Project AI (INFPRJ01)

Versie 1.0
Final Version
9 April 2023

Comment Checker

Sentiment Analysis

Hakan Dalama (0963609)

Rotterdam University of Applied Sciences

Inhoudsopgave

Inleiding

Dit is het einddocument voor de minor AI, dit document is volledig herschreven. Ik ben de afgelopen maanden druk bezig geweest om verschillende modellen te implementeren. In de volgende hoofdstukken valt de probleemstelling te lezen, alsmede de resultaten, tools, etc. Verder is er een conclusie getrokken op basis van de gegevens, en opgeschreven wat er geleerd is van dit project. Tot slot staan de contributies vermeld, en in de bijlage is het onderzoek te vinden en de volledige documentatie van alle code die is geschreven.

Probleemstelling en achtergrond

Probleemstelling

Het probleem is dat er in de huidige samenleving veel kritiek wordt geleverd op mensen, sommige mensen kunnen dit makkelijker aan dan anderen. Het AI model dat ik wil creëren zou ervoor moeten zorgen dat negatieve comments worden gevonden en er uit worden gefilterd. Dit gebeurt met gelabelde data. In de eerste instantie heb ik een eigen dataset opgesteld met ongeveer 400 samples, dit was niet voldoende dus heb ik besloten om een dataset met 50.000 IMDB film reviews te nemen om zo beter te kunnen trainen. De dataset bevat 25.000 negatief gelabelde reviews en 25.000 positief gelabelde reviews. De dataset is afkomstig van de website Kaggle.

Criteria

In het voorafgaande onderzoek zijn er vooral base models getest en gemaakt en alleen op accuracy getest, dit zal in dit onderzoek beter aangepakt worden en er zullen betere modellen gemaakt worden. De parameters zullen worden aangepast en getoetst op welke het beste schikt bij een model. De keuzes voor bepaalde parameters zullen ook worden onderbouwd. Er zullen ook meer dan 1 model (ML en DL) worden gemaakt en die worden met elkaar vergeleken. Fouten zullen worden bijgehouden en zullen worden beschreven waarom wat fout ging en hoe het is opgelost.

- Methoden

• Beschrijf de methoden die je hebt verkend en onderbouw je methoden.
• Begin met het verzamelen van data, vervolgens schoonmaken en repareren, dan transformatie, dan analyse en eventuele visualisaties die je hebt gemaakt.
• Parameterkeuzes kunnen een grote invloed hebben op de prestaties. Zorg ervoor dat je weet welke parameters (vooral standaardinstellingen) je hebt gebruikt en dat ze redelijk goed werken met jouw data.
• Zorg ervoor dat je elke methode die je hebt geprobeerd opneemt, zelfs als het niet "werkt" of niet zo goed presteert als jouw uiteindelijke aanpak. Bij het beschrijven van methoden die niet werkten, maak duidelijk hoe ze faalden.

- Resultaten

• Geef een gedetailleerde samenvatting van de resultaten van jouw werk. Hier specificeer je de exacte prestatie-maatregelen die je hebt gebruikt. Zorg ervoor dat je jouw maatregel(en) rechtvaardigt in termen van de doelstellingen van jouw project. Meestal zal er een soort nauwkeurigheids- of kwaliteitsmaatregel zijn. Geef je resultaten over enkele variaties van jouw oplossing, zoals verschillende typen modellen of verschillende parameters keuzes.
• Gebruik visualisaties en grafieken wanneer mogelijk.

- Tools

Beschrijf de tools die je hebt gebruikt en de redenen voor hun keuze. Onderbouw ze in termen van het probleem zelf en de methoden die je wilt gebruiken.
Hoe heb je ze ingezet? Welke functies werkten goed en welke niet?

- Geleerde lessen

Vergelijk de prestaties van jouw baseline model en primaire model en bespreek/verklaar de verschillen.
Idealiter geef je resultaten over enkele variaties van jouw oplossing, zoals verschillende typen modellen of verschillende parameter keuzes.
Gebruik visualisaties en grafieken wanneer mogelijk. Inclusief links naar interactieve visualisaties als je deze hebt gemaakt.

- Contributies

Geef een percentage van wat er is gedaan door de groepsgenoten, hierbij zal ik gebruik maken van de logboeken, zorg er voor dat iedereen het er mee eens is

Achtergrond

Zoals al deels aangegeven in de probleemstelling, is het tegenwoordig steeds makkelijker om je negatief uit te laten op social media. En het ergste is dat de boosdoeners vaak niet eens gestraft worden. Ze ontlopen vrijwel altijd de consequenties, terwijl degene op wie de comment is gericht mentaal beschadigd achterblijft. Om dit in de toekomst te voorkomen wil ik een model ontwikkelen waarbij negatieve comments eruit worden gefilterd. Op die manier zullen veel mensen zich mentaal beter in hun vel voelen als de social media platformen dit model toepassen.

Methoden

Er zijn verschillende methodes gebruikt voor dit project. Hieronder zie je in het algemeen welke methodes en waarom deze methodes zijn gebruikt. De volledige code documentatie met elk onderdeel staat vermeld in *Bijlage 2 - Code Documentatie.*

Machine Learning modellen (Naïve Bayes, Random Forest en SVM)

Gegevensverzameling:
Verzamel de dataset met reviews en hun sentiment labels.

Gegevens Voorbereiding:
1. Importeer de benodigde library's, waaronder pandas en scikit-learn.
2. Laad de dataset in een pandas-dataframe.
3. Maak de gegevens schoon door eventuele ongewenste tekens, cijfers of symbolen uit de tekst te verwijderen.
4. Converteer de tekst naar numerieke functie vectoren met behulp van de TfidfVectorizer-klasse van scikit-learn.
5. Splits de dataset in trainings- en testsets met behulp van de train_test_split-functie van scikit-learn.

Modeltraining en -evaluatie:
1. Train de Naïve Bayes-, Random Forest-modellen met verschillende aantallen estimators en SVM-modellen met zowel lineaire als RBF-kernels op de trainingsset.
2. Evalueer de prestaties van elk model met behulp van de testset en bereken resultaten zoals nauwkeurigheid, precision, recall en F1-score.
3. Kies het best presterende model op basis van de resultaten.

Deep Learning modellen (CNN)

Gegevensverzameling:
Verzamel de dataset met beoordelingen en hun overeenkomstige sentiment labels.

Gegevens Voorbereiding:
1. Importeer de benodigde library's, waaronder pandas, scikit-learn en Keras.
2. Laad de dataset in een pandas-dataframe.
3. Maak de gegevens schoon door eventuele ongewenste tekens, cijfers of symbolen uit de tekst gegevens te verwijderen.
4. Tokenize de tekst en converteer ze naar sequenties van getallen.
5. Vul de sequenties aan tot een vaste lengte.
6. Splits de dataset in trainings- en testsets.

Modeltraining en -evaluatie:
1. Definieer een CNN-model met behulp van Keras.
2. Compile het model met een loss-function en optimizer.
3. Train het model op de trainingsset.
4. Evalueer de prestaties van het model met behulp van de testset en bereken resultaten zoals nauwkeurigheid, precision, recall en F1-score.

Hieronder is een tabel te zien met de modellen die zijn gemaakt met de bijbehorende kenmerken voor het model. er zijn een aantal wijzigingen van hyperparameters er uit gelaten omdat deze steeds geen/nauwelijks invloed hadden op voorspellingen, deze wijzigingen zijn wel terug te vinden in de code.

