Sifferigenkänning med MNIST och scikit-learn

Linus Rundberg Streuli

EC Utbildning, Data Scientist 2023

2024-03-22

Abstract

Digit recognition is a field in machine learning with many applications. Here I use it to build an application to identify numbers in a sudoku puzzle so the computer can solve it.

Förkortningar och begrepp

MNIST: Ett dataset med 70 000 handskrivna siffror, anpassade för att användas vid maskininlärning.

Sudoku: ett logikpussel med siffror där siffrorna 1-9 ska placeras ut i rader, kolumner och rutor i ett 9×9-rutnät.

Innehållsförteckning

Abstract 2
Förkortningar och begrepp 2
Innehållsförteckning 3
Inledning 4
  Syfte 4
  Frågeställningar 4
Teori 5
  Maskininlärning 5
  Typer av problem 5
  Evaluering av modeller 5
    Accuracy 5
    Confusion matrix 5
  Bildbearbetning 5
Metod 6
  Hämta MNIST 6
  Dela upp datan 6
    Förberedelse av datan 6
Resultat och diskussion 7
  Resultat 7
    EDA 7
    Träning av modeller 8
    Sudoku-lösare 10
  Diskussion 11
Slutsatser 13
Appendix A: Bilder 14
Appendix B 16
  Teoretiska frågor 16
Källförteckning 18

Inledning

Maskininlärning är användbart inom många områden på grund av datorers förmåga att snabbt göra avancerade matematiska beräkningar, långt bortom den mänskliga förmågan.

En gren inom maskininlärning är att tolka innehållet i bilder. Inom detta fält finner vi bland annat sifferigenkänning. Ett användningsområde för sifferigenkänning rör sig i gränssnittet mellan den analoga och digitala världen. En vältränad maskininlärningsmodell kan tolka och bearbeta stora mängder data som annars skulle behöva föras in manuellt.

Beroende på användningsområde kan det dock ställa stora krav på modellens förmåga då feltolkningar av datan kan få allvarliga konsekvenser och det inte alltid är rimligt att övervaka alla delar av processen.

Syfte

Syftet med projektet är att skapa en applikation som med hjälp av maskininlärning kan identifiera siffrorna i en bild av ett sudoku och sedan lösa sudokut och visa lösningen för användaren. För att uppnå syftet kommer följande frågeställningar besvaras:

Frågeställningar

1. Vilken maskininlärningsmodell presterar bäst på MNIST-datasetet?
2. Vilka bearbetningar av bilderna behövs för modellen ska kunna identifiera siffrorna på bästa sätt?

Teori

Maskininlärning

I grunden kan maskininlärning definieras som en process där en maskin lär sig att dra slutsatser av data utan att explicit programmeras till det (Géron, 2019, s. 2). Man drar helt enkelt nytta av datorers förmåga att snabbt göra många komplicerade beräkningar.

Typer av problem

Maskininlärning kan grovt delas upp i två grupper: *supervised* och *unsupervised learning*.

Supervised learning definieras av att modellen har tillgång till ”facit” och utifrån det tränas i att antingen förutspå ett visst kontinuerligt värde (regression) eller placera en instans i en viss klass (klassificering).

I *unsupervised learning* finns inget ”facit” att tillgå utan modellen finner själv mönster i datan och kan sedan använda dessa för att exempelvis placera ny data i en viss klass.

Det problem som ligger till grund för rapporten är ett klassificeringsproblem inom *supervised learning*.

Evaluering av modeller

När det kommer till klassificering finns det ett antal olika mått man kan använda sig av för att avgöra hur väl en modell presterar.

Accuracy

Accuracy beskriver helt enkelt hur stor andel av instanserna som modellen klassificerade korrekt. I många fall kan *accuracy* fungera bra som mått, men det finns också tillfällen då en hög *accuracy* inte nödvändigtvis betyder att modellen presterar bra. Ett exempel är när det är obalans mellan klasserna i datan (Géron, 2019, s.80).

Confusion matrix

En *confusion matrix* är ett tydligare mått på hur en klassificeringsmodell presterar. Det är en tabell som beskriver förhållandet mellan de faktiska och de predikterade värdena. Utifrån en *confusion matrix* kan man också få ut måtten *precision*, *recall*, och *f1*.

Bildbearbetning

För att ge modellen bästa möjliga förutsättningar att göra en korrekt identifiering behöver bilder på siffror bearbetas för att vara lika träningsdatan. Detta innefattar steg som att förkasta färginformation, skala ner bilden till de 28×28 pixlar som modellen är tränad på, samt att platta ut bilden till en lång rad med 784 kolumner med pixeldata.

Metod

Hämta MNIST

MNIST-datasetet är ett klassiskt dataset inom maskininlärning och finns tillgängligt i många former (Géron, 2019, s. 85). För det här arbetet har jag använt det som finns tillgängligt på openml.org.

Dela upp datan

För att kunna avgöra hur väl modellen presterar på okänd data är det viktigt att inte träna modellen på all tillgänglig data utan dela upp datan i åtminstone ett tränings- och ett test-set (Géron, 2019, s. 30). Det är också viktigt att inte använda sig av test-setet för tidigt i processen utan spara det tills man har en modell som ger bra resultat på träningsdatan. För att ändå kunna avgöra vilken modell som presterar bäst på okänd data kan man antingen dela upp datan i ytterligare en del som kallas validerings-set, eller evaulera modellen på slumpmässigt utvalda delar av träningsdatan, något som kallas *cross validation*.

Siffrorna i MNIST-datasetet ligger slumpmässigt fördelade så ett enkelt och vanligt sätt att dela upp datan är att ta de första 60 000 instanserna till träning och spara 10 000 till test.

Förberedelse av datan

MNIST-datasetet är redan väl förberett för att användas till maskininlärning. Den enda extra förberedelse jag gjorde var att normalisera datan genom att dela varje pixelvärde med 255 för att få ett värde mellan 0 och 1.

Resultat och diskussion

Resultat

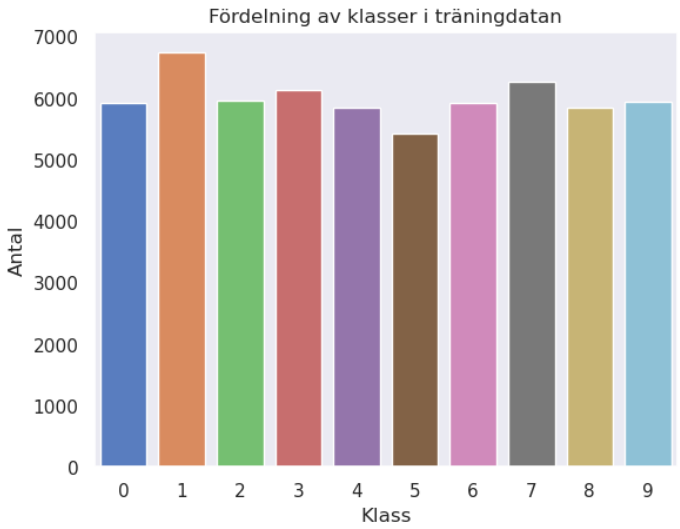
EDA

MNIST-datasetet består alltså av 70 000 handskrivna siffror. Datan är bearbetad så att varje bild av en siffra är skalad till 28×28 pixlar och sedan utplattad till en *array* med 784 värden, ett för varje pixel. Värdet motsvarar pixelns färgintensitet.

Figur 1: De första 25 siffrorna i träningsdatan

Figur 1 visar att bilderna också är inverterade, med vita siffror på svart bakgrund.

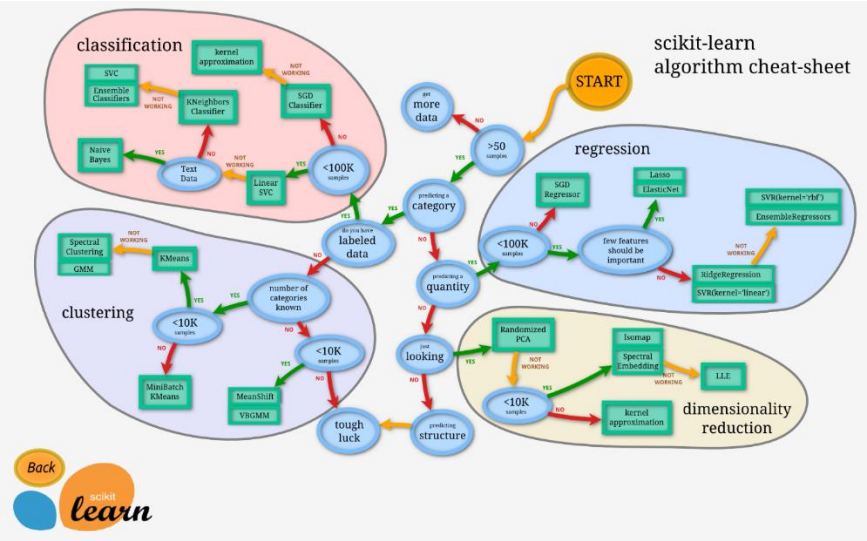
Figur 2 visar fördelningen av siffror i träningsdatan. En liten övervikt ettor och något färre femmor men inget jag bedömer ska skapa några problem.



Figur 2: Fördelning av klasser i träningsdatan

Träning av modeller

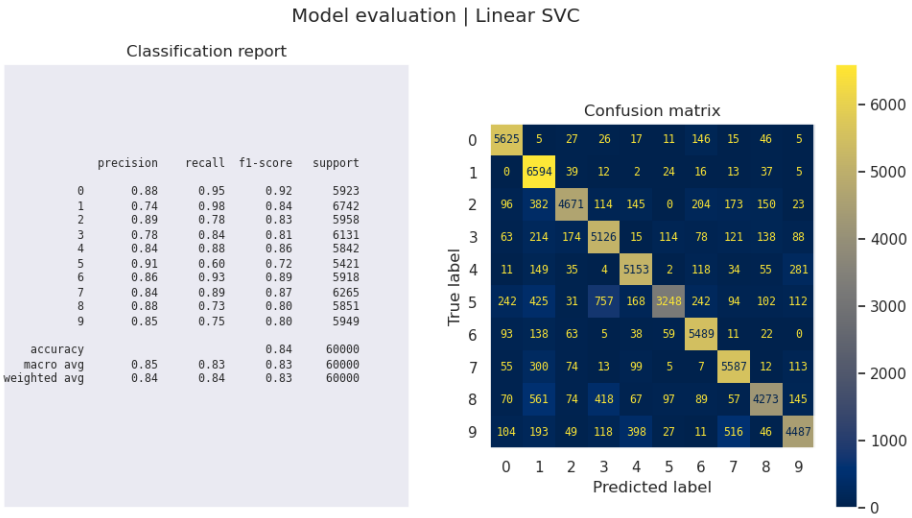
För att välja modeller utgick jag från den här bilden från Sciki-Learns hemsida:



Figur 3: Scikit-Learn algorithm cheat sheet

Linear Support Vector Classifier

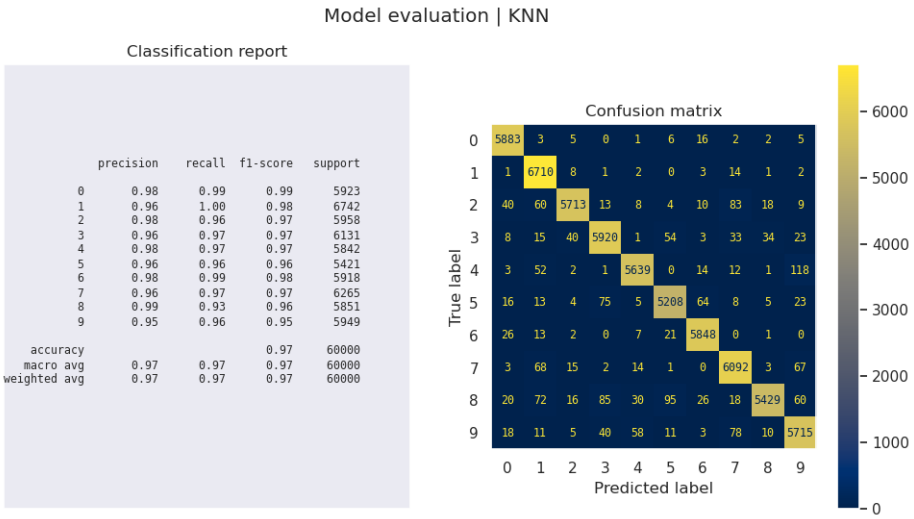
Utifrån schemat i bilden ovan började jag med att träna en *Linear Support Vector Classifier* med standard-hyperparametrar. Figur 4 visar resultaten. Istället för att försöka förbättra modellen gick jag vidare till nästa.



Figur 4: Evaluering av Linear SVC

KNeighbors Classifier

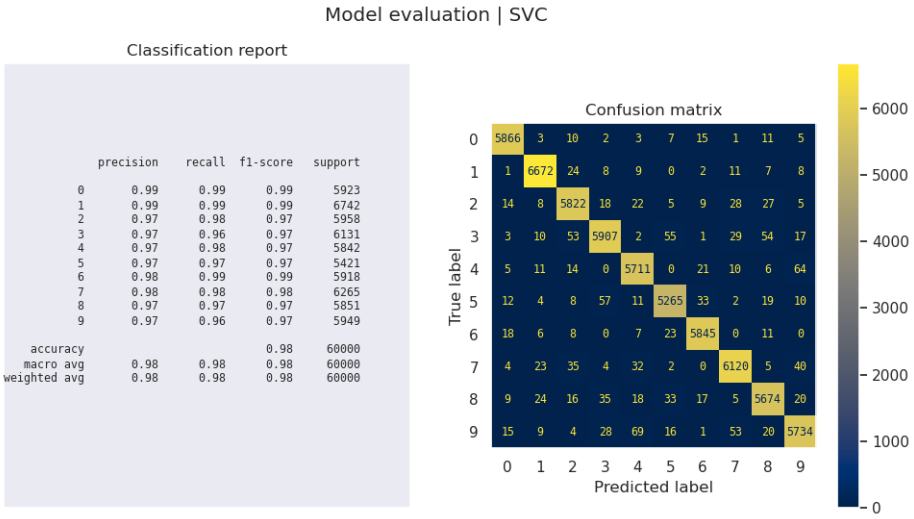
Nästa modell jag tränade var en KNeighbors Classifier. Figur 5 visar resultaten, en klar förbättring, från 0,84 accuracy till 0,97.



Figur 5: Evaluering av KNeighbors Classifier

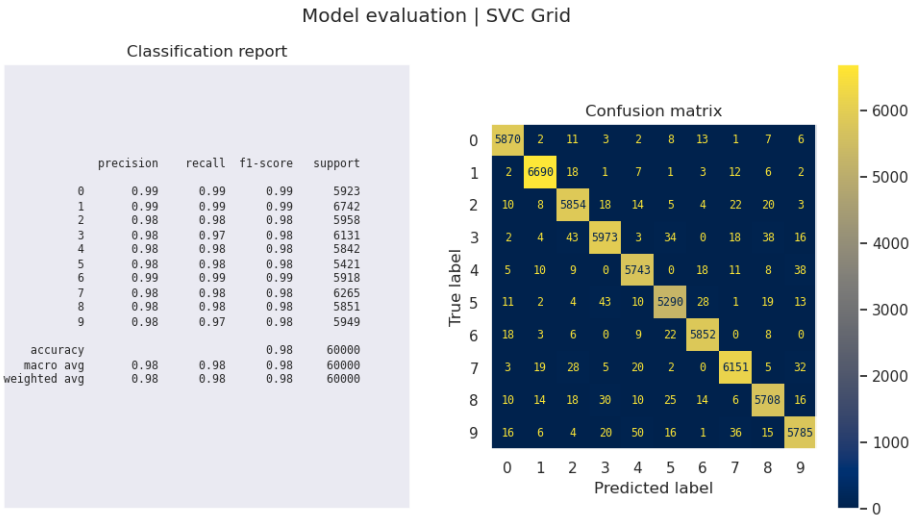
Support Vector Classifier

Jag bestämde för för att testa en *Support Vector Classifier* för att se hur den skiljde sig från en *Linear SVC*. Med standard-hyperparametrarna, (bland annat kernel='rbf') blev resultaten de vi ser i Figur 6.



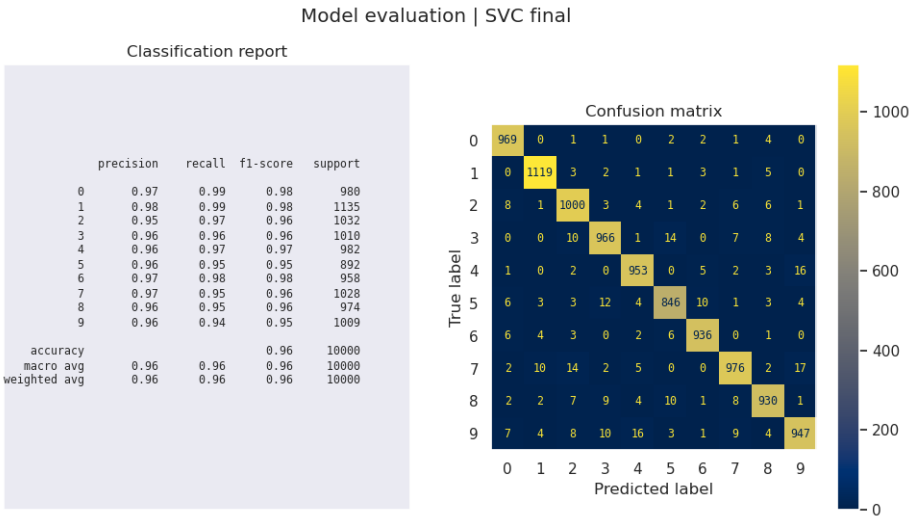
Figur 6: Evaluering av Support Vector Classifier

Direkt "ur lådan" fick SVC:n en accuracy på 0.98. En *grid search* pekade på att ett högre värde på hyperparametern C, det vill säga en nåt mindre regulariserad modell, presterade bättre på valideringsdatan. Det bästa värdet på C var 10, vilket också var det högsta värdet jag testade. Jag hade alltså kunnat gå ännu högre, men nöjde mig med en accuracy på 0,98.



Figur 7: Evaluering av SVC efter GridSearch. C=10.

SVC-modellen hade högst accuracy, så jag evaluerade den på testsetet. Figur 8 visar de slutliga resultaten.



Figur 8: Evaluering av den slutliga modellen

Sedan sparade jag modellen via `joblib` och använde i min sudoku-app.

Sudoku-lösare

Jag gjorde en enkel app i *Streamlit* för att testa min idé. (Bilder på appen finns i appendix A.)

Appen är upplagd som följer:

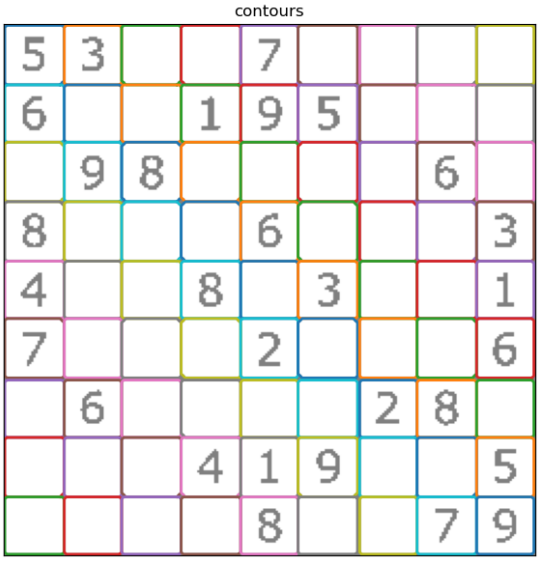
- Användaren laddar upp en bild eller tar en bild med enhetens kamera
- Användaren beskär bilden så att den bara innehåller själva sudokut.
- En bildbearbetningsfunktion går igenom ett antal steg för att förbereda bilden för modellen:

    - Omvandla bilden till gråskala
    - Skala bilden till 252×252 pixlar för att enkelt kunna dela upp den i 81 stycken 28×28 stora rutor
    - Invertera färgerna
    - Applicera ett tröskelvärde på bilden för att skärpa kanterna
- Med hjälp av en *meshgrid* loopar en funktion över bilden i 28×28 pixlar stora rutor, rad för rad, och läser av medelvärdet av pixlarna i rutan. Är det under ett visst värde pekar det på att det finns en siffra i rutan och modellen försöker identifiera siffran. Prediktionen läggs till i en lista. Är rutan tom läggs ett tomt värde till i listan istället.
- De färdiga prediktionerna visas för användaren som en tabell med möjlighet att redigera felaktiga värden.
- När tabellen stämmer med det faktiska sudokut kan användaren klicka på knappen ”Lös sudokut”, varefter appen försöker lösa det med hjälp av *brute force*-metoden. Lyckas den kan användaren sedan klicka på ”Visa lösning” för att få se det lösta sudokut.

Diskussion

Vägen fram till en någotsånär fungerande app var inte spikrak.

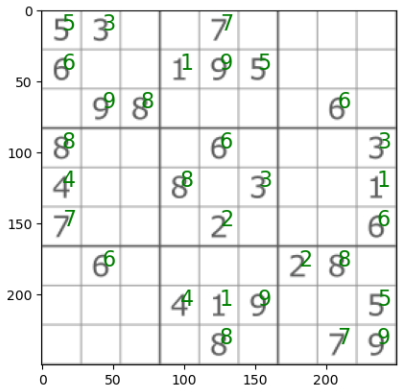
För att kunna identifiera siffrorna använde jag först `find_contours` från scikit-image och filtrerade resultatet på en viss storlek för att bara behålla själva rutorna. Figur 7 visar hur det kunde se ut.



Figur 9: Scikit-images find_contours

Sedan kunde jag loopa igenom rutorna och mäta medelvärdet av pixlarna i var och en. Om värdet var under ett tröskelvärde kunde jag anta att den rutan innehöll en siffra och

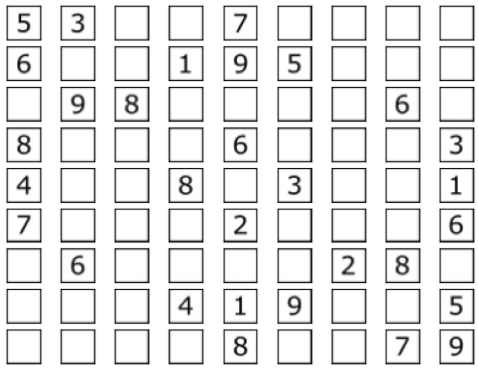
kunde försöka identifiera den med modellen. Figur 8 visar resultatet av en sådan process.



Figur 10: Ett sudoku med identifierade siffror

Metoden var dock känslig för skillnader i pixelintensitet, och klarade inte av för tunna linjer mellan rutorna.

Då hittade jag `st-cropper`, ett tillägg till Streamlit som gjorde att jag kunde beskära bilden innan jag skickade den till bearbetning. Då kunde jag strunta i att hitta konturer – det enda jag behöver göra är att skala bilden till 252×252 pixlar och sedan loopa över den i 28×28 pixlar stora rutor.



Figur 11: Sudokut uppdelat i 81 rutor

Dock kräver det att sudokut på bilden är rakt. En lösning jag tänker mig men inte implementerat är att istället för att beskära med en ruta, projicera sudokut på en *canvas* och låta användaren markera de fyra hörnen, för att sedan räta ut bilden och beskära.

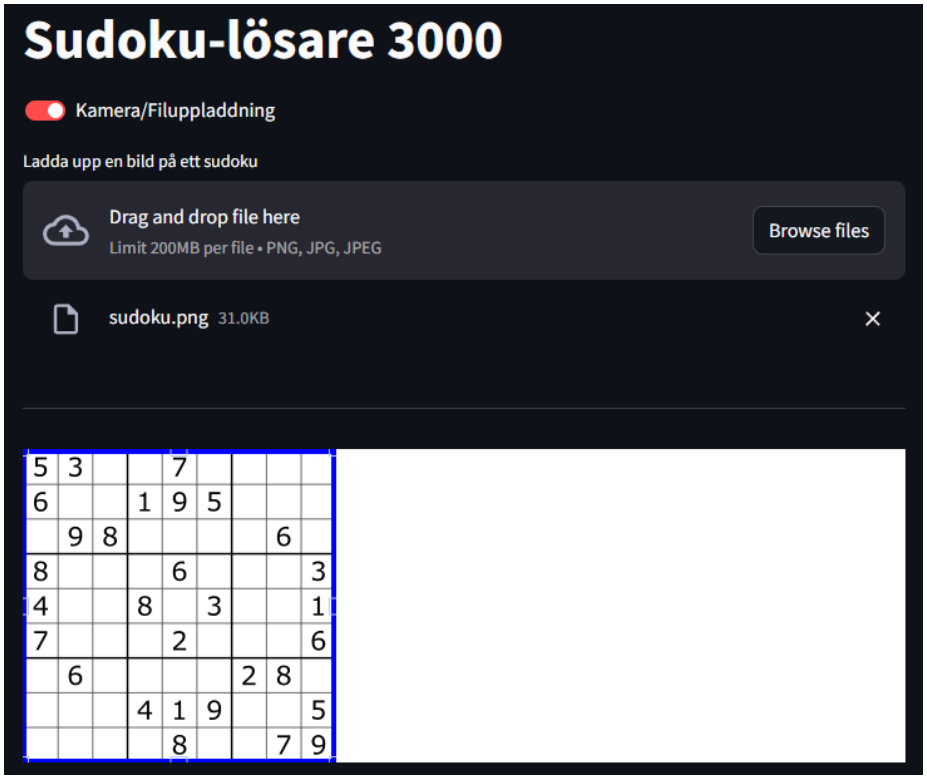
Modellen är också fortfarande känslig för lokala skillnader i intensitet som till exempel skuggor. Mer arbete på att bearbeta bilden skulle behövas, men resultatet visar på att det är görbart.

Slutsatser

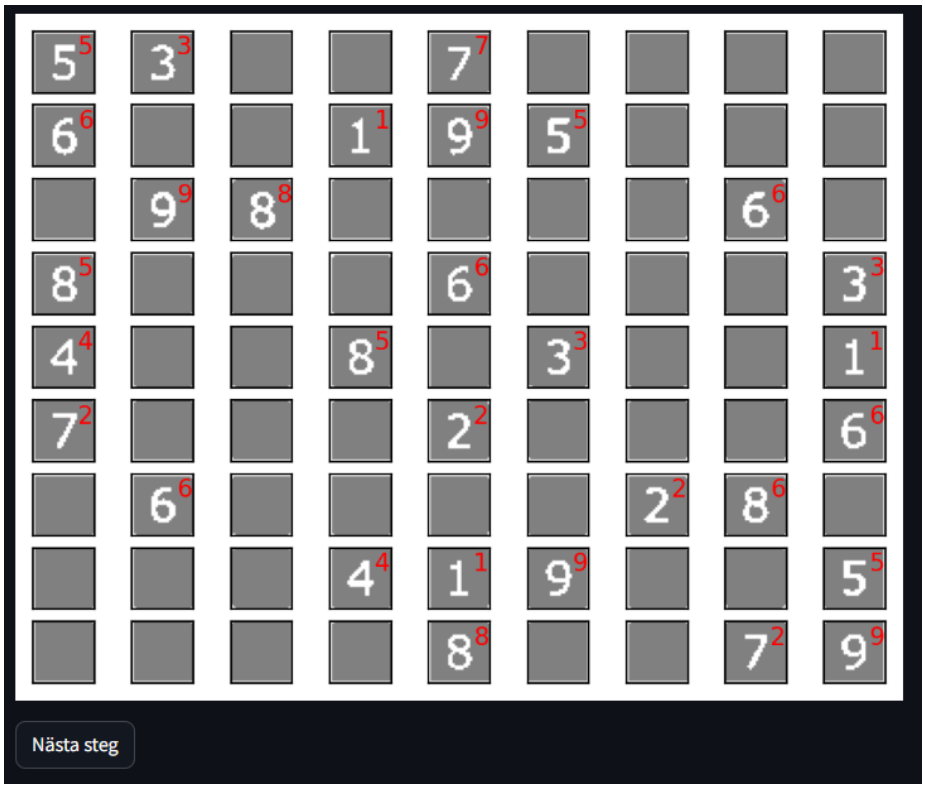
Problemen jag stött på har egentligen mindre handlat om själva maskininlärningen och mer om att lära hur Streamlit hanterar data, samt att bearbeta bilder så att det går att hantera dem som ett sudoku. En perfekt modell hade visserligen inte behövt ett sätt att kontrollera och ändra de identifierade siffrorna, men en perfekt modell går knappast att skapa.

Jag kan också konstatera att de flesta siffror modellen fick identifiera inte är handskrivna utan tryckta. Ett (analogt) sudoku är ofta en blandning av båda. Att istället träna modellen på *EMNIST*-datasetet skulle vara en möjlig utveckling av projektet (Cohen m fl, 2017) .

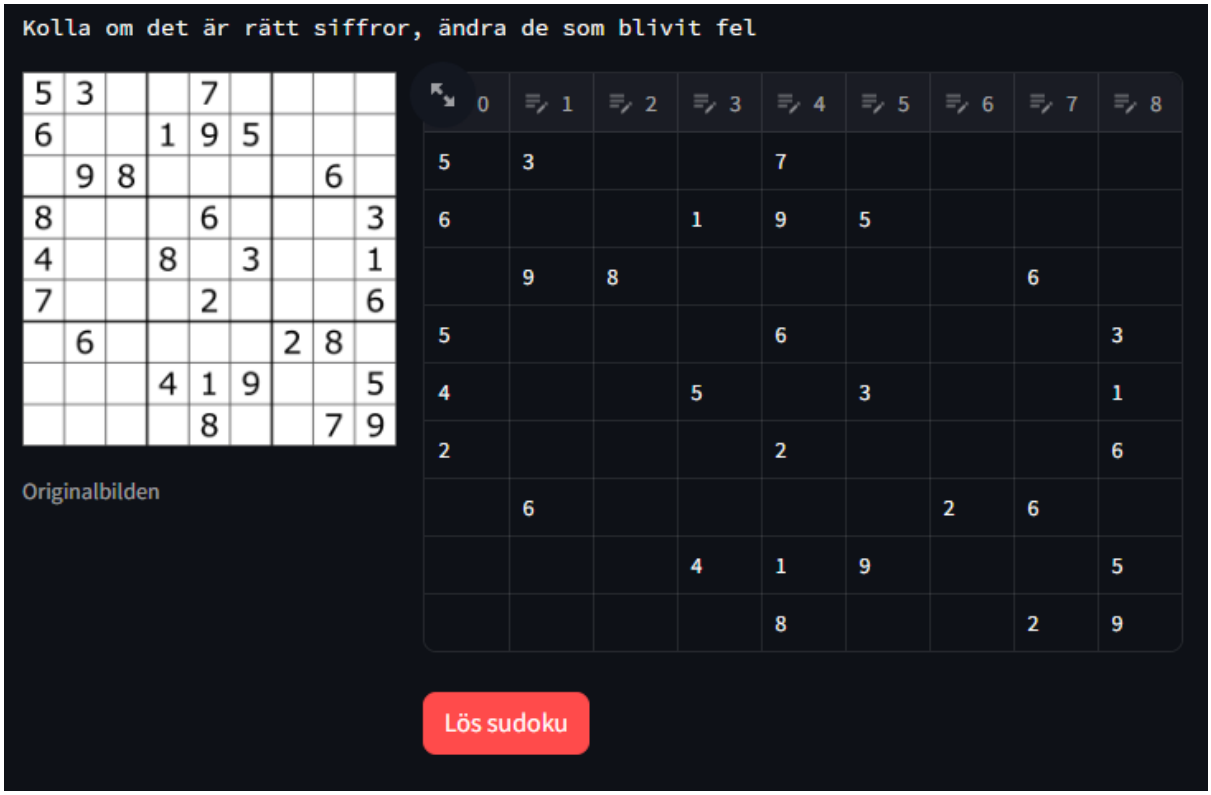
Appendix A: Bilder



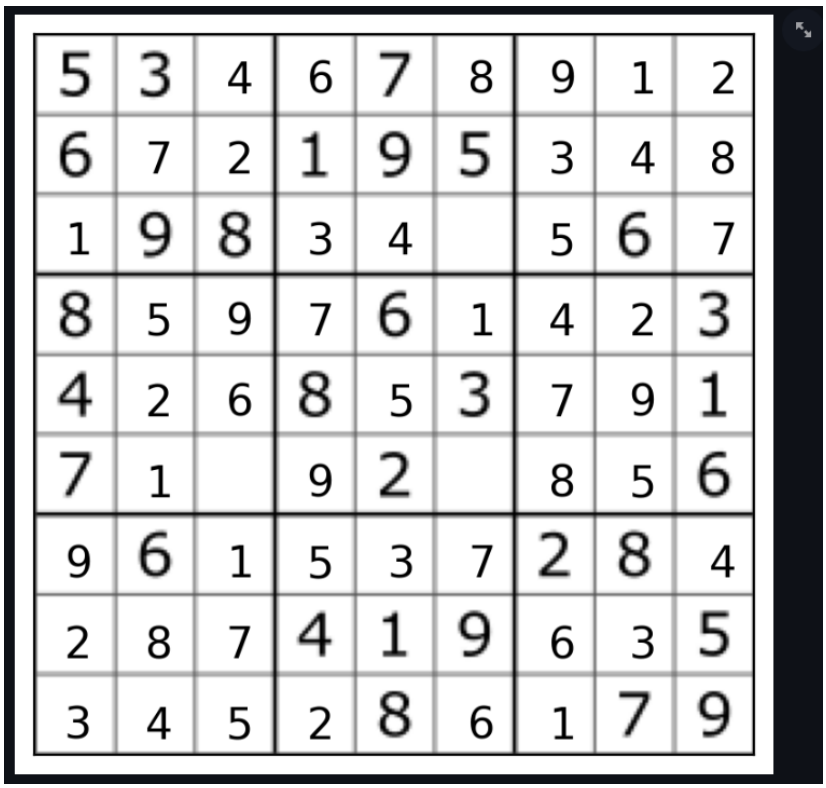
Figur 12: En användare har laddat upp en bild på ett sudoku och beskurit bilden.



Figur 13: Modellen har identifierat siffrorna i sudokut. Inte alla stämmer!



Figur 14: Användaren kan redigera resultaten och rätta de siffror som modellen felidentifierat.



Figur 15: Det lösta sudokut!

Appendix B

Teoretiska frågor

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

Träningsdelen används för att träna flera olika modeller. För att undvika att använda test-datan för tidigt kan han sedan använda validerings-delen för att se hur bra modellerna hanterar osedd data. Test-datan använder han i slutskedet av processen för att testa den modell som har presterat bäst på valideringsdatan.

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

Julia kan använda *cross validation* som ett alternativ till att använda ett explicit validerings-set.

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

Regressionsproblem är inom maskininlärning när en modell ska ta fram kontinuerliga värden. Exempel på modeller som används är *Linear Regression*, *Support Vector Machines* och *Random Forest*. Potentiella användningsområden är att förutsäga exempelvis fastighetspriser eller löneutveckling.

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

RMSE står för *Root Mean Square Error* och är roten ur kvadraten av medelavstånden mellan de faktiska och de av modellen skattade värdena. Det används som ett mått på hur bra en regressionsmodell presterar.

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

Klassificeringsproblem är inom maskininlärning när en modell ska placera en instans i en viss klass baserat på dess egenskaper. Exempel på modeller som används är *Logistic Regression, K-nearest Neighbors och Random Forest*. Potentiella användningsområden är exempelvis inom medicin för att klassificera tumörer som god eller elakartade. En *confusion matrix* är en tabell som visar hur modellen har presterat genom att ställa korrekta prediktioner mot inkorrekta.

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

K-means är en modell som används för *klustring*, som är en del inom *unsupervised learning*. Den kan användas för att dela upp data i klasser utan tillgång till *labels* som visar den korrekta klassen. Ett användningsområde är för att förbereda data innan klassificering, som ett sätt att dimensionsreducera datan (Geron, s. 251).

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

Maskininlärningsmodeller behöver datan som siffror. *Ordinal encoding* är ett sätt att omkoda data till en typ som modellen kan använda och är specifikt till för värden som har en inneboende ordning mellan sig. Exempelvis kan data som har en kolumn ”Storlek” med värdena ”Stor”, ”Mellan”, ”Liten” omkodas så att alla instanser med värdet ”Liten” får en 0:a istället, alla ”Mellan” en 1:a, och så vidare.

I de fall värdena inte har en inneboende ordning kan *ordinal encoding* göra mer skada än nytta. Då är det bättre med *one-hot* eller *dummy variable*-encoding, som är ungefär samma sak. Här skapas det lika många kolumner i datan som det finns unika värden i kolumnen som ska kodas, och instansen får en 1:a i den kolumn som motsvarar värdet i originalkolumnen. Skillnaden mellan *one hot*- och *dummy variable*-encoding är att *dummy variable*-encoding droppar den första kolumnen och låter det värdet motsvaras av 0:or i alla de andra kolumnerna.

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?

Även om hennes exempel kanske inte är det bästa har Julia rätt i att datan generellt behöver tolkas, och färger kan i vissa sammahang representera ordinal data – men inte alltid!

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

Streamlit är ett ramverk för att med relativt enkel Python-kod bygga appar för att hantera och presentera data.

Källförteckning

Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Hämtad den 22 mars 2024 från http://arxiv.org/abs/1702.05373

Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Ed.)*. Sebastopol: O'Reilly.

MNIST Dataset. Hämtad den 17 mars 2024 från https://openml.org.

Scikit-Learn (2024). Choosing the right estimator. Hämtad den 21 mars 2024 från Scikit-Learns hemsida: https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html