Handskriven sifferigenkänning

Ett projekt i maskininlärning

Martin Blomqvist

EC Utbildning

Kunskapskontroll 2

202503

Abstract

This Machine Learning project used the MNIST dataset to train, test and evaluate three different models. The Random Forest model was chosen and utilized in the creation of a Streamlit app for automatic recognition of handwritten numbers, written on a given canvas. The end result was an app with very high precision.

Innehållsförteckning

1 Inledning

Att per automatik kunna känna igen ansikten, nummerplåtar på bilar och signalement på eftersökta personer med mera är en ständigt mer eftertraktad förmåga (Li, 2015). Alla ovanstående används redan idag med stor framgång i samhället. Med hjälp av maskininlärning kan detta bli verklighet, ett ytterligare område är igenkänning av handskrift - något som denna rapport kommer att fokusera på. Eftersom handskrift varierar kraftigt från person till person och kan läsas in både roterat och med olika storlek, kan detta vara en stor utmaning för traditionella algoritmer att känna igen. Det är här AI, maskininlärning och robusta modeller verkligen kan göra en stor skillnad, i allt från att tolka adressetiketter på post till att digitalisera gamla och nya dokument.

Detta projekt handlar om att med hjälp av maskininlärning skapa en app som känner igen handskrivna siffror. Inom ramen för projektet ska flertalet modeller tränas, testas och utvärderas i förhållande till prestanda, precision och lämplighet. Detta tillvägagångssätt säkerställer ett konsekvent och kvalitetsorienterat arbete och slutprodukt.

1.1 Syfte och frågeställning

Syftet med detta projekt är att utveckla en applikation som kan känna igen handskrivna siffror med hög precision. Målet är att uppnå minst 90 % noggrannhet på testdata, det vill säga data som modellen inte har tränats på. Detta anses vara ett godtagbart resultat för en grundläggande maskininlärningsmodell (Prgomet 2025).

- Kan en modell tränas, testas och utvärderas så att den uppnår minst 90 % noggrannhet vid prediktioner på tidigare osedd testdata från MNIST?

2 Teori

2.1 Maskininlärning och klassificering

Maskininlärning är ett område inom artificiell intelligens (AI) som utvecklar algoritmer som lär sig mönster från data och kan därifrån göra förutsägelser. Klassificering är en av de mest centrala uppgifterna som kommer upp inom området, där målet är att förutsäga vilken kategori en specifik datapunkt tillhör.

I detta projekt klassificeras och tolkas handskrivna siffror. Modellen tränas på ett dataset med en stor mängd handskrivna kända siffror och lär sig då att se de mönster som kan användas för att identifiera nya, för modellen tidigare osedda siffror.

2.2 Modellval

För att kunna genomföra klassificeringen måste en modell väljas ut. Inom detta projekt har tre olika modeller utvärderats: K-Nearest Neighbors (KNN), Random Forest och Extra Trees. Dessa tre modeller har valts ut på grund av deras olika styrkor och hantering av klassificeringsproblem.

2.2.1 K-Nearest Neighbors (KNN)

KNN är en s.k. icke-parametrisk algoritm som inte gör några antaganden om hur datan ser ut, den klassificerar en ny datapunkt baserat på dess närmaste grannar i träningsdatan (Keen 2021). Hyperparametern K (antalet grannar) används för att finjustera resultatet. KNN är en i sammanhanget enkel modell att implementera men kan underprestera i stora dataset då varje enskild prediktion måste beräkna avståndet till varje träningspunkt.

2.2.2 Random Forest

Random Forest är en ensemblemodell som bygger på beslutsträd (Géron 2023, s. 220). Genom att träna varje träd på en slumpmässig del av träningsdatan och sedan göra majoritetsomröstning bland träden så nås ett slutresultat. Detta blir i sin tur kraftfull mot överanpassning och bra på att hantera komplexa mönster.

2.2.3 Extra Trees

Extra Trees är även den en modell baserad på beslutsträd (Géron 2023, s. 220). Den viktigaste skillnaden här är att medan Random Forest använder den optimala delningspunkten för varje träd, så väljs delningspunkterna här slumpmässigt. Detta kan ibland förbättra generaliseringsförmågan, det vill säga vidare minska risken för överanpassning.