Model	**Characteristics**
Naïve Bayes	base
Random Forest	n_estimators: ● 50 ● 100 ● 150
SVM	Kernels: ● Linear ● RBF
CNN Model 1	**Sequential** Layers: ● Embedding (5000,32) & input length 200 ● Flatten (2D) ● Dense (128) ReLU ● Dropout ● Dense (1) Sigmoid Optimizer: Adam (0,001)
CNN Model 2	**Sequential** Layers: ● Embedding (5000,32) & input length 200 ● Flatten (2D) ● Dense (128) ReLU ● Dense (64) ReLU ● Dense (32) ReLU ● Dropout ● Dense (1) Sigmoid Optimizer: Adam (0,0001)
CNN Model 3	**Sequential** Layers: ● Embedding (5000,32) & input length 200 ● Flatten (2D) ● Dense (128) ReLU ● Dense (64) ReLU ● Dense (32) ReLU ● Dropout ● Dense (1) Sigmoid Optimizer: Adam (0,0001)

CNN Model 4	**Sequential** Layers: ● Embedding (5000,128) & input length 500 ● Convolutional (filter = 128, kernel_size = 3) ReLU ● Global Max Pool ● Dense (1) Sigmoid Optimizer: Adam (0,0001)
CNN Model 5	**Sequential** Layers: ● Embedding (5000,128) & input length 500 ● Convolutional (filter = 64, kernel_size = 3) ReLU ● Global Max Pool ● Dense (1) Sigmoid Optimizer: Adam (0,0001)
CNN Model 6	**Sequential** Layers: ● Embedding (5000,128) & input length 500 ● Convolutional (filter = 128, kernel_size = 3) ReLU ● Convolutional (filter = 128, kernel_size = 3) ReLU ● Convolutional (filter = 128, kernel_size = 3) ReLU ● Global Max Pool ● Dense (1) Sigmoid Optimizer: Adam (0,0001)
CNN Model 7	**Sequential** Layers: ● Embedding (5000,128) & input length 500 ● Convolutional (filter = 128, kernel_size = 3) ReLU ● Convolutional (filter = 128, kernel_size = 3) ReLU ● Convolutional (filter = 128, kernel_size = 3) ReLU ● Global Max Pool ● Dense (64) ReLU ● Dense (32) ReLU ● Dense (1) Sigmoid Optimizer: Adam (0,0001)

Resultaten

- **Naïve Bayes:** Het model had een nauwkeurigheid van 0,87, wat redelijk goed is, maar niet erg hoog. De precision, recall en F1-scores waren allemaal boven de 0,80, wat aangeeft dat het model redelijk goed presteerde. De confusion matrix laat zien dat het model relatief veel valse negatieven had, wat suggereert dat het mogelijk niet het beste model is voor deze specifieke dataset.

	precision	recall	f1-score	support
0	0.86	0.88	0.87	4961
1	0.88	0.85	0.86	5039
accuracy			0.87	10000
macro avg	0.87	0.87	0.87	10000
weighted avg	0.87	0.87	0.87	10000

```
[[4352  609]
 [ 737 4302]]
```

Scores Naïve Bayes

- **Random Forest:** Het random forest-model met 50 estimators had een nauwkeurigheid van 0,841, wat hoger is dan het decision tree-model. De precision, recall en F1-scores waren allemaal boven de 0,84, wat aangeeft dat het model goed presteerde. De confusion matrix laat zien dat het model relatief weinig valse negatieven had (708), wat suggereert dat het een goed model is voor deze dataset. Het verhogen van het aantal estimators naar 150 verlaagde de nauwkeurigheid iets naar 0,865, maar de precision, recall en F1-scores bleven boven de 0,85. Het verhogen van het aantal estimators naar 150 verbeterde de nauwkeurigheid tot 0,865, waarbij alle andere evaluation metrics ook verbeterden.

	precision	recall	f1-score	support
0	0.86	0.86	0.86	4961
1	0.86	0.86	0.86	5039
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

```
[[4279  682]
 [ 708 4331]]
```

Scores Random Forest

- **SVM:** Het lineaire SVM-model had een nauwkeurigheid van 0,8968, wat de hoogste is van alle tot nu toe geprobeerde ML modellen. De precision, recall en F1-scores waren allemaal boven de 0,89, wat aangeeft dat het model zeer goed presteerde. De confusion matrix laat zien dat het model relatief weinig valse negatieven (475) had, wat suggereert dat het een goed model is voor deze dataset. Het SVM-model met de

RBF-kernel had een iets hogere nauwkeurigheid van 0,9007, waarbij alle andere evaluation metrics hoog bleven.

	precision	recall	f1-score	support
0	0.91	0.88	0.90	4961
1	0.89	0.92	0.90	5039
accuracy			0.90	10000
macro avg	0.90	0.90	0.90	10000
weighted avg	0.90	0.90	0.90	10000

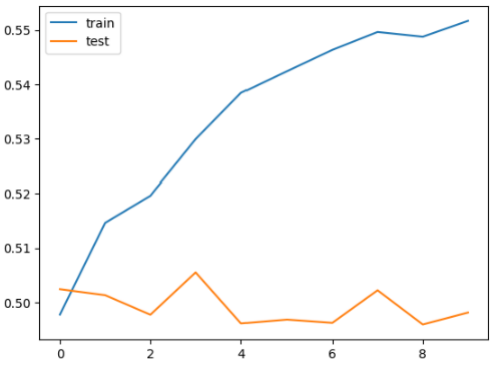
```
[[4389  572]
 [ 421 4618]]
```

Scores SVM

Over het algemeen presteerden de SVM-modellen het best op deze dataset, waarbij het lineaire SVM-model iets beter presteerde dan het SVM-model met de RBF-kernel. De random forest-modellen presteerden ook goed, maar niet zo goed als de SVM-modellen. Het decision tree-model presteerde redelijk goed, maar had relatief veel valse negatieven.

- **CNN Model 1:** Dit model had een erg lage nauwkeurigheid van 0,496, dit is lager dan als een model random zou voorspellen want dan is de kans 0,5. Hier moest dus aan gesleuteld worden om een betere nauwkeurigheid te krijgen. De confusion matrix toont het aantal true positives, false positives, true negatives en false negatives voor elke klasse. Het model voorspelde correct 4673 instanties van klasse 0 en 308 instanties van klasse 1, maar heeft 4731 instanties van klasse 1 verkeerd geclassificeerd als klasse 0 en 288 instanties van klasse 0 verkeerd geclassificeerd als klasse 1.

```
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.94      0.65      4961
           1       0.52      0.06      0.11      5039

    accuracy                           0.50     10000
   macro avg       0.51      0.50      0.38     10000
weighted avg       0.51      0.50      0.38     10000

Confusion Matrix:
 [[4673  288]
 [4731  308]]
```

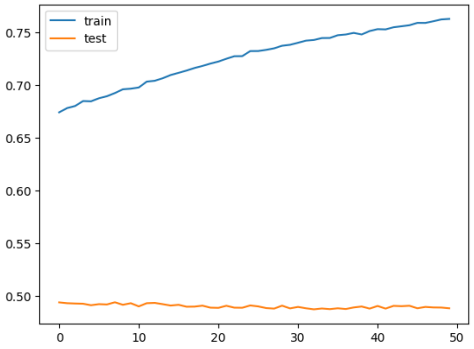


Scores en accuracy grafiek CNN Model 1

- **CNN Model 2:** Dit model had een lagere nauwkeurigheid van 0,492, dit was het model waar er was gesleuteld met de learning rates. In dit geval laat de confusion matrix zien dat er 2964 true negatives waren (samples correct voorspeld als 0), 1997 false positives (samples voorspeld als 0 maar waren eigenlijk 1), 3121 false negatives (samples voorspeld als 1 maar waren eigenlijk 0) en 1918 true positives (samples correct voorspeld als 1). Over het algemeen suggereren de evaluatie metrics dat het model niet erg effectief is bij het classificeren van de samples en dat

er ruimte is voor verbetering. De precision- en recall-waarden zijn relatief laag, wat aangeeft dat het model een aanzienlijk aantal valse voorspellingen maakt. De confusion matrix bevestigt dat het model moeite heeft om beide klassen correct te voorspellen. Verder onderzoek is nodig om de redenen achter de lage prestaties te begrijpen en het model te verbeteren.

