MODELLERING AV MNIST DATASETET

SIFFERIGENKÄNNING MED SUPERVISED

MACHINE LERNING MODELLER

Astrid Hansen

EC Utbildning

Kunskapskontroll 2

202403

Abstract

This report focuses on testing various classification models on the MNIST dataset and evaluating their predictions based on selected performance metrics. The analysis reveals that the Voting Classifier achieved the highest accuracy score, demonstrating the effectiveness of ensemble methods. The performance of the models was generally high and improved with fine-tuning the hyperparameters through GridSearch. The Voting Classifier will be utilized as the prediction model for a digit recognition app developed in Streamlit.

Innehållsförteckning

1 Inledning

Bildigenkänning har varit ett stort ämne under lång tid och fortsätter att växa i takt med den tekniska utvecklingen inom AI och maskininlärning. Tack vare bildigenkänning har många arbetsprocesser blivit mer effektiva. Det har blivit möjligt att automatisera klassificering av bilder, genomföra ansiktsigenkänning vid inloggning, analysera bilder för sjukdomsdiagnoser och skapa appar för blinda som hjälper dem att läsa och 'se' världen, för att nämna några exempel. Möjligheterna är många.

Bildigenkänning kan utföras med maskininlärning, som är en gren av artificiell intelligens. Målet med maskininlärning är att utbilda datorer att utföra specifika uppgifter eller göra prediktioner baserade på tillgängliga data. Träningen av modellerna sker genom att anpassa dem till träningsdata, vilket möjliggör att de kan förutsäga värden när de utsätts för nya data. Det finns många olika modeller inom maskininlärning, var och en med sina egna för- och nackdelar. Valet av modell beror på flera faktorer, inklusive datans struktur och volym samt syftet med projektet

1.1 Frågeställningar

Denna rapport visar hur man kan skapa ett maskininlärningsflöde baserat på modellering av MNIST-datasättet, vilket är ett mycket välkänt och användbart datasätt för att testa klassifikationsalgoritmer. Genom att utvärdera de testade modellerna kommer vi att välja den som presterar bäst och använda den som produktionsmodell i en sifferigenkännings-app som utvecklas med Streamlit. För att uppnå detta kommer följande frågar besvaras:

1. Hur ser datastet ut och behöver man transformera om datan innan modellträningen och innan prediktionen görs i streamlit-appen?
2. Vilket mått kan användas för att utvärdera modellerna och vilka insikter får man genom de olika måtten?
3. Vilken av modeller presterar bäst utifrån de valda måtten?

1.2 Rapportens struktur

I teoridelen beskrivs de modeller som kommer att användas, samt reflektioner kring valet av modeller och utvärderingsmått. I den efterföljande metoddelen beskrivs hur jag har gått till väga för att undersöka datan, samt träna och utvärdera de olika modellerna. I avsnittet Resultat och Diskussion kommer jag att presentera och diskutera resultaten, vilket leder till slutsatsen av rapporten. Avslutningsvis besvarar jag de teoretiska frågorna och gör en utvärdering av eget arbete.

2 Teori

2.1 MNIST

MNIST är ett välkänd dataset som ofta används för att testa klassifikationsalgoritmer. Datasetet består av 70,000 bilder av handskrivna siffror från 0 till 9. Varje bild har dimensionerna 28x28 pixlar, vilket ger totalt 784 features när man transformera om bilden till en 1-dimensionell array. Pixlarna representeras med intensitet från 0 till 255 (Géron, 2019, s. 86).

2.2 Klassifikationsmodeller

Inom klassificeringsproblem skiljer man mellan binära och multiclass-klassifikationsproblem. I modelleringen av MNIST-datasetet i detta projekt behandlar vi ett multiklass-problem med targetvariabler från 0 till 9, vilket resulterar i 10 olika klasser. Nedan följer en beskrivning av de modeller som jag har valt att testa. Dessa modeller är av typen supervised learning, vilket innebär att de har en targetvariabel, till skillnad från unsupervised learning modellerna.

2.2.1 Random Forrest Classifier

Random Forest är en Ensemble-metod, vilket innebär att den består av flera modeller, i detta fall av en grupp Decision Trees, som är en annan typ av supervised learning model. Decision Trees kan användas för både klassificerings- och regressionsproblem och fungerar genom att dela in instanserna i en trädliknande struktur baserat på regler som skapas utifrån features i datasetet. Det är en enkel och intuitiv modell, vilket gör den lätt att förstå och använda. Random Forest fungerar genom att samla in förutsägelser från de enskilda Decision Trees och sedan välja den förutsägelse som får flest 'röster'.

2.2.1.1 Hyperparametrar

Hyperparametrar för en Random Forest modell inkluderar bland annat att bestämma antalet träd i skogen (n_estimators), där 100 träd är standardvärdet. En annan hyperparameter är max_depth som styr hur ’djupt’, eller hur stor trädet ska växa. Ju större träd, desto mera komplexitet i modellen. Här är standardvärdet *”none”* (scikit-learn.org, 2024*)*.

2.2.2 KNN-Classifier

En annan modell som tränas är k-nearest neighbors (kNN), som också kan användas både för klassificerings- och regressionsproblem. kNN fungerar som ett 'röstningssystem'. När man gör en prediktion undersöker man de k närmaste datapunkterna och klassificerar utifrån dem. Detta bygger på antagandet att datapunkter som är nära varandra också tillhör samma klass. Eftersom modellen använder avstånd mellan punkterna kan det ofta vara fördelaktigt att skala datan.

2.2.2.1 Hyperparametrar

Vid användning av modellen kan man specificera värdet på k (antalet grannar) som modellen ska undersöka, där standardvärdet är 5. Att välja rätt värde för k kan vara utmanande och kräver ofta testning. En av modellens andra hyperparametrar är 'weights', som kan ställas in på 'uniform' eller 'distance'. Om man väljer 'distance' väger avstånden mellan punkterna tyngre i röstningen, medan 'uniform' ger samma vikt åt varje av de k närmaste grannarna i röstningen (scikit-learn.org, 2024**)**

2.2.3 Support Vektor Machines

Den tredje modellen som används är en Support Vector Machine (SVM). Det är en flexibel och populär modell som används både för regressions- och klassifikationsproblem, och den passar bra för modellering av små och medelstora dataset. Modellen fungerar genom att skapa ett ”hyperplan” mellan de olika instanserna i datasetet och på så sätt klassificera instanserna i olika klasser. Området kring planet, kallat en 'street', fungerar som en gränsregion mellan linjerna. De datapunkter som finns i detta område kring linjen kallas för Support Vectors och det är dessa som modellen använder i beräkningarna. SVM-modeller är känsliga för skalan på features, och det rekommenderas i de flesta fall att transformera datasetets features, tex. med StandardScalar, för att uppnå bästa resultat (Géron, 2019, s. 153ff).

SVM är egentligen en binär klassificerare, men den kan konfigureras för att hantera multiklassklassificeringar. Det kan göras genom OvO (One versus One) eller OvR (One versus the Rest) klassificerare. För SVM är det rekommenderat att använda OvO, vilket också är standardinställningen om man använder modellen för multiklassproblem. OvO innebär att man jämför en siffra åt gången med en annan siffra, till exempel jämför man 0 och 1, 0 och 2, 0 och 3, osv (Prgomet, 2024).

2.2.3.1 Hyperparametrar

Några av de mest använda hyperparametrarna för SVM är C och Gamma. C styr bredden på gränsen kring hyperplanet, med andra ord, hur många support vectors som tillåts av modellen. Ju högre C-värde, ju smalare bredd och desto färre support vectors tillåts. Det kan vara fördelaktigt att minska värdet på C om vi märker att modellen överanpassar sig till träningsdatan (Géron, 2019, 2. 155). Gamma styr hur mycket de individuella datapunkterna påverkar, ”drar i”, linjerna. Ett lågt gamma-värde ger mindre dragkraft än ett högt gamma-värde. Om vi upplever underanpassning kan gamma ökas, medan det kan sänkas om det är för lite regularisering och modellen överanpassar (Géron, 2019, s. 160ff).

2.2.4 Voting Classifier

Den sista modellen som används är också en Ensemble-modell, vilket även gäller för Random Forest-klassificeraren. Båda Ensemble-metoderna bygger på principen om 'Wisdom of the crowds', där tanken är att man uppnår bättre prediktioner genom att använda flera olika prediktionsmodeller, oavsett deras individuella kvalitet. Detta kan leda till mer precisa prediktioner än vad en enskild modell kan uppnå. En bra Voting-modell som inkluderar flera modeller med lite lägre prediktionsförmåga, kan ändå producera mer precisa prediktioner än en enskild högpresterande modell. En generell tumregel är att använda modeller med olika styrkor och svagheter för att komplettera varandra och därigenom skapa mer robusta prediktioner (Géron, 2019, s. 191).

2.2.4.1 Hyperparametrar

I en Voting Classifier har vi möjlighet att använda soft eller hard voting. Vid soft voting beaktas sannolikheterna som de enskilda modellerna ger, medan hard voting enbart baseras på den klass som varje modell förutsäger. Det kan vara fördelaktigt att använda soft voting eftersom det ger mer vikt åt prediktioner med högre sannolikhet. En förutsättning för att använda soft voting är dock att alla modeller i ensemblet kan beräkna sannolikheterna för klasserna (Géron, 2018, s. 192). För SVM-modellen krävs att hyperparametern 'probability' sätts till True. Voting Classifier har också en

hyperparameter för vikter (weights), där man kan ange hur mycket varje modells sannolikheter ska väga in i den sammanslagna sannolikheten.

2.3 GridsearchCV

GridSearchCV används för att testa modellens hyperparametrar och hitta de kombinationer som ger bäst resultat. Genom att anpassa modellens hyperparametrar kan dess prediktioner förbättras avsevärt. Hyperparametrar kan också motverka överanpassning genom att öka regulariseringen, eller tvärtom motverka underanpassning genom att sänka regulariseringen.

Att hitta de optimala hyperparametrarna kräver oftast en process av testning och utvärdering av olika kombinationer. GridSearchCV underlättar denna process avsevärt genom att automatiskt testa modellen med olika parametrar och utvärdera deras prestanda med hjälp av cross-validation.

2.4 Utvärderingsmått

Det finns flera mått som kan användas för att utvärdera en klassifikationsmodell, och vilka som ska användas beror på syftet med modellen. Ett populärt mått är accuracy score, som visar hur väl modellen predikterar targetvariablerna. Det är andelen av korrekta prediktioner delad med det totale antal prediktioner. Valet av utvärderingsmått beror på syftet med modellens prediktioner samt datans struktur. Det kan vara en nackdel att använda accuracy score på obalanserade dataset. Om exempelvis endast 10% av datan representeras av kvinnor, och modellen ska förutsäga kön, kan modellen gissa "man" varje gång och ändå få en score på 90%. MNIST-datasetet är balanserat, vilket gör att det går att använda accuracy score som utvärderingsmått i bedömning av modellerna.

Andra användbara mått är Precision och Recall. I vissa klassifikationsproblem är det viktigt med hög precision, medan det i andra situationer är viktigare med hög recall. Precision mäter ”*Andelen av de positiva prediktionerna som faktiskt är korrekta*” (Prgomet, 2024). Och recall mäter ”*Andelen av den positiva klassen som vi predikterar korrek*t” (Prgomet, 2024). Utifrån dessa två mått kan F1-score beräknas, där en hög F1-score är önskvärd, då det motsvarar en hög Recall och hög Precision.

En annan användbar utvärderingsmetod är att skapa en confusion matrix, som visualiserar modellens prediktioner jämfört med de sanna y-värden. En confusion matrix kan ge värdefulla insikter om modellens styrkor och svagheter, vilket vi kommer att se senare i rapporten.

3 Metod

3.1 Data

MNIST-datasetet finns i flera versioner och är en del av ett större dataset som kallas för NIST. För detta projekt har jag hämtat datasetet från OpenML. Härefter har jag separerat datan i X (features) och y (target), och kontrollerat att det finns 70 000 instanser och 784 features. Dessa delades sedan upp i ett träningsset, ett valideringsset och ett testset. Tanken är att träna modellerna på träningssetet, validera dem på valideringssetet och slutligen träna den valda modellen på både tränings- och valideringssetet för att sedan utvärdera dess prestanda på testsetet. Målet med denna process är att minska överanpassning på träningsdatan, så jag alltid kan validera modellerna på ”ny” data. Innan den slutgiltiga modellen ska användas i appen, kommer jag att träna modellen på hela datasetet.

3.2 Exploratory Data An alysis

Syftet med explorativ dataanalys (EDA) är att få insikter om datan och avgöra om, och hur, vi behöver strukturera och förbereda datan för våra modeller. Det kan innebära att hitta nollvärden, skala datan, bearbeta kategoriska data och få grundläggande förståelse för hur datasetet är strukturerat. Vi vet redan en del om hur MNIST är strukturerat, vilket underlättar analysarbetet.

Nedan ser vi de första 6 siffrorna i vår X_train. Redan här märker vi att det kan vara svårt att avgöra siffrorna även för oss människor. Siffran tredje från vänster ser mer ut som ett frågetecken än som en sjua.

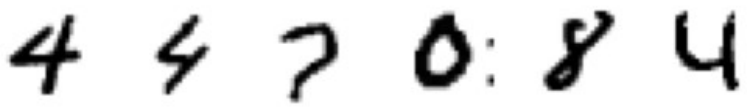

Bild 1: bild på de första 6 siffrorna i y_train (4, 4, 7, 0, 8, 4)

Vi ser att siffrorna är svartvita, men om vi kollar på själva pixel-värdarna för en siffra ser vi att den består mestadels av nollor. Pixelskalan går från 0 till 255, där 0 representerar svart och 255 representerar vitt. När vi observerar alla nollvärden i pixlarna kan vi konstatera att bilderna är inverterade, där 0 motsvarar vitt och 255 motsvarar svart. Detta kommer att ha betydelse för hur vi transformerar bilderna i vår app.

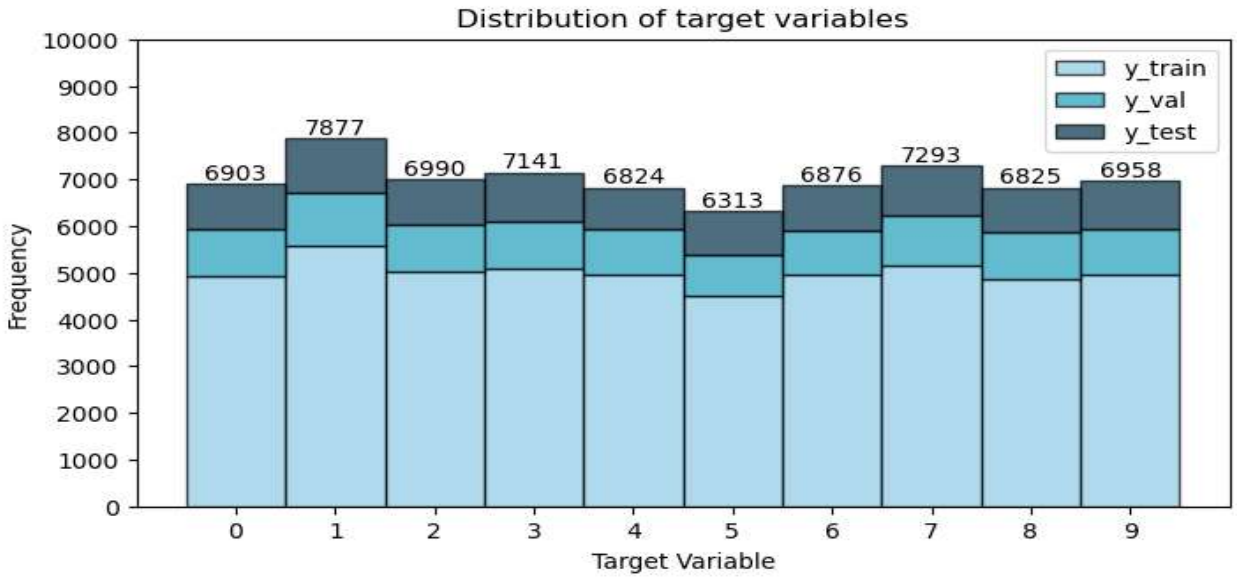

Bild 2: Distribution av target variablens värden

Ovan ser vi fördelningen av siffrorna för targetvariabeln. Vi kan konstatera att fördelningen av siffrorna i y_train verkar vara relativt lika jämfört med hela y. Därmed kan vi anta att y_train är en bra representation av hela y. Vi kan också observera att det finns färre femmor och flest ettor. Detta kan potentiellt skapa utmaningar vid prediktionen av femmor, vilket kan bli tydligt när vi utvärderar modellerna.

3.3 Data Preproccessing

MNIST-datasetet som används är mycket tacksamt att arbeta med. Bilderna är redan standardiserade till samma storlek och är centrerade. Det är då ett bra dataset som kan användas direkt utan att behöva bearbeta eller formatera datan ytterligare. Alla värden är numeriska och det finns inga nollvärden (OpenML, 2014).

Det kan vara en fördel att använda dimensionsreducering om man planerar att använda en SVM-modell, eftersom det annars kan ta lång tid att träna modellen (Géron, 2019, s. 224). Att det var en fördel med dimensionsreducering med en SVM-modell, kan jag bekräfta. Det tog avsevärd längre tid utan dimensions reducering. Det kan likaså vara fördelaktigt att skala datasetet. Av den anledningen har jag skapat en pipeline som reducerar och skalar träningsdatan, när jag testar SVM-modellen, samt skalar datan när jag testar kNN-modellen. Det kommer därför inte finnas ett separat avsnitt i koden med Data Processing, utan datan transformeras genom pipelines när modellerna testas.

3.4 Model Training

Träningsavsnittet börjar med träning av en Random Forest-modell med standardvärden för hyperparametrarna, följt av en Random Forest-modell med Gridsearch där jag testar olika värden på n_estimators och max_depth. Målet var att se hur stor effekt denna Gridsearch hade på modellens prestanda.

Jag utförde samma procedur för KNeighbors-klassificeraren. En med standardhyperparametrar och en med Gridsearch. Jag testade också en pipeline med en kNN-modell där jag använde StandardScaler innan modellen kördes. I vissa fall kan det vara fördelaktigt att skala x-värdena innan man kör kNN, så jag ville undersöka om det var fallet.

I modellerna med GridSearch körde jag cross-validation med 3 Folds och fick ut en bra accuracy score för modellerna, över 90%.

Härnäst testade jag en pipeline med StandardScaler och PCA-dimensionsreducering med en SVM-modell. Jag satte probability=True för att kunna använda modellen i den sista modellen, som är en Voting Classifier med Soft Voting. Voting-klassificeraren innehåller tre modeller: en Random Forest-modell, en kNN-modell (båda med standardvärden på hyperparametrarna) och pipeline-SVM-modellen med StandardScaler och PCA. Alla modellerna sparades med joblib så att de enkelt kunde åtkommas senare

3.5 Model Evaluation

Först körs en for-loop som beräknar accuracy score för alla modeller och genererar en tabell över dessa. Därefter definieras en funktion som genererar classification reports och confusion matrix för varje modell. Jag använder denna funktion på de tre modeller med högst accuracy score samt på den modell med lägst accuracy score för att undersöka skillnaderna i deras prediktionsförmåga.

3.6 Final Model

I den sista del av koden tränar jag om den valda modellen, som i detta fall blev Voting-klassificeraren, på X_train_val-datasetet. Sedan testar jag denna modell på X_test-datasetet för att slutligen beräkna accuracy score för modellens prediktioner. Om det inte finns tecken på överanpassning eller underanpassning, då tränar jag om modellen på hela datasetet, innan den sparas i joblib för att kunna användas i streamlit-appen.

3.7 Streamlit App

Det viktigaste i Streamlit-appen är hur bilden bearbetas innan modellen ska göra sin prediktion av siffran. Det är av yttersta vikt att den array som modellen ska förutsäga från, är av samma typ som den data som modellen har tränats på.

När en bild har laddats upp i appen, visas vilka steg som bilden genomgår innan den matas in i modellen. Först visas den uppladdade bilden, sedan visas den omvandlad till gråskala, eftersom vi endast är intresserade av nyanser i denna skala. Därefter inverteras bilden så det svarta blir vitt och det vita blir svart. Detta görs eftersom vi i EDAen konstaterade att pixlarna var inverterade, så att 0 motsvarade vitt. I den inverterade bilden är siffran nu vit och bakgrunden svart, vilket kan vara missvisande, men i array bakom blir det rätt. Därefter formateras bilden till en storlek på 28x28 pixlar. Slutligen görs bilden binär genom att ett tröskelvärde sätts för när pixlarna ska vara vita och när de ska vara svarta. Slutligen formateras pixelarrayen till en 1-dimensionell array. När bilden har bearbetats kan vi klicka på knappen "Prediction" för att få vår prediktion. Se screen shot på nästa sida där de olika bearbetningssteg och prediktionen syns.

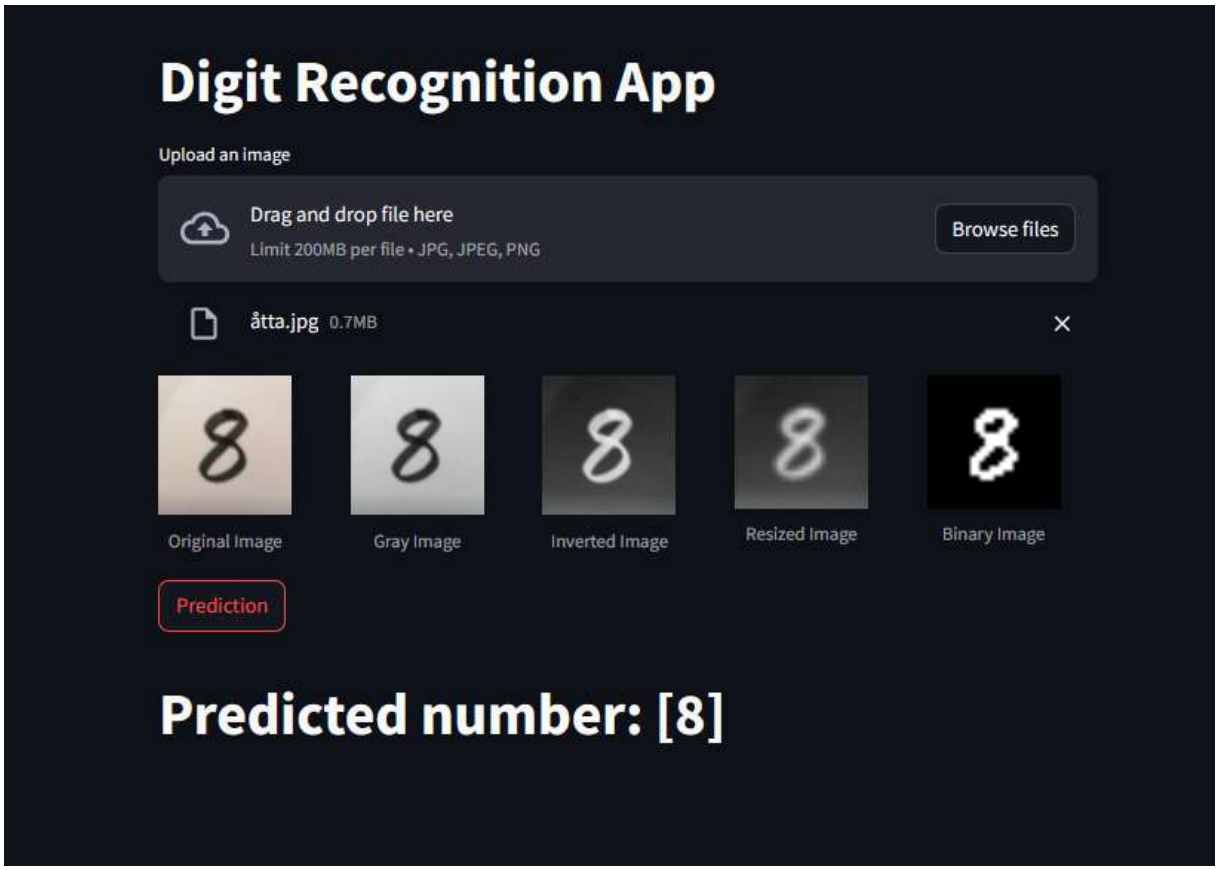

Bild 7: Screen shot på Streamlit app

4 Resultat och Diskussion

4.1 Val av modeller

I valet av modeller hade jag redan från början klart för mig, att jag ville testa hur en Voting-klassificerare presterade jämfört med enskilda bra klassificerare. Mot denna bakgrund valde jag tre modeller med ganska olika beräkningsalgoritmer som förhoppningsvis kunde komplettera varandra väl. Eftersom problemet i sig är relativt komplext, med 10 klasser, valde jag att testa Random Forest-klassificeraren i stället för ett enskilt Decision Tree. Denna ingick i ett ensemble med en kNN-modell och en SVM-modell, vilka båda är populära och kraftfulla modeller. Det visade sig att Voting-klassificeraren var den som presterade bäst, vilket beskrivs i följande avsnitt.

4.2 Modellutvärdering

I tabellen nedan visas accuracy scoren för de olika modellerna. Samtliga modeller presterade relativt bra, och endast en av dem låg under 95% accuracy score. Det är dock viktigt att notera att detta resultat är baserat på valideringsdatasetet och inte på testdatasetet, men hittills ser resultaten lovande ut.

Som förväntat visade Gridsearch en positiv effekt både för Random Forest- och kNN-modellerna, även om skillnaderna var små. Om vi hade utforskat fler kombinationer av hyperparametrar hade skillnaderna kanske varit större.

Voting-klassificeraren presterade också mycket bra, vilket var förväntat då den drar nytta av de andra modellernas samlade kunskap. Jag testade modellen med en Random Forest och kNN med standardvärden på hyperparametrarna, och en Pipeline med SVM. Det hade varit intressant att testa med de hyperparametrar som presterade bäst när vi använde Gridsearch för Random Forest- och kNN-modellerna. Trots det är jag nöjd med resultatet från Voting-klassificeraren, och det är den modellen som kommer att användas som den slutgiltiga modellen.

Den modell som presterade sämst var en pipeline med skalade X-värden och en kNN-modell. Jag ville testa detta eftersom skalning av features ibland kan förbättra prediktionerna för kNN-modeller, men det visade sig inte vara fallet här. Det kan vara fördelaktigt att skala datan om man har många features med stora utsvängningar i de numeriska värdena. Detta är dock inte fallet för MNIST-datasetet, då X-värdena har en relation till varandra. Med relation menas att de är på samma färgskala (0-255) och därmed har ett visst inbördes förhållande, vilket kan vara orsaken.

Name	Score
Voting Classifier	0.9782
KNeighborsClassifier with Gridsearch	0.9731
RandomforrestClassifier with Gridsearch	0.9720
KNeighborsClassifier	0.9702
RandomforrestClassifier	0.9691
Pipeline with SVM	0.9645
Pipeline with KneighborsClassifier	0.9431

Bild 3: accuracy score for alla modellerna

4.3 Classification Report och Confusion Matrix

Om vi ser på klassificeringsrapporten för Voting-klassificeraren, kan vi observera att både precision, recall och F1-score är relativt höga, alla över 95%. De lägsta siffrorna, på 96%, är för recall för siffrorna 3 och 8. Recall representerar andelen korrekta positiva förutsägelser i förhållande till det totala antalet faktiska positiva fall (i detta fall antalet 3:or och 8:or). Detta innebär att dessa siffror i större utsträckning förväxlas med andra siffror, eventuellt med varandra.

De siffror som har högst F1-score är 0 och 6, med både precision och recall uppe på 99%, vilket även gäller för både precision och recall-värdena. Detta indikerar att modellen är mycket bra på att korrekt identifiera dessa siffror.

Classification report

	precision	recall	f1-score	support
0	0.99	0.99	0.99	997
1	0.98	0.99	0.98	1158
2	0.98	0.97	0.98	1007
3	0.98	0.96	0.97	1028
4	0.98	0.98	0.98	966
5	0.97	0.98	0.98	885
6	0.99	0.99	0.99	945
7	0.97	0.98	0.97	1070
8	0.98	0.96	0.97	988
9	0.97	0.97	0.97	956
accuracy			0.98	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.98	0.98	0.98	10000

Bild 4. Classification report för Voting Classifier på prediktionerna på valideringsdatasetet

I bild 5 visas en confusion matrix för Voting-klassificeraren, med de faktiska värdena längs y-axeln och de förutsagda värdena längs x-axeln. Här kan vi observera vilka siffror som kan vara problematiska, och det verkar särskilt som att 9:or ofta ser som 7:or. Som tidigare nämnts syns det också att det är färre 5:or, och att modellen ofta tror att det är en 5:or när det är en 3:or eller en 8:or. Vi noterar att både 8:or och 3:or ofta förväxlas med andra siffror (3:or med 5, 7 och 8, och 8:or med flera av siffrorna), vilket resulterar i en lägre recall, vilket vi såg i klassifikationsrapporten.

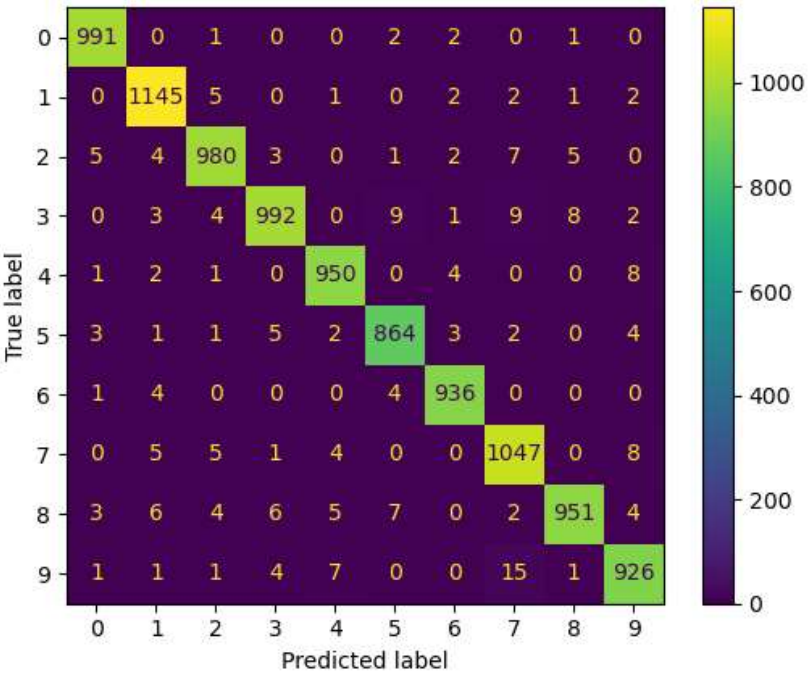

Bild 5: Confusion Matrix för Voting Classifier på prediktionerna på valideringsdatasetet

Nedan ser vi en confusion matrix för modellen som presterade sämst, vilket var pipeline med kNN. Här är det tydligt hur stor skillnad det är i prediktionsförmåga mellan modellerna, även om skillnaderna är små i procentandelar. Till skillnad från Voting-modellen var denna modell dålig på att känna igen 9:oerna, som ofta blev predikterade som 4:or eller 7:or. Det syns också att det är svårt att korrekt prediktera 8:or. Hela 39 gånger förväxlades en 8:a med en 5:a.

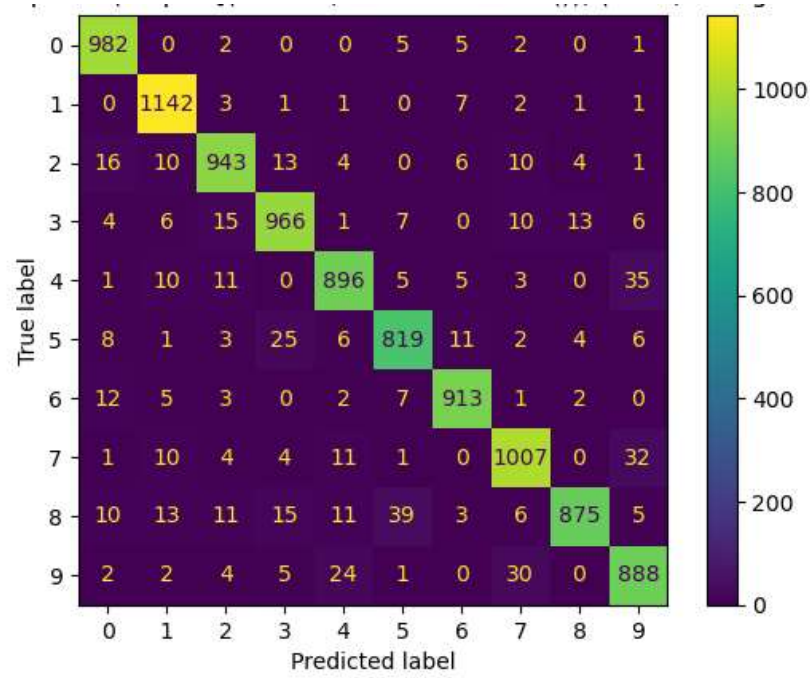

Bild 6: Confusion Matrix för Pipeline med kNN på prediktionerna på valideringsdatasetet

Generellt sett kan man säga att de flesta av modellerna hade problem med att korrekt prediktera 3:or och 8:or. Med Gridsearch och kNN hade 8:orna en recall på 94% och 3:or på 96%, medan Random Forest med Gridsearch hade en recall på 95% för 3:or och 96% för 8:or.

4.4 Slutgiltiga modellen

I det sista steget av maskininlärningsflödet tränades Voting-klassificeraren på datamängden som består av både X_train och X_val för att sedan göra förutsägelser på X_test. Efter avslutad träning och beräkning av accuracy score på test-setet fick vi en poäng på 0.9772, vilket endast är något under resultatet vi fick när vi validerade modellen på valideringsdatamängden. Det tyder varken på överanpassning eller underanpassning, och modellen är därför acceptabel för användning i appen.

I testningen av appen använde jag flera bilder för att säkerställa att bilderna blev inlästa korrekt och att modellen gav korrekta prediktioner. Om man skulle vidareutveckla appen kunde man genomföra ytterligare bildbehandling, såsom att centrera bilderna och reducera brus. Det skulle behövas om appen skulle kunna hantera mer "stökiga" bilder än de jag testade med.

Slutsatser

Vi kan nu återvända till de frågor som jag ställde i början av rapporten och se om de går att besvara. Frågeställningarna var följande.

4.5 Frågeställningar

1. Hur ser data ut och behöver man transformera datan innan modellträningen och innan prediktionen görs i streamlit-appen?

Det behövdes inte många transformationssteg av datasetet innan vi kunde köra modellerna, faktiskt ingen. Det enda som behövde göras var att dela upp datan i ett tränings-, validerings- och testset, och sedan kunde man börja testa. Det är långt ifrån verkligheten inom machine learning, där en betydande del av arbetet innebär att transformera datan innan modellerna kan tränas. De transformationer som gjordes blev gjort genom pipelines, vilket visade sig vara särskilt effektivt för SVM-modellen, där träningstiden sjönk avsevärd efter en dimensionsreduktion. De transformationer som inte var nödvändiga för modellträning behövde dock utföras för bilderna som skulle laddas upp i Streamlit-appen. Här krävdes en del bearbetning av bilden för att anpassa den till modellen.

2. Vilket mått kan användas för att utvärdera modellerna och vilka insikter får man genom de olika måtten?

Det finns flera mått som kan användas för att utvärdera klassifikationsproblem. Jag använde främst accuracy score eftersom detta mått passade bra till datasetets struktur och för att utvärdera modellens prestation i förhållande till dess syfte; att ha så många korrekta prediktioner som möjligt. Genom att även granska Precision och Recall, samt undersöka confusion matrix för modellerna, kunde vi identifiera vilka siffror modellerna hade svårast att känna igen (speciellt 8:or visade sig vara svåra att prediktera). Dessutom kunde vi tydligt se var de enskilda modellerna hade svårigheter.

3. Vilken av modeller presterar bäst utifrån de valda måtten?

Den slutgiltiga modellen, som blev en Voting klassificerare, landade på en accuracy score på testsetet på 97.72%. Det är en hög score och visar på styrkan i ensemblemodellerna. Både Random Forest- och kNN-modellen presterade också bättre när vi tränade dem med GridSearchCV, och hittade de bästa hyperparametrarna inom de parametrar som jag hade valt att testa.

4.6 Reflektioner

Avslutningsvis kan vi konstatera att våra modeller presterade tillräckligt bra för deras syfte, vilket var att användas i en sifferigenkänningsapp. Det finns flera åtgärder som möjligen kunde ha förbättrat den slutgiltiga modellens prediktioner. Till exempel kunde vi ha inkluderat fler klassificerare i Voting-klassificeraren. Jag hade också kunnat testa fler hyperparametrar för modellerna och sedan använt de bästa i de modeller som ingick i Voting-klassificeraren. Det hade också varit intressant att testa GridSearch på en Pipeline med SVM-modellen, men det tog ganska lång tid att köra denna, även med dimensionsreducering.

Det hade varit användbart med en funktion i appen som returnerade en topplista över de tio mest sannolika siffrorna och deras respektive sannolikheter. Voting-klassificeraren använder soft voting, och funktionen predict_proba() skulle därför kunna användas för detta ändamål. På så sätt kunde användaren få en uppfattning om hur säker modellen var på sin förutsägelse.

Appendix

Kod från Streamlitappen

```
# Streamlit app
def main():
    st.title("Digit Recognition App")

    # File uploader widget
    uploaded_image = st.file_uploader("Upload an image", type=["jpg", "jpeg",
"png"])
    if uploaded_image is not None:
        # Read the uploaded image
        image = np.array(Image.open(uploaded_image))
        # Convert the resized image to grayscale
        image_gray = rgb2gray(image)
        # invert image
        image_invert = invert(image_gray)
        # Resize the image to 28x28 pixels
        image_resized = resize(image_invert, (28, 28))
        # scale pixels to 0-255
        scaled_image = (image_resized * 255).astype(np.uint8)
        # create a binary image
        binary_image = np.where(scaled_image < 125, 0, 255)
        # Flatten the image into a 1D array
        img_array = binary_image.flatten()

        # Create a layout for displaying images horizontally
        col1, col2, col3, col4, col5 = st.columns(5)
        with col1:
            st.image(image, caption='Original Image', width=100)
        with col2:
            st.image(image_gray, caption='Gray Image', width=100)
        with col3:
            st.image(image_invert, caption='Inverted Image', width=100)
        with col4:
            st.image(image_resized, caption='Resized Image', width=100)
        with col5:
            st.image(binary_image, caption='Binary Image', width=100)

        # Prediction
        if st.button('Prediction'):
            model = joblib.load("model_final.pkl")
            prediction = model.predict(img_array.reshape(1, -1))
            st.markdown("# Predicted number: " + str(prediction))

if __name__ == "__main__":
    main()
```

Källförteckning

DataCamp (2024). K-Nearest Neighbors (KNN) Classification with scikit-learn. Hämtad 19. Mars 2024 från https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn

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

Prgomet, A. (2024). 02_klassificering. Hämtad 19. Mars 2024, från https://github.com/AntonioPrgomet/maskininlaerning

OpenML (2014). mnist_784. Hämtad 19. Mars 2024, från https://openml.org/search?type=data&status=active&id=554

Scikit-learn (2007-2024). Sklearn.ensemble.RandomForrestClassifier. Hämtad 19. Mars 2024, från https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Scikit-learn (2007-2024). Sklearn.neighbors.KNeighborsClassifier. Hämtad 19. Mars 2024, från https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html