

Att finjustera eller inte?

En jämförelse av optimerade och
ooptimerade ML-modeller



Peter Lantz
EC Utbildning
Maskininlärning
202503

Abstract

Even though many machine learning models can be optimized by fine-tuning their hyperparameters, the question remains whether the extra accuracy gained justifies the time spent on optimization.

To address this, I trained some models with hyperparameter tuning and compared them with models that were not fine-tuned.

The aim of this work was to compare the performance of models with and without hyperparameter tuning, to assess whether the additional effort in optimization justifies the improvement in accuracy.

The result showed only a slight difference in performance between the two, but the time spent on tuning the hyperparameters could take hours of training.

My conclusion is that tuning may not be worth the extra time if the goal is to quickly train a good model to solve everyday problems. However, if you are willing to invest additional time for a marginal improvement in accuracy, it may be worth it.

Innehåll

1	Inledning	1
1.1	Bakgrund	1
1.2	Syfte	1
2	Teori.....	2
2.1	Grundläggande begrepp	2
2.1.1	Dela upp data i träning och test	2
2.1.2	EDA	2
2.1.3	Estimerare.....	2
2.1.4	Transformatorer	2
2.1.5	Predikterare	2
2.2	Datapreparering.....	2
2.2.1	Varför behövs datapreparering?	2
2.2.2	StandardScaler.....	2
2.3	Klassificeringsmodeller	3
2.3.1	SVC.....	3
2.3.2	LogisticRegression	3
2.3.3	SGDClassifier	3
2.3.4	DecisionTree	4
2.3.5	RandomForest	4
2.3.6	ExtraTree.....	5
2.4	Ensemble metoder.....	5
2.4.1	Voting classifiers	5
2.5	Modellutvärdering och hyperparameteroptimering	6
2.5.1	Cross Validation	6
2.5.2	GridSearchCV	6
2.5.3	Mått och Score.....	6
2.5.4	Evaluering	7
3	Metod	8
3.1	Importering av bibliotek	8
3.2	Hämtning och inläsning av dataset.....	8
3.3	Databeskrivning	8
3.4	Splittning av data	8
3.5	EDA.....	8
3.5.1	Datatyp	8

3.5.2	Fördelning i MNIST	9
3.5.3	Urval av MNIST-bilder.....	9
3.6	Bearbetning av data.....	9
3.7	Modeller.....	9
3.7.1	Val av modell	9
3.7.2	Träning av modeller.....	10
3.7.3	Utvärdering av modeller.....	10
3.7.4	Utvärdering av modellens generalisering.....	10
3.7.5	Spara och ladda modeller.....	10
3.8	Streamlit-applikation	10
4	Resultat och Diskussion.....	11
4.1	Resultat	11
4.1.1	Första utvärderingen	11
4.1.2	Andra utvärderingen	11
4.1.3	Tredje utvärderingen.....	12
4.1.4	Val av modeller att utvärdera med valideringsdata	13
4.1.5	Modeller utan hyperparameteroptimering.....	14
4.1.6	Sammanställning av validerade modeller	15
4.1.7	Träning och test av Voting Classifier.....	15
4.1.8	Storlek på modeller	16
4.1.9	Streamlit	16
4.2	Diskussion	17
4.2.1	Optimering mot cooptimering	17
4.2.2	Analys av ensemble-modeller	17
4.2.3	Validering av modeller.....	18
4.2.4	Analys av den valda modellen - Voting Classifier	18
4.2.5	Storleken har betydelse.....	18
5	Slutsatser	19
5.1	Vidare undersökning	19
6	Teoretiska frågor	20
7	Självutvärdering.....	23
Appendix A	24
Källförteckning.	25

1 Inledning

1.1 Bakgrund

Behovet av att lösa ett problem kan ge upphov till att vilja använda maskininlärning. Vi hoppas genom att skapa en eller flera modeller kunna lösa allt från vardagliga problem till mer komplexa sådana, inom diverse olika områden. Ett sätt att göra det är genom att träna en eller flera modeller på den data vi har tillgänglig för att den i slutändan skall kunna prediktera önskat resultat vilket slutligen skall lösa vårt problem.

När det gäller maskininlärningsmodeller finns det ett stort antal olika modeller, och alla presterar olika bra för olika ändamål. För att förbättra en modells prestation kan man finjustera den med olika hyperparametrar. Det här medför både att det kan vara svårt att hitta rätt hyperparametrar och att det är resurskrävande.

1.2 Syfte

Syftet med denna rapport är därför att undersöka hur en modell utan anpassade hyperparametrar presterar jämfört med en modell som har optimerade hyperparametrar, samt att utvärdera om det är värt tiden det tar att anpassa en eller flera modeller i relation till deras prestanda. För att uppfylla syftet kommer följande frågeställningar att besvaras:

1. Hur presterar en modell som inte har anpassade hyperparametrar jämfört med en modell som har det?
2. Är det värt tiden det tar att anpassa en eller flera modeller i relation till dess prestanda?

2 Teori

2.1 Grundläggande begrepp

2.1.1 Dela upp data i träning och test

För att kunna träna och utvärdera en modell innan den tas i produktion delas datan vanligtvis upp i en träningsdel och en testdel, och ibland även en valideringsdel. Detta gör det möjligt att träna modellen på en del av datan och sedan utvärdera dess generaliseringsförmåga genom att låta den göra prediktioner på testdatan (Géron, 2023, s. 34).

2.1.2 EDA

När vi delat upp vår data så vi vill få en förståelse för datan. Vi kanske redan nu kan hitta samband mellan olika variabler. Vi kanske ser att den innehåller felaktigheter som saknade värden som måste hanteras. Vi kanske inte behöver all data eller har kategoriska data som måste hanteras. För att komma till dessa insikter så behöver vi se över vår data och där det lämpar sig även visualisera detta i grafer för att få en enklare och bättre förståelse (Géron, 2023, s. 51).

2.1.3 Estimerare

En estimerare är ett objekt som har tränats på data och lärt sig att uppskatta parametrar baserat på den informationen (Géron, 2023, s. 70).

2.1.4 Transformatörer

Vissa estimerare kan också transformera data, och de kallas för transformatörer. Ett exempel på en sådan är SimpleImputer (Géron, 2023, s. 70).

2.1.5 Predikterare

Vissa estimatörer har också möjlighet att prediktera på nya data, och dessa kallas för predikterare (Géron, 2023, s. 70).

2.2 Datapreparering

2.2.1 Varför behövs datapreparering?

För att modellerna ska kunna göra bättre estimeringar och i slutändan bättre prediktioner, behöver vi ibland transformera datan innan träning. För vissa modeller är detta till och med nödvändigt för att de ska fungera optimalt (Géron, 2023, s. 67).

2.2.2 StandardScaler

Med StandardScaler justeras skalan så att vi får ett medelvärde 0 och standardavvikelse 1. Vilket gör variablerna mer jämförbara och kan förbättra modellens prestanda och stabilitet. Det kan också snabba upp träningen för algoritmer som använder gradientbaserad optimering (Géron, 2023, s. 75).

$$Z = \frac{X - \mu}{\sigma}$$

Man använder **fit_transform** på träningsdata för att beräkna skalningsparametrarna (medelvärde och standardavvikelse) och applicera dem, medan man endast använder **transform** på validerings- och testdata för att säkerställa att de transformeras på samma sätt som träningsdata, utan att läcka information (Géron, 2023, s. 75).

2.3 Klassificeringsmodeller

När vi vill göra någon typ av klassificering så är det för att särskilja klasser åt. Exempelvis om man är en man eller kvinna (Prgomet, 2025).

