TEK300

Maskinlæring

Eksamensinnlevering ved

Høyskolen Kristiania - Institutt for teknologi

HØST 2018

Kandidatnummer:

700009

Intro

Oppgaven går ut på å identifisere siffer fra MNIST datasettet med forskjellige algoritmer. Gjennom teksten vil det bli gjort rede for prosess, hvilke algoritmer som er tatt i bruk og sammenlikning av output.

Prosess

Det første jeg valgte å gjøre i prosessen var å allokere datasettet, og bli bedre kjent med det. Dette er et datasett med håndskrevne siffer med en kolonne som er fasit på hvilket siffer som er skrevet. Deretter valgte jeg å bli kjent med de ulike algoritmene som kan være anvendelig i bruk av siffer gjenkjenning, da jeg anså det som hensiktsmessig med grunnkunnskap på plass før jeg startet å kode. Ettersom oppgavesettet sier at det skal brukes supervised learning[1], var det noe lettere å minimere scopet. Det jeg så for meg var at fire ulike algoritmer vil være representativt for å sammenligne resultater, samt bredde om hva som har blitt lært gjennom semesteret.

Da jeg vurderte algoritmene jeg ville anvende i denne eksamen, tenkte jeg at det ville være mest gunstig å ikke ta i bruk algoritmer jeg har jobbet med tidligere. Siden vi brukte clustering under arbeidskravet, tenkte jeg det var greit å holde seg innen supervised learning. Etter litt frem og tilbake valgte jeg å gå fremover med Decision tree classifier, K-nearest, Naive bayes og multilayer perceptron.
Det enkleste ville absolutt vært å bruke kode fra forelesningene og endre litt om slik at jeg kunne brukt det til MNIST datasettet. Jeg mener at det er mer gunstig for meg å danne vaner med å finne informasjon på egenhånd, og valgte derfor å gå fremover ved å lete på nettet.

Etter at jeg hadde fått implementert alle algoritmene i kode inn i løsningen min og fått ut valid data. Gikk jeg inn for å gjøre slik at dataen kan være representativ. Det må være noe visuelt som bruker kan se, og faktisk validere at dataen er slik som den gir

[1] "Supervised learning - Wikipedia." https://en.wikipedia.org/wiki/Supervised_learning. Åpnet 22 nov.. 2018.

inntrykk av. Brukeren må kunne bruke programmet til noe. Da denne delen var på plass flyttet jeg fokuset over til å analysere dataene jeg fikk ut av testene. Da fokuserte jeg på hvor lang tid det tok å komme frem til kode som gjør alt den skal, hvor lang tid den bruker på å bygge modellen/treet[2], samt å kjøre koden. Årsaken til at jeg mener dette er en viktig del av prosessen, er fordi man ønsker å få mest mulig ut av tiden man investerer i et prosjekt. Det vil da være hensiktsmessig å danne seg formeninger om hva som vil være gunstig og mindre gunstig. Det vil eksempelvis ikke være like gunstig å bruke 30 minutter på å bygge en modell som får samme presisjon som en annen algoritme med kjøretid på under ett sekund.

Da jeg så meg ferdig med dette bevegde jeg meg over til å se på presisjonen de ulike algoritmene får. Jeg så etter trender over 10 tester, for å se om det var noen avvik i rapportene. Da mener jeg om det er noen tall med spesielle former som regelmessig får lavere presisjon enn tall med mindre spesielle former. Man kan da fundere over hva årsaken til dette er ved å se etter mønster i formene og trekke konklusjoner ut av det.

Algoritmer

Denne seksjonen av dokumentasjonen gjør rede for de forskjellige algoritmene som har blitt brukt i demoen. Den går gjennom hver algoritme og forteller litt kort om funksjon, hensikt og bruksområde.

Decision tree classifier

Dette er en trebasert læringsalgoritme og er en av de mest brukte algoritmene innen supervised learning. Treets struktur er en rotnode med indre noder, grener og blad-noder. Algoritmen er svært utbredt blant data mining[3], maskinlæring og statistikk. Målet er da å lage en modell som kan predikere en ut-verdi(output)basert

[2] "Predictive modelling - Wikipedia." https://en.wikipedia.org/wiki/Predictive_modelling. Åpnet 25 nov.. 2018.
[3] "Data mining - Wikipedia." https://en.wikipedia.org/wiki/Data_mining. Åpnet 21 nov.. 2018.

på en større mengde inn-verdi(output). Treet vil bli lært opp ved at man analyserer dataene med et sett med kriterier.
Algoritmen i seg selv var enkel å implementere, ettersom oppsettet av koden er veldig simplistisk. Treet trenes opp med et trening datasett og fasit som parameter, og returnerer presisjon ved å iterere over all test dataen med hvor mange ganger den predikerte riktig svar.