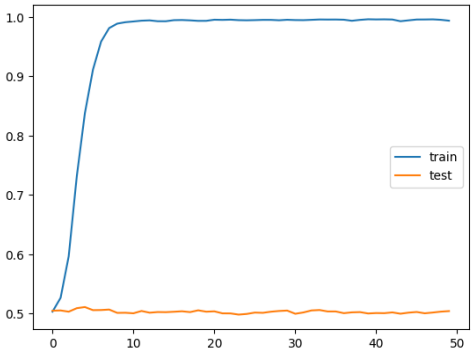
```
Classification Report:
              precision    recall  f1-score   support

           0       0.49      0.60      0.54      4961
           1       0.49      0.38      0.43      5039

    accuracy                           0.49     10000
   macro avg       0.49      0.49      0.48     10000
weighted avg       0.49      0.49      0.48     10000

Confusion Matrix:
 [[2964 1997]
 [3121 1918]]
```



Scores en accuracy grafiek CNN Model 2

- **CNN Model 3:** Hier ging de nauwkeurigheid weer iets omhoog door de aanpassingen in het model, maar niet significant genoeg, namelijk een nauwkeurigheid van 0,500. Als we naar het rapport kijken, is de precision voor klasse 0 0,50, wat betekent dat van alle samples die als 0 zijn voorspeld, 50% daadwerkelijk 0 waren. Op dezelfde manier is de recall voor klasse 0 0,50, wat betekent dat van alle werkelijke 0-samples slechts 50% correct als 0 zijn voorspeld. De F1-score voor klasse 0 is 0,50. Op dezelfde manier zijn de precision, recall en F1-score voor klasse 1 0,51, 0,51 en 0,51

```
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.50      0.50      4961
           1       0.51      0.51      0.51      5039

    accuracy                           0.50     10000
   macro avg       0.50      0.50      0.50     10000
weighted avg       0.50      0.50      0.50     10000

Confusion Matrix:
 [[2493 2468]
 [2492 2547]]
```

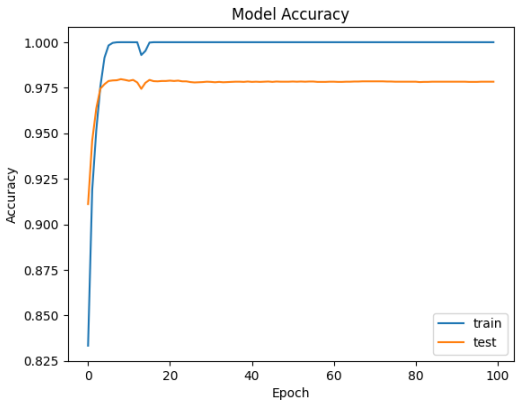


Scores en accuracy grafiek CNN Model 3

- **CNN Model 4:** Bij dit model kreeg ik eindelijk de resultaten die ik zocht, de accuracy was gestegen naar 0,897. De precision, recall en F1-score voor zowel positieve (1) als negatieve (0) sentimenten zijn ook dicht bij 0,9, wat aangeeft dat het model goed presteert bij het correct classificeren van zowel positieve als negatieve recensies. Bij het bekijken van de confusion matrix heeft het model 487 negatieve recensies foutief

geclassificeerd als positief (false positives) en 540 positieve recensies als negatief (false negatives).

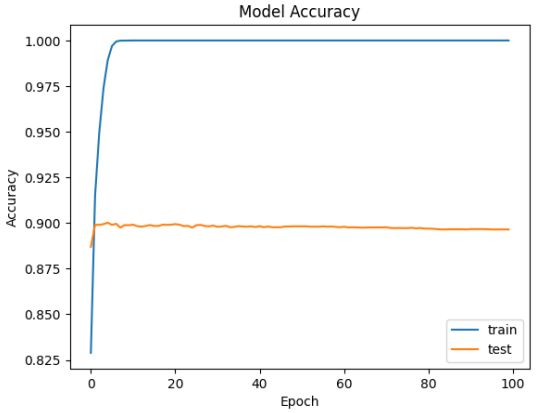
```
Test loss: 1.23
Test accuracy: 0.897
313/313 [==============================] - 0s 1ms/step
              precision    recall  f1-score   support

           0       0.89      0.90      0.90      5011
           1       0.90      0.89      0.90      4989

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000

[[4524  487]
 [ 540 4449]]
```



Scores en accuracy grafiek CNN Model 4

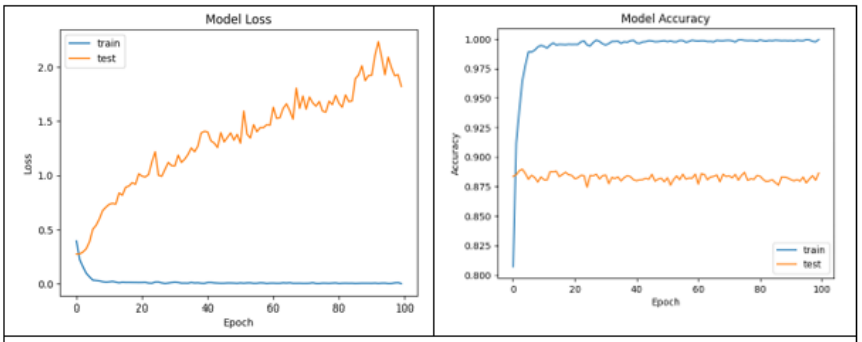
- **CNN Model 5:** Dit model had een kleine verbetering ten opzichte van het vorige model met het voorspellen van positieve segmenten. Hier kan je de overweging maken in welke scenario het model zou willen gebruiken (zijn er meer positieve of negatieve segmenten en wat is belangrijker om goed te voorspellen).Bij het bekijken van de confusion matrix heeft het model 488 negatieve beoordelingen foutief geclassificeerd als positief (false positives) en 547 positieve beoordelingen als negatief (false negatives).

```
Test accuracy: 0.896
313/313 [==============================] - 0s 1ms/ste
              precision    recall  f1-score   support

           0       0.89      0.90      0.90      5011
           1       0.90      0.89      0.90      4989

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000

[[4523  488]
 [ 547 4442]]
```



Scores en accuracy grafiek CNN Model 5

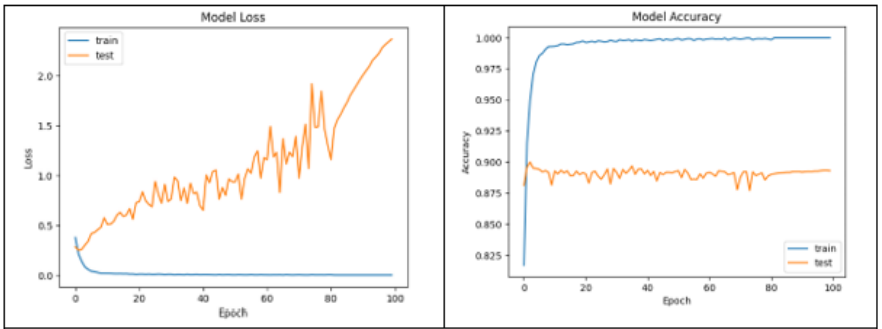
- **CNN Model 6:** Dit model presteerde iets minder goed dan het model 5, dit model laat dus zien dat niet altijd de prestaties worden verbeterd met het toevoegen van meerdere convolutional layers.




```
Test accuracy: 0.886
313/313 [==============================] - 1s 2ms/step
              precision    recall  f1-score   support

           0       0.89      0.88      0.89      5021
           1       0.88      0.89      0.89      4979

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

Scores en accuracy grafiek CNN Model 6

- **CNN Model 7:** Dit model presteerde weer iets beter maar niet het beste van allemaal, wel aardig in de buurt. De loss is ook iets hoger, wat betekent dat het model wel wat minder nauwkeurig zou zijn bij bepaalde voorspellingen dan het vorige model.




```
Test loss: 2.37
Test accuracy: 0.893
313/313 [==============================] - 1s 2ms/step
              precision    recall  f1-score   support

           0       0.91      0.88      0.89      5021
           1       0.88      0.91      0.89      4979

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

Scores en accuracy grafiek CNN Model 7

Hieronder zijn alle resultaten onder elkaar gezet om een beter overzicht te krijgen. Ook zijn de resultaten van de verschillende waardes van de machine learning modellen er bij gezet. In de code zijn de testresultaten te zien van eventuele andere aanpassingen van parameters, alleen bleek het vaak geen effect te hebben op het model en is daarom niet meegenomen in dit bestand met de resultaten. Deze resultaten zijn wel te vinden in de code.

