# Handin 2

Machinea Learning 2016

Af Jonas Herskind Sejr (20021728)(Group 9)

Denne rapport er en fortsættelse af en tidligere handin 1.
I handin 1 var målet at klassifisere håndskrevne tal fra et til ti ved brug af logistic regression.
Med logistic regression opnåede jeg en fejlrate på 7.46% hvilket ikke er specielt godt.
Denne gang bruges i stedet support vector machines og neurale netværk.
I den tidligere opgave arbejde jeg både på MNIST datasættet og et datasæt kreeret, som en del af machine learning undervisningen på Århus universitet over de seneste år. I denne opgave vil jeg kun se på datasættet udviklet, som en del af undervisningen og jeg vil kalde dette dataset AU datasættet.
Til evaluering af modellerne er funktionen `classification_report` i `sklearn.metrics` brugt. Den giver tre mål for kvaliteten af klassifikationen:
Precision som defineres til true positives over true positives plus false positives.

$$P = \frac{T_p}{T_p+F_p},$$

Recall som defineres til true positives over true positives plus false negatives .

$$R = \frac{T_p}{T_p+F_n}$$

Og F1 score, som defineres til harmonic mean of precision and recall.

$$F1 = 2\frac{P \times R}{P+R}$$

I denne opgave bruges precision, som for multiclass klassifisering beskriver gennemsnittet af precision for de enkelte klasser. Antager vi at talene er uniformt fordelt vil dette være det samme som andelen af klassificeringerne, der er korrekte.

# OCR med SVM

I dette afsnit beskrives en række eksperimenter med SVM.
Der er blevet lavet forsøg med lineære, radial basis og polynomiske kernels.
Til model selection inden for hver af de tre typer af kernels er brugt grid search med cross validation, som fåes ved hjælp af Python klassen `GridSearchCV.` Modellerne evalueres med multiclass precision, som beskrevet ovenfor (precision_macro i python).

## Lineær kernel

```
Best parameters set found on development set:

{'kernel': 'linear', 'C': 1}

Grid scores on development set:

0.917 (+/-0.005) for {'kernel': 'linear', 'C': 1}
0.915 (+/-0.005) for {'kernel': 'linear', 'C': 10}
0.915 (+/-0.005) for {'kernel': 'linear', 'C': 100}
0.915 (+/-0.005) for {'kernel': 'linear', 'C': 1000}

Detailed classification report:
             precision    recall  f1-score   support

       0      0.96   0.99   0.97         258
       1      0.90   0.92   0.91         258
```

```
          2       0.93      0.94      0.94         258
          3       0.90      0.93      0.91         258
          4       0.93      0.93      0.93         258
          5       0.90      0.89      0.89         258
          6       0.96      0.96      0.96         258
          7       0.94      0.95      0.95         258
          8       0.90      0.84      0.87         258
          9       0.89      0.87      0.88         258

avg / total           0.92      0.92      0.92      2580
```

Af ovenstående ses det at den bedste model med lineær kernel har parametren C = 1 og at denne model rammer 91,7 % af billederne in sample (på datasættet brugt til cross validation) og ca. det samme out of sample 92%.
Det lader altså ikke til at modellen er overfitted, hvilket heller ikke var forventet med en meget simple model (lav VC dimension). Det må derfor også forventes at en model med højere kompleksitet vil kunne performe bedre.

## Polynomisk kernel 2 grad

Best parameters set found on development set:

{'gamma': 0.001, 'kernel': 'poly', 'C': 1000, 'degree': 2}

Grid scores on development set:

0.880 (+/-0.004) for {'gamma': 0.001, 'kernel': 'poly', 'C': 1, 'degree': 2}
0.764 (+/-0.013) for {'gamma': 0.0001, 'kernel': 'poly', 'C': 1, 'degree': 2}
0.932 (+/-0.003) for {'gamma': 0.001, 'kernel': 'poly', 'C': 10, 'degree': 2}
0.779 (+/-0.014) for {'gamma': 0.0001, 'kernel': 'poly', 'C': 10, 'degree': 2}
0.957 (+/-0.006) for {'gamma': 0.001, 'kernel': 'poly', 'C': 100, 'degree': 2}
0.880 (+/-0.004) for {'gamma': 0.0001, 'kernel': 'poly', 'C': 100, 'degree': 2}
0.959 (+/-0.004) for {'gamma': 0.001, 'kernel': 'poly', 'C': 1000, 'degree': 2}
0.932 (+/-0.003) for {'gamma': 0.0001, 'kernel': 'poly', 'C': 1000, 'degree': 2}

Error Rate:
0.0437984496124
Detailed classification report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.98 | 0.99 | 0.98 | 258 |
| 1 | 0.92 | 0.97 | 0.94 | 258 |
| 2 | 0.98 | 0.97 | 0.98 | 258 |
| 3 | 0.93 | 0.97 | 0.95 | 258 |
| 4 | 0.98 | 0.93 | 0.95 | 258 |
| 5 | 0.97 | 0.94 | 0.95 | 258 |

```
          6       0.98      0.97      0.98       258
          7       0.96      0.97      0.96       258
          8       0.94      0.93      0.94       258
          9       0.92      0.93      0.92       258

avg / total       0.96      0.96      0.96      2580

Training time: 4131.786187410355 seconds
```

Polynomier af anden grad er et supersæt af de lineære modeller og burde derfor in sample kunne performe bedre og, hvis vi kan undgå overfitting, også out of sample.
Det ses at dette også er tilfældet, da den bedste model ({'gamma': 0.001, 'kernel': 'poly', 'C': 1000, 'degree': 2}) ved crossvalidation har en precision på 95.9% in sample og vi får det samme out of sample, hvilket indikerer at der stadig ikke er tale om overfitting og potentiel mulighed for at bruge en endnu mere kompleks model.

## Poly grad 3

```
Best parameters set found on development set:

{'degree': 3, 'gamma': 0.001, 'C': 1000, 'kernel': 'poly'}

Grid scores on development set:

0.811 (+/-0.005) for {'degree': 3, 'gamma': 0.001, 'C': 1, 'kernel': 'poly'}
0.751 (+/-0.026) for {'degree': 3, 'gamma': 0.0001, 'C': 1, 'kernel': 'poly'}
0.900 (+/-0.013) for {'degree': 3, 'gamma': 0.001, 'C': 10, 'kernel': 'poly'}
0.751 (+/-0.026) for {'degree': 3, 'gamma': 0.0001, 'C': 10, 'kernel': 'poly'}
0.948 (+/-0.006) for {'degree': 3, 'gamma': 0.001, 'C': 100, 'kernel': 'poly'}
0.753 (+/-0.025) for {'degree': 3, 'gamma': 0.0001, 'C': 100, 'kernel': 'poly'}
0.962 (+/-0.004) for {'degree': 3, 'gamma': 0.001, 'C': 1000, 'kernel': 'poly'}
0.811 (+/-0.005) for {'degree': 3, 'gamma': 0.0001, 'C': 1000, 'kernel': 'poly'}

Error Rate:
0.0372093023256
Detailed classification report:
          precision       recall  f1-score   support

          0       0.98      0.99      0.99       258
          1       0.95      0.98      0.96       258
          2       0.99      0.97      0.98       258
          3       0.94      0.97      0.95       258
          4       0.98      0.95      0.96       258
          5       0.97      0.94      0.95       258
          6       0.98      0.97      0.98       258
          7       0.96      0.98      0.97       258
```

```
          8       0.95    0.95    0.95    258
          9       0.93    0.94    0.93    258

avg / total       0.96    0.96    0.96    2580

Training time: 5366.22421169281 seconds
```

Med den sidste polynomiske model opnås kun samme resultat for bedste model ({'degree': 3, 'gamma': 0.001, 'C': 1000, 'kernel': 'poly'}), som med 2. Grad polynomierne: ca. 96% in sample (validation set) og out of sample (test set).
Igen kunne det indikere at der kan bruges endnu mere komplekse modeller (4. 5. Grads polynomier), men som det også ses tager det lang tid at køre modellerne og fokus i denne opgave vil i stedet være på neurale netværk.

## Radial Basis Kernel

```
Best parameters set found on development set:

{'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}

Grid scores on development set:

0.917 (+/-0.008) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
0.870 (+/-0.004) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}
0.945 (+/-0.007) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.001}
0.914 (+/-0.009) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.0001}
0.949 (+/-0.004) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}
0.933 (+/-0.005) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}
0.949 (+/-0.005) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}
0.928 (+/-0.004) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}

Error Rate:
0.0527131782946
Detailed classification report:
             precision    recall  f1-score   support

          0       0.96    0.99    0.98    258
          1       0.93    0.95    0.94    258
          2       0.97    0.97    0.97    258
          3       0.92    0.95    0.94    258
          4       0.96    0.94    0.95    258
          5       0.96    0.91    0.93    258
          6       0.97    0.97    0.97    258
          7       0.95    0.96    0.96    258
          8       0.94    0.91    0.93    258
          9       0.91    0.93    0.92    258

avg / total       0.95    0.95    0.95    2580
```

| |
|---|
| Training time: 2410.673439025879 seconds |

Denne sidste gruppe af svm modeller opnår lidt dårligere resultater end polynomierne og er derfor mindre interessante baseret på ovenstående resultater.
Generelt er der flere metoder, som vil kunne forbedre resultaterne herunder udvidelse af datasættet med billed transformationer, parametre til læringsalgoritmen osv., men jeg vil i stedet gå videre og se på neurale netværk.

# OCR with Neural Networks

Ser man på top resultater på OCR med MNIST datasættet (http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html) er neurale netværk overlegne. Anden halvdel af rapporten dedikeres til denne type classifier.
Den første opgave er ifølge opgaveformuleringen at afprøve et simpelt netværk med et hidden layer. Herefter er opgaven at finde den bedst mulige classifier.
Alle modeller er implementeret i Googles TensorFlow og til alle modeller er brugt batch gradient descent med Adam Optimizer (tf.train.AdamOptimizer) til hvert step.
Da min computer kører Windows er modellerne kørt på en virtuel maskine med Linux. Dette har gjort at modellerne har taget meget lang tid at køre, da den kun har haft 2 kerner til rådighed og ingen GPU.

## Simpelt 3 lags netwærk

Det første netværk, der er blevet lavet eksperimenter med er et tre lags netværk med et hidden layer. Alle modellerne er trænet med batch size 50 og 200 iterationer.
Fordi det har taget meget lang tid at køre modellerne og der er mange parametre at skrue på er parametrene varieret en ad gangen ud fra følgende default parametre:
{reg_rate=0, hidden_layer_breadth=784, drop_out_keep_prop=1},
hvor reg_rate er L2 regulariserings parametren, hidden_layer_breadth er breden af hidden layer og  drop_out_keep_prop er sandsynligheden for at en node bliver i grafen for et givet batch step i gradient descent.
I tabellen herunder ses resultatet. Kun parametre der afviger fra default er vist.

| **Configuration** | Error Rate |
|---|---|
| | 0.0563583815029 |
| reg_rate=0.0001 | 0.0568400770713 |
| reg_rate=0.001 | 0.0679190751445 |
| reg_rate=0.01 | 0.196050096339 |
| drop_out_keep_prop=0.01 | 0.231695568401 |
| drop_out_keep_prop=0.1 | 0.0895953757225 |

| | |
|---|---|
| drop_out_keep_prop=0.5 | 0.0472061657033 |
| drop_out_keep_prop=0.6 | 0.0462427745665 |
| drop_out_keep_prop=0.7 | 0.0467244701349 |
| hidden_layer_breadth=484 | 0.142100192678 |
| hidden_layer_breadth=1084 | 0.0534682080925 |

I ovenstående tabel kan det ses at L2 regularisering gør modellen dårligere.
Det ses også at gøres hidden layer bredere, hvilket giver en højere kompleksitet i modellen, bliver den bedre, hvor imod den bliver værre hvis hidden layer gøre mindre.
Det ser derfor ud til at den meget simple model med 784 brede på hidden layer er for simpel.
Drop out giver derimod en forbedring, hvilket understreger at regularisergin ikke bare er regularisering. Effekten af L2 regularisering er en preference på modeller med færre vigtige (med høje værdier) noder. Drop out giver derimod en robusthed, fordi hver batch forhindres i at bruge et antal features og derfor må modellen flades ud og bruge flere forskellige features.
Da det af ovenstående virkede til at mere kompleksitet i modellen ville være en fordel gik jeg hurtigt videre til dybe netværk, med en ide om at drop out ville fungere bedre en L2 regularisering.

## Deep and Convoluitonal networks

Efter at have afprøvet det simple neurale netværk afprøvede jeg et dybere neuralt netværk samt det convolutional netværk, der er beskrevet i “Deep MNIST for Experts” og konstaterede at mit convolutional netværk var bedst. Da der ikke var så meget tid (da modellerne tager meget lang tid at køre på to virtuelle kerner) gik jeg direkte i gang med at optimere og eksperimentere med convolutional netværk.
Ud af boksen gav netværket en præcision på 97.5%.
Det første netværk bestod af følgende lag:

1. Et indput lag på 784 noder
2. Et 5*5 convolutional lag med 32 kanaler og ReLu activation function
3. Et 2*2 max_pooling lag (tager max i hver 2*2 område. Outputter derfor 14*14*32)
4. Et 5*5 convolutional lag med 64 kanaler og ReLu activation function
5. Et 2*2 max_pooling lag (tager max i hver 2*2 område. Outputter derfor 7*7*32)
6. Et simpelt hidden layer med 1024 out
7. Et output layer med 10 out
8. Softmax

Herfra er lavet en række eksperimenter, som ikke er dokumenteret med data, da fokus har været på at forbedre modellen. Følgende ændringer er lavet på modellen.
Bredden af det første convolutional layer er parametriseret, og der er blevet lavet eksperimenter med flere kanaler. Uden andre ændringer gav dette ingen forbedring.
Bredden af det simple hidden layer er parametriseret og der er eksperimenteret med forskellige bredder, hvilket heller ikke med AU datasættet og default parametre gav bedre resultater.

Herefter blev drop out introduceret over det simple hidden layer, hvilket gav en lille forbedring.
For at kunne træne modellen hurtigere introduceredes batch normalization over det simple hidden layer, hvilket gav en tydelig forskel på, hvor hurtigt modelen konvergerede.
Til sidst blev data sættet udviddet ved at dreje billedet tilfældigt mellem -15 og 15 grader og derefter flyttet tilfældigt mellem -5 og 5 pixels på både y og x aksen.
Da modellerne tog meget lang tid at køre, er det ikke lavet gridsearch, men i stedet har jeg vurderet, hvad der skulle til for at forbedre modellen ud fra følgende principper.

1. Hvis datamængden øges skal modellen have større komplexitet
2. Hvis insample er lavere end out of sample skal, der regulariseres mere eller modellen skal simplificeres
3. Mere data er bedre

Alle modellerne er sammenlignet på et validerings data sæt, som i starten bliver udtaget som 20 % af training data.
Den endelige model er sendt med som zip fil og giver på test AU datasættet en precision på

**0.985**

## Retrospektiv

Når jeg ser tilbage på forløbet valgte jeg tidligt at køre på en virtuel maskine. Det kan jeg se nu var en fejl, da modellerne tog alt for lang tid at køre. Det havde været godt givet ud at sætte en server  med GPU support op i starten.
Det er vigtigt at have masser af power når man arbejder med neurale netværk.
Havde jeg haft længere tid ville jeg også gerne have prøvet med endnu mere genereret data og måske også have forsøgt at transformere MNIST til et format der ligner AU datasættet.
Det er der ikke noget at gøre ved nu andet end at nyde hvad man har lært :)