JÄMFÖRELSE AV MASKININLÄRNINGSMODELLER FÖR HANDSKRIVEN SIFFERKLASSIFICERING

Matthew H. Motallebipour

30 april 2024

Yrkeshögskolan EC Utbildning

Sammanfattning

Abstract: The present report elaborates on a comparison between three different machine learning models for classification of handwritten digits: Support Vector Machines (SVM), Random Forests (RF), and Extreme Gradient Boosting (XGBoost). The models are compared and ranked based on accuracy. In the second part of the report, an ensemble learning model is also presented, which is used to create a Python application in the Streamlit framework that classifies uploaded images with an accuracy of about 98%.

Sammanfattning

Abstract: Föreliggande rapport presenterar en jämförelse mellan tre olika maskininlärningsmodeller för klassificering av handskrivna siffror: Support Vector Machines, Random Forest, och Extreme Gradient Boosting. Modellerna jämförs och rangordnas med avseende på noggrannhet. I den andra delen av rapporten presenteras också en ensemble inlärning modell som används för att framställa en Python-applikation i ramverket Streamlit som via en server klassificerar uppladdade bilder med en noggrannhet på cirka 98%.

Acknowledgements

I would like to thank my supervisor, Mr. Antonio Prgomet, for his outstanding teaching and his guidance throughout the whole project. I would also like to thank my dear friend, Mr. Robert Shaw for his invaluable contributions; a peer through whose journalistic scrutiny I was able to realize the shortcomings in my work and improve upon them. Finally, thank you to all my fellow students that are always there for a good discussion throughout all courses.

Förkortningar

SVM	Support Vector Machines
RF	Random Forest
XGBoost	Extreme Gradient Boosting
ML	Machine Learning
DL	Deep Learning
CNN	Convolutional Neural Network
GUI	Graphical User Interface
API	Application Programming Interface
IDE	Integrated Development Environment
TPE	Tree-structured Parzen Estimator

Figurer

Tabeller

Innehåll

1 Inledning

För bara några år tillbaka i tiden kunde man inte föreställa sig att algoritmer inom det nytillkomna, heta området maskininlärning skulle på så kort tid revolutionera takniken och människors syn på framtiden. Speciellt efter introduktionen av djupinlärning (DL) och faltningsneuronnät (CNN) har maskininlärning (ML) blivit så populär att skolor och universitet runtom Sverige har börjat erbjuda kurser inom området för lärare för att kunna bemöta den ökande efterfrågan på ML-kunnandet (Skolverket, 2020). Därför är det ytterst väsentligt att vi som studenter och forskare inom området maskininlärning förstår de olika modellerna och deras styrkor och svagheter.

1.1 Syfte

Denna rapport ämnar behandla tre olika ML-modeller, nämligen Support Vector Machines (SVM), Random Forest (RF), och Extreme Gradient Boosting (XGBoost). Syftet är att besvara frågan om dessa modeller i allmännhet kan klassificera handskrivna siffror med en acceptabel noggrannhet. I synnerhet kommer vi att jämföra skillnaden mellan dessa tre och se vilken som fungerar bäst.

1.2 Frågeställning

Den specifika frågan är vilken av de tre modellerna SVM, RF och XGBoost är det bästa alternativet för klassificering av handskrivna siffror.

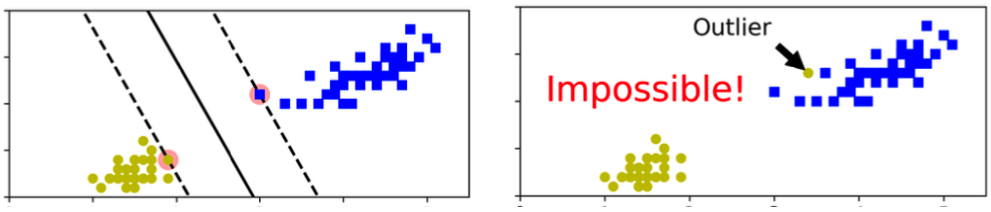
1.3 Textens disposition

Först kommer vi att titta på våra tidigarenämnda modeller och vad de är. Därefter presenteras det lite olika metoder för att förbehandla och senare klassificera bilderna som används som träningsdata. I resultatdelen kommer vi att presentera hur våra modeller klarat av utmaningen. I diskussionsdelen kommer vi att jämföra och rangordna modellerna och i den sista delen kommer vi presentera vår slutsats och förslag på framtida arbeten för att förbättra resultaten.

