Dokumentasjon

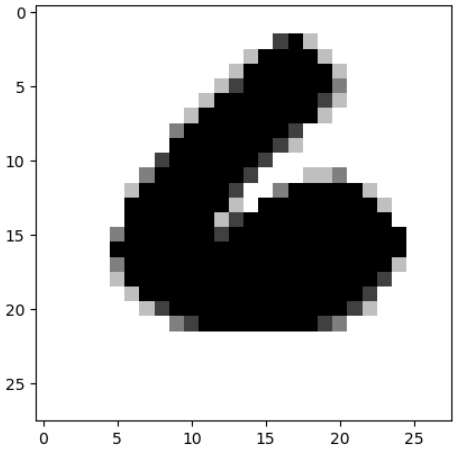
Forord

Etter å jobbet med andre eksamener følte jeg meg litt rusten i python og maskinlæring. Da var det bare å begynne i en ende. Planen var å først få en av dem til å funke. Jeg gikk rett til Sentdex sin Youtube kanal da jeg vet at han har gode videoer. Fant en video om SVM og MNIST datasettet. Etter å ha fått det til å funke innså jeg at det er endel smådetaljer som jeg ikke skjønner når det gjelder python syntax og scikit learn bibliotekene. Måten jeg har taklet dette på før er å få mange ulike eksempler på samme problemstilling til å funke. Om jeg enda ikke skjønner alt så prøver jeg å "merge" de ulike fungerende eksemplene sammen. Om en metode eller linje med kode fungerer slik jeg tror den gjør så tar jeg to eller flere eksempler å bytter om på kode som gjør noe ala det samme. Deretter tar jeg linjene med kode jeg skjønner eller liker best. Det gjorde jeg! Dette brukte jeg en hel dag på. Jeg merket godt dagen etter hvor lurt det var. Neste algortime gikk veldig mye fortere. MNIST datasettet er lastet ned fra kaggle. Denne er bearbeidet på forhånd, klar til maskinlæring. Den inneholder bl.a train.csv og test.csv. Planen var jo da å trene på train.csv og teste på test.csv. Men etter å ha eksperimentert med train.csv og brukt den til både testing og trening så konkluderte jeg med at det ville tatt for lang tid. Det varierer, men går fort opp mot 10 minutter med venting på min gamle slitne laptop ved bruk av 40000 bilder til test og trening.

I og med at de ulike algoritmene responder nokså ulikt på mengde treningsdata så har jeg valgt å teste nettopp dette. Velger å teste med 500, 5000, 20000 og 40000. 80% er satt av som treningsdata, de resterende 20% som testdata. Rekkefølgen under er den samme rekkefølgen som jeg arbeidet i.

Kode

Det er flere kodesnutter som er helt lik i alle filene. Etter å ha brukt mye tid på skjønne hvordan ting funker så oppsto det ett fint fundament som kunne brukes overalt. Random state er satt til 0 for å produsere like resultat. Jeg har valgt å navngi test og treningsdataen med noe som er mer forståelig. F.eks istedenfor y_train heter den test_labels. Dette hjalp veldig på forståelsen for min egen del. Helt nederst ligger kode for en finere version av confusion matrix. Jeg har valgt å ikke kommentere koden fordi det er såpass lite og lett å forstå. Jeg mener det ikke er noe som trenger forklaring.

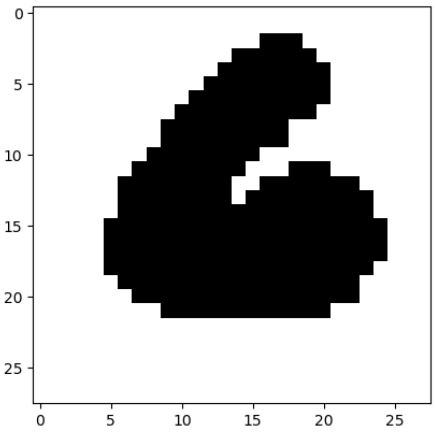


SVM

Support Vector Machines er en supervised maskinlæringsalgortime som kan bli brukt innen klassifisering og regression Men mest brukt innen klassifisering. SVM finner en "hyper-plane"(linje i 2d, plane i 3d) som separer treningsdataen på en slik måte at avstanden mellom "hyper-plane" og de nærmeste punktene fra vær klasse er maksimert.

Treffsikkerheten er nokså tilfeldig visst man bruker bildene som de er. De ulike punktene er av forskjellig gråfarge. Om man setter verdien til 1 på alle punkter så skyter treffsikkerheten til himmels. Gjøres med:

```
test_images[test_images>0]=1
train_images[train_images>0]=1
```



Antall bilder	Treffsikkerhet
500	86 %
5000	91 %
20000	95 %
40000	96 %

Jeg er nokså fornøyd med treffsikkerheten. Den gir som vist 86% ved kun 500 bilder. Jeg ligger til sammenligning på rundt 54 % ved 500 bilder når jeg bruker ett nevralt nettverk. Den fungerer da greit med lite treningsdata.
På de to største ligger den på fantastiske 95 og 96%. Men det tar tid..