2.3.1 SVC

En support vector machine är en kraftfull modell som kan användas för både linjära och ickelinjära samband. Modellen kan användas både för regression och klassificering. Modellen är väl lämpad för små eller medelstora dataset men är inte lämplig att använda på stora dataset. Den klarar av upp till några tusen innan det börjar bli allt för mycket (Géron, 2023, s. 175).

2.3.2 LogisticRegression

Logistisk regression används normalt för att estimera sannolikheten för en instans, dvs vilken klass den tillhör. Logistisk regression är binär, dvs den sätter antingen värdet 0 eller 1 för en instans.

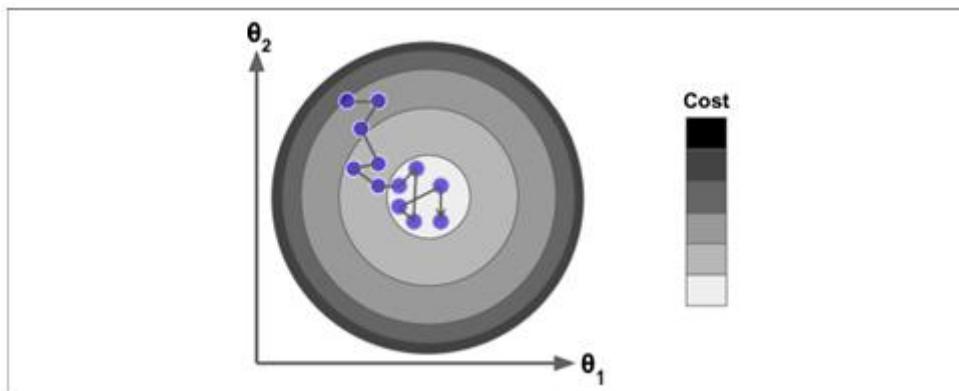
Antingen så tillhör instansen klassen och får då siffran 1 eller så tillhör den inte klassen och får då siffran 0. När modellen estimerar så tittar den på sannolikheten att en instans tillhör en klass eller ej, ofta är 50% gränsen. Är sannolikheten större så sätter den värdet till 1 och annars värdet 0 (Géron, 2023, s. 164).

Logistisk regression estimerar sannolikheten enligt:

$$\hat{p} = h_{\theta}(x) = \sigma(\theta^T x) \text{ (Géron, 2023, s. 164).}$$

2.3.3 SGDClassifier

SGD står för Stochastic gradient descent och bygger på principen om slumpmässig inlärning för vald datapunkt när den försöker hitta bästa optimeringen i stället för att den går steg för steg, dvs använder hela datasetet samtidigt. Man får en snabbare inlärning men mindre stabil.



Figur 2-1 Med SGD är varje träningssteg mycket snabbare men också betydligt mer slumpmässigt. Återgiven från Géron (2023, s. 145).

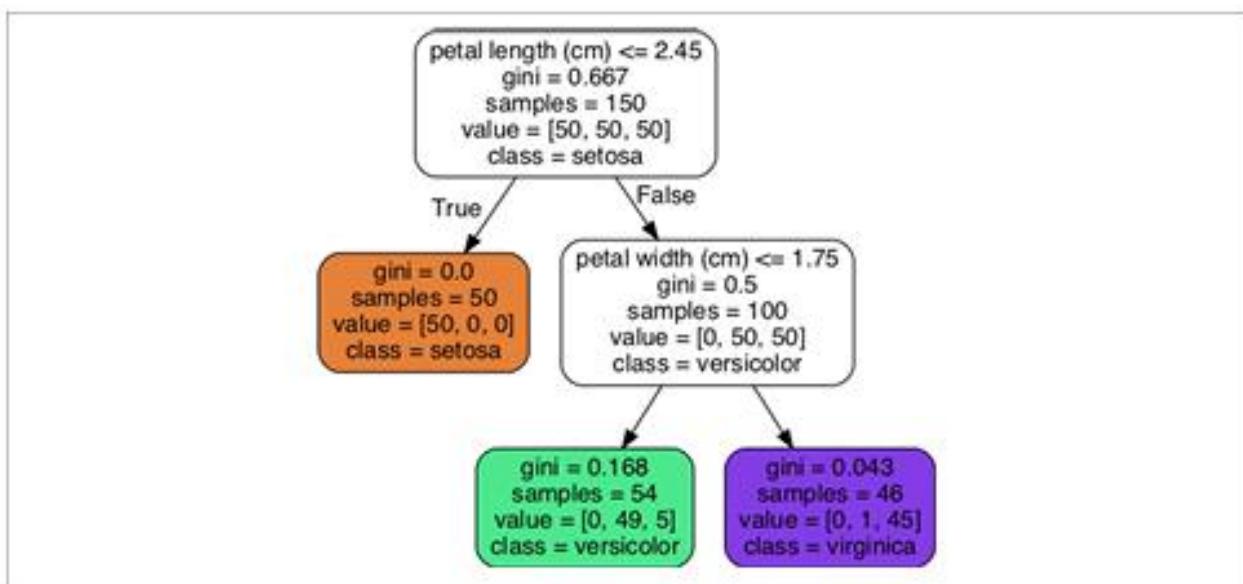
SGD Classifier används för klassificeringsproblem och lämpar sig väl för stora dataset till följd av SGD vilket gör att modellen tränar snabbare och effektivare.

2.3.4 DecisionTree

En DecisionTree modell kan användas för flera ändamål såsom regression, klassificering och även för att förutsäga flera utdata samtidigt, dvs. återge flera värden baserat på den inmatade datan. Modellen är kraftfull och kan hantera komplexa data.

Modellen är också grunden till RandomForest (Géron, 2023, s. 195).

Nedan bild visar hur beslutsträdet fördelar datan i olika grupperingar kallade noder och löv. Fördelningen görs utifrån att modellen testar om instansen uppfyller villkoret eller ej. Om den gör det så fördelas den till det vänstra lövet, True, annars om det är False så skapas en ny nod med motsvarande två löv. Modellen försöker att hålla en så hög "renlighet" som möjligt, dvs ett så lågt värde som möjligt. Det vi i bilden ser som gini (Géron, 2023, s. 196).



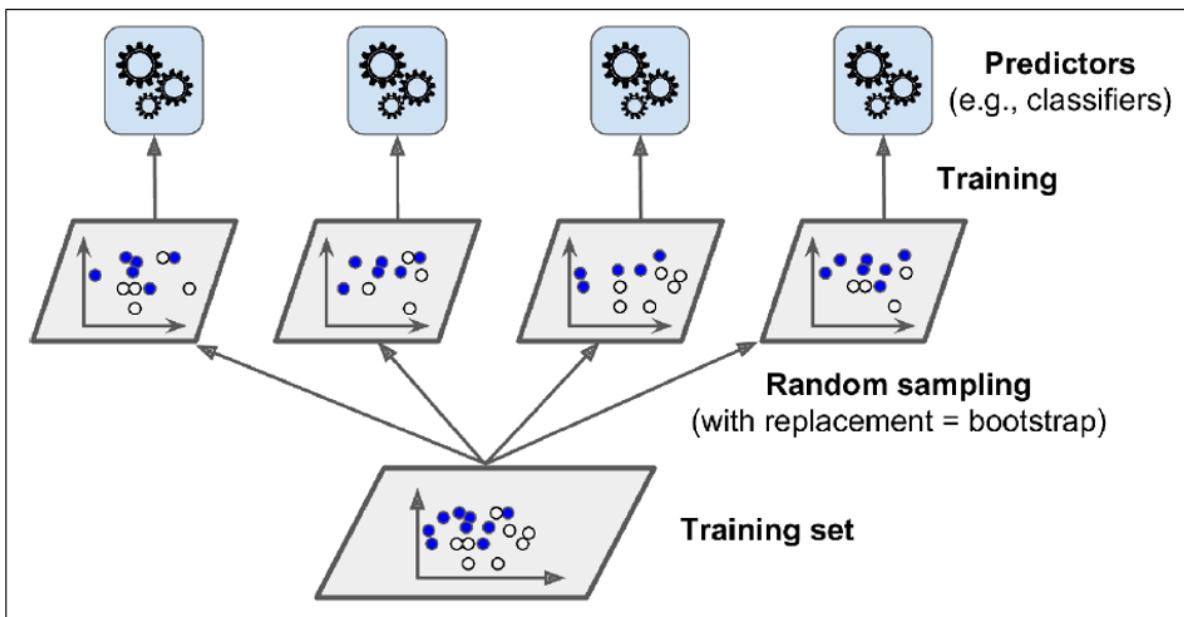
Figur 2-2 Beslutsträd. Återgiven från Géron (2023, s. 196).