Naïve Bayes

	precision	recall	f1-score	support
0	0.86	0.88	0.87	4961
1	0.88	0.85	0.86	5039
accuracy			0.87	10000
macro avg	0.87	0.87	0.87	10000
weighted avg	0.87	0.87	0.87	10000

```
[[4352  609]
 [ 737 4302]]
```

Random Forest

50

	precision	recall	f1-score	support
0	0.84	0.85	0.85	4961
1	0.85	0.84	0.85	5039
accuracy			0.85	10000
macro avg	0.85	0.85	0.85	10000
weighted avg	0.85	0.85	0.85	10000

```
[[4224  737]
 [ 796 4243]]
```

100

	precision	recall	f1-score	support
0	0.86	0.86	0.86	4961
1	0.86	0.86	0.86	5039
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

```
[[4279  682]
 [ 708 4331]]
```

150

	precision	recall	f1-score	support
0	0.86	0.87	0.86	4961
1	0.87	0.86	0.87	5039
accuracy			0.86	10000
macro avg	0.86	0.87	0.86	10000
weighted avg	0.87	0.86	0.87	10000

```
[[4294  667]
 [ 683 4356]]
```

SVM

Linear

	precision	recall	f1-score	support
0	0.90	0.89	0.90	4961
1	0.89	0.91	0.90	5039
accuracy			0.90	10000
macro avg	0.90	0.90	0.90	10000
weighted avg	0.90	0.90	0.90	10000

```
[[4404  557]
 [ 475 4564]]
```

RBF

	precision	recall	f1-score	support
0	0.91	0.88	0.90	4961
1	0.89	0.92	0.90	5039
accuracy			0.90	10000
macro avg	0.90	0.90	0.90	10000
weighted avg	0.90	0.90	0.90	10000

```
[[4389  572]
 [ 421 4618]]
```

CNN Model 1

```
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.94      0.65      4961
           1       0.52      0.06      0.11      5039

    accuracy                           0.50     10000
   macro avg       0.51      0.50      0.38     10000
weighted avg       0.51      0.50      0.38     10000

Confusion Matrix:
 [[4673  288]
 [4731  308]]
```

CNN Model 2

```
Classification Report:
              precision    recall  f1-score   support

           0       0.49      0.60      0.54      4961
           1       0.49      0.38      0.43      5039

    accuracy                           0.49     10000
   macro avg       0.49      0.49      0.48     10000
weighted avg       0.49      0.49      0.48     10000

Confusion Matrix:
 [[2964 1997]
 [3121 1918]]
```

CNN Model 3	Classification Report: precision recall f1-score support 0 0.50 0.50 0.50 4961 1 0.51 0.51 0.51 5039 accuracy 0.50 10000 macro avg 0.50 0.50 0.50 10000 weighted avg 0.50 0.50 0.50 10000 Confusion Matrix: [[2493 2468] [2492 2547]]
CNN Model 4	Test loss: 1.23 Test accuracy: 0.897 313/313 [==============================] - 0s 1ms/step precision recall f1-score support 0 0.89 0.90 0.90 5011 1 0.90 0.89 0.90 4989 accuracy 0.90 10000 macro avg 0.90 0.90 0.90 10000 weighted avg 0.90 0.90 0.90 10000 [[4524 487] [540 4449]]
CNN Model 5	Test accuracy: 0.896 313/313 [==============================] - 0s 1ms/step precision recall f1-score support 0 0.89 0.90 0.90 5011 1 0.90 0.89 0.90 4989 accuracy 0.90 10000 macro avg 0.90 0.90 0.90 10000 weighted avg 0.90 0.90 0.90 10000 [[4523 488] [547 4442]]
CNN Model 6	Test accuracy: 0.886 313/313 [==============================] - 1s 2ms/step precision recall f1-score support 0 0.89 0.88 0.89 5021 1 0.88 0.89 0.89 4979 accuracy 0.89 10000 macro avg 0.89 0.89 0.89 10000 weighted avg 0.89 0.89 0.89 10000

