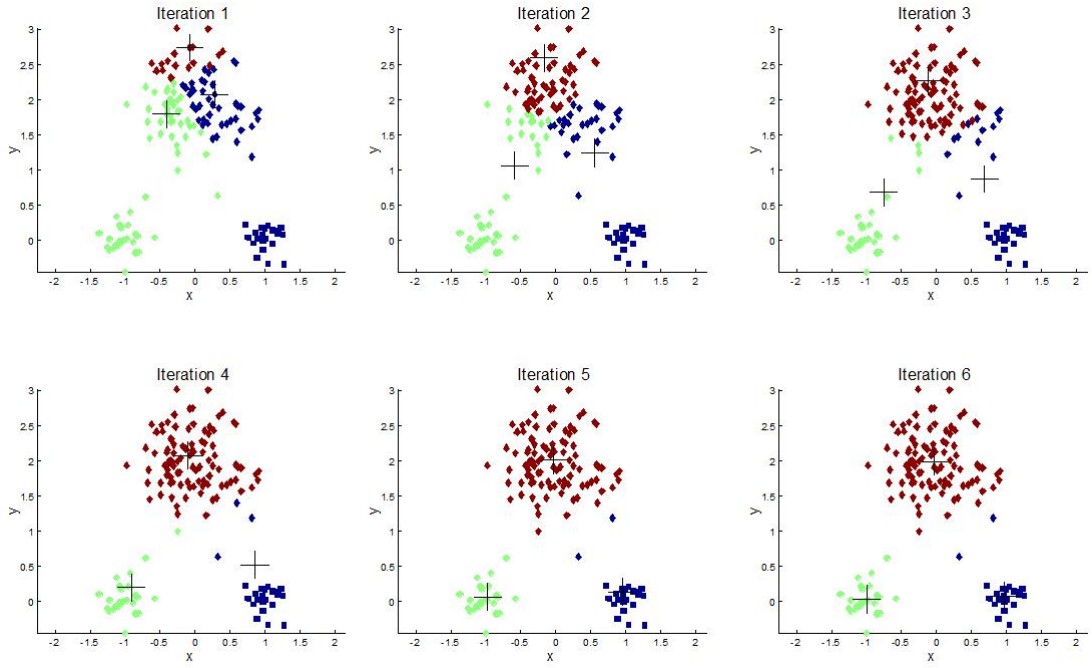
där k är antalet klasser och c_i är den i:te klassen [9]. I kapitel 3 presenteras implementation av NB i Python och i kapitel 4 ges ett exempel på en förbättrad version av NB med resultat.

2.3 K-means-klustring

K-means-klustring eller K-means är en enkel metod inom icke-vägledd maskininlärning och baserar sig på att kategorisera data genom att dela in den i så kallade kluster [2]. I praktiken placeras *k* antal mittpunkter (means) ut, helst så långt

ifrån varandra som möjligt för bästa resultat, och sedan paras varje datapunkt ihop med det kluster som ligger på närmaste avstånd [10]. Då alla datapunkter blivit tilldelade ett kluster beräknar algoritmen mittpunkten för varje kluster på nytt och samma indelning repeteras [10]. Detta steg upprepas tills mittpunkten för varje kluster inte längre förflyttas vilket innebär att indelningen är slutförd [10]. I figur 2.2 visualiseras K-means-algoritmens iterationer.

Eftersom ett initialvärde på k måste väljas manuellt och antalet kluster direkt påverkar resultatet kan man argumentera att K-means-algoritmen inte är icke-vägledd till 100 % [11]. Eftersom algoritmen behandlar data utan attribut och kategoriserar den i kluster betraktas algoritmen allmänt taget ändå som icke-vägledd. I kapitel 3 presenteras implementation av K-means i Python och i kapitel 4 föreslås en förbättrad version av K-means-algoritmen.



Figur 2.2 K-means-algoritmens olika iterationer där de olika färgerna representerar olika kluster [12].

2.4 Optimeringsmetoder av första graden

De vanligaste optimeringsmetoderna av första graden är metoder baserade på kurvans lutning (eng. *Gradient Descent, GD*) [7]. Den enklaste och mest använda varianten av GD innebär att välja en punkt på kurvan som startvärde och sedan uppdatera den iterativt genom att gå i motsatt riktning till kurvans gradient [7]. Målet med metoden är att efter tillräckligt många iterationer konvergera till optimum [7]. Algoritmen beskrivs av följande triviala exemplet, där minimum söks för f(x):

$$f(x) = x^2 + 6x + 6 \;\; \Rightarrow \;\; f'(x) = 2x + 6$$

Steg 1: välj startvärde $x_0 = -2$

Steg 2: välj inlärningstakt $\eta = 0.1$

Steg 3: utför iterationer enligt $x_n = x_{n-1} - \big(\eta \cdot f'(x_{n-1})\big), n \geq 1, n \in \mathbb{Z}$

$$x_1 = x_0 - \big(\eta \cdot f'(x_0)\big) = -2{,}2$$

$$x_2 = x_1 - \big(\eta \cdot f'(x_1)\big) = -2{,}36$$

$$x_3 = x_2 - \big(\eta \cdot f'(x_2)\big) = -2{,}48$$

...

$$x_{14} = x_{13} - \big(\eta \cdot f'(x_{13})\big) = -2{,}956 \ldots$$

$$x_{15} = x_{14} - \big(\eta \cdot f'(x_{14})\big) = -2{,}964 \ldots$$

x-värdet ser ut att konvergera mot -3, vilket är $f(x)$ globala minimum. Inlärningstakten η bestämmer storleken på varje steg i iterationen och påverkar antalet iterationer som behövs för att nå funktionsminimum [7].

Eftersom GD är svår att implementera när det gäller stora mängder data och gradienten kan vara svår att beräkna exakt används en vidareutvecklad version av metoden ofta i praktiken istället, nämligen stokastisk GD (eng. *Stochastic Gradient Descent*) [7]. Denna metod bygger på att uppskatta gradienten istället för att beräkna den och uppnår ofta bättre resultat än den vanliga GD när stora mängder data behandlas.

2.5 Optimeringsmetoder av högre grad

Metoder av andra grad används främst då funktionen man vill minimera är strängt icke-linjär och de flesta metoderna bygger på användningen av Hessematriser (eng. *Hessian Matrix*) [7], [13]. En av de välkändaste optimeringsmetoderna av andra grad är Newtons metod [13]. Metoden bygger på att approximera funktionen i fråga som en andragradsfunktion med hjälp av gradient (första ordningens derivata) och Hessematris (andra ordningens derivata) och sedan iterativt minimera funktionen [7]. Newtons (flerdimensionella) iterationsformel är angiven enligt:

$$x_{n+1} = x_n - \nabla^2 f(x_n)^{-1} \nabla f(x_n), \qquad n \geq 0 \text{ [7]}$$

där f är funktionen man vill minimera och $\nabla^2 f$ är dess Hessematris. Trots att metoden är väldigt effektiv är den samtidigt dyr vad beträffar lagring och beräkningstid eftersom metoden kräver beräkning av funktionens Hessematris samt dess invers för varje iteration [7], [13]. På grund av detta utvecklades quasi-Newton-metoden som istället för att beräkna Hessematrisens invers för varje iteration approximerar den [7]. Denna metod har visat sig vara effektivare än Newtons metod i vissa fall och är över lag en av de mest effektiva metoderna när det kommer till icke-linjära optimeringsproblem [7].

2.6 Deriveringsfri optimering

I praktiken går det ändå inte alltid att beräkna derivatan för en funktion som används i någon viss maskininlärningsalgoritm. I detta fall är man tvungen att använda sig av optimeringsmetoder som inte är baserade på funktionens derivata. Detsamma gäller då derivatan är väldigt komplicerad att beräkna. En deriveringsfri metod man kan använda för att minimera funktionen med är *Coordinate Descent* (CD) [7]. Metoden baserar sig på att minimera funktionen turvis i varje axelriktning och liknar GD i och med sättet den konvergerar på [7]. Metoden beskrivs mera detaljerat i [14].

3. Praktiska implementationer

Detta kapitel behandlar implementering av maskininlärningsmetoderna presenterade i kapitel 2. I avsnitt 3.1 presenteras implementation av dessa metoder i *Scikit-learn* som är ett maskininlärningsbibliotek för Python. Programpaketet beskrivs på ett allmänt plan, med exempel på implementation i kod.

3.1 Scikit-learn i Python

Scikit-learn är ett gratis bibliotek för Python med kodimplementation för ett stort antal populära maskininlärningsmetoder för vägledd- samt icke-vägledd inlärning [15]. Biblioteket är speciellt användbart för att ge icke-specialister inom området ett relativt enkelt sätt att implementera maskininlärning på i praktiken. Trots detta används det inte bara i akademiskt syfte utan i dagens läge alltmer även inom industrin [15]. *Scikit-learn* är publicerad under BSD-licens och bygger långt endast på Pythons *numpy* och

scipy till skillnad från många andra liknande bibliotek som kräver extern programvara för att fungera [15].

Bland de maskininlärningsmetoder som *Scikit-learn* erbjuder implementering för finns även SVM, NB och K-means. I avsnitt 3.1.1, 3.1.2 samt 3.1.3 presenteras exempelkod för implementering av metoderna med hjälp av *Scikit-learn*.

3.1.1 Stödvektormaskin med Scikit-learn

Följande implementeringsexempel av SVM med *Scikit-learn* i Python är taget ur [16].

Steg 1: importera nödvändiga bibliotek.

```
import pandas as pd
import numpy as np
```

Steg 2: läs valfri datamängd, i detta exempel ”bill_authentication.csv” som innehåller bankdata.

```
bankdata = pd.read_csv("D:/Datasets/bill_authentication.csv"
```

Steg 3: dela in data i attribut och klasser (X attribut, y klass).

```
X = bankdata.drop('Class', axis=1)
y = bankdata['Class']
```

Steg 4: dela in data i tränings- och testmängder.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.20)
```

Steg 5: träna modellen.

```
from sklearn.svm import SVC
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
```

Stödvektormaskinmodellen är nu tränad och klar att användas. För resultat av modellen, se [16].

3.1.2 Naive Bayes med Scikit-learn

Följande implementeringsexempel av NB med *Scikit-learn* i Python är taget ur [17].

Steg 1: importera nödvändiga bibliotek.

```
import pandas as pd
```

Steg 2: läs valfri datamängd, i detta exempel ”SMSSpamCollection” som innehåller spammeddelanden.

```
df = pd.read_table('SMSSpamCollection',
                   sep='\t',
                   header=None,
                   names=['label', 'message'])
```

Steg 3: omformatering av data, i detta fall konvertera ’label’ från sträng till binär form, konvertera tecken till små bokstäver samt ta bort punkter, kommatecken med mera. Se [17] för detaljer.

Steg 4: dela in data i tränings- och testmängder.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(counts, df['label'], test_size=0.1, random_state=69)
```

Steg 5: träna modellen.

```
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB().fit(X_train, y_train)
```

NB-modellen är nu tränad och klar att användas. För närmare detaljer och modellens resultat, se [17].

3.1.3 K-means med Scikit-learn

Följande implementeringsexempel av K-means med *Scikit-learn* i Python är taget ur [18].

Steg 1: importera nödvändiga bibliotek.

```
import numpy as np
from sklearn.cluster import KMeans
```

Steg 2: dela upp data i datapunkter.

X = np.array([[5,3], [10,15], [15,12], [24,10], [30,45], [85,70], [71,80], [60,78], [55,52], [80,91]])

Steg 3: skapa kluster.

```
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
```

K-means-modellen är nu klar. En visualisering av modellen samt resultat hittas i [18].

4. Förbättrade maskininlärningsalgoritmer

I praktiken kan alla maskininlärningsmetoder presenterade i kapitel 2 tillämpas för datautvinning, svårprogrammerade applikationer samt självanpassande mjukvara, så optimering av dessa kommer i denna avhandling att presenteras i form av förbättrade versioner av tidigare nämnda maskininlärningsmetoder. Förbättringsexemplen i avsnitt 4.1, 4.2 och 4.3 bygger på tidigare experiment och behandlas mera detaljerat i de artiklar där de ursprungligen publicerats.

4.1 Förbättrad stödvektormaskin

En förbättrad version av SVM är ”Successive Over-Relaxation” (SOR) som bygger på matrisindelning och beskrivs noggrannare i [19]. I ett experiment som presenteras i

samma artikel jämförs SOR med vanliga SVM samt med SVM där en optimeringsmetod känd som ”Sequential Minimal Optimization” (SMO) utnyttjas. Resultatet av experimentet presenteras i tabell 4.1 nedan.

Training Set Size	CPU Sec	Non-Bound SVs	Bound SVs	Iter	Test Set Accuracy
	SOR SMO SVM^l	SOR SMO SVM^l	SOR SMO SVM^l	SOR SMO SVM^l	SOR SMO SVM^l
1605	0.3	49	635	924	84.06
	0.4	42	633	3230	
	5.4	44	634	292	84.25
2265	1.2	51	930	1142	84.24
	0.9	47	930	4635	
	10.8	49	929	383	84.43
3185	1.4	52	1221	962	84.23
	1.8	57	1210	6950	
	21.0	62	1208	732	84.40
4781	1.6	65	1794	1819	84.28
	3.6	63	1791	9847	
	43.2	67	1788	865	84.47
6414	4.1	64	2379	1799	84.30
	5.5	61	2370	10669	
	87.6	63	2370	956	84.43
11221	18.8	81	4085	2642	84.37
	17.0	79	4079	17128	
	306.6	85	4078	1625	84.68
16101	24.8	86	5852	2995	84.62
	35.3	67	5854	22770	
	667.2	77	5849	2234	84.83
22697	31.3	103	8209	6061	85.06
	85.7	88	8209	35822	
	1425.6	109	8199	3960	85.17
32561	83.9	201	11550	4702	84.96
	163.6	149	11558	44774	
	2184.0	166	11554	5075	85.05

Tabell 4.1 Jämförelse av SOR, SMO & SVM [19]

Ur Tabell 4.1 framgår det att SOR-metoden drastiskt reducerar tiden som används för beräkningarna jämfört med vanliga SVM samt att noggrannheten nästan fullständigt bevaras, dock med en liten minskning. Det framgår också att SOR är märkbart snabbare än SMO, speciellt vid stora datamängder. Noterbart är också att antalet stödvektorer som utnyttjas av de olika metoderna är så gott som samma och att antalet iterationer för SOR jämfört med SVM också är nästan lika.

Experimentet presenterat i [19] lyfter fram SOR-metodens inverkan på beräkningstiden och ur resultatet framgår vikten av förbättring av SVM, speciellt vid behandling av större datamängder.

4.2 Förbättrad Naive Bayes för datautvinning

Inom datautvinning utnyttjas Naive Bayes vanligen för klassifikationsproblem [20]. I ett experiment utfört av [21] jämförs tre implementeringsmetoder för Naive Bayes med bland annat vanliga Naive Bayes och SVM. De tre metoderna (M1, M2 & M3) bygger på klassannolikheter och betingad sannolikhet (för noggrannare beskrivning samt matematisk modellering av metoderna, se [21]).

Experimentet bygger på datautvinning ur 14 datamängder med ett antal olika metoder, däribland Naive Bayes (NB), stödvektormaskin (SVM) samt NB-metoder M1, M2 och M3 [21]. Attributen för datamängdernas data har också gjorts diskreta för att direkt kunna tillämpa NB [21]. Resultatet framgår ur tabell 4.2 nedan där noggrannheten bland de olika metoderna framgår.

Data	NB	TAN	SVM	C4.5	1-NN	M1	M2	M3
Breas	96.37	95.92	95.32	91.21	96.71	97.40	97.62	**97.99**
Cong	91.43	91.74	96.76	95.71	95.78	96.83	96.89	**96.97**
Credit	84.92	82.90	85.51	87.49	83.21	84.89	88.86	**89.98**
Diabet	75.93	76.51	76.72	76.12	74.94	76.92	77.35	**78.65**
Germa	75.32	74.02	76.15	72.14	72.23	75.65	75.91	**78.36**
Haber	75.22	73.98	73.54	71.79	70.14	75.93	77.98	**78.65**
Heart	81.67	84.72	80.54	81.73	80.03	86.86	87.67	**88.87**
Hepat	83.76	83.44	83.43	82.99	83.58	83.94	84.42	**84.43**
Ionos	82.95	84.57	85.98	86.35	**88.97**	84.61	85.93	88.68
Liver	61.84	61.87	60.05	60.17	62.85	63.01	65.64	**65.72**
Sonar	75.16	75.48	76.99	76.69	74.15	76.41	76.65	**79.97**
Spam	90.31	89.89	90.47	91.96	92.41	90.24	92.56	**93.57**
Svm1	92.72	92.16	93.31	93.72	95.89	93.75	94.83	**96.89**
Svm3	81.34	83.01	80.11	81.23	80.58	83.15	83.72	**86.65**
Ave	82.06	82.15	82.49	82.09	82.24	83.56	84.71	**86.09**

Tabell 4.2 Medelnoggrannheten för metoderna över 50 testkörningar [21]

Som det framgår ur tabell 4.2 har NB där metod M3 använts högsta medelnoggrannhet av alla metoder i experimentet och presterar bättre än vanliga NB och SVM för alla datamängder. Förbättrade NB har också över lag högre noggrannhet än de andra metoderna. Resultatet visar att förbättring av NB påverkar noggrannheten av datautvinningen positivt trots att den enkla strukturen hos NB bibehålls.

4.3 Förbättrad K-means

Den vanliga K-means-algoritmen bygger på att iterativt tilldela datapunkterna en mittpunkt tills mittpunkterna har nått sina slutliga positioner [10]. Det har dock visat sig att endast en liten del av punkterna uppdateras efter ett visst antal iterationer, vilket leder till onödiga uppdateringar som inverkar på prestanda [22]. Detta tas i beaktande i ett experiment utfört av [22] som bygger på att reducera antalet uppdateringar som görs för varje iteration och på så sätt förbättra algoritmen genom att reducera körtiden.

Den experimentella förbättringsmetoden baserar sig på att dela in datapunkterna i två kategorier, beroende på om en punkt ligger nära yttre gränsen av ett kluster eller ej

[22]. Ifall den gör det placeras den i en lista med andra gränspunkter [22]. Istället för att sedan uppdatera varje punkt skilt för sig kan hela listan med gränspunkter uppdateras samtidigt [22]. I experimentet använde man slumpmässigt genererade datamängder, med allt från 100 000 till 5 000 000 datapunkter per mängd, som man delade upp i 4, 8 och 12 kluster. Punkterna bestod av 2D punkter med koordinater mellan 0 och 1. Resultatet av experimentet presenteras nedan (för närmare detaljer samt beskrivning av algoritmen, se [22]).

Data set size	Running Time Standard K-Means (ms)	Running Time Optimized K-Means (ms)	Improved by(%)
100,000	6300	1967	68.77
250,000	14382	4528	68.51
500,000	40321	11402	71.72
750,000	55088	15943	71.05
1,000,000	73957	21436	71.01
2,000,000	140339	42962	69.38
5,000,000	420516	116630	72.26

Tabell 4.3 Förbättrade K-means och vanliga K-means körtider med 4 kluster [22]

Data set size	Running Time Standard K-Means (ms)	Running Time Optimized K-Means (ms)	Improved by(%)
100,000	75664	28418	62.44
250,000	148472	57485	61.28
500,000	629777	224277	64.38
750,000	1004100	359230	64.22
1,000,000	1096291	396923	63.79
2,000,000	2798319	1006918	64.01
5,000,000	9685205	3355996	65.34

Tabell 4.4 Förbättrade K-means och vanliga K-means körtider med 8 kluster [22]

Data set size	Running Time Standard K-Means (ms)	Running Time Optimized K-Means (ms)	Improved by(%)
100,000	53879	27388	49.16
250,000	186923	90140	51.77
500,000	323584	158888	50.89
750,000	681331	317328	53.42
1,000,000	809675	377522	53.37
2,000,000	1650657	776873	52.93
5,000,000	4835146	2324173	51.93

Tabell 4.5 Förbättrade K-means och vanliga K-means körtider med 12 kluster [22]

Som det framkommer ur tabell 4.3, 4.4 och 4.5 har förbättringen en märkbar inverkan på körtiderna hos algoritmerna. Man kan också observera att inverkan av förbättringen minskar då antalet kluster ökar vilket beror på att gränsområdena blir bredare när klustren blir flera [22].

5. Diskussion

Syftet med avhandlingen var att lyfta fram vikten av att använda så effektiva versioner av maskininlärningsmetoder som möjligt, vilket har gjorts genom att presentera och evaluera konkreta exempel där förbättrade metoder använts. I exemplen jämfördes bland annat körtiden hos förbättrade och oförbättrade algoritmer och resultaten tyder på en klart bättre effektivitet hos de förbättrade metoderna. Som framkom ur resultatet i kapitel 4.3 minskade körtiden för K-means-algoritmen för fyra kluster som mest med 72,26 %, för åtta kluster som mest med 65,34 % och för tolv kluster som mest med 53,42 %. I kapitel 4.1 och 4.2 jämfördes noggrannheten bland de förbättrade

algoritmerna med de oförbättrade algoritmerna och även här syntes en positiv inverkan på noggrannheten bland de förbättrade metoderna.

Trots att resultaten verkar tyda på en klart bättre effektivitet bland de förbättrade metoderna i exemplen ovan kan man ändå inte dra slutsatsen att de skulle fungera för andra implementationsområden än de som presenterades i denna avhandling. Detta område är öppet för framtida forskning där enskilda förbättringsmetoders prestanda inom flera tillämpningsområden kan analyseras. Studiens resultat styrker ändå påståendet att förbättring av maskininlärningsalgoritmer inverkar direkt på resultatet, vilket var avhandlingens huvudsakliga syfte.

6. Sammanfattning

I denna avhandling presenterades ett antal förbättringsmetoder för några vanliga maskininlärningsalgoritmer och deras inverkan på effektiviteten bland algoritmerna analyserades. De metoder som förbättrades i avhandlingen var stödvektormaskin (SVM), Naive Bayes (NB) samt K-means-klustring och resultatet tydde på en förbättring i både tidseffektivitet och noggrannhet bland de olika algoritmerna.

Avhandlingen behandlade också maskininlärning på ett allmänt plan, genom att kort beskriva vägledd inlärning, icke-vägledd inlärning samt förstärkningsinlärning. Optimering som teori behandlades också på ett allmänt plan och optimering av första graden och högre grad samt deriveringsfri optimering lyftes fram.

Resultaten i kapitel 4 visade att förbättring av de olika metoderna direkt inverkar på effektiviteten. Förbättrad K-means var upp till 72,26 % snabbare i tid än oförbättrad K-means då fyra kluster användes. Noggrannheten hos SVM och NB påverkades också positivt då förbättring av algoritmerna förekom.

På grund av att endast ett fåtal handplockade exempel analyserades och samma förbättringsmetod inte tillämpades på flera tillämpningsområden kan man inte dra slutsatsen att förbättringarna skulle fungera lika väl i alla sammanhang. Ett framtida forskningsobjekt kan vara att jämföra förbättringsmetoder presenterade i denna avhandling inom flera tillämpningsområden än endast ett. Resultaten lyfter ändå fram vikten av effektivisering av algoritmer inom maskininlärning, vilket var den huvudsakliga meningen med avhandlingen.

Källförteckning

[1] S. Sra, S. Nowozin and S. J. Wright, Optimization for machine learning, Mit Press, 2012.

[2] B. Mahesh, ”Machine Learning Algorithms - A Review,” *International Journal of Science and Research,* vol. 9, nr 1, pp. 381-386, 2020.

[3] T. M. Mitchell, ”Does Machine Learning Really Work?,” *AI Magazine,* vol. 18, nr 3, pp. 11-20, 1997.

[4] F. O. Isinkaye, Y. O. Folajimi och B. A. Ojokoh, ”Recommendation systems: Principles, methods and evaluation,” *Egyptian Informatics Journal,* vol. 16, nr 3, pp. 261-273, 2015.

[5] A. L. Fradkov, ”Early History of Machine Learning,” *ScienceDirect IFAC PapersOnLine,* vol. 53, nr 2, pp. 1385-1390, 2020.

[6] T. M. Mitchell och M. I. Jordan, ”Machine learning: Trends, perspectives, and prospects,” *Science,* vol. 349, nr 6245, pp. 255-260, 2015.

[7] S. Sun, Z. Cao, H. Zhu och J. Zhao, ”A Survey of Optimization Methods From a Machine Learning Perspective,” *IEEE Transactions on Cybernetics,* vol. 50, nr 8, pp. 3668-3681, 2020.

[8] T. Hastie, R. Tibshirani och J. Friedman, The Elements of Statistical Learning, New York: Springer, 2008.

[9] G. I. Webb och C. Sammut, ”Naïve Bayes,” i *Encyclopedia of Machine Learning and Data Mining*, Boston, Springer, 2011.

[10] T. M. Kodinariya och P. R. Makwana, ”Review on determining number of Cluster in K-Means Clustering,” *International Journal of Advance Research in Computer Science and Management Studies,* vol. 1, nr 6, pp. 90-95, 2013.

[11] K. P. Sinaga och M. Yang, ”Unsupervised K-Means Clustering Algorithm,” *IEEE Access,* vol. 8, pp. 80716-80727, 2020.

[12] R. Talimi. [Online]. Available: http://talimi.se/ml/kmeans/. [Använd 17 februari 2022].

[13] H. H. Tan och K. H. Lim, ”Review of second-order optimization techniques in artificial neural networks backpropagation,” *IOP Conference Series: Materials Science and Engineering*, vol. 495, 2019.

[14] S. J. Wright, ”Coordinate descent algorithms,” *Mathematical Programming*, vol. 151, pp. 3-34, 2015.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot och E. Duchesnay, ”Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

[16] U. Malik, ”StackAbuse,” 9 maj 2019. [Online]. Available: https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/. [Använd 18 mars 2022].

[17] D. Shahrokhian, ”StackAbuse,” 10 juli 2018. [Online]. Available: https://stackabuse.com/the-naive-bayes-algorithm-in-python-with-scikit-learn/. [Använd 18 mars 2022].

[18] S. Robinson, ”StackAbuse,” 21 november 2021. [Online]. Available: https://stackabuse.com/k-means-clustering-with-scikit-learn/. [Använd 18 mars 2022].

[19] O. L. Mangasarian och D. R. Musicant, ”Successive Overrelaxation for Support Vector Machines,” *IEEE Transactions on Neural Networks*, vol. 10, nr 5, pp. 1032-1037, 1999.

[20] S. M. Gorade, A. Deo och P. Purohit, ”A Study of Some Data Mining Classification Techniques,” *International Research Journal of Engineering and Technology*, vol. 4, nr 4, pp. 3112-3115, 2017.

[21] S. Taheri och M. Mammadov, ”Learning the Naive Bayes Classifier With Optimization Models,” *International Journal of Applied Mathematics and Computer Science*, vol. 23, nr 4, pp. 787-795, 2013.

[22] C. M. Poteras, M. C. Mihaescu och M. Mocanu, ”An Optimized Version of the K-Means Clustering Algorithm,” *2014 Federated Conference on Computer Science and Information Systems*, pp. 695-699, 2014.

[23] T. M. Mitchell, Machine Learning, McGraw-Hill Science/Engineering/Math, 1997.