K-nearest neighbor
Den klassifiserer nye punkt utifra avstanden til de ulike klassene. Altså, ett nytt punkt vil prøve å finne ut hvilket klaser som er nærmest seg selv. Her har jeg prøvd å teste litt forskjellige verdier av K. Men som vanlig får jeg hakket dårligere resultat om jeg velger noe annet enn default-parametere. Hm. Ved 500 bilder gir den 77 %, ikke så altfor ille. Ved 5000 bilder hopper den til 91%, nokså bra. Det tar nokså lang tid ved testing med både 20000 og 40000 bilder. Dette fordi algoritmen beregner avstanden til alle punktene. 96 % vil jeg si er temmelig bra for en så simpel algoritme. Men jeg kan steke egg på pcen mens jeg venter på det resultatet.
Testet i etterkant med andre måter å beregne avstand på. Både, “euclidian”, “manhattan” og “chebyshev” ga dårligere resultat enn minkowski. Valgte derfor å ikke inkludere dem i tabellen.

Antall bilder	Treffsikkerhet (minkowski)
500	77 %
5000	91 %
20000	95 %
40000	96 %

Gaussian Naive bayes
Det er en klassifiseringsmetode som er basert på Bayes Theorem. En Naive Bayes klassifiserer går ut ifra at ulike “features” ikke er relatert til andre features. Den bruker lite regnekraft og derfor rask. Jeg trodde da først at denne ville være godt egnet til å trene på mye data. Som man ser i tabellen under så synker treffsikkerheten veldig etter man runder ca 500 bilder. Dette synes jeg var rart. Etter å ha undersøkt litt mener jeg dette er ett godt eksempel på overfitting. Etterhvert som datasettet blir mer komplekst vil den tilpasse støy i treningsdataen, ulike mønster som ikke vil komme tilbake. Altså, den tilpasser seg så godt treningsdataen at den ikke klarer å forutsi ny data.

	Treffsikkerhet	**Treffsikkerhet**	**Treffsikkerhet**
Antall bilder	Gaussian	Bernoulli	Multinominal
500	70 %	81 %	81 %
5000	57 %	82 %	81 %
20000	55 %	83 %	82 %
40000	56 %	84 %	83 %

At den makser på 70% er ikke så imponerenes. Men det er vel og merke med en simpel metode ved bruk av kun 500 bilder. Jeg syntes 70% var endel mindre enn hva jeg har sett i andre eksempler. Valgte derfor å teste med bernoulli og multinomial som modell. Ettersom jeg leste at Bernoulli Naive Bayes blir brukt kun til “binære features” så tenkte jeg da at det her må jo være bra! Det var det! Treffsikkerhet steg med mellom 10 og 30 % over heile fjøla. Alt ettersom hvor mye treningsdata som ble brukt.

Decicion trees
Det er en type model brukt for både klassifisering og regression. Treet besvarer etterfølgende spørsmål som så sender oss nedover treet utifra svaret. Litt som en noe lengre if/else setning. I og med at random forest er en samling av decicion trees der resultatetene blir slått sammens til ett endelig resultat så valgte jeg å inkludere den også. Tenkte nesten det ble litt rart uten. Bruk av random forest minsker feil pga av “fordommer” og feil pga store forskjeller mellom ulik data.

Treffsikkerhet med 500 bilder ved bruk av ett tre ligger på 61 %. Med andre ord responder den dårlig på lite treningssdata. Men det gjør opp for seg ved å regne fort, også på større mengder data. Ved 40000 bilder ligger den på 86 %. Ved bruk av random forest(mange trær) så øker den med ca 10 % over hele linjen. Trodde først det ville ta fryktelig lang tid med random forest. Men det går faktisk overraskende fort. 94% med 40000 bilder! Ganske bra!

Gradient boosting så jeg ble nevnt i en en slide fra forelesning. Måtte jo bare teste. Gir en god del bedre resultat. 14% bedre en decicion tree og 4% bedre enn random forest ved bruk av 5000 bilder. Fikk ikke testet med mer data fordi det tok for lang tid. Den er muligens god på mellomstore datasett der tid ikke er en faktor.

Antall bilder	Decicion tree	Random forest	Gradient boosting
500	61 %	73 %	74 %
5000	77 %	86 %	90 %
20000	82 %	92 %	
40000	86 %	94 %	

Logistic regression
Jeg oppdaget i etterkant at vi ikke har gått gjennom dette. Men vi har hatt om lineær regression som er nesten det samme. I motsetning til lineær regression har logistic regression ett begrenset antall mulige utfall. Nå skal jeg innrømme at jeg prøvde å få lineær regression til å funke med datasettet. Men ser nå at den ikke egner seg til klassifisering. Resultet vil jeg si er ganske greit, litt bedre enn Naive Bayes. 91% ved 40000 bilder er det nesten bra vil jeg si.

Antall bilder	Treffsikkerhet
500	80 %
5000	84 %
20000	89 %
40000	91 %

Neural network
Nevrale nett er laget med enheter som gjør beregninger, altså ett nevron. Alle disse har forbindelser til de ulike lagene. Dette nettverket transformerer data helt til den kan klassifisere den som output. Det er ikke en algoritme, men en samling av ulike algoritmer. Jeg har hørt mye bra om nevrale nett og bildegjenkjenning. Så derfor for meg at denne vil gå seirende ut av testingen. Det gjorde den nesten…med 40000 bilder. Det som er så flott med nevrale nett er at de blir bedre og bedre ettersom hvor mye treningssdata de mates med. Hadde jeg trent på hele settet og brukt test filen til testing så tror jeg den hadde økt med ett par prosent til iallfall. Det er også godt mulig resultetet hadde vært bedre om parameterene hadde blitt finjustert. Det å skulle velge ting som hidden layers er ikke lett.

Antall bilder	Treffsikkerhet
500	54 %
5000	80 %
20000	92 %
40000	94 %

Konklusjon
Utifra antall bilder så går SVM av med førsteplass. KNN og SVM ligger 2% høyere enn nevralt nett ved 40000 bilder. Ulempen er at de bruker så hinsides lang tid på å gjøre opp en mening.

Antall bilder	Vinner
500	SVM(86%)
5000	K-nearest neighbor/SVM (91%)
20000	K-nearest neighbor/SVM (95 %)
40000	K-nearest neighbor/SVM (96 %)

Men!
Av de som gir gode resultat og bruker kort tid på å beregne større mengder, så er det to som skiller seg ut. Random forest og neural network.

Antall bilder	Random forest	**Antall bilder**	Neural network
500	73 %	**500**	54 %
5000	86 %	**5000**	80 %
20000	92 %	**20000**	92 %
40000	94 %	**40000**	93,88 %

Jeg har til nå rundet av til nærmeste heltall. Om man tar med to desimaler så vinner random forest marginalt ved bruk av 40000 bilder. Den vinner derfor på alle utifra antall bilder. Treffsikkerheten til det nevrale nettet ville nok slått random forest visst mengden data var større. Etterhvert som datakraften øker og vi får større mengder data så vil nok de nevrale nettene bli fremtidens vinner. Jeg skulle så inderlig ønske jeg implementerte noe som målte brukt tid fra start til slutt. Men kom på det litt vel sent. Jeg vil også nevne at denne testen ikke er helt god. Den rettferdiggjør ikke de ulike algoritmene. Dette fordi det er veldig mye som kan finjusteres. Det er også noen algoritmer som varier litt fra tid til anen, selv ved lik trenings og testdata. Hm. De ulike resultatene ville nok sett noe annerledes ut om alt var implementert optimalt.
Jeg har lært veldig mye i løpet av denne innleveringen og ser meg derfor fornøyd med arbeidet.

Datasett: https://www.kaggle.com/c/digit-recognizer/data

Kilder:

https://www.youtube.com/watch?v=j9_yzC-x-js
https://stackoverflow.com/questions/12146914/what-is-the-difference-between-linear-regression-and-logistic-regression
https://www.youtube.com/watch?v=JcI5Vnw0b2c
https://www.youtube.com/watch?v=KTeVOb8gaD4
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
https://www.youtube.com/watch?v=oYbVFhK_olY
https://www.kaggle.com/archaeocharlie/a-beginner-s-approach-to-classification
https://www.youtube.com/watch?v=aZsZrkIgan0
https://en.wikipedia.org/wiki/Artificial_neural_network
https://towardsdatascience.com/logistic-regression-using-python-sklearn-numpy-mnist-handwriting-recognition-matplotlib-a6b31e2b166a
https://www.youtube.com/channel/UCfzlCWGWYyIQ0aLC5w48gBQ
https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/#nine
https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
https://scikit-learn.org/stable/auto_examples/svm/plot_iris.html
https://en.wikipedia.org/wiki/Naive_Bayes_classifier
https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/
https://towardsdatascience.com/decision-tree-ensembles-bagging-and-boosting-266a8ba60fd9
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
https://blog.statsbot.co/neural-networks-for-beginners-d99f2235efca
https://towardsdatascience.com/hype-disadvantages-of-neural-networks-6af04904ba5b
Slides fra forelesning

#Oppdaget i etterkant at jeg muligens har rotet litt med detaljene, tror det skal være fikset. Men er #så trøtt at det kan være noe feil her og der