2 Teori

För klassificering av data i olika former såsom bild, text eller ljud finns det många olika alternativ inom maskininlärning. Dessa alternativ kallas modeller varav de mest kända är Logsitic Regression, Decision Trees, Random Forest, Gradient Boosting Machines, Support Vector Machines, k-Nearest Neighbors, Naive Bayes, Neural Networks, Linear Discriminant Analysis, Adaboost, Bootstrap Aggregating (Bagging), extra trees (Géron, 2019), Quadratic Discriminant Analysis (Bishop, 2006), Gaussian Process Classifiers (Gibbs and MacKay, 1997), Passive Aggressive Classifiers (Crammer and Singer, 2003) och många fler.

Vi valde att begränsa oss till en jämförelse mellan tre av dessa modeller: Support Vector Machines, Random Forest och Extreme Gradient Boosting. Dessa modeller är kända för att vara effektiva och används ofta i praktiken.



(a) & (b)

Figur 1: Support Vector Machines: [Géron] (a) Linjär separerbar data. De två streckade linerna visar marginalen medan den hela linjen är beslutsgränsen (b) Icke-linjär separerbar data.

2.1 Support Vector Machines (SVM)

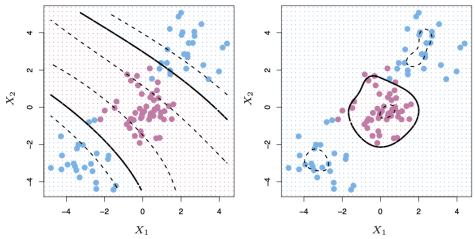
SVM är en linjär klassificerare som försöker hitta den bredaste marginalen som kan separera två klasser. Den används både för klassificering och regression men används mest för klassificering. SVM är en övervakad algoritm baserad på Vapnik-Chervonenkis teori, se Figur 1 från Géron (2019). Om klasserna inte är linjärt separerbara kan vi använda icke-linjär SVM, t ex polynomial kernel eller radial kernel; se Figur 2.

Random Forest är en ensemble inlärningsmodell som består av flera beslutsträd. Varje träd i skogen är en enkel klassificerare som bygger på slumpmässigt valda egenskaper som, rent praktiskt, är kolumnerna i befinlig data-tabell. RF är också en mycket effektiv algoritm som tvärtemot SVM är robust mot outliers. Det är därför som vi har inkluderat den i implementeringen av vår sifferklassificerare. RF är med andra ord en aggregerad modell vars utfall är baserad på en majoritsröstning av alla träd.

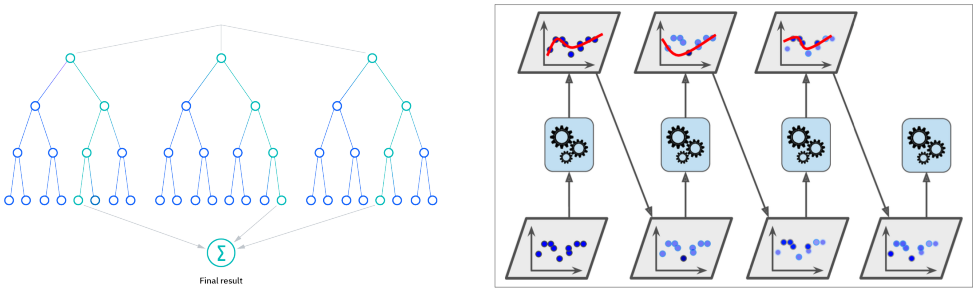
Leo Breiman, who is also known for introducing Bootstrap Aggregating (Bagging) (Géron, 2019), is the creator of this well-known model, along with Adele Cutler (Izenman, 2008).

2.2 Extreme Gradient Boosting (XGBoost)

XGBoost är inte från samma katalog som de ovannämnda modellerna, utan den har en egen katalog i Python. XGBoost är en optimerad version av Gradient Boosting Machines som är en ensemble inlärningsmodell som liksom RF också består av flera beslutsträd. Skillnaden är att denna innehåller träd som byggs på varandra i en sekventiell ordning och inte parallellt som i RF. Varje träd i XGBoost bygger på felet, de så kallade residualer som är skillanderna mellan de predikterade och de reela värdena, från det föregående trädet. Denna modell använder också early stopping för att undvika överanpassning till data och därmed minska bias (Géron, 2019). Early stopping innebhär helt enkelt att det



Figur 2: Vänster: SVM med polynomkärna av grad 3 applicerad på icke-linjär separerbar data. Höger: SVM radiell kärna applicerad på samma data. Båda visar framgångsrik klassificering. Mer om detta kan läsas i James et al. (2013), pages 353–358.



(a) (b)

Figur 3: Random Forest (IBM, 2021) (a) och Gradient Boosting (Géron, 2019) (b)

enskilda trädet inte byggs på när residualen inte längre minskar avsevärt. XGBoost har visat sig vara snabbare än sin föregångare, i scikit-learn implementaterade versionen av gradient boosting, och används därför ofta inom ML-tävlingar (Guido, 2016).

Vad vi vet är att beslutsträd i allmänhet har en inneboende varaians, eftersom de har en tendens att anpassa sig till träningsdata mycket väl, samt att den minsta förändringen i hyperparameterarna kan leda till att skapa helt olika modeller (Géron, 2019). RF och XGBoost är mycket effektiva modeller som båda motverkar den egenskapen (Géron, 2019). Därför är det intressant att jämföra dessa två med SVM för att se vilken som är bäst för klassificering av handskrivna siffror.

2.3 PCA

Principalkomponentanalys (PCA) är enkelt sagt den matematiska methoden som används för att eliminera överflödig information i form av dimensionen på data utan att minska variationen av informationen avsevärt. Det innebär rent praktiskt att man transformerar det aktuella koordinatsystemet till ett nytt sådant där de nya ortonormala komponenterna, eller variablerna, rangordnas efter hur stor variation i datan de representerar.

PCA används ofta för att minska dimensionen på data och för att visualisera data i en lägre dimensionell rum (Jolliffe, 2002). Enligt Jolliffe (2002) var PCA introducerad av två oberoende personer, Pearson i 1901 och Hotelling i 1933, baserad på Beltramis forskning i 1873 and Jordans i 1874 kring singulärvärdesuppdelning, som i korta ordalag består av en rotation följd av en omskalning och en till rotation (Wikipedia, 2021).

En viktig tumregel är att man gör PCA sist av alla förbehandlingar, eftersom PCA förändrar data och förhållandena mellan features och gör det därför svårt att tolka data och därmed även omöjliggör andra förbehandlingar som normalisering och standardisering som råkar användas efter just PCA (Datacamp, 2021).

2.4 Stratifierad k-faldig korsvalidering

Stratifierad k-faldig sampling är en metod som används för att dela upp träningsdata i k separata grupper, där varje grupp kallas för en *fold*. Den genererande algoritmen ser till att varje *fold* representerar en lika stor andel av varje klass som finns inom träningsmängden. På det sättet undviker man att vissa klasser inte används i träningen alls. Problemet med att inte ha med alla klasser vid träningstillfället är att den genererade modellen inte kan generalisera såsom det är tänkt. Ett mycket bra exempel på varför stratifierad k-faldig korsvalidering är att föredra före en vanlig sådan kan man läsa på sidan 257 i Müller and Guido (2018).

2.5 Optuna

Optuna är en ramverk för hyperparameteroptimerings som använder sig av en teknik som kallas för Tree-structured Parzen Estimator (TPE) för att optimera hyperparametrar. TPE i sin tur bygger på Bayesiansk optimering. Optuna är byggd av Takuya Akiba och hans kollegor på Preferred Networks. (Akiba et al., 2019).

2.6 Streamlit

Streamlit är en ramverk avsedd att hjälpa vid utveckling av webb-applikationer i Python. Applikationen kan både ta emot data från användaren och presentera färdiga, resultat och interaktiva grafer på webben (Richards, 2023). Streamlit installeras enkelt genom den sedvanliga instruktionen *pip* (Preferred Installer Program) och kan köras på den lokala home-servern så gott som på en vanlig webbserver.

Streamlit grundades av Adrien Treuille, Amanda Kelly och Thiago Teixeira (TechCrunch, 2022).

3 Metod

För att jämföra de tre modellerna SVM, RF och XGBoost använde vi oss av Python-biblioteken Scikit-learn och XGBoost. All kod implementerades i Jupyter Notebook, bäddad in i VSCode, och på en Mac dator med 1.4 GHz Quad-Core Intel Core i5, 8 GB 2133 MHz LPDDR3, och Intel Iris Plus Graphics 645 1536 MB. Nu ska vi titta närmare på vilka steg vi tog för att komma till mål.

3.1 Undersökning

Det första man brukar göra i alla projekt inom ML och data science är att titta på data och se hur den ser ut, vilka egenskaper den har samt vad som kan vara av värde att notera och använda.

Vår givna data bestod av MNIST-datasetet som innehåller bilder på 70 000 handskrivna siffror i storleken 28x28 pixlar. Pixlarna är i gråskala med värden från 0 till 255, som för bättre prestanda standardiserades till att anta värden mellan 0 ch 1[1]. Vi delade upp bilderna i 60 000 träningsbilder och 10 000 testbilder. Dessa laddades in i programmet genom

3.2 Datarensning

Därefter försöker man rensa data från information som inte är användbar, eller inte är komplett. I det andra fallet, och om man redan har tillräckligt många datapunkter, eliminerar man de datapunkter som inte är kompletta, dvs att de rader där en eller flera kolumner saknar värden. Alternativet är att man försöker med olika metoder ”gissa” eller resonera sig fram till vad de saknade värdena kan vara och fyller i det som saknas.

[1]Var uppmärksam här på att divisionen med 255 inte är skalning utan standardisering av data. Att skala innebär att subtrahera medelvärdet och dividera resultatet med standardavvikelsen, vilket omvandlar datavärdena till att få medelvärdet 0 och standardavvikelsen 1. Det är viktigt för vissa algoritmer som SVM, som är känsliga mot alltför stora värdeförändringar i data, att man normaliserar värdena. I föreliggande problem är detta dock inte nödvändigt, eftersom värdena redan ligger mellan 0 och 1.

```
from keras.datasets import mnist
    (X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Tabell 1: Kod för att ladda in data från MNIST-datasetet

I vårt MNIST dataset fanns det av ganska naturliga skäl inga saknade datapunkter, dvs det fanns inga bilder där de hade värden utanför intervallet 0 och 255. Däremot fanns det en misstanke om att en eller flera rader och kolumner, som ligger närmast kanterna på alla bilder, kunde med fördel separeras för att minska antalet punkter som behövde behandlas under träningen. Detta visade sig inte stämma, eftersom eliminering av tre pixlar från alla fyra kanter inte ledde till en avsevärd förbättring av noggrannheten. Detta kommer att visas och diskuteras på sin plats.

3.3 Förbehandling

Till sist organiserar man data så att man kan förbereda den för att utvinna största möjliga information i de efterföljande stegen. När man väl har komplett och rensad data kan man fortsätta med att applicera olika metoder för att minska tid- och minneskostnaden vid träningstillfället, som brukar vara den mest resurskrävande delen vad gäller just tid och minne.

Eftersom vår data redan är i gråskala behövs ingen analys av färger. Däremot kan det vara värt att titta på hur mycket av ytan på varje bild som tas upp av siffrorna. För detta superpositionerade vi alla bilderna, dvs att motsvarande pixelvärdena från alla bilder lades ihop och delades med antalet bilder, alltså att medelvärdet av pixlarna beräknades. Vi tittade på den del av bilderna där det inte förekom några icke-vita pixlar. Det visade sig att siffrorna tar upp en relativt liten del av bilden, vilket kan vara en anledning till att modellerna inte presterade bättre än de gjorde.

Det finns flera olika metoder utav vilka vi använde PCA för den smidighet men samtidigt kapacitet den presenterar.

3.4 Experiment

Vi gjorde våra experiment med betydligt färre data, dvs 18 000 träningsbilder, som delades upp i tre lika stora delar. Detta för att spara tid vid många iterationer och test av olika modeller och metoder. På detta sättet tog varje modell högst 10 minuter per iteration.

3.5 Val av modell(er)

Nästa steg blir att använda en eller flera maskininlärningsmodeller för att memorera den underliggande strukturen i data och bygga en färdig motor som kan generalisera och, förhoppningsvis, med god sannolikhet klassificera eller förutse värden för bilder som matas in.

Här använde vi oss av de tre modeller som presenterades redan i teoridelen. Dessa ansåg vi vara de bästa och samtidigt snabbaste man kunde använda.

4 Resultat

Resultatet av körning av de tre utvalda modellerna är sammanfattat i tabellen ovan, Tabell 2. Som vi påpekade tidigare användes endast 18 000 bilder vid varje körning för att minska tiden för varje iteration. Tabellen visar att roterade bilder tar upp mer tid än de andra bildformaterna att klassificera, XGBoost är den snabbaste modellen, speciellt vid klassificering av testdata och SVM är den effektivaste när det gäller noggrannhet. Rent allmänt ser vi ingen större skillnad i noggrannhet vid klassificering av data som är trunkerad, roterad eller både trunkerad och roterad. En aneldning till att man kanske sparar lite tid på att trunkera bilderna men att noggrannheten ändå inte påverkas avsevärt kan vara att det ändå försvinner en del pixlar i bilderna som kan vara av värde för vissa bilder. Detta kräver tester med fler bilder och fler iterationer för att kunna avgöra.

I Tabell 3 kan vi se en typisk klassificeringsrapport för en normal bild. Därutav kan vi läsa precision, recall, f1-score och support. Precision är andelen korrekt klassificerade bilder av en viss siffra utav alla observationer som klassificerades som just den siffran, vare sig rätt eller fel. Om siffran s är den siffra vi söker efter ser formeln ut som följer

asr = antalet bilder av s som vi klassificerat rätt
asf = antalet bilder som vi klassificerat felaktigt som s men som inte innehåller s

$$precision = \frac{asr}{asr + asf}$$

Recall är andelen korrekt klassificerade bilder på en viss siffra delat med summan av antalet bilder som modellen har rätt om att innehålla den siffran plus antalet bilder som modellen säger helt rätt att de inte innehåller just den siffran. Om vi lägger till

aisr = antalet bilder som inte innehåller s och som vi klassificerat rätt, då kan formeln för recall uttryckas som

$$recall = \frac{asr}{asr + aisr}$$

F1-score är den harmoniska medelvärdet av precision och recall, dvs

$$f1 = 2 \times \frac{precision \times recall}{precision + recall}$$

Support är det totala antalet observationer i varje klass.

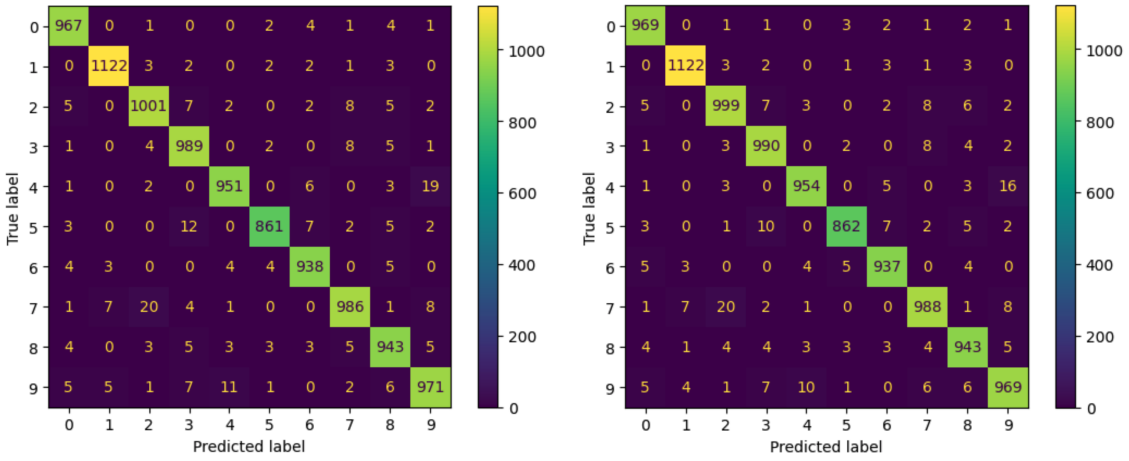
5 Diskussion

Generellt fungerade modellerna nöjaktigt men i praktiken finns det alltid saker som bryter mot teorin och under påverkan från olika faktorer, från implementering till val av parametrar, plattform och dylikt.

5.1 Val av modell

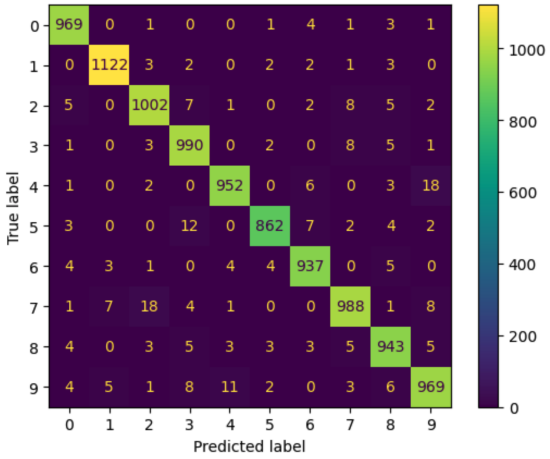
Valet av Random Forest som en av tre modeller som användes för detta projekt och enligt Jeroen Boeye[2] var att ”while simple in design, random forests often manage to be highly accurate and avoid overfitting even with the default Scikit-learn settings.”

[2]Instruktör för Datacamp-kursen Dimensionality Reduction in Python, kapitel 3, 00:05 - 00:34

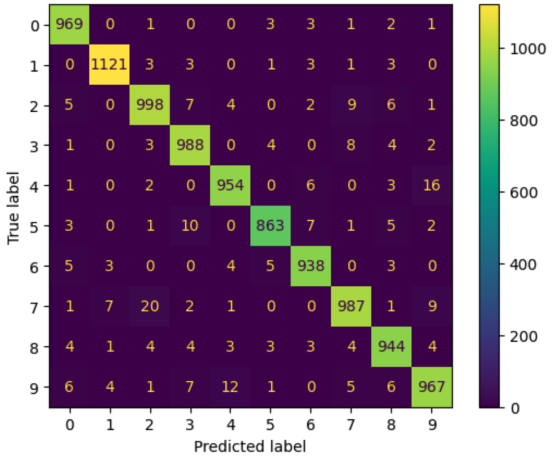


(a) Endast normala bilder

(b) Inklusive trunkerade bilder



(c) Inklusive roterade bilder



(d) Inklusive all formaterade bilder

Figur 4: Confusion matrix för all de olika formateringarna. Det som utmärker sig i alla fyra matriser är i första hand likheten mellan siffrorna 2 och 7, och i andra hand 3 och 5 samt 4 och 9. Det som förefaller oväntat är att i de roterade bilderna blir sannolikheten att blanda ihop 2 och 7 i (c) blir mindre än i den första (a) samt att de olika formateringarna påverkar sannolikheten för att blanda ihop 7 och 9.

5.2 Overfitting

När vi tittar på tabell 2 kan vi konstatera att speciellt RF och XGBoost visar bättre noggrannhet vid test jämfört med träning. Även för SVM är minskningen i noggrannhet i storleken ett fåtal hundradelar av en procent. Detta tyder på att våra modeller inte är övertränade på träningsdata och har låg bias.

5.3 Beräkningskomplexitet

Beräkningskomplexiteten hos ML-algoritmer kan variera baserad på olika faktorer, däribland implementeringen, storleken och komplexiteten hos data samt valet av hyperparameterar.

Komplexiteten för träning av en SVC är enligt Géron (2019) någonstans mellan $O(r^2 \times c)$ och $O(r^3 \times c)$, där m är antalet rader och n är antalet kolumner i data. Av den anledningen är den modellen bättre lämpad för mindre data men i vårt fall har det ändå fungerat nöjaktigt men det kostade oss en nedskärning av antalet träningsexempel, speciellt när vi skulle jämföra de fyra modellerna gång på gång.

Gradient Boosting har en beräkningskomplexitet på $O(r \times c \times \log_2(c))$ som kan redu-

ceras till $O(b \times c)$ med b som antalet intervall, om histogram-baserad implementering av denna model används, vilket gör modellen uppemot 100 gånger snabbare (Géron, 2019).

För ett beslutsträd är beräkningskomplexiteten lika med $O(r \times c \times \log c)$ (Géron, 2019) och eftersom en RF modell består av många parallella beslutsträd kan komplexiteten för en sådan modell uppgå till $O(r \times r \times c \times \log c) = O(r^2 \times c \times \log c)$

I enlighet med denna tabell är den snabbaste modellen respektive XGBoost, RF och SVC. Resultattabellen, Tabell 2, säger oss dock att ordningen för träningstid är tvärtom SVC, RF och XGBoost, medan för testtid är ordningen samma som i teorin, dvs XGBoost, RF och SVC. Anledningen till den avsevärda skillnaden är inte klar men vad vi redan vet är att val av parametrar kan påverka mycket (Géron, 2019).

6 Slutsats

I denna rapport gick vi igenom de steg man behöver ta för att träna en modell och sedan använda det till att klassificera bilder från MNIST databasen över handskrivna siffror. I detta arbete kunde testbilder klassificeras med en noggrannhet över 97 procent. Det finns bättre algoritmer som är ännu mer träffsäkra varav Convolutional Neural Networks (CNN) och deep learning är de mest kända. Men för de klassiska modellerna som vi använde här är resultatet ändå mycket bra.

6.1 Framtida arbeten

En sak som skulle kunna förbättra resultatet genom att minska träningstiden är just motsvarande vad vi redan har gjort i vår jämförelse mellan olika förbehandlingar, nämligen trunkering av bilderna. Ytterligare en sak skulle vi kunna göra och det är att titta på statistiken för varje pixel och se hur mycket de varierar. En pixel vars värde inte varierar mycket mellan olika siffror spelar en mindre roll för att urskilja de olika siffrorna och därmed kan det elimineras från mängden pixlar som ingår i träningen. Detta minskar träningstiden och kan därför förbättra träningen då man kan använda sig av fler träningsbilder.

I början var koden olika för olika modeller. Men det tog ett bra tag innan koderna blev enhetliga, vilket med tanke på tidsbrist inte kunde organiseras på rätt sätt. Det goda resultatet har inte påverkats men det kunde organiseras bättre i form av funktioner som kunde förenkla de omskrivningar som har gjorts för varje modeller. Detta är en del av det pågående och framtida arbetet.

Vi skulle kunna använda SGDClassifier för en bättre implementering av SVC, som motsvarar linjär SVC med tidskomplexiteten $O(m \times n)$. Detta är något att tänka på vid en nästa version.

6.1.1 Explained Variance Ratio

För varje egenvektor som spänner det transformerade koordinatsystemet för den originella rymden som är spänd av datasetets vektorer, finns ett motsvarande egenvärde. Man definierar Exaplained Variance Ratio (EVR) (Izenman, 2008; Kuhn and Johnson, 2019; OpenClassrooms, 2021) eller Variance Explained Ratio (VER) (Raschka and Mirjalili, 2019) för varje egenvektor som dess motsvarande egenvärde delat med summan av samtliga egenvärdena för alla egenvektorer som spänner upp det transformerade systemet.

Det vill säga att i matematiska termer, om λ_i representerar det egenvärde som motsvarar egenvektor nummer i i principal komponenten och $\lambda_1, \lambda_2, ..., \lambda_n$ är egenvektorerna sorterade i fallande ordning, ges EVR för i-te principalkomponenten av

$$EVR_i = \frac{\lambda_i}{\Sigma_{j=1}^{n} \lambda_j}$$

Enklare sagt, EVR visar hur mycket information eller varians reflekteras i den lägre dimensionerade rymden som spänns av principalkomponenterna. Detta ger en förstahands information om varje individuell komponent. Koden som används för att utnyttja EVR är som framgår av tabell 5.

Det är klart att vi använder mer eller mindre samma princip när vi utnyttjar cross validation eller grid search men dels tar detta betydligt mindre tid att genomföra/ beräkna och dels blir det lättare för oss att ha kontroll över vilka komponenter som används och i vilken utsträckning dessa påverkar variansen i modellen. Detta skulle vi kunna använda till exempel för att se vilka komponenter som är inblandade i igenkänningen av varje siffra. Huruvida detta skulle kunna generera bättre resultat kan undersökas i en framtida version.

Referenser

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning.* Springer.

Crammer, K. and Singer, Y. (2003). A family of additive online algorithms for category ranking. *Journal of Machine Learning Research.*

Datacamp (2021). Dimensionality reduction in python. *Datacamp.*

Gibbs, A. and MacKay, D. J. (1997). Variational gaussian process classifiers. *IEEE Transactions on Neural Networks.*

Guido, S. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media.

IBM (2021). Random forest. *IBM.*

Izenman, A. J. (2008). *Modern Multivariate Statistical Techniques: Regression, Classification, and Manifold Learning.* Springer.

Jolliffe, I. T. (2002). *Principal Component Analysis.* Springer.

Kuhn, M. and Johnson, K. (2019). *Feature Engineering and Selection: A Practical Approach for Predictive Models.* CRC Press.

Müller, A. C. and Guido, S. (2018). *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media.

OpenClassrooms (2021). Analyze the results. *OpenClassrooms*.

Raschka, S. and Mirjalili, V. (2019). *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing.

Richards, T. (2023). *Streamlit for Data Science*. Packt Publishing.

TechCrunch (2022). Snowflake acquires streamlit for $800m to help customers build data-based apps. *TechCrunch*.

Wikipedia (2021). Singular value decomposition. *Wikipedia*.

Tabell 2: Tider är räknade i sekunder och högsta och minsta värdet på varje rad är markerad med grön, respektive röd

		Bildformat →	Normal	Trunkerad	Roterad	Trunkerad och roterad
SVM						
	Träning	Tid	132.4	155.9	180	155.2
		Noggrannhet	97.75%	97.74%	97.77%	97.77%
	Test	Tid	40.9	51.6	55	52.7
		Noggrannhet	97.74%	97.70%	97.71%	97.72
RF						
	Träning	Tid	172.6	170.6	187.8	170.7
		Noggrannhet	93.33%	93.30%	93.06%	93.18%
	Test	Tid	44.8	43.8	48.9	43.6
		Noggrannhet	93.59%	93.74%	93.60%	93.51%
XGBoost						
	Träning	Tid	717.2	263.4	347.5	273.3
		Noggrannhet	96.44%	96.38%	96.46%	96.38%
	Test	Tid	56.2	31.4	35.1	33.2
		Noggrannhet	94.70%	94.95%	94.66%	94.94%
Ensemble						
	Test	Tid	292.3	189.3	326.3	189.1
		Noggrannhet	97.29%	97.33%	97.34%	97.29%

	precision	recall	f1-score	support
0	0.98	0.99	0.98	980
1	0.99	0.99	0.99	1135
2	0.97	0.97	0.97	1032
3	0.96	0.98	0.97	1010
4	0.98	0.97	0.97	982
5	0.98	0.97	0.97	892
6	0.98	0.98	0.98	958
7	0.97	0.96	0.97	1028
8	0.96	0.97	0.97	974
9	0.96	0.96	0.96	1009

Tabell 3: Klassificeringsrapport för MNIST databasen, där bilderna inte genomgått några av ytterligare behandlingarna rotation eller trunkering

Tabell 4: Sammanfattning av beräkningskomplexiteten för de tre modellerna

Modell	Tid	Minne
SVM	$O(r^2 \times c)$ - $O(r^3 \times c)$	$O(n \times p)$
RF	$O(r^2 \times c \times \log c)$	$O(n \times p \times m)$
XGBoost	$O(r \times c \times \log_2(c))$ - $O(r \times c)$	$O(n \times p)$

Tabell 5: Kod som redan finns i sklearn och kan användas för beräkning av Explained Veriance Ratio (EVR)

```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(std_df)
print(pca.explained_variance_ratio_)
```