2.3 Prestandamått

För att utvärdera modellernas prestanda har flera mått använts. Noggrannhet (Accuracy) mäter hur stor andel av alla utförda klassificeringar som blir korrekta Géron 2023, s. 107). Detta ger en god övergripande bild över modellens prestanda men kan också vara missvisande ifall datasetet är obalanserat. En Klassificeringsrapport har även skapats som innehåller mått såsom precision, recall och F1-score. Precision betyder här andelen korrekta prediktioner av en viss klass, medan recall andelen av de verkliga instanserna av klassen som modellen har identifierat korrekt (Prgomet 2023). F1-score är ett genomsnittsmått mellan dessa två .

Metoderna ovan ger en detaljerad inblick i prestandan för vardera modell och gör det möjligt att efter jämförelse fatta kvalitetssäkrade och säkra beslut inför valet av modell att implementera i projektet.

3 Metod

3.1 Datainsamling och förberedelse

I detta projekt användes datasetet MNIST, som innehåller 70 000 bilder av handskrivna siffror (28x28 pixlar), fördelade i tio klasser som är siffrorna 0-9 (Géron 2023, s. 103). Datasetet laddades in med hjälp av Scikit-learn och funktionen fetch_openml och normaliserades genom att skala pixelvärdena till [0,1].

```
from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784')

X, y = mnist["data"], mnist["target"]

X = X / 255.0
```

Modellen som används i applikationen är en Random Forest-modell som tränades på 80% av datan och resterande 20% användes som testdata. Denna modell valdes på grund av dess robusthet och förmåga att hantera stora mängder data utan överanpassning. Modellen använder 400 beslutsträd, vilket valdes baserat på resultat från hyperparameteroptimering där prestandan stabiliserades vid denna nivå. För att ytterligare validera modellens prestanda och precision användes en femfaldig korsvalidering innan train-test-splitten.

3.2 Utveckling av applikationen

Applikationen byggdes med Python och Streamlit för att ge en interaktiv användarupplevelse där användaren ser resultatet av sin handskrift i realtid genom att rita på en digital canvas. Innan resultatet når användaren preprocessas bilden innan den skickas till modellen för klassificering.

3.2.1 Preprocessing

För att möjliggöra och optimera igenkänningen av användarens ritade siffror genomgår denna flertalet steg innan klassificering är möjlig. För att förbereda data användes olika pre-processing tekniker från Scikit-learn (Scikit-learn, n.d.). Vidare användes funktioner från scikit-image för bildbehandling (Van der Walt et al., 2014):

1. Binarisering – Otsus tröskelvärde konverterar bilden till svartvitt. Detta gör det enklare för modellen att upptäcka mönster och konturer.
2. Centrering – Bilden centreras baserat på dess masscentrum. Detta minskar risken att modellen förväxlar siffror på grund av placering.
3. Dilation – En morfologisk behandling hanterar små detaljer och brutna linjer. Genom att fylla i dessa brister görs siffran mer lättläslig.
4. Skalning och normalisering – Bilden omformas till en 1D-array och normaliseras. Detta hjälper modellen att konvergera snabbare under träning - det vill säga att det inte behövs inte lika många iterationer eller justeringar för att optimera parametrarna, vilket leder till en snabbare träningsprocess.

Genom att använda dessa preprocessing-tekniker gör du bilderna mer konsekventa och lättare för modellen att förstå, vilket leder till bättre och mer pålitliga resultat. Efter ovanstående steg skickas bilden vidare till modellen för klassificering.

3.3 Testning och utvärdering

Modellens noggrannhet utvärderades genom dels femfaldig korsvalidering på hela datasetet, dels genom en train-test-split (80/20). Femfaldig korsvalidering ger en mer robust uppskattning av modellens prestanda genom att använda hela datasetet, medan train-test-split ger en enklare men snabbare utvärdering (Géron 2023, s. 89).

Eftersom modellen tränades på MNIST-data kan prestandan av handritade siffror i appen skilja sig från ovanstående. På grund av detta ges användaren själv möjlighet att undersöka detta genom att klicka i om prediktionen var korrekt eller felaktig efter varje inmatning. Dessa räknas sedan ihop under hela användarens session, för att sedan nollställas. Detta ger en subjektiv utvärdering av modellens faktiska prestanda, och kan därför inte användas för att fullt ut verifiera att modellen konsekvent uppnår minst 90% rätt prediktioner. Den subjektiva utvärderingen baseras på användarens interaktion i appen, vilket ger en indikation på hur väl modellen kan generalisera till nya, handritade siffror.

4 Resultat och Diskussion

4.1 K-Nearest Neighbor (KNN)

För KNN-modellen uppnåddes en genomsnittlig noggrannhet på **97.05%** vid korsvalidering, vilket visar att modellen generellt presterar mycket bra för klassificeringen av handskrivna siffror. Modellen visade också låg standardavvikelse (**0.0022**), vilket tyder på stabilitet och tillförlitlighet över olika deluppsättningar. Vid testuppsättningen presterade modellen med **97.13%** noggrannhet, vilket bekräftar att den inte overfittar träningen och håller en jämn prestanda.

4.2 Random Forest

Random Forest-modellen presterade också mycket bra, med en genomsnittlig noggrannhet på **96.95%** från korsvalidering och en standardavvikelse på **0.0025**. Den slutliga noggrannheten på testuppsättningen var **96.94%**, vilket var mycket nära KNN:s resultat. Random Forest hade en något bättre precision och recall för de flesta klasser jämfört med KNN, särskilt för klasser som 0 och 1, där modellen presterade nästan perfekt.

Valet av Random Forest berodde på att denna modell var snabbare att exekvera, vilket var viktigt för att kunna genomföra fler experiment på kortare tid. Med tanke på att modellen också gav nästan lika bra resultat som KNN och Extra Trees, valdes Random Forest för sin bättre balans mellan hastighet och noggrannhet.

4.3 Extra Trees

Extra Trees-modellen gav också bra resultat, med en genomsnittlig noggrannhet på **97.20%** och en standardavvikelse på **0.0008**. Den presterade liknande Random Forest och KNN men kräver mer beräkningskraft, vilket gjorde att den inte valdes i det slutliga projektet.

4.4 Reflektioner och förbättringsområden

En intressant observation är att alla tre modeller visade på otroligt stabil prestanda, vilket tycker på att datans integritet och kvalitet är gedigen. Trots att modellerna presterade bra skulle det vara intressant att vidare utforska ytterligare optimering för att maximera prestanda. En potentiell förbättring hade kunnat vara att använda Grid search för att ytterligare finjustera hyperparametrarna, istället för att göra det manuellt. Detta hade eventuellt kunnat förfina precisionen ytterligare, men med tanke på den redan höga noggrannheten så blev resultatet lyckat.

5 Slutsatser

Detta projekt har undersökt och jämfört prestandan hos tre olika maskininlärningsmodeller – K-Nearest Neighbors (KNN), Random Forest och Extra Trees – för att klassificera handskrivna siffror från MNIST-datasetet. Genom att använda flera prestandamått, såsom noggrannhet, precision, recall och F1-score, har det visats att alla modeller presterade bra, med mycket höga noggrannheter kring 97%.

Bland de tre modellerna var KNN den näst snabbaste att exekvera, medan Extra Trees var den långsammaste. Random Forest valdes till slut som den mest effektiva modellen, både när det gäller hastighet och prestanda. Den presterade nästan lika bra som KNN och Extra Trees men hade en betydligt snabbare exekveringstid, vilket gjorde den till det mest praktiska valet för detta projekt.

För applikationen där handskrivna siffror skulle kännas igen i realtid, visade Random Forest sig vara det bästa valet. Kombinationen av snabb exekvering och nästan lika hög prestanda som de andra modellerna gjorde den till det mest optimala valet för detta projekt.

De tester som utfördes på användardata gav en god indikation på modellens prestanda i en verklig applikation, även om subjektiv utvärdering inte kan ersätta en fullständig, kvantitativ utvärdering av modellen.

Sammanfattningsvis kan det konstateras att Random Forest-modellen erbjuder en bra balans mellan noggrannhet och effektivitet, vilket gör den till ett utmärkt val för projektet. Vidare optimering av hyperparametrar, till exempel genom användning av Grid Search, kan potentiellt ytterligare förbättra prestanda, men den uppnådda noggrannheten var redan tillräcklig för det aktuella syftet.

6 Teoretiska frågor

1. Kalle delar upp sin data i "Träning", "Validering" och "Test", vad används respektive del för?

- **Träning** (cirka 70%) används för att träna modellen på den aktuella datan, för att på så vis kunna förutsäga framtida datapunkter.
- **Validering** (cirka 15%) används för att testa modellens prestation under träningen, innan man går vidare till nästa steg med ny data. Genom valideringsdatan kan man justera hyperparametrar och överanpassning, för att se till att modellen inte bara lär sig träningsdatan utan även hanterar okänd data.
- **Test** (cirka 15%) används för att testa modellens prestanda på ny okänd data efter att träningen är klar.

2. Julia delar upp sin data i träning och test. På träningsdatan tränar hon tre modeller: "Linjär Regression", "Lasso regression" och en "Random Forest modell". Hur ska hon välja vilken av de tre modellerna hon ska fortsätta använda när hon inte har skapat ett explicit "valideringsdataset"?

Genom att använda Cross Validation. K-Fold Cross Validation med hjälp av sklearn.model_selection och cross_val_score ger en bild av vilken modell som ger lägst MSE (Mean Squared Error). Om två modeller har väldigt lika eller samma MSE, väljer Julia den enklare eller snabbare modellen. Vid överanpassning använder hon den modell som är mest generell, såsom Lasso, som är regulariserad. Julia bör också komma ihåg att använda minustecknet på resultatet för neg_mean_squared_error för att få rätt resultat.

3. Vad är ett "regressionsproblem"? Kan du ge några exempel på modeller som används och potentiella tillämpningsområden?

Regressionsproblem innebär att förutsäga ett kontinuerligt värde, såsom försäljningsintäkter, bostadspriser och temperaturdiagnoser, baserat på den data modellen tränas på. Vanliga modeller inkluderar:

- **Linear Regression** som används för att modellera linjära samband.
- **Lasso Regression**, som liknar Linear Regression men med regularisering, vilket gör modellen enklare genom att straffa alltför stora koefficienter och sätta dem till noll för att förhindra överanpassning.
- **Random Forest Regression**, som tränar på slumpmässigt valda delar av datan på flera beslutsträd, där varje träd gör en egen förutsägelse. Slutresultatet är medelvärdet av alla trädens förutsägelser, och denna metod är särskilt bra för komplexa och stora datamängder.

4. Hur kan du tolka RMSE och vad används det till?

RMSE (Root Mean Squared Error) används för att beräkna hur stor avvikelsen är mellan

regressionsmodellens förutsägelser och de verkliga värdena. Ju lägre RMSE, desto bättre kan modellen förutsäga framtida data. Eftersom RMSE mäter det genomsnittliga felet och även kvadrerar det, syns stora fel mer tydligt.

5. Vad är ett "klassificeringsproblem"? Kan du ge några exempel på modeller som används och potentiella tillämpningsområden? Vad är en "Confusion Matrix"?

Klassificeringsproblem innebär att förutsäga indelningar i diskreta kategorier, till exempel binära (sjuk/frisk) eller multiklass (katt/råtta/kamel). Modeller som används för klassificering är till exempel:

- Random Forest
- Logistisk Regression
- Neurala nätverk (MLP, CNN)

Ett **Confusion Matrix** är en tabell som visar hur många förutsägelser som är korrekta och felaktiga, med fyra celler: TP (True Positives), FP (False Positives), TN (True Negatives) och FN (False Negatives).

6. Vad är K-means modellen för något? Ge ett exempel på vad den kan tillämpas på.

K-means är en klustringsalgoritm som delar upp datapunkter i K olika kluster baserat på deras avstånd till centrala punkter. Den kan användas för att exempelvis dela upp sjukdomsgrupper i medicinsk data, filtrera skräppost i dokumentklassificering eller segmentera kunder i marknadsföringsdata.

7. Förklara (gärna med ett exempel): Ordinal encoding, one-hot encoding, dummy variable encoding.

- **Ordinal Encoding** används när kategorierna har en inbördes ordning. Varje kategori ersätts med ett nummer.
    - Exempel: "Låg" = 0, "Medel" = 1, "Hög" = 2.
- **One-hot Encoding** används när varje kategori får en egen kolumn, där värdet är 1 om kategorin är aktiv och 0 om den inte är det.
    - Exempel: "Låg" = [1, 0, 0], "Medel" = [0, 1, 0], "Hög" = [0, 0, 1].
- **Dummy Variable Encoding** är likt One-hot Encoding, men tar hänsyn till analyser där multikollinearitet kan skapa problem. Här tas en kategori bort och används som referenskategori.
    - Exempel: Om "Låg" är referenskategori blir "Medel" = 1, "Hög" = 0; om båda är 0, betyder det att referenskategorin "Låg" är aktiv.

8. Göran påstår att datan antingen är "ordinal" eller "nominal". Julia säger att detta måste tolkas. Hon ger ett exempel med att färger såsom {röd, grön, blå} generellt sett inte har någon inbördes

ordning (nominal), men om du har en röd skjorta så är du vackrast på festen (ordinal) – vem har rätt?

- **Nominal data** består inte av någon inbördes ordning. Exempelvis har färgerna röd, blå och grön ingen objektiv rangordning, men de kan tillskrivas en subjektiv rankning och blir då **Ordinal**.
- **Ordinal data** är data som har en inbördes ranking. Exempelvis "Fult", "Okej", "Vackrast" är en subjektiv ranking, där "röd skjorta gör en vackrast" är en sådan ordning, vilket gör den till ordinal.

9. Vad är Streamlit för något och vad kan det användas till?

Streamlit är ett Pythonbibliotek som gör det möjligt att snabbt skapa webbapplikationer för maskininlärning och datavisualisering. Det används för att skapa interaktiva applikationer där användaren kan träna, utvärdera och justera modeller i realtid. Genom Streamlit kan man skapa enkla och tydliga gränssnitt till maskininlärningsmodeller eller andra Pythonbaserade applikationer utan att behöva kunna frontendutveckling.

7 Självutvärdering

1. En av de största utmaningarna jag stötte på var att hantera prestanda och exekveringstid för de olika modellerna. Extra Trees visade sig vara alltför långsam, vilket gjorde att jag behövde omvärdera mitt val av modell. Jag valde att använda Random Forest, som inte bara var snabbare utan också gav mycket bra resultat i termer av noggrannhet. Genom att noggrant analysera och jämföra modellerna med hänseende till både prestanda och exekveringstid såg jag till att optimera valet för mitt specifika projekt.

   En annan utmaning var att säkerställa att jag hade en korrekt utvärdering av mina modeller genom att använda olika metoder som korsvalidering och precision/recall. Jag hanterade detta genom att hålla ett kontinuerligt fokus på att utvärdera mina resultat från flera olika perspektiv för att få en så rättvis och överskådlig bild som möjligt.

   En tredje utmaning var att skapa själva appen och skapa ett tydligt och användarvänligt gränssnitt i Streamlit, en miljö som var helt ny för mig. Det var dock inspirerande att se vilket kraftfullt och lättillgängligt verktyg det är.

2. Jag anser att jag har nått upp till resultatet VG baserat på detta projekt och studierna under kursens gång. Jag har nått samtliga läranderesultat för kursen genom att inte bara implementera maskininlärning, utan också kritiskt analysera de valda modellerna, deras prestanda och exekveringstider. Jag har löst ett problem genom att tillämpa flera maskininlärningstekniker på ett fördjupat sätt och med hög säkerhet, där jag har diskuterat både styrkor och svagheter hos de olika modellerna.

   Jag har redogjort för mina val av modeller (till exempel Random Forest) och motiverat dem utifrån både teoretisk grund och experimentell utvärdering. Jag har även kritiskt diskuterat modellernas anpassning till uppgiften, inklusive val av hyperparametrar och utvärderingstekniker, vilket enligt mig uppfyller de krav som ställs för VG.

3. Du är en grym pedagog! Maskininlärning är nu en självklar passion framåt i yrkeslivet. Tack för denna kurs!

Källförteckning

7.1 Tryckta källor

Geron, A. (2023). *Hands-on machine learning with Scikit-learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems* (3rd ed.). O'Reilly Media.

7.2 Audiovisuella källor

Li, F.-F. (2015-03). *How we're teaching computers to understand pictures* [Video]. TED. https://www.ted.com/talks/fei_fei_li_how_we_re_teaching_computers_to_understand_pictures

Keen, M. (2024-09-02). *K-Nearest Neighbors (KNN) algorithm explained* [Video]. YouTube. https://www.youtube.com/watch?v=b6uHw7QW_n4

Prgomet, A. (2023-05-27). *Klassificering* [Video]. YouTube. https://www.youtube.com/watch?v=-QzFLifZgNw

Prgomet, A. (2025, 11 mars). *Machine Learning* [Onlineföreläsning]. EC Utbildning.

7.3 Digitala källor online

Scikit-learn documentation. (n.d.). Preprocessing. Hämtad [2023-03-14] från https://scikit-learn.org/stable/modules/preprocessing.html

Van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J., Yager, N., Gouillart, E., & Kerherve, R. (2014). scikit-image: Image processing in Python. Hämtad [2023-03-14] från https://scikit-image.org/