CNN Model 7	

```
Test loss: 2.37
Test accuracy: 0.893
313/313 [==============================] - 1s 2ms/step
              precision    recall  f1-score   support

           0       0.91      0.88      0.89      5021
           1       0.88      0.91      0.89      4979

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

Al met al lijkt het dat het CNN Model 4 & 5 het beste presteerden en uiteindelijk zullen worden gebruikt voor de tool, deze 2 modellen hadden een hoge nauwkeurigheid van ongeveer 0.9 en een relatief lage loss, ook de andere waarden waren goed aan beide modellen.

Tools

Omgeving

Ik programmeer in Python voor dit project. Ik heb een omgeving, interpreter of IDE nodig waar ik mijn code kan runnen. Google Collab was mijn keuze. Google bracht Collab, een populaire online IDE voor Python, uit in 2017. Google Collab kan je gebruiken voor het uitvoeren van deep learning en machine learning projecten.

Google Collab biedt veel functies, zoals:

• Schrijf, voer Python-code uit en werk samen.

• Het is eenvoudig om bestanden te maken, te distribueren en te downloaden.

• Externe dataset bestanden van Google Drive, Kaggle en andere bronnen kunnen worden geïmporteerd.

Het belangrijkste voordeel van het gebruik van Colab Notebook is dat het niet nodig is om alle bibliotheken op lokale systemen te downloaden, omdat ze al aanwezig zijn in het cloud netwerk en kunnen worden geïmporteerd. Bovendien biedt het een gratis cloud-GPU-service met een beperking van 12 uur. Het biedt 100 GB tijdelijk geheugen en 12 GB tijdelijke RAM.

Bovendien biedt de Colab-laptop drie processing units: lokaal, TPU en GPU, maar RAM en geheugen zijn beperkt in TPU en GPU.

Omdat Pandas, NumPy, Matplotlib en andere Python-bibliotheken al zijn geïnstalleerd in de Anaconda Jupyter Notebook, is het aanbod van Google Collab gunstig voor toepassingen voor deep learning. Naast het aanbieden van deze bibliotheken, wordt Google Collab ook geleverd met de vooraf geïnstalleerde machine learning-bibliotheken Keras, PyTorch en TensorFlow. Meerdere ontwikkelaars kunnen samenwerken aan hetzelfde project met behulp van Collab. Dat is erg handig voor groepsprojecten. Bovendien kan je je werk delen met andere ontwikkelaars, en omdat al je Collab-bestanden worden bewaard in uw Google Drive, kun je ze bekijken vanaf elk apparaat. Toegang tot GPU en TPU is ook beschikbaar via Google Collab. Meerdere processen kunnen worden uitgevoerd door de GPU (Graphic Processing Unit).

Andere software-opties

Naast Google Colab had ik ook verschillende mogelijkheden waar ik gebruik van had kunnen maken. Notebook wordt voornamelijk beschouwd als een van de oplossingen voor deep learning en machine learning model training voor Anaconda, Pycharm, Spider en Jupyter. Ze werken allemaal op lokale runtimes en maken gebruik van lokale systeemkracht.

Waarom Google Colab belangrijk is

Zoals eerder vermeld, heeft Google Colab gratis toegang tot de GPU en werkt het in de cloud met behulp van cloudgebaseerd netwerk-RAM en -opslag. De andere oplossingen, zoals Pycharm, Jupyter Notebook, Spider en Anaconda, maken gebruik van je lokale systeem. Hierdoor heeft men een apparaat nodig met sterkere componenten. Omdat ik geen krachtige machine heb met hele goede GPU en RAM waardoor de wachttijd langer is, dacht ik dat Google Colab een van mijn beste opties was. Daarom worden Google Drive en Colab gebruikt om alle codes en resultaten te maken.

Frameworks en bibliotheken

Deep learning projecten kunnen worden ontwikkeld met behulp van verschillende frameworks. Hieronder zal ik een paar open source frameworks doornemen die je kunt gebruiken om een model in Python te bouwen.

-TensorFlow

Het meest populaire deep learning framework is gecreëerd door het Google Brain team om de ontwikkeling van deep learning modellen mogelijk te maken met verschillende programmeertalen, zoals Python, R en C++. Google Translate maakt gebruik van het TensorFlow framework, dat functies bevat zoals tekstclassificatie, samenvatting, audio-, beeld- en handschriftherkenning. Effectieve netwerkmodellering en datavisualisatie worden geboden door TensorFlow. Een op Python gebaseerd framework genaamd TensorFlow wordt onderhouden door Google. TensorFlow ondersteunt GPU's sterk. Het wordt gebruikt om gegevens uit grafieken, SQL-databases en foto's te combineren.

-Keras

Een open source Python-bibliotheek genaamd Keras biedt een Python-interface voor kunstmatige neurale netwerken en wordt voornamelijk gebruikt in deep learning. Bovendien kan Keras worden gebruikt om diepe modellen te maken voor het web en Java Virtual Machines.

-PyTorch

PyTorch is een framework voor wetenschappelijk rekenen dat veel ondersteuning biedt voor machine learning-methoden. Het is een op Lua-based deep learning-systeem dat veel wordt gebruikt door grote bedrijven zoals Facebook, Google etc.

Ik heb de TensorFlow- en Keras-frameworks gebruikt in dit project; alle bibliotheken die in de code zijn gebruikt, waren voornamelijk afkomstig van TensorFlow en Keras. Bovendien heb ik de NumPy- en panda's-bibliotheken gebruikt voor wiskundige bewerkingen en het werken met dataframes. De layers en model architectuur voor de vooraf getrainde netwerken die ik in dit project heb gebruikt waren meestal afkomstig van de tensorflow- en keras-bibliotheken. Ook de optimizers en afbeeldingsgegevens generatie API van de Keras-bibliotheken werden gebruikt. Naast het gebruik van de open cv library heb ik het ook gebruikt voor het lezen en verwerken van afbeeldingen.

Trainen van modellen

Aangezien het een lange tijd kost om elk model te trainen voordat ik het resultaat kon krijgen, wordt het trainen van modellen beschouwd als een van de grotere geduld testende taken in deep learning. Dus naast het oplossen van codeerfouten, was een van mijn grootste problemen het trainen van de modellen en het wachten op de resultaten. Ik heb dit probleem opgelost door gewoon geduldig te wachten tot de training is voltooid.

Geleerde Lessen

Zoals bij de resultaten is beschreven kan je zien dat er 2 modellen goed presteerden, de SVM model met een RBF kernel en de CNN model 4&5. Het CNN model heeft veel tijd gekost om te maken en heeft lang geduurd om goed te trainen. Dit kan je zien aan het aantal modellen dat niet goed voorspelden. Zo zijn er verschillende dingen geprobeerd om uiteindelijk een goed presterend model te maken.

Het SVM model had al een aardige accuracy voor de gegeven dataset door gebruik te maken van een RBF kernel, het doel was dus om een beter presterende Deep learning model te creëren om beter te kunnen voorspellen.

In het begin bij het eerste sequential CNN model heb ik gebruik gemaakt van een embedding layer met een input dimensie van 5000, een output dimensie van 32 en een input lengte van 200, vervolgens werd de output van de embedding layer afgevlakt en doorgegeven aan de dense layer met een ReLU-activation function, daarna een drop out layer en vervolgens weer een dense layer met een sigmoid-activation function. Dit werd allemaal gedaan met een Adam-optimizer met een learning rate van 0,001. De resultaten vielen hiertegen dus ik ben gaan spelen met de parameters, zo heb ik de learning rate veranderd maar dat gaf niet hele goede resultaten.
Daarna heb ik meer dense layers toegevoegd met 32 en 64 neurons, maar hierdoor gingen de scores er maar een heel klein beetje op vooruit.

Ik ben het vervolgens iets anders gaan doen, omdat de waarden veranderen en meer layers toevoegen geen effect bleek te hebben. Ik heb een sequential model gemaakt met 1 convolutional layer (filter = 128, kernel_size = 3, activation = ’relu’) en 1 dense layer, de optimizer bleef de Adam-optimizer. Hieruit kwamen eindelijk goede resultaten, een gemiddelde accuracy van 0,896 dit was een grote sprong van de vorige accuracy van 0,5. vervolgens heb ik nog de dense layer neurons aangepast naar 64 om te kijken of dit effect zou hebben, op de accuracy had het een kleine positieve verandering namelijk 0,897. De confusion matrix liet zien dat het model beter was in het voorspellen van positieve comments, en iets slechter met negatieve.

Bij de eerste 3 modellen gebruikte ik dus een flatten 2D layer, toen ik het veranderde naar een layer met een globalmaxpool, werden de resultaten een stuk beter.

Dus welke je zou moeten kiezen ligt aan welk sentiment er vaker voor zou komen. Voor deze case zou dus de sequential CNN model met 128 dense layer neurons beter werken, omdat je liever zoveel mogelijk negativiteit eruit wilt filteren en dit model daarin beter presteert.