K-Nearest neighbors

K-Nearest neighbors(KNN) er en parametfri algoritme som brukes for klassifisering. Algoritmen er utrolig simplistisk i den form at den er en lat algoritme som ikke lager noen modell. Den plasserer alle verdier på ett plan og finner K-nærmeste naboer basert på likheter i gitt parameter. Det finnes ingen fasit på hvilket antall K som er best. Det vil variere fra på datasett, treningsmengde og lignende. Det vil derfor være naturlig å forskjellige verdier.
Ved testing vil algoritmen sette de ulike objektene inn i klasser og undersøker om det er korrekt ved å validere med label-kolonnen.

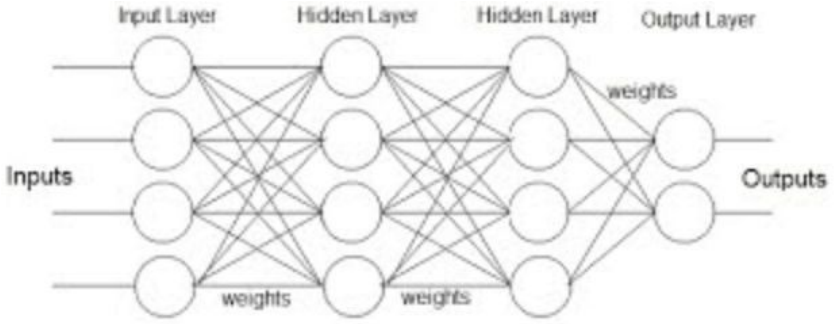
Naive bayes

Dette er en av de vanligste probabilistiske klassifiserings algoritmene, det vil si at den har en observasjon av input og predikerer output. Algoritmen er naiv i den forstand at den vurderer hvert parameter helt uavhengig av andre parametere, som er årsaken til at den har fått navnet. I dette eksempelet så bruker jeg Gaussian naive bayes som er en klassifiseringsmodell. Årsaken til at jeg valgte denne modellen er på bakgrunn at den er svært enkel å implementere, samt at den er veldig populær med gode eksempler.

Koden er basert på den samme implementasjonen jeg brukte i Decision tree classifier, som gjorde at jeg kun behøvde å importere modellen, og å legge den i en variabel.

Multilayer perceptron

Multilayer perceptron(MLP) går under klassen kunstige nevrale nett. Nettet er bygd opp celler/nevroner og forbindelser/synapser mellom disse cellene. Man trenger minimum tre ulike lag i det nevrale nettet. Input, hidden og output. MLP består av flere lag av noder i en rettet graf med hvert sitt lag tilkoblet til det neste. Derfor er hver node tilkoblet en annen node med vekting mellom hvert tilkoblet lag. Størrelsen på antall nevroner og lag øker kompleksiteten på algoritmen. Algoritmen lærer ved å endre vektingen etter hver bit av data har blitt prosessert, dette baseres på mengden feil i output sammenliknet med prediksjon.



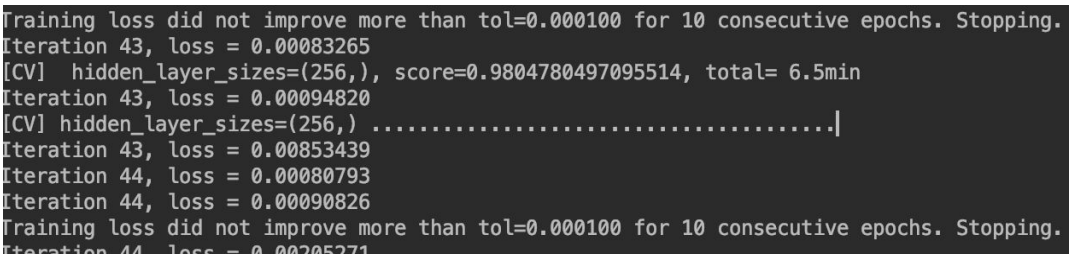
Figur 1.

Figuren over viser den overordnede strukturen. Det vil naturligvis være forskjellig fra bruk til bruk, ettersom man selv kan modifisere kompleksiteten ved å skalere opp antallet skjulte lag og noder. Første gang man kjører modellen uten en predefinert modell i direktoratet, tar det ekstremt mye lengre tid å kjøre koden. På min fire år gamle MacBook Pro tok det over 35 minutter å bygge modellen. Etter modellen har blitt lagd og koden kjøres igjen tar det maks to sekunder å gjøre en ny prediksjon.

```
[CV] hidden_layer_sizes=(256,) .....
[CV] hidden_layer_sizes=(256,) .....
[CV] hidden_layer_sizes=(256,) .....
[CV] hidden_layer_sizes=(256,) .....
Iteration 1, loss = 0.41964772
Iteration 1, loss = 0.41329584
Iteration 1, loss = 0.40715395
Iteration 1, loss = 0.41072613
Iteration 2, loss = 0.18710159
Iteration 2, loss = 0.18125407
Iteration 2, loss = 0.18209571
Iteration 2, loss = 0.18568641
Iteration 3, loss = 0.13209798
Iteration 3, loss = 0.13058362
Iteration 3, loss = 0.12901178
Iteration 3, loss = 0.13603400
```

```
Iteration 25, loss = 0.00235844
Iteration 25, loss = 0.00306971
Iteration 25, loss = 0.00225713
Iteration 26, loss = 0.00214283
Iteration 26, loss = 0.00201316
Iteration 26, loss = 0.00238123
Iteration 26, loss = 0.00212542
Iteration 27, loss = 0.00186524
Iteration 27, loss = 0.00170985
Iteration 27, loss = 0.00214812
Iteration 27, loss = 0.00181543
Iteration 28, loss = 0.00176089
Iteration 28, loss = 0.00162899
```

Figur 2 og 3.



Figur 4.

Ved å iterere gjennom de ulike parameterne og finne den beste verdien med minst mulig tap, bruker modellen den beste estimatoren for å predikere ved senere bruk.

Grunnen til at jeg valgte å implementere et nevralt nett, er på bakgrunn av at jeg synes det er spennende og super relevant. Nevrale nett og deep learning er av de mest hype temaene innen maskinlæring, derfor var det logisk å opparbeide seg erfaring. Det var også interessant å være med i prosessen å bygge en modell.

Sammenligning

Her tar jeg for meg resultatene som har blitt returnert fra algoritmene. Jeg går dypere inn i hva jeg mener er bra- og dårlig. Samt analyser klassifisering rapportene og hvilke saker det vil være hensiktsmessig å anvende de ulike algoritmene.

Classification_report

Bygger en tekst rapport som viser hoved klassifiseringen av beregningene. Dette er da et sammendrag om presisjon, recall, F1 score og support. Årsaken til at dette er en utrolig fin metode, er at brukeren får ut valid data som kan brukes. Det er da en oversikt over alle klassene som i dette tilfellet er sifrene 0-9. Her kan man se hvor mange ganger algoritmen treffer på testene.

Decision tree classifier

Det jeg merket var at den analyserer dataene utrolig raskt, som vil si at den kan jobbe med store datasett uten at man behøver enorme ressurser. Det som kan ta litt tid er å bygge treet(modellen), men jeg mener si at det er ikke noe man vil legge

merke til. Da må det i tilfelle være snakk om enorme datasett med store objekter, eks satellittbilder.
Ved all testingen som har blitt utført kan jeg se at det jevnlig er 3, 5, 8 og 9 som får lav presisjon. Ettersom presisjonen er så lav sammenlignet med de andre algoritmene, ville jeg ikke brukt et tre for siffer gjenkjenning. Jeg vil dog nevne at det var utrolig enkelt å implementere koden, som mest sannsynlig er årsaken til at det er en utbredt form for maskinlæring som leverer gode prediksjoner, men ikke i dette tilfellet.

K-Nearest neighbors

Med denne algoritmen oppnår vi 99.26% treffsikkerhet. Etter litt testing av antall naboer, kan jeg ved å iterere med ett heltall hver gang få ut valid presisjon på testingen. Fra 1-16 naboer får jeg 99.26% treffsikkerhet, som vil si at den skalerer veldig bra for dette datasettet. Som for decision tree classifier og naive bayes får jeg lavere presisjon på 8 og 9. Presisjonen minker marginalt ved å øke antall naboer. Det vil dog i få tilfeller være hensiktsmessig å ha så stor K-naboer at det går utover presisjonen.

Koden var noe vanskeligere å implementere ettersom det var her jeg ble kjent med “train_test_split”. Forutenom det kan jeg se for meg at denne algoritmen vil være fin å bruke for applikasjoner med siffer gjenkjenning og lignende. Den tar lite plass, trenger ingen modell og bruker svært kort tid på å kjøre. Ypperlig for dette datasettet.

Multilayer perceptron

Dette er definitivt den mest presise modellen for MNIST datasettet med 99.5 presisjon. Den kjører utrolig fort etter at modellen har blitt lagd. Med 17 500 test objekter så analyserer den alle på en svært høy hastighet. Det som tar lang tid er å lage selve modellen. Jeg prøvde å bygge modellen 5 ganger, da brukte den over 30 minutter hver gang på min MacBook pro fra 2014. Jeg opplevde at det krevde en del finpussing i parameterne, ettersom det tok lengre tid og returnerte feilmeldinger ved endringer. Det var uten tvil den mest komplekse algoritmen å implementere i hele

løsningen. Koden var mye mer kompleks enn de andre algoritmene, men det fungerte til slutt.

Ettersom KNN hadde den samme presisjonen som MLP, vil jeg si at det vil være enklere og mindre tidkrevende å bruke KNN. Den bruker kortere tid på å kjøre, samt at man ikke behøver å lage en modell. Nå som jeg har fått erfaring med ulike algoritmer, vil jeg påpeke at det finner flere algoritmer som er lettere å trene, modellere og å teste enn MLP. Det vil derfor avhenge helt at tilfellene jeg hadde valgt å gå for denne i en løsning.

Naive bayes

Her oppnår jeg 56% treffsikkerhet på testene med Naive bayes, grunnet at den naive antagelsen åpenbart ikke er korrekt. Dette er en svært lav treffsikkerhet som kan gjenspeiles i antagelse om at den naive predikeringen ofte er inkorrekt, tilnærmet 44% inkorrekt av 21 000 testtilfeller.

```
Naive Bayes results:
The algorithm predicted correct in 11758 of 21 000 cases.
Accuracy =  55.99047619047619
-----------------------------------

Evaluation of the test data:
              precision    recall  f1-score   support

           0       0.66      0.93      0.77      2088
           1       0.75      0.95      0.84      2312
           2       0.84      0.22      0.35      2069
           3       0.69      0.41      0.51      2174
           4       0.83      0.15      0.25      1997
           5       0.70      0.07      0.13      1897
           6       0.61      0.95      0.75      2062
           7       0.88      0.33      0.48      2234
           8       0.32      0.58      0.42      2058
           9       0.39      0.93      0.55      2109

   micro avg       0.56      0.56      0.56     21000
   macro avg       0.67      0.55      0.50     21000
weighted avg       0.67      0.56      0.51     21000

-----------------------------------
```

Figur 5.

Det man kan se på figuren over, er at de rapporterte tallene i “precision” kolonnen har relativt OK verdier. Ved å trimme gjennomsnittet med å fjerne de tallene 8 og 9, har algoritmen en treffsikkerhet på 74,5%. Derav er den ikke like dårlig som den

tilsynelatende kan virke ut som. Det at den treffer så utrolig lavt på 8 og 9 gjør at den ikke vil være realistisk å bruke i noen sammenheng av siffer gjenkjenning med så lav oppløsning, på bakgrunn av at den treffer relativt lavt på samtlige siffer.
Vel å merke så var det utrolig enkelt å implementere algoritmen, det at den ikke var passelig for dette datasettet, betyr ikke at man kan få høy presisjon på andre datasett. Jeg vil derfor gjøre rede for at det absolutt vil være hensiktsmessig å teste den på større datasett, ettersom at den kjører veldig fort og det er lite som må til kodemessig, for å returnere valid data.

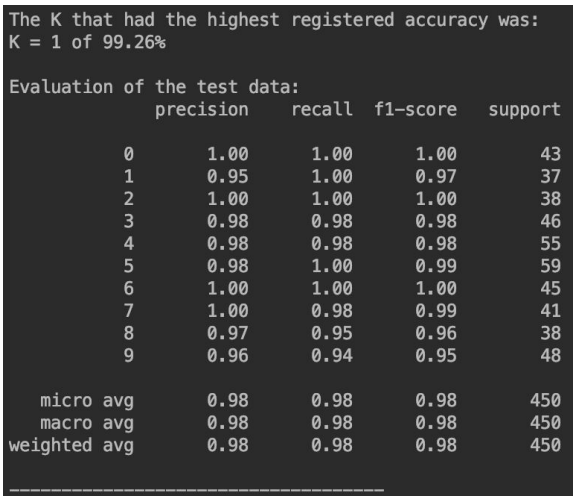
Resultat

Det man kan understreke med disse resultatene er at decision tree classifier og naive bayes får lav presisjon på sifrene 8 og 9, spesielt naive bayes. Det kan antas at de sliter med å analysere de sirkulære formene av de håndskrevne sifrene. Det er mulig at mennesker skriver de sirkulære formene litt over allerede eksisterende streker, som kan resultere i at maskinen vil predikere noe annet. Dette er på bakgrunn av at K-nearest som har presisjon på 99%, også får lavest presisjon på sifrene 8 og 9.

```
Training Decision tree classifier algorithm...

Decision tree classifier results:
The algorithm predicted correct in 17618 of 21 000 cases.
Accuracy =  83.8952380952381
-----------------------------------

Evaluation of the test data:
              precision    recall  f1-score   support

           0       0.91      0.91      0.91      2088
           1       0.92      0.94      0.93      2312
           2       0.80      0.82      0.81      2069
           3       0.80      0.78      0.79      2174
           4       0.84      0.85      0.85      1997
           5       0.77      0.76      0.77      1897
           6       0.87      0.86      0.87      2062
           7       0.88      0.89      0.88      2234
           8       0.78      0.77      0.77      2058
           9       0.79      0.81      0.80      2109

   micro avg       0.84      0.84      0.84     21000
   macro avg       0.84      0.84      0.84     21000
weighted avg       0.84      0.84      0.84     21000
```

```
Naive Bayes results:
The algorithm predicted correct in 11758 of 21 000 cases.
Accuracy =  55.99047619047619
-----------------------------------

Evaluation of the test data:
              precision    recall  f1-score   support

           0       0.66      0.93      0.77      2088
           1       0.75      0.95      0.84      2312
           2       0.84      0.22      0.35      2069
           3       0.69      0.41      0.51      2174
           4       0.83      0.15      0.25      1997
           5       0.70      0.07      0.13      1897
           6       0.61      0.95      0.75      2062
           7       0.88      0.33      0.48      2234
           8       0.32      0.58      0.42      2058
           9       0.39      0.93      0.55      2109

   micro avg       0.56      0.56      0.56     21000
   macro avg       0.67      0.55      0.50     21000
weighted avg       0.67      0.56      0.51     21000

-----------------------------------
```

Figur 6 og 7.



Figur 8.

Det at tallene 8 og 9 har en rund form på toppen, gjør at det må være der algoritmene sliter mest.

Alle algoritmene bruker ca. like lang tid på å kjøre. Det er ingen merkbar forskjell på runtime. Det tar ikke lang tid å bygge modellene, med unntak av MLP-en. Med alle resultatene jeg har fått fra denne oppgaven, vil jeg si at den algoritmen jeg har fått best inntrykk av er K-Nearest. Den leverte svært gode resultater med ikke så mye anstrengelse. Det er absolutt en algoritme jeg skal bli bedre kjent med.

Refleksjon

Her må jeg bare nevne igjen at det var utrolig artig å opparbeide erfaring med nevrale nettverk. Jeg synes det er kult at vi får egne erfaringer med maskinlæring som forhåpentligvis kan hjelpe oss i karrieren. Man hører nevrale nett og leser om det, men det er først nå jeg mener jeg har noe relevant å komme med. Forutenom generell kunnskap om temaet. Det har vært moro å jobbe med maskinlæring i kode som har gjort hele prosessen triveligere. Jeg har fått vist bredde, samt forståelse av hva som er hensikten med bruk av maskinlæring, som avslutningsvis har gjort meg veldig fornøyd med sluttresultatet.

Referanser

Decision tree classifier: “Handwritten Digits Recognition in python using scikit-learn”
https://www.youtube.com/watch?v=aZsZrkIgan0
Pyimagesearch: “K-nearest neighbors”
https://gurus.pyimagesearch.com/lesson-sample-k-nearest-neighbor-classification/
Multilayer perceptron: “1.17. Neural network models(supervised)”
https://scikit-learn.org/stable/modules/neural_networks_supervised.html?fbclid=IwAR2rs_kG3qe5jb5feWCThumxYJnWHmciAXTFRnyw8P1olWP0nIF5eidirSo