2.3.5 RandomForest

Random forest utgörs av en samling av flera beslutsträd (DecisionTrees) som oftast tränas utifrån konceptet "bagging" men även ibland "pasting".

- Bagging
Innebär att modellerna tränas på slumprövade data med återläggning.
- Pasting
Innebär att modellerna tränas på slumprövade data utan återläggning.

Nedan är en bild som illustrerar hur modellerna tränas på den slumprövade datan enligt bagging och pasting (Prgomet, 2025).



Figur 2-3 Bilden illustrerar hur modeller tränas på slumpmässiga data genom bagging och pasting. Återgiven från Prgomet (2025, slide 11).

2.3.6 ExtraTree

Extra tree bygger på principen från Random Forest, att flera träd skapas. Skillnaden med Extra tree är att den inte försöker hitta den optimala fördelningen vid varje nod, utan i stället använder slumpmässigheten i funktioner (features) och uppdelningspunkter (thresholds). Detta innebär att olika features väljs slumpmässigt för att "bygga träden" och innebär att inte alla funktioner används vid varje uppdelening.

Vid noderna används sedan också slumpmässiga uppdelningspunkter, vilket skapar ännu mer slumpmässighet i lärandet. Syftet är att skapa en modell som är bättre på att generalisera och inte bli överanpassad så kallad "over fitted". Eftersom processen är mer slumpmässig tenderar modellen också att träna snabbare (Géron, 2023, s. 220).

2.4 Ensemble metoder

2.4.1 Voting classifiers

När vi tränar olika modeller var för sig så uppnår de efter träning och validering en viss score, det vill säga hur bra de är på att klassificera instanser och därefter göra prediktioner på nya data. Olika modeller presterar olika bra beroende på vilken typ av data de tränas på.

Ett sätt att förbättra modellernas prestanda är att använda Ensemble learning, vilket innebär att man kombinerar flera modellers estimeringar och prediktioner för att uppnå ett bättre resultat. Det är inte alltid detta leder till en förbättring, men ofta kan det ge mer robusta och noggranna prediktioner.

En vanlig ensemble-metod är **Voting Classifiers**, som kan delas in i **hard voting** och **soft voting**.

- **Hard voting** är en typ av majoritetsröstning, där varje modell gör en binär prediktion, det vill säga att de olika modellerna predikterar "Ja" eller "Nej" och flest antal röster vinner.

- **Soft voting** tar även hänsyn till sannolikheten för prediktionerna. Det medför att om en modell är mer säker på en viss prediktion, så får den större vikt i röstningen. Detta leder ofta till bättre prediktioner, men är inte alltid fallet.

Syftet med ensemble metoder är att uppnå bättre prediktioner men så är alltså inte alltid fallet (Prgomet, 2025).

2.5 Modellutvärdering och hyperparameteroptimering

2.5.1 Cross Validation

Cross Validation är en metod för att utvärdera en modells prestanda genom att dela upp datan i flera delar. Modellen tränas på vissa delar och testas på andra, och detta upprepas flera gånger. Resultatet blir ett genomsnittligt score från alla iterationer, vilket ger en mer pålitlig utvärdering än en enkel train/test-split (Géron, 2023, s. 90).

2.5.2 GridSearchCV

Man kan optimera en modells prestanda genom att justera dess hyperparametrar. Att göra detta manuellt genom att testa olika kombinationer kan dock vara tidskrävande. Ett smidigare sätt är att använda **GridSearchCV** från Scikit-Learn, som automatiskt testar olika hyperparametrar och utvärderar dem med hjälp av **cross-validation**. De hyperparametrar som ger bäst resultat väljs och används sedan för att träna den slutliga modellen (Géron, 2023, s. 91).

När **GridSearchCV** har genomfört optimeringen kan man hämta de bästa parametrarna, den bästa estimatorn och det bästa måttet med:

- `best_params_`
- `best_estimator_`
- `best_score_` (Géron, 2023, s. 93).

2.5.3 Mått och Score

För att utvärdera hur bra modellen presterar kan vi använda ett mått, exempelvis `cross_val_score`. Med `cross_val_score` tränas modellen på datan baserat på ett valt antal folds. För varje iteration beräknas ett genomsnittligt score. När alla iterationer är genomförda och genomsnittliga score har beräknats, tas det totala genomsnittet av alla scores, vilket ger oss ett övergripande mått på modellens prestanda (Géron, 2023, s. 107).

Genom att använda `scoring='accuracy'` anger vi att vi vill mäta modellens prestanda i form av noggrannhet, alltså andelen rätta förutsägelser (Géron, 2023, s. 107).

2.5.4 Evaluering

2.5.4.1 Accuracy_score

Accuracy mäter andelen korrekt klassificerade instanser i en modell. Det är ett vanligt mått men kan vara mindre användbart vid obalanserad data (Scikit-learn Developers, n.d.).

2.5.4.2 Confusion_Matrix

En confusion matrix är ett verktyg som används för att utvärdera prestandan hos en klassificeringsmodell. Den visar antalet korrekta och felaktiga förutsägelser genom att jämföra de faktiska etiketterna med de predikterade etiketterna. Den består av fyra huvudsakliga element:

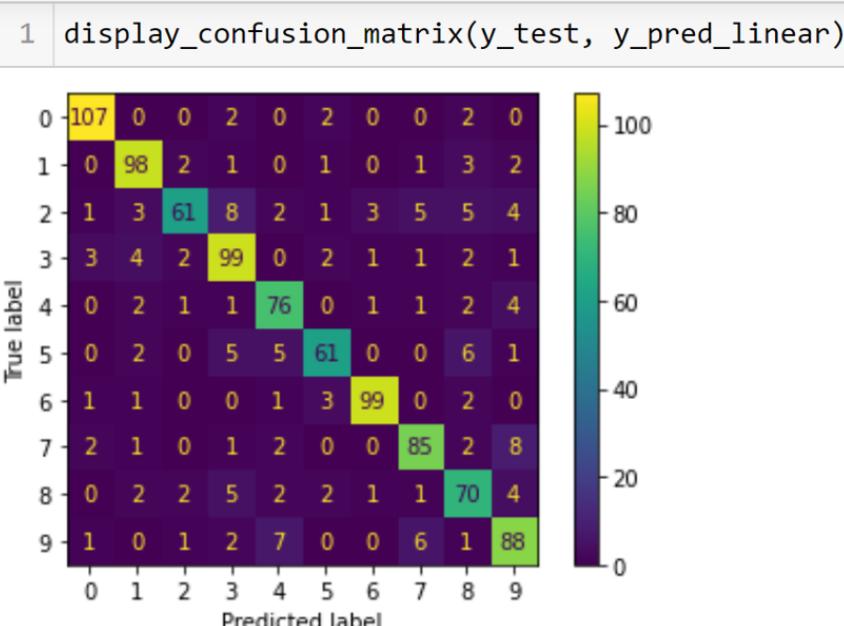
1. **True Positives (TP)**: Korrekt förutsagda positiva resultat.
2. **False Positives (FP)**: Felaktigt förutsagda positiva resultat.
3. **True Negatives (TN)**: Korrekt förutsagda negativa resultat.
4. **False Negatives (FN)**: Felaktigt förutsagda negativa resultat.

Genom dessa värden kan man beräkna olika prestandamått som noggrannhet, precision, recall och F1-score (Prgomet, 2025).

2.5.4.2.1 Confusion Matrix Display

Confusion Matrix Display är ett sätt att visualisera en confusion matrix. Den visar hur bra modellen har klassifierat instanser genom att jämföra de faktiska etiketterna med de predicerade etiketterna. Matrisen visar antalet korrekta och felaktiga klassificeringar för varje klass, vilket gör det lättare att se hur modellen presterar på de olika klasserna (Géron, 2023, s. 122).

Confusion Matrix



Figur 2-4 Confusion Matrix som visar de sanna värdena mot de predikterade värdena. Återgiven från Prgomet (2025, slide 25).

3 Metod

3.1 Importering av bibliotek

Importerar nödvändiga bibliotek för att kunna arbeta med datan, bearbeta den och visualisera den. NumPy och Pandas för datahantering, Matplotlib och Seaborn för visualisering, samt Scikit-learn för modellträning och utvärdering. Joblib används för att spara och ladda modeller, medan Streamlit möjliggör skapandet av en interaktiv applikation. PIL används för att bearbeta bilder i Streamlit-applikationen.

3.2 Hämtning och inläsning av dataset

Datasetet hämtades från OpenML med hjälp av funktionen `fetch_openml` från Scikit-learn (Scikit-learn, 2025).

3.3 Databeskrivning

Arbetet har utförts med MNIST datasetet hämtat från OpenML.

Mnist består av 70 000 handritade bilder, motsvarande siffror mellan 0 – 9. Där varje bild har 784 features som representeras av enskilda pixlar. Varje enskild bild består 28 x 28 pixlars gråskalebild (OpenML, 2025).

3.4 Splittning av data

Eftersom det ursprungliga datasetet är stort (70 000 bilder) och kan ta lång tid att träna modeller på, delas datan upp i tre olika storlekar för att kunna jämföra resultat snabbare och mer effektivt. Först skapas X och y där X representerar de oberoende variablerna och y de beroende variablerna (target).

- **Hela datasetet:** Datat delas in i 50 000 tränings-, 10 000 validerings- och 10 000 testbilder. De bästa modellerna tränas på hela tränings-datasetet, utvärderas med valideringsdata. Den vinnande modellen tränas om på tränings- och valideringsdata och slutligen testas modellen på testdatan.
- **Medium datasetet:** Här tränas modeller på 10 000 träningsbilder utan validering eller test, där Cross-validation används för att utvärdera prestanda och identifiera de bästa modellerna för vidare tränning på hela datasetet.
- **Litet dataset:** Består av 3 000 träningsbilder och används för att undersöka träningshastigheten för olika modeller, med Cross-validation för att testa deras prestanda.

Denna uppdelning gör det möjligt att snabbare utvärdera modeller och samtidigt spara tid vid tränning.

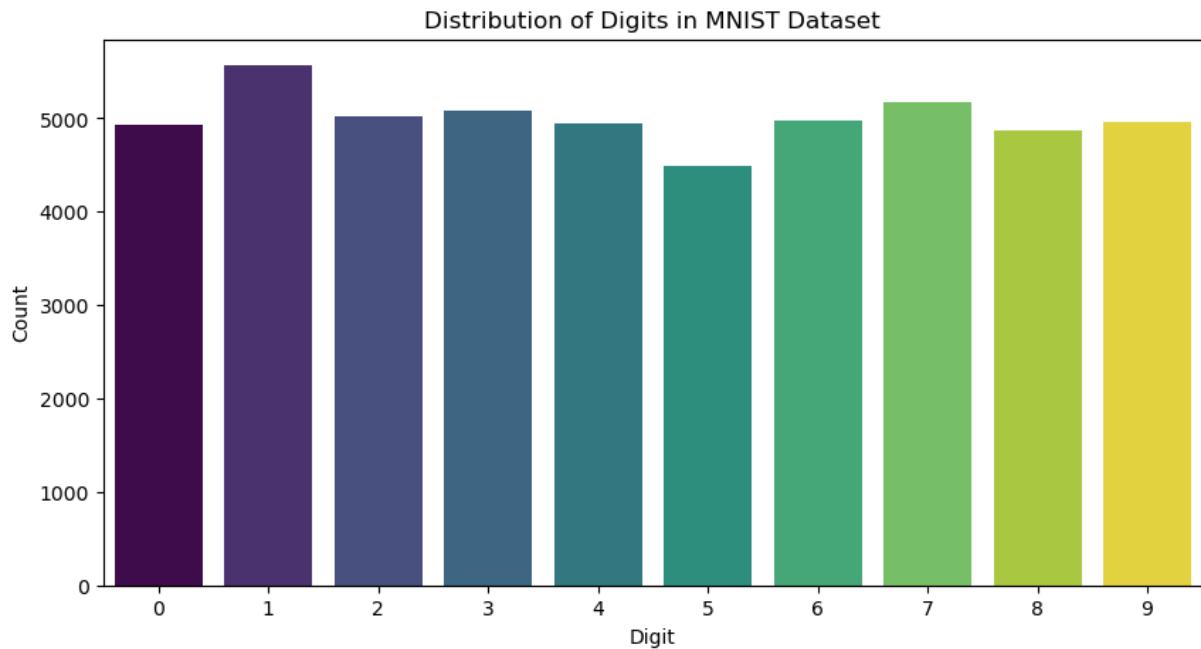
3.5 EDA

3.5.1 Datatyp

Varje pixel i MNIST-bilderna lagras som ett uint8 värde (0–255). För att modellen ska tolka dem korrekt behöver de normaliseras till float inom intervallet 0–1. Ny data måste därför bearbetas på samma sätt för att ge tillförlitliga prediktioner.

3.5.2 Fördelning i MNIST

För att säkerställa att datan inte var överrepresenterad av något specifikt målärde eller för snedfördelad, valde jag att undersöka och visualisera dess fördelning. Detta gjorde det lättare att få en uppfattning om hur datan var fördelad.



Figur 3-1 Fördelningen av handskrivna siffror i MNIST-datasetet, som visar hur varje siffra förekommer i träningsdata och ger en översikt över datasetets balans.

3.5.3 Urval av MNIST-bilder

För att få en bättre förståelse av hur MNIST-bilderna ser ut och hur modellen behandlar dem, valde jag att skriva ut några av bilderna. Det blir tydligt att bilderna har en svart bakgrund med vita siffror.



Figur 3-2 Exempel på handskrivna siffror från MNIST-datasetet.

3.6 Bearbetning av data

För att kunna träna alla modeller på data behöver den förberedas genom att skala den, vilket jag gör med hjälp av StandardScaler. Även om inte alla modeller kräver detta, gör vissa det, och därför bearbetas data för att optimera träningsprocessen. Detta gör processen både snabbare och mer stabil.

3.7 Modeller

3.7.1 Val av modell

Eftersom problemet är ett klassificeringsproblem valde jag att använda modeller som är väl lämpade för den här typen av uppgifter. Jag valde att använda de modeller jag redan var bekant med och hade erfarenhet av, vilket gav mig en bra grund för att snabbt komma igång och få resultat. Utöver detta

valde jag även att inkludera en Support Vector Machine (SVM), eftersom jag hade hört att den är särskilt effektiv för att hantera bilddata, vilket gör den lämplig för att klassificera handskrivna siffror från MNIST-datasetet.

3.7.2 Träning av modeller

Vid träning av modellerna började jag med att träna ett antal modeller på det lilla datasetet utan att optimera hyperparametrarna. Därefter tränade jag samma modeller på det medelstora datasetet och använde hyperparameteroptimering via funktionen GridSearchCV från Scikit-Learn för att hitta de bästa parametrarna. De modeller som gav de bästa resultaten vid optimeringen tränades sedan på det större datasetet med de optimerade parametrarna. För att kunna jämföra de optimerade modellerna med icke-optimerade modeller skapade jag också några modeller som inte genomgick någon hyperparameteroptimering.

3.7.3 Utvärdering av modeller

För att mäta modellernas prestationer använde jag 5-fold cross-validation för att beräkna accuracy score. Det är viktigt att påpeka att varken det lilla eller medelstora datasetet innehöll testdata, utan endast träningsdata. Modellerna tränades och utvärderades enbart på dessa dataset, där cross-validation användes för att få en rättvis bedömning av deras prestationer.

Efter att ha tränat modellerna på det medelstora datasetet och utvärderat dem, valde jag de modeller som visade bäst resultat att gå vidare med. Dessa, tillsammans med några nya modeller, tränades sedan på det större datasetet (50 000 instanser). Modellerna utvärderades slutligen med en accuracy score på valideringsdatan.

3.7.4 Utvärdering av modellens generalisering

När jag valt en vinnande modell gick jag vidare och testade dess generaliseringsförmåga på testdatan. För att utvärdera modellens prestation använde jag återigen accuracy som mått.

3.7.5 Spara och ladda modeller

För att undvika onödig omträning sparade jag mina modeller successivt under arbetets gång med Joblib. Detta gjorde det möjligt att ladda in dem vid behov istället för att träna om dem. Slutligen sparade jag även den bästa modellen tillsammans med min standardscaler för framtida användning.

3.8 Streamlit-applikation

Modellen har integrerats i en Streamlit-applikation för att kunna testas och användas interaktivt. Streamlit är ett verktyg för att snabbt bygga applikationer och visualisera maskininlärningsmodeller. I applikationen genomfördes viss förbearbetning av bilder för att anpassa dem till modellens format och säkerställa korrekta prediktioner. Till följd av modellens storlek är den inte produktionssatt. För en mer detaljerad beskrivning av koden, se Appendix A.

4 Resultat och Diskussion

4.1 Resultat

4.1.1 Första utvärderingen

Här är resultaten från modellen utan att ha optimerat hyperparametrar. Jag mätte accuracy på träningsdatan.

Modell	Medel CV Score	Tidsåtgång (i sekunder)
RandomForest	0.926667	3.806844
SVC	0.908667	4.156310
LogisticRegression	0.875667	0.933862
SGDClassifier	0.873667	3.799834
DecisionTree	0.730000	1.230171

Figur 4-1 Resultat för modellerna med 5-fold Cross-validation på träningsdatan (utan optimering av hyperparametrar).

4.1.2 Andra utvärderingen

Här presenteras resultaten för modellerna efter att ha optimerat hyperparametrarna med GridSearchCV och genomfört 5-fold Cross-validation på träningsdatan. Jag mätte accuracy för varje modell och noterade även träningstiden för att kunna jämföra prestanda och tidsåtgång.

SVC Score: 0.9482 {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'} Tid: 5 minuter och 45 sekunder.
LogisticRegression {'C': 1, 'penalty': 'l2', 'solver': 'saga'} Score: 0.9095 Tid: 16 minuter och 56 sekunder
SGDClassifier {'alpha': 0.0001, 'learning_rate': 'optimal', 'max_iter': 1000, 'penalty': 'l2'} Score: 0.9011 Tid: 18 minuter och 50 sekunder.
DecisionTree {'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2} Score: 0.8064 Tid: 1 minut och 56 sekunder
RandomForest {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200} Score: 0.9496 Tid: 9 minuter och 22 sekunder

Figur 4-2 Resultat för modellerna efter hyperparameternoptimering med GridSearchCV och 5-fold Cross-validation på träningsdatan. Tiden för träning noterades också.

4.1.3 Tredje utvärderingen

För att säkerställa konsekventa resultat, genomförde jag en tredje utvärdering där varje modell tränades individuellt istället för i en loop. Denna justering var för att undvika varningsmeddelanden. Den tredje utvärderingen använde samma förutsättningar som i den andra undersökningen, med Cross-validation (5-fold) och accuracy som mått genom GridSearchCV.

SVC

Score: 0.948199999999999
{'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
SVC(C=10, gamma='auto')
Tid: 6 min

LogisticRegression

Score: 0.9064
{'C': 0.1, 'penalty': 'l2', 'solver': 'saga'}
LogisticRegression(C=0.1, max_iter=500, solver='saga')
Tid: 40 min och 23 sek

SGDClassifier

Score: 0.9015000000000001
{'alpha': 0.0001, 'learning_rate': 'optimal', 'max_iter': 1000, 'penalty': 'l2'}
Felmeddelande: 60 fits failed out of a total of 120.
Tid: 17 min 21 sek

DecisionTree

Score: 0.8069
{'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2}
DecisionTreeClassifier(criterion='entropy', max_depth=10)
tid: 1 min 52 sek

RandomForest

Score: 0.9486000000000001
{'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
RandomForestClassifier(max_depth=20, n_estimators=200)
Tid: 9 min 3 sek

ExtraTree

Score: 0.956199999999999
{'bootstrap': False, 'criterion': 'gini', 'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 500}
ExtraTreesClassifier(max_depth=20, n_estimators=500)
Tid: 148 min 57 sek

Figur 4-3 resultat för modeller efter optimering av hyperparametrar med GridSearchCV och 5-fold Cross-validation, inklusive accuracy och träningstid på träningsdatan.

4.1.3.1 Sammanställning av score efter 3e körning

För att underlätta jämförelsen av accuracy scores efter den tredje körningen sammanställde jag modellerna och deras score i fallande ordning, vilket gör det enklare att identifiera de bästa presterande modellerna.

Modell	Accuracy	Tid
ExtraTree	0.9562	148 min 57 sek
RandomForest	0.9486	9 min 3 sek
SVC	0.9482	6 min
Logistic Regression	0.9064	40 min och 23 sek
SGDClassifier	0.9015	17 min 21 sek
DecisionTree	0.8069	1 min 52 sek

Figur 4-4 Jämförelse av modellerna baserat på accuracy score efter tredje körningen, sorterade i fallande ordning för att enkelt identifiera de bästa presterande modellerna.

4.1.4 Val av modeller att utvärdera med valideringsdata

Trots att de tre bästa modellerna hade bättre resultat och hade varit tillräckliga, valde jag ändå att inkludera en annan typ av modell för att få en mer mångsidig utvärdering. Därför valde jag att gå vidare med de fyra bästa modellerna för att utvärdera dem på valideringsdata.

4.1.4.1 Validering av optimerade modeller på valideringsdata

Alla modeller som genomgått hyperparameteroptimering tränas nu på det stora datasetet och valideras på valideringsdata för att utvärdera deras prestanda. I detta steg valde jag att inte mäta träningstiden, eftersom den skulle vara längre än vid den initiala optimeringen av hyperparametrarna. Fokus låg istället på att utvärdera modellerna baserat på deras accuracy score.

Nedan presenteras resultaten

Modell	Accuracy
ExtraTreesClassifier	0.9719
RandomForestClassifier	0.97
SVC	0.9715
LogisticRegression	0.9169

Figur 4-5 Tabellen visar de fyra modeller som valdes att gå vidare efter den initiala utvärderingen och nu har validerats på valideringsdata. Resultaten presenteras som accuracy-scores för varje modell.

4.1.5 Modeller utan hyperparameteroptimering

För att jämföra prestandan mellan modeller med och utan hyperparameteroptimering skapade jag tre modeller utan att använda GridSearch. Dessa modeller tränades utan några specifika inställningar för hyperparametrar, vilket möjliggjorde en snabbare träning och prediktion. Syftet var att undersöka om det fanns en märkbar skillnad i prestanda mellan de icke-optimerade och de optimerade modellerna.

Under denna process skapade jag också två olika voting classifiers, där en tränades med 'hard' voting och den andra med 'soft' voting. Dessa modeller tränades efter varandra, och jag använde dem för att förbättra prestandan genom att kombinera flera modeller. Dessutom använde jag probability=True i min SVC för att kunna utnyttja sannolikheten för klassificering, vilket också ökade tidsåtgången jämfört med en standard SVC.

4.1.5.1 Resultat

Tabellen nedan sammanställer resultaten för dessa modeller, där accuracy på valideringsdata samt träningstider redovisas. Detta gör det möjligt att jämföra prestandan hos de modeller som tränades utan hyperparameteroptimering med de som genomgick optimering.

RandomForest Score: 0.9692 Tid: 22 sek.
ExtraTree Score: 0.9715 Tid: 18 sek
SVC, Probabiltiy = True (för att kunna räkna på sannolikheter) Score: 0.9632 Tid: 19 min 42 sek
Ensemble VotingClassifier (Hard) Score: 0.9731 Tid: 20 min 43 sek
Ensemble VotingClassifier (Soft) Score: 0.9754 Tid: 20 min 58 sek

Figur 4-6 Modellprestanda utan hyperparameteroptimering.

4.1.6 Sammanställning av validerade modeller

Här följer en sammanställning av betygen på de olika modeller som validerats på valideringsdata.

Modell	Accuracy
Ensemble VotingClassifier (Soft)	0.9754
Ensemble VotingClassifier (Hard)	0.9731
ExtraTreesClassifier (Optimerad)	0.9719
ExtraTree (ooptimerad)	0.9715
SVC (Optimerad)	0.9715
RandomForestClassifier (Optimerad)	0.97
RandomForest (ooptimerad)	0.9692
SVC, Probabilty = True	0.9632
LogisticRegression (Optimerad)	0.9169

Figur 4-7 Tabellen visar alla modeller som validerades på valideringsdata och deras accuracy.

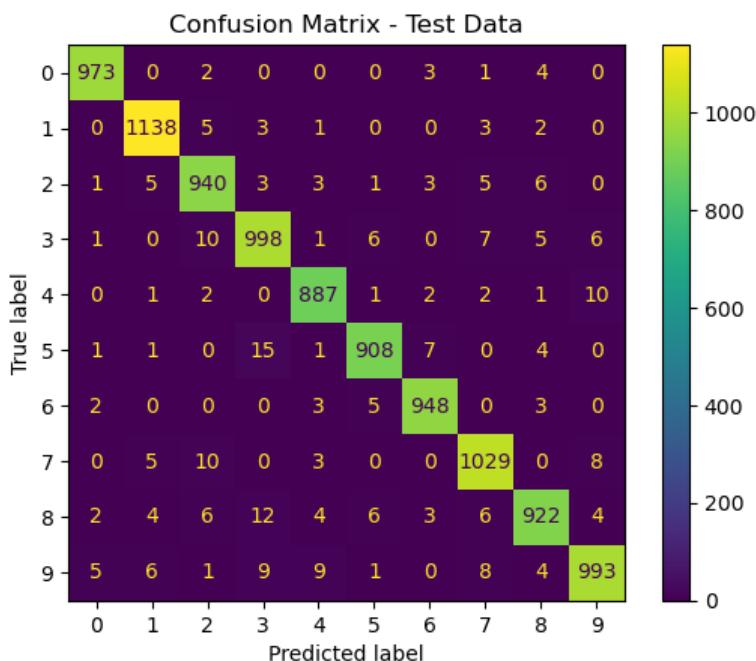
4.1.7 Träning och test av Voting Classifier

Efter att ha identifierat Voting Classifier (soft voting) som den bäst presterande modellen, tränade jag om den på hela tränings- och valideringsdata (X_train_val_scaled) för att utnyttja all tillgänglig data. Jag utvärderade dess accuracy på denna data och fick resultatet 1.0.

Därefter testades modellen på den tidigare hållna (osedda) testdata (X_test_large_scaled) för att bedöma dess generaliseringsförmåga. Accuracy på testdata blev 0.9736.

4.1.7.1 Confusion Matrix

Nedan visas confusion matrix för modellens prediktioner på testdata. Matrisen ger en detaljerad bild av hur väl modellen klassificerat varje klass, inklusive antalet korrekta och felaktiga prediktioner.



Figur 4-8 Confusion matrix för Voting Classifier-modellen på testdata, som visar antalet korrekt och felaktigt klassificerade exemplar för varje klass.

4.1.8 Storlek på modeller

Jag noterade även storleken på de sparade modellerna och såg att vissa modeller tog upp betydligt mer utrymme än andra. Även om detta inte är ett problem idag, kan det bli relevant vid mycket stora datamängder.

Modell	Storlek
ExtraTree (optimerad)	1100 MB
Voting Classifier (Hard voting)	911 MB
Voting Classifier (Soft voting)	911 MB
ExtraTree (ooptimerad)	249 MB
RandomForest (optimerad)	234 MB
RandomForest (ooptimerad)	121 MB
SVC (ooptimerad)	85 MB
SVC (optimerad)	78 MB
Linear Regression (optimerad)	63 kB

Figur 4-9 Filstorlekar för de sparade modellerna i MB. Tabellen visar hur mycket lagringsutrymme varje modell kräver, vilket kan vara relevant vid hantering av stora dataset.

4.1.9 Streamlit

Resultatet från Streamlit-applikationen visar att modellen kan identifiera handskrivna siffror, även om prediktionerna inte alltid är korrekta. En testbild illustrerar hur modellen tolkar en ritad siffra.

**Welcome to this amazing world of
number prediction with a white-box
model!**

**Just draw a number and watch the magic
unfold!**



Figur 4-10 Exempel på hur modellen tolkar en ritad siffra i Streamlit-applikationen.

4.2 Diskussion

4.2.1 Optimering mot ooptimering

4.2.1.1 Optimering

Fördelar:

Genom att optimera modellerna med de bästa hyperparametrarna kan vi förbättra modellens inlärning på träningsdatan, vilket i sin tur leder till bättre prediktioner på ny data. Målet är att uppnå så hög accuracy som möjligt, vilket innebär att minska antalet felaktiga prediktioner.

Nackdelar:

Det kan ta väldigt lång tid att leta bland och optimera för de olika hyperparametrarna. Beroende på modellen och dess komplexitet kan optimering ta från några minuter till flera timmar. Detta beror på faktorer som antalet hyperparametrar att testa, datasetets storlek och cross-validation.

Att testa enbart en modell kan ta lång tid och att testa fler modeller och optimera dessa, kan ta ännu längre tid.

4.2.1.2 Ooptimerade modeller.

Det var väldigt tydligt att de modeller som inte genomgick en extra optimering av hyperparametrar presterade lika bra. Därför bör man verkligen ställa sig frågan om man har råd att avsätta den tid som behövs för att hitta och anpassa hyperparametrarna, särskilt när de förbättringar man kan uppnå i prestanda ofta handlar om mycket små marginaler, ibland så lite som skillnader på tredje – fjärde decimalen.

Om man vill lösa vardagliga problem snabbt och marginalen inte är avgörande, är det tydligt att det är bättre att träna modellerna med de redan givna hyperparametrarna. Dessa är uppenbarligen redan väldigt bra, vilket gör att man kan komma igång snabbt, spara mycket tid och ofta uppnå lika hög prestanda.

4.2.1.3 Slutsats om optimering

När man tittar på resultatet för de olika modellerna så ser man att de modeller som har optimerats har presterat marginellt bättre, med skillnader som ofta avgörs på tredje eller fjärde decimalen. Samtidigt kan det ta lång tid, ibland flera timmar, för att optimera och komma vidare i arbetet. Därför bör man ställa sig frågan om det verkligen är värt den tid som krävs för att optimera en modell, givet den lilla förbättring i prestanda man ofta får.

4.2.2 Analys av ensemble-modeller

Att kombinera flera modeller i en ensemble har visat sig vara effektivt, då det drar nytta av deras individuella styrkor. Träningen tar längre tid än för en enskild modell, men resultatet blir ofta bättre, vilket är målet. Hard voting presterade bra, men soft voting gav ännu bättre resultat. Genom att väga in sannolikheten för varje klassificering får den modell som ger högst sannolikhet större inflytande, vilket leder till en mer flexibel och noggrann modell med högre accuracy och bättre prediktioner.

4.2.3 Validering av modeller.

Vid validering av både optimerade och icke-optimerade modeller ser man att deras accuracy score ligger väldigt nära varandra. Detta visar att den extra tid som krävs för optimering inte alltid ger en betydande förbättring. Därför kan det vara en mer effektiv strategi att, som tidigare nämnts, välja ut ett fåtal lovande modeller, träna och validera dem på valideringsdata och sedan välja den bäst presterande modellen för att utvärdera dess generaliseringsförmåga på testdata.

4.2.4 Analys av den valda modellen - Voting Classifier

Eftersom Voting Classifier är en ensemblemodell tog den längre tid att träna jämfört med en enskild modell. Jag valde dock att investera den tiden för att utvärdera hur en ensemblemodell skulle prestera. Det visade sig att den presterade bäst, vilket gjorde att jag gick vidare med den.

Vid träning på hela träningsdatan (träning + validering) uppnådde modellen en perfekt score på 1.0, vilket väckte misstankar om överanpassning. För att säkerställa modellens generaliseringsförmåga testade jag den på testdatan, vilket är det korrekta tillvägagångssättet. Resultatet blev en mycket hög score på 0.9736, vilket tyder på att modellen generaliseras väl på osedd data och att oron för överanpassning inte var befogad.

Den höga prestandan beror på styrkan i att kombinera flera modeller, där svagheter hos en modell kan vägas upp av styrkor hos en annan. Exempelvis kan vissa modeller vara mindre känsliga för varians, medan andra är bättre på att lära från viss typ av data. Genom att kombinera dessa i en ensemble kan man uppnå en starkare och mer träffsäker modell jämfört med en enskild modell som saknar denna kompensation.

4.2.5 Storleken har betydelse

Det är tydligt att vissa modeller tar upp mer lagringsutrymme än andra när de sparas, vilket kan vara en viktig faktor att beakta. Vid modellval kan datasetets storlek verka obetydlig om det endast rör sig om några tusen instanser, men när datasetet växer till miljontals instanser blir även modellens lagringskrav mer relevant. Även om hårdvarans prestanda förbättras och lagringskapaciteten ökar, så sjunker inte alltid priserna i samma takt som tidigare. Därför är det särskilt viktigt att väga in lagringskraven när man väljer mellan modeller som presterar likvärdigt.

5 Slutsatser

Genom mitt arbete har jag utvärderat hur modeller utan anpassade hyperparametrar presterar jämfört med modeller där hyperparametrarna har optimerats. Resultaten visar att optimerade modeller generellt presterar något bättre, men skillnaden är ofta så liten att den endast märks på tredje decimalen. Därför kan man dra slutsatsen att en modell utan optimering i praktiken presterar lika bra som en optimerad modell i många fall.

Den marginella förbättringen i prestanda måste vägas mot den tid det tar att optimera en modell. Optimering kan ta avsevärt mycket längre tid, vilket gör det mer praktiskt att använda standardinställningarna om den högsta möjliga accuracy-scoren inte är avgörande. För enklare problem är det därför mer effektivt att träna modeller direkt, utvärdera dem och välja den bästa utan att lägga tid på hyperparameteroptimering.

En ytterligare insikt från arbetet är att valet av modell blir viktigt när man arbetar med stora dataset. Vissa modeller växer betydligt i storlek när de tränas på fler instanser, vilket kan påverka lagringskrav och hanterbarhet vid mycket stora datamängder.

5.1 Vidare undersökning

En möjlig vidare undersökning skulle kunna vara att utforska om vissa typer av modeller gynnas mer av hyperparameteroptimering än andra.

Det skulle också vara intressant att utforska om **ensemble-metoder** (som Voting Classifier) påverkas av hyperparameteroptimering på samma sätt som enskilda modeller, eller om de redan balanserar ut sina brister genom kombinationen av flera modeller.

6 Teoretiska frågor

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

Svar:

För att kunna bygga en modell som skall kunna prediktera på ny data så är det viktigt att dela upp vårt dataset i tre delar där viss data utgör träning, validering och test.

Träning

Den del av data som delats upp i träning brukar utgöra ca 70-80 % av hela datasetet, dvs innan vi delade upp datan och används till att träna vår modell på. Det är här modellen studerar datan och lär sig samband mellan oberoende och beroende variabler för att sedan kunna prediktera på ny data.

Validering

För att kunna veta om modellen presterar bra behöver vi kunna mäta detta och modellen behöver därför någon typ av mått, exempelvis RMSE. Genom att mäta detta kan vi veta hur bra modellen presterat och det är detta som valideringsdatan används till. Dvs efter att modellen tränats med träningsdata låter vi den prediktera på valideringsdatan. Vi mäter resultatet mot den faktiska datan och får ut ett mått (ex RMSE) som visar hur bra modellen presterat. Detta är något vi gör även när vi vill jämföra flera modeller för att vi skall kunna välja ut den bästa modellen som vi slutligen skall köra på testdata.

Valideringsdatan brukar bestå någonstans mellan 10-15% av hela datasetet men kan även vara mindre om vi har riktigt stora dataset.

Test

När vi valt vilken modell vi vill gå vidare med använder vi slutligen testdatan för att se hur väl modellen nu predikterar på helt osedd data. Dvs hur bra den är på att generalisera på ny data. Även här använder vi vårt mått/score för att se hur väl den presterat.

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 "validerings-dataset"?

Svar:

Då Julia inte skapat ett valideringsset så kan hon använda sig av Cross Validation när hon tränar modellerna. Cross Validation delar upp träningsdatan i olika folds (mappar) beroende på vilken Cross Validation vi väljer. Om vi exempelvis väljer CV=5 vilket är normalt, så delas datan upp i fem lika stora delar där den i 5 iterationer tränar på fyra delar och testar mot den sista. I första iterationen skulle den exempelvis kunna träna på D1, D2, D3, D4 och validera på D5. I andra iterationen så skulle den köra D2, D3, D4, D5 och validera på D1. Den gör så här för varje iteration tills dess att den tränat och validerat mot alla delar. När den är klar så har den tränat modellen 5 gånger och för varje träning så har den fått ett medelmått (ex accuracy score) och alla dessa summeras och divideras

med antal iterationer vilket slutligen ger oss medelmåttet av alla körningar. En nackdel med detta tillvägagångssätt är att om det är väldigt mycket data så kan det ta väldigt lång tid att träna modellen. Vid oerhört stora dataset är det kanske mer lämpligt att dela upp i träning, validering och test.

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

Svar:

Förslag på några modeller för regressionsproblem: LinearRegression, Lasso, RandomForest.

Ett regressionsproblem är när vi vill försöka förutsäga ett värde i siffror. Exempelvis värdet på ett hus, hur stor lön någon förväntas ha osv. Vi använder oss då av en eller flera oberoende variabler för att försöka skatta det beroende värdet.

- Hur kan du tolka RMSE och vad används det till.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

Svar:

Roten ur det kvadrerade medelfelet.

RMSE används för att mäta medelfelet i modellens prediktioner. Det mäter avståndet mellan de faktiska värdena och de predikterade värdena

- Vad är ”klassificeringsproblem? Kan du ge några exempel på modeller som används och potentiella tillämpningsområden? Vad är en ”Confusion Matrix”?

Svar:

Ett klassificeringsproblem innebär att vi vill dela in vår data i olika klasser.

Modeller som används för klassificering är exempelvis Random Forest, Extra Trees, Logistic Regression och SVC.

En Confusion matrix används för att utvärdera hur väl en klassifierare lyckats med sina prediktioner. Att se om en modell predikterat korrekt i relation till de faktiska värdena. Vi har True positives, True negatives, False positives och False negatives. Detta ger oss olika mått vi kan kolla på såsom Precision och Recall och även F1 score.

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

Svar: En K-means modell är en klustringsmodell som klstrar olika data som är unlabeled, dvs unsupervised learning. Modellen letar efter olika kluster i datan (olika grupperingar).

Exempel på när K-means kan användas är om vi vill hitta olika grupperingar bland kunder på ett bolag för använda detta vid exempelvis marknadsföring, merförsäljning.

7. Förklara (gärna med ett exempel): Ordinal encoding, one-hot encoding, dummy variable encoding. Se mappen "l8" på GitHub om du behöver repetition.

Svar: I dataseten så kan det finnas kategorisk data som man vill använda. Exempelvis kan det vara bilmärken eller geografiska områden eller dylikt. Detta kan vi behöva transformera med olika transformatörer då inte alla modeller direkt kan bearbeta kategorisk data, och det kan då ske på tre olika vis.

Ordinal encoding är när det finns en inbördes ordning i kategorin, exempelvis, bra, bättre, bäst där den ena väger tyngre än den andra. Genom att använda Ordinal encoding så tar modellen hänsyn till detta.

One-hot encoding används när det inte finns någon innehördes ordning.

Dummy variable används för att vi kan spara plats genom att inte spara värden på något som vi egentligen redan vet. Det innebär att vi droppar ett kolumnvärde. Ett exempel är om vi har tre bilmärken så får två av bilmärkena en binär 1a i sin kolumn och två nollor. Det tredje bilmärket får istället bara tre nollor vilket kännetecknar att det är det bilmärket vi syftar på.

Vissa ML-modeller har också svårt att hantera One-hot encoding varför Dymmy behövs.

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?

Svar: Det beror på! Ordinal innebär att det finns en inbördesordning medan det inte gör det i nominal. Bland de nämnda färgerna tros det inte vara någon direkt inbördes ordning och det bör då vara som Göran säger att det är nominal. Men om Julia nu har arrangerat en fest där röd skjorta är vackrast på festen, så verkar det finnas en inbördesordning och på hennes fest kan man då tolka det som ordinal 😊.

9. Kolla följande video om Streamlit: <https://www.youtube.com/watch?v=ggDa-RzPP7A&list=PLgzaMbMPEHEx9Als3F3sKKXexWnyEKH45&index=12> Och besvara följande fråga: - Vad är Streamlit för något och vad kan det användas till?

Svar:

Streamlit är ett digitalt verktyg som låter dig bygga en interaktiv app där man bland annat kan ladda in en ML modell för att prediktera siffror men man kan också presentera en massa modeller och visualiseringar. Styrkan ligger i att det går snabbt att komma igång, stödjer de vanliga biblioteken, koden är ganska straight forward m.m.

7 Självutvärdering

1. Utmaningar du haft under arbetet samt hur du hanterat dem.

En stor utmaning jag haft har varit att begränsa mitt arbete vilket jag initialt inte tänkte på när jag började. Detta märktes under resan när jag prövade flera modeller och det tog lång tid att träna och optimera. Men inte minst att min rapport blev lika omfattande för att få med alla delar jag gjort. Detta är något jag får ha med mig till nästa arbete. Att även om det är kul att testa på och pröva när man har chansen så kan det kosta i andra änden 😊

2. Vilket betyg du anser att du skall ha och varför.

Jag tycker att jag förtjänar ett VG. Jag upplever att jag uppfyller de krav som ställs för att få betyget. Jag har tagit till mig kunskapen från lektioner, videos m.m och kunnat tillämpa detta i kodning och framför allt projektet. Men har även haft möjlighet att hjälpa andra för att jag förstått hur det fungerar och kunnat dela med mig. Jag har också engagerat mig på lektioner för att dela tankar och få en ökad förståelse.

3. Något du vill lyfta fram till Antonio?

Det var lika roligt som vanligt att se dig och jag har lärt mig mycket under kursen. Precis som med Sannolikheteskursen så var det lite svårrott i början men för varje vecka så blir det klarare och klarare och man kan koppla ihop ett och ett.

Appendix A

Kunskapskontoll 2- kodning

https://github.com/lantzpeter/05_ml/blob/main/Kunskapskontroll_2_peter_lantz_21032025.ipynb

Streamlit - kodning

https://github.com/lantzpeter/05_ml/blob/main/streamlit_digit.py

Modeller

Jag har inte valt att ladda upp modellerna då det var flertalet och då många var större filstorlek.

Källförteckning

Joblib. (n.d.). Joblib documentation. Hämtad 21 mars 2025, från
<https://joblib.readthedocs.io/en/latest/>

Géron, A. (2023). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques for Building Intelligent Systems* (3rd ed.). O'Reilly Media.

Matplotlib. (n.d.). Using Matplotlib. Hämtad 21 mars 2025, från
<https://matplotlib.org/stable/users/index>

NumPy. (n.d.). NumPy user guide. Hämtad 21 mars 2025, från
<https://numpy.org/doc/stable/user/index.html>

OpenML. (2025). *MNIST dataset*. Hämtad 20 mars 2025, från
<https://www.openml.org/search?type=data&status=active&id=554&sort=runs>

Pandas Development Team. (n.d.). User guide. Hämtad 21 mars 2025, från
https://pandas.pydata.org/docs/user_guide/index.html

Prgomet, A. (2025). *Klassificering* [PowerPoint-slides]. EC-utbildning. Hämtad från:
https://github.com/AntonioPrgomet/ds24_ml/blob/main/l1/02_klassificering.pptx

Prgomet, A. (2025). *Ensemble learning & Random forest* [PowerPoint-slides]. EC-utbildning.
Hämtad från:
https://github.com/AntonioPrgomet/ds24_ml/blob/main/l7/09_ensemble_learning_random_forest.pptx

Python Software Foundation. (n.d.). time — Time access and conversions. Hämtad 21 mars 2025, från
<https://docs.python.org/3/library/time.html>

Scikit-learn. (2025, 18 mars). accuracy_score. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

Scikit-learn. (2025, 20 mars). fetch_openml. Hämtad från https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_openml.html#sklearn.datasets.fetch_openml

Seaborn. (n.d.). seaborn: statistical data visualization. Hämtad 21 mars 2025, från
<https://seaborn.pydata.org/>

Streamlit. (n.d.). Getting started with Streamlit. Hämtad 21 mars 2025, från
<https://docs.streamlit.io/library/get-started>

OpenAI. (n.d.). ChatGPT. Hämtad 21 mars 2025, från <https://chat.openai.com/>