Deep learning modellen worden in het algemeen vaker gebruikt voor beeld- en spraakherkenning, Machine learning modellen presteren in het algemeen vaker beter op kleinere datasets en minder ingewikkelde functies. In dit geval presteerden ze bijna even goed alleen was het CNN model een stuk ingewikkelder en kostte meer tijd. Als ik in het vervolg een oplossing zou moeten bedenken voor een classificatie probleem met niet al te veel data zal dus de keuze sneller vallen op de SVM Machine learning model.

Contributies

Dit project was een verbetering van het oude project dat was gemaakt door het groepje dat bij het titelblad staat van het onderzoeksrapport. Het project was ver onder de maat en is daarom verbeterd. De andere groepsgenoten besloten een andere minor te gaan volgen en ik heb dus dit project zelf moeten oppakken. Wat ik heb gebruikt om uiteindelijk een goed project neer te zetten is het onderzoek dat we hadden gedaan met het groepje, ik vond dat dit niet nodig was om dit helemaal opnieuw te gaan doen aangezien we al aardig veel informatie hebben kunnen verzamelen, verder is het idee van het project ook overgenomen. Alle code/modellen en dit verslag is volledig geschreven door mij, ook heb ik een andere grotere dataset genomen om zo beter te kunnen trainen. De verdeling van het werk zou dus een percentage van 85% zijn wat ik heb verricht en de overige 15% heb ik van het oude project overgehouden.

Bijlagen

Bijlage 1 - Onderzoeksverslag

Project AI (INFPRJ01)

Versie 1.0
Final Version
1 Maart 2023

Onderzoeksverslag

Artificial Intelligence

Hakan Dalama (0963609)
(Gizem Sazak, Rahul Udhayua, Remco de Zeeuw, Hoes Fortuin)

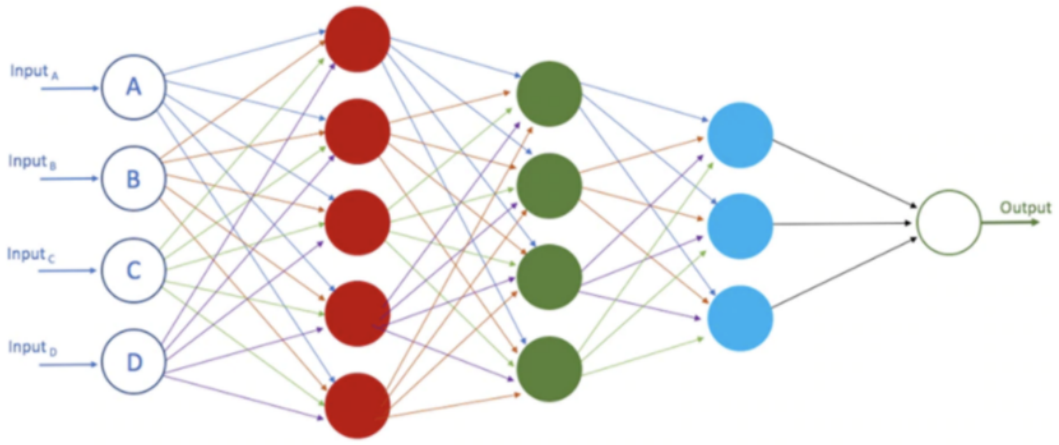
Rotterdam University of Applied Sciences

Wat is Deep Learning?

Laten we de volgende definitie erbij pakken:

“Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example.” (MathWorks, sd)

Deep Learning is dus min of meer het aanleren van een computer om te leren zoals mensen dat doen. Een goed voorbeeld hiervan is een zelfrijdende auto die verkeersborden kan herkennen en hier op de juiste manier op reageren (bv. Stoppen voor een stopbord, snelheid aanpassen aan een maximum snelheidsbord, etc.). (MathWorks, sd)
Deep Learning verschilt van andere soorten AI door het gebruik van zogenaamde lagen. Als er meerdere opdrachten tegelijk worden uitgevoerd (elk in een eigen laag) spreken we over Deep Learning. Dit geheel van lagen wordt een neuraal netwerk genoemd. (Meer, 2019)



Figuur 1: Neuraal network (Tierney, sd)

Wat is Machine Learning?

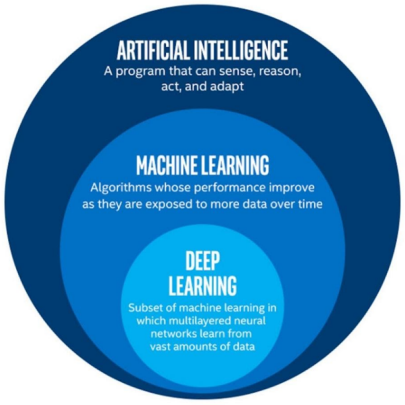
Net als bij Deep Learning pakken we er eerst een definitie bij:

“Machine learning is een vorm van kunstmatige intelligentie (AI) die is gericht op het bouwen van systemen die van de verwerkte data kunnen leren of data gebruiken om beter te presteren.” (Oracle, sd)

Daarnaast valt Machine Learning op te delen in twee varianten, namelijk supervised learning en unsupervised learning. (Oracle, sd)
Aangezien dit onderzoek vooral gericht is op Deep Learning, gaan we hier verder niet op in.

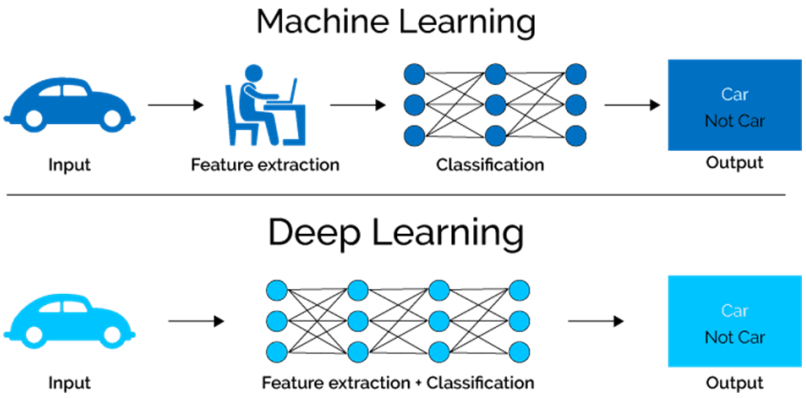
Verschil tussen Machine Learning en Deep Learning

Machine Learning en Deep Learning zijn erg verwant aan elkaar. Sterker nog, Deep Learning is een onderdeel van Machine Learning, zoals te zien in onderstaand figuur.



Figuur 2: Relatie tussen Machine Learning en Deep Learning (Oracle, sd)

Machine Learning en Deep Learning delen dus dezelfde werking. Het voordeel van Deep Learning is dat het gebruik maakt van zogenaamde layers (lagen), waardoor er meer data tegelijkertijd verwerkt kan worden. Dit heeft echter als nadeel dat het veel meer rekenkracht kost, vooral van de GPU (videokaart). (Pilli, sd)
Ook maakt Deep Learning gebruik van neurale netwerken, waardoor het systeem uiteindelijk zelf patronen leert herkennen. Machine Learning is in dat opzicht een stuk minder complex, en kan geen patronen leren herkennen (zie ook figuur 4). (Pilli, sd)



Figuur 3: Verschillen tussen Machine Learning en Deep Learning (Pilli, sd)

	Deep Learning	Machine Learning
Wordt helaas en ten onrechte verward met AI/Kunstmatige Intelligentie	V	V
Maakt gebruik van neurale netwerken	V	O
Draait om ontwikkelen algoritmes	O	V
Leert dankzij het herkennen van patronen	V	O

Figuur 4: Deep Learning en Machine Learning met elkaar vergeleken (Meer, 2019)

Automatic Text Classification

Text Classification is een methode binnen Machine Learning (en dus ook Deep Learning) die tekst kan sorteren op categorieën op basis van verschillende criteria. Een goed voorbeeld hiervan is het sorteren van nieuwsberichten op onderwerp. Deze methode lijkt in eerste instantie ideaal voor de doeleinden van dit project. Deze methode kan zowel handmatig als automatisch uitgevoerd worden (supervised en unsupervised). Ook biedt Python genoeg keuze wat betreft libraries die Text Classification ondersteunen. Enkele voorbeelden hiervan zijn Scikit-Learn en NLTK. Met TensorFlow is het ook mogelijk om Text Classification toe te passen in Deep Learning. (Pascual, 2019)

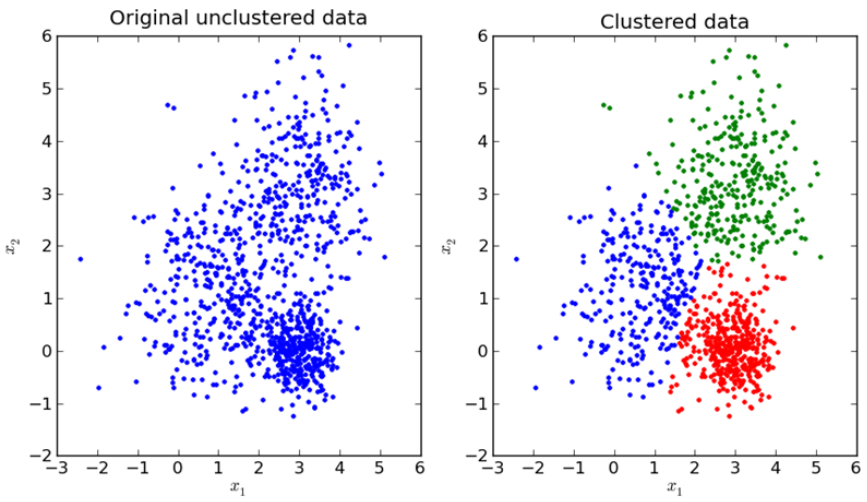
Text Classification werkt als volgt: Je “voert” het systeem een aantal goede voorbeelden. Na een tijdje gaat het systeem leren wat wat is, en heeft het geen voorbeelden meer nodig. Het systeem is dan goed genoeg getraind om zelfstandig te kunnen functioneren. (Pascual, 2019)

Supervised Text Classification

In deze vorm van Text Classification geeft men gelabelde data aan het systeem. Het systeem wordt hiermee getraind, en zal steeds beter worden in het zelfstandig sorteren van de data. Een goed voorbeeld hiervan is een spamfilter in een emailontvanger. Na een tijdje volgt de testperiode, waarin het systeem data krijgt zonder labels, waarna het systeem zelf de data moet indelen op categorie. Als dit goed gaat is het systeem voldoende getraind. Zo niet, dan zal er nog meer gelabelde data nodig zijn. (KDnuggets, sd)

Unsupervised Text Classification

Unsupervised Text Classification wordt uitgevoerd zonder externe informatie te geven aan het systeem. De data heeft dus geen labels, zoals dit wel het geval is bij de supervised method. Bij deze method probeert het algoritme structuur in de data te vinden. Het algoritme maakt op basis van de resultaten clusters. Alle punten die dicht bij elkaar liggen worden gezien als categorie. Op die manier bouwt het algoritme alle mogelijke categorieën op. (KDnuggets, sd)



Figuur 5: Geclusterde data na unsupervised learning

Kunnen wij Deep Learning in ons project gebruiken?
Om die vraag te beantwoorden zullen we moeten kijken naar de precieze werking van Deep Learning. Zoals hierboven gezegd is het een manier om het menselijk leren te simuleren. Het is voor ons natuurlijk heel makkelijk om een negatieve comment te herkennen. Een computer kan dit echter niet uit zichzelf.

Deep Learning zou een goede manier zijn om onze AI te trainen. Een van de manieren waarop Deep Learning werkt is het “voeren” van gelabelde data, om zo de AI te trainen. Op basis hiervan wordt een model opgesteld waarmee de AI in de toekomst geen label meer nodig heeft. Hoe meer training (gelabelde data, of comments in dit geval), des te beter het model zal worden. (MathWorks, sd)
Daarnaast heeft Deep Learning, in tegenstelling tot Machine Learning, geen menselijke input nodig. Ook dit maakt deze methode meer geschikt voor ons project.

Echter, omdat Deep Learning krachtigere hardware nodig heeft, is het onwaarschijnlijk dat we dit kunnen laten uitvoeren door onze eigen PC’s. We zullen dus gebruik moeten maken van cloud computing als we Deep Learning willen inzetten. Hogeschool Rotterdam biedt ons deze mogelijkheid aan, als we dit zouden willen. (Pilli, sd)

Welke programmeertaal is het meest geschikt?

Voor Machine Learning/Deep Learning zijn er redelijk wat programmeertalen geschikt, maar de drie die er met kop en schouders bovenuit steken zijn C#, R en Python. In dit onderzoek zullen wij alleen deze drie talen onder de loep nemen. (SSA DATA, 2020)

C#
In principe is dit een goede keuze voor Machine Learning. Echter, het aanbod van libraries met betrekking tot AI is beperkt. Ook is C# voornamelijk gericht op het Windows-platform, en zal het dus vrij lastig zijn om een cross-platform applicatie te bouwen met deze taal. (SSA DATA, 2020)

R
Deze taal is vooral gericht op het verwerken van data. Ideaal voor AI dus. Echter zijn de mogelijkheden buiten dataverwerking om vrij minimaal, en is de taal ook berucht om zijn traagheid in het maken van berekeningen. (SSA DATA, 2020)

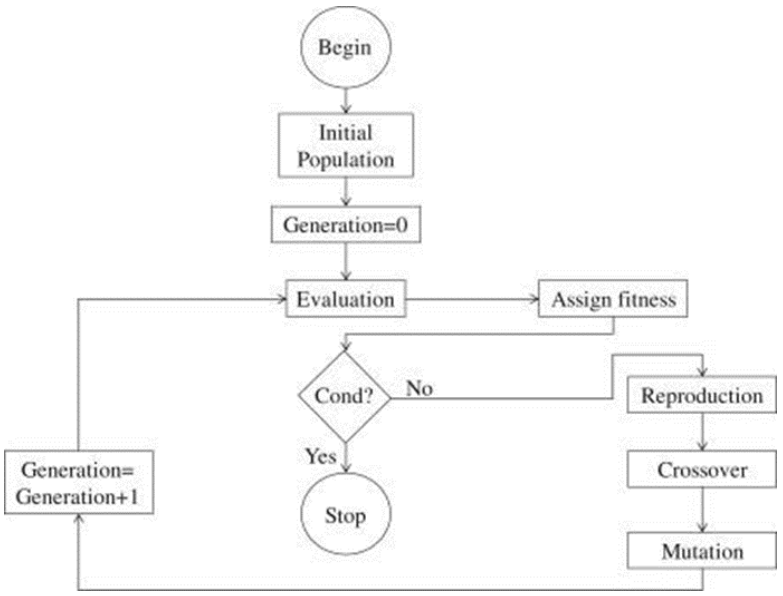
Python
Deze taal wordt het meest gebruikt bij AI-gerelateerde applicaties. Python heeft een ruim aanbod aan libraries voor AI, is makkelijk te leren, en werkt op vrijwel alle platforms. (SSA DATA, 2020)

Aan de hand van bovenstaande informatie is Python voor ons project het meest geschikt. Sommige van onze projectleden hebben al ervaring met Python, ook dit weegt mee in onze beslissing.

Wat is Genetic Algorithm?

Genetic Algorithm is gebaseerd op de Darwin theorie. Het is een langzaam proces waarbij steeds de twee beste resultaten worden gekozen. Dit zijn de parents. Vervolgens wordt er gebaseerd op die parents weer een nieuwe generatie gemaakt. Van die nieuwe generatie worden de twee besten weer gekozen, enzovoort. Deze methode is langzaam, maar zorgt er wel voor dat de AI zich steeds verbeterd. (Gad, 2018)

Dit proces kent een aantal stappen:
1. Initial Population – Dit is een willekeurige populatie gebaseerd op de data.
2. Fitness function – Zoeken naar het fitness niveau van alle uitkomsten.
3. Selection – Selecteer de twee uitkomsten met de hoogste fitness.
4. Cross-over – Creëer een nieuwe populatie gebaseerd op de zojuist gekozen parents.
5. Mutation – Mutatie uitvoeren voor een of meerdere eigenschappen van de nieuwe generatie. Zo ontstaat er meer diversiteit. (Choudhary, sd)
Voer vervolgens stap 2-5 uit voor elke opeenvolgende generatie, net zolang tot de nieuwere generaties niet meer verbeteren of totdat er verzadiging is opgetreden. (Choudhary, sd)



Figuur 6: Genetic Algorithm uitgebeeld (Science Direct, sd)

Wanneer gebruik je Genetic Algorithms

Je gebruikt Genetic Algorithms als je iets wilt optimaliseren. In veel gevallen in AI wordt iets niet verbeterd, maar met GA kun je elke generatie verbeteringen doorvoeren. Op die manier creëer je steeds een efficiënter product, met een hogere accuracy. (Sommerville, 2021)

Kunnen wij Genetic Algorithms gebruiken in ons project?

Wij kunnen dit zeker in ons project gebruiken. Met de methode die we tot nu toe hebben gebruikt krijgen we slechts één resultaat gebaseerd op een bepaalde dataset. Dit betekent dat het systeem zich daarna niet meer verbeterd, en dus niet meer bijleert. Op die manier zal de efficiëntie nooit toenemen. Zelfs met een dataset van duizenden records was de efficiëntie niet hoger dan 75%. Als we Genetic Algorithms gebruiken zal deze efficiëntie veel hoger kunnen worden naarmate de generaties toenemen.

Wat is Reinforcement Learning?

Reinforcement Learning is een vorm van kunstmatige intelligentie (AI). Het is een manier van zelflerende vermogen onder de categorie Machine Learning. Het doel is om het ideale gedrag te ontdekken voor de best haalbare prestaties.
Een aantal voorbeelden uit de praktijk zijn bijvoorbeeld:

- Robots producten laten uitpakken
- Prijzen automatisch veranderen
- Automatisch beleggen

Het werkt met observatie, ontdekking en een trial and error beloningssysteem. Een voorbeeld uit de praktijk kan zijn een hond, als een hond iets goeds doet wordt hij beloond met een snoepje bijvoorbeeld. Als de hond niet doet wat hij hoort te doen krijgt de hond geen beloning en zal dit dan ook hoogstwaarschijnlijk niet meer doen.
Basic reinforcement is gemodelleerd als een “Markov decision process” **(MDP)**:

Types of Reinforcement

Positief
Positieve Reinforcement wordt gedefinieerd als wanneer een gebeurtenis optreedt als gevolg van een bepaald gedrag, de kracht en de frequentie van het gedrag toeneemt. Met andere woorden, het heeft een positief effect op gedrag .GeeksforGeeks. (2020, 18 mei).

Voordelen van Reinforcement learning zijn:
-Maximaliseert prestaties
-Ondersteun verandering gedurende een lange periode

Nadelen van Reinforcement learning:
-Te veel versterking kan leiden tot een overbelasting van toestanden, waardoor de resultaten kunnen verminderen Sutton, R. S., & Barto, A. G. (2018).

Negatief
Negatieve Reinforcement wordt gedefinieerd als het versterken van een gedrag doordat een negatieve toestand wordt gestopt of vermeden.

Voordelen van Reinforcement learning:
-Verhoogt gedrag
-Verzet tegen de minimale prestatie norm

Nadelen van Reinforcement learning:
-Het biedt alleen genoeg om aan het minimum gedrag te voldoen

Kan het gebruikt worden voor het product?

Waarvoor wordt Reinforcement Learning normaliter gebruikt?
Reinforcement Learning is een vorm van technologie dat gebruikt wordt om producten beter te kunnen beheren (optimaliseren) , implementeren (personaliseren), realiseren (efficiënter maken) en ontwerpen (verbeteren van applicatie) (2019, 11 juli). APPelit. Q-learning, double Q learning en policy gradient methods zijn algorithms onder Reinforcement Learning

Kan het specifiek gebruik worden voor ons product?
Er is een mogelijkheid om Automated Text Classification te gebruiken voor ons project. Automated Text Classification kan taken toewijzen tot 1 of meer member Gupta, S. (2018, 21 juni)

Bronnen

Choudhary, A. (sd). Generic Algorithm in Machine Learning using Python. DataScience+. https://datascienceplus.com/genetic-algorithm-in-machine-learning-using-python/

Gad, A. (2018, juli 3). Introduction to Optimization with Genetic Algorithm. Towards Data Science.
https://towardsdatascience.com/introduction-to-optimization-with-genetic-algorithm-2f5001d9964b

KDnuggets. (sd). Automated Text Classification Using Machine Learning. KDnuggets.
https://www.kdnuggets.com/2018/01/automated-text-classification-machine-learning.html

MathWorks. (sd). What Is Deep Learning? MathWorks.
https://www.mathworks.com/discovery/deep-learning.html

Meer, B. v. (2019, mei 31). Wat is deep learning? Een korte introductie tot deep learning. Moqod. https://moqod.com/nl/wat-is-deep-learning-een-korte-introductie-tot-deep-learning/

Oracle. (sd). Wat is machine learning. Oracle.
https://www.oracle.com/nl/data-science/machine-learning/what-is-machine-learning/

Pascual, F. (2019, oktober 14). Text Classification with Python. MonkeyLearn.
https://monkeylearn.com/blog/text-classification-with-python/#:~:text=Text%20classification%20is%20the%20automatic,categories%20to%20unstructured%20text%20data.&text=That%27s%20a%20lot%20of%20social,companies%20make%20data%2Dbased%20decisions.

Pilli, N. (sd). Difference between Machine Learning and Deep Learning. Morioh.
https://morioh.com/p/ed56b4fdbf1c

Science Direct. (sd). Genetic Algorithm.
https://www.sciencedirect.com/topics/engineering/genetic-algorithm

Sommerville, G. (2021, februari 4). Using genetic algorithms on AWS for optimization problems. AWS.

https://aws.amazon.com/blogs/machine-learning/using-genetic-algorithms-on-aws-for-optimization-problems/

SSA DATA. (2020, januari 20). TOP Programming Languages For Machine Learning (Python, C#, R). Geraadpleegd van https://www.ssa-data.com/blog/archive/best-programming-languages-for-machine-learning/

Tierney, B. (sd). Understanding, Building and Using Neural Network Machine Learning Models using Oracle 18c. Geraadpleegd van https://developer.oracle.com/databases/neural-network-machine-learning.html

GeeksforGeeks. (2020, 18 mei). Reinforcement learning. Geraadpleegd van https://www.geeksforgeeks.org/what-is-reinforcement-learning/

Gupta, S. (2018, 21 juni). Automated Text Classification Using Machine Learning. Medium. Geraadpleegd van https://towardsdatascience.com/automated-text-classification-using-machine-learning-3df4f4f9570b

Reinforcement learning: slimmere & succesvoller bedrijf. (2019, 11 juli). APPelit. Geraadpleegd van https://www.appelit.com/reinforcement-learning/

Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning. Amsterdam University Press.

Wikipedia contributors. (2021, 1 maart). Reinforcement learning. Wikipedia. Geraadpleegd van https://en.wikipedia.org/wiki/Reinforcement_learning

Glen, S. (1 januari 2017). "Probabilistic: Definition, Models and Theory Explained". Geraadpleegd van https://www.statisticshowto.com/probabilistic/

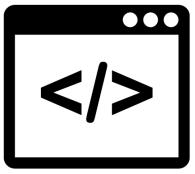
Ghahramani, Z. (2017). “Probabilistic machine learning and AI”. Geraadpleegd van https://www.microsoft.com/en-us/research/wp-content/uploads/2017/03/Ghahramani.pdf

Ratnayaka, G. (29 april 2020). “Probability and Machine Learning? — Part 1- Probabilistic vs Non-Probabilistic Machine Learning Models”. Geraadpleegd van https://gathikaapoorwa.medium.com/probability-and-machine-learning-570815bad29d#:~:text=Some%20examples%20for%20probabilistic%20models,input%20data%20instance%20belongs%20to

Ministry of Education (24 september 2013). “Deterministic and probabilistic models”. Geraadpleegd van https://seniorsecondary.tki.org.nz/Mathematics-and-statistics/Achievement-objectives/AOs-by-level/AO-S8-4/Deterministic-and-probabilistic-models

McGarigal, K. (2008). “Analysis of Environmental Data Conceptual Foundations: deterministic models”. Geraadpleegd van https://www.umass.edu/landeco/teaching/ecodata/schedule/deterministic.pdf

Bijlage 2 - Code Documentatie



Project AI (INFPRJ01)

Versie 1.0
Final Version
9 April 2023

Code Documentatie

Dit bestand bestaat uit de code die ik heb geschreven, uitgelegd per onderdeel, ook wat de resultaten zijn en waarom bepaalde keuzes zijn gemaakt.

Hakan Dalama (0963609)

Rotterdam University of Applied Sciences

Dataset

In [2]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
```

In [3]:
```
# load data
data = pd.read_csv(r"C:\Users\hakand\OneDrive\Desktop\rvws\IMDB Dataset.csv")

# drop any rows with missing values
data.dropna(inplace=True)
data
```

Out[3]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
...	...	...
49995	I thought this movie did a down right good job...	positive
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative
49997	I am a Catholic taught in parochial elementary...	negative
49998	I'm going to have to disagree with the previou...	negative
49999	No one expects the Star Trek movies to be high...	negative

50000 rows × 2 columns

De bovenstaande code leest een CSV-bestand in met filmrecensies van IMDB, verwijdert rijen met ontbrekende waarden en past vervolgens een Naive Bayes-classifier toe om te voorspellen of een recensie positief of negatief is. De code gaat als volgt:

1. import pandas as pd: importeert de Pandas-bibliotheek, die wordt gebruikt voor datamanipulatie en analyse.
2. from sklearn.feature_extraction.text import TfidfVectorizer: importeert de TfidfVectorizer-klasse van scikit-learn, die wordt gebruikt om tekst om te zetten in numerieke kenmer vectoren die kunnen worden gebruikt voor machine learning.
3. from sklearn.model_selection import train_test_split: importeert de train_test_split-functie van scikit-learn, die wordt gebruikt om de gegevens in trainings- en testsets te splitsen.
4. from sklearn.naive_bayes import MultinomialNB: importeert de MultinomialNB-class van scikit-learn, die wordt gebruikt om een Naive Bayes-classifier te trainen en toe te passen.
5. from sklearn.metrics import accuracy_score: importeert de accuracy_score-functie van scikit-learn, die wordt gebruikt om de nauwkeurigheid van het model te berekenen.
6. data = pd.read_csv(r"C:\Users\hakand\OneDrive\Desktop\rvws\IMDB Dataset.csv"): leest het CSV-bestand met de filmrecensies in en slaat het op in een Pandas DataFrame genaamd data.
7. data.dropna(inplace=True): verwijdert alle rijen met ontbrekende waarden uit het DataFrame.

Over het algemeen lijkt deze code een goede start voor het analyseren van sentiment in filmrecensies. Het mist echter enkele belangrijke stappen, zoals het voorbewerken van de tekst gegevens (bijvoorbeeld het verwijderen van stopwoorden) en het evalueren van de prestaties van het model op een aparte testset. Het kan ook nuttig zijn om de gegevens te visualiseren voordat het model wordt toegepast.

In [4]:
```
# convert sentiment to binary labels (0=negative, 1=positive)
data['sentiment'] = data['sentiment'].apply(lambda x: 0 if x=='negative' else 1)

data
```

Out[4]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	1
1	A wonderful little production. The...	1
2	I thought this was a wonderful way to spend ti...	1
3	Basically there's a family where a little boy ...	0
4	Petter Mattei's "Love in the Time of Money" is...	1
...	...	...
49995	I thought this movie did a down right good job...	1
49996	Bad plot, bad dialogue, bad acting, idiotic di...	0
49997	I am a Catholic taught in parochial elementary...	0
49998	I'm going to have to disagree with the previou...	0
49999	No one expects the Star Trek movies to be high...	0

50000 rows × 2 columns

Dit deel van de code zet de sentiment kolom in het DataFrame om van string labels (negatief en positief) naar binaire labels (0 voor negatief en 1 voor positief). Dit is nodig voor het trainen van de classifier, omdat numerieke labels als invoer verplicht zijn.

De apply() methode wordt gebruikt om een lambda functie toe te passen op elk element in de sentiment kolom, die de string label omzet naar een binaire label met behulp van een ternary operator. Als de string label negatief is, wordt de lambda functie 0, anders wordt deze 1. De binaire labels worden vervolgens terug opgeslagen in de sentiment kolom van het DataFrame.

```
In [5]:
# preprocess text data
tfidf = TfidfVectorizer(stop_words='english')
X = tfidf.fit_transform(data['review'])
y = data['sentiment']
```

```
In [6]: X

Out[6]: <50000x101583 sparse matrix of type '<class 'numpy.float64'>'
        with 4434500 stored elements in Compressed Sparse Row format>
```

```
In [7]: y

Out[7]: 0        1
        1        1
        2        1
        3        0
        4        1
               ..
        49995    1
        49996    0
        49997    0
        49998    0
        49999    0
        Name: sentiment, Length: 50000, dtype: int64
```

Dit deel van de code voert tekstverwerking uit op de review kolom van de DataFrame.

1. TfidfVectorizer(stop_words='english'): maakt een instantie van de TfidfVectorizer-klasse aan, die wordt gebruikt om tekst gegevens om te zetten in numerieke eigenschap vectoren. De parameter stop_words is ingesteld op 'english', waarmee veel gebruikte Engelse woorden die niet nuttig zijn voor sentimentanalyse worden verwijderd.
2. X = tfidf.fit_transform(data['review']): past de TfidfVectorizer toe op de review kolom van de Data Frame om de tekst gegevens om te zetten in een confusion matrix. De resulterende confusion matrix wordt opgeslagen in X.
3. y = data['sentiment']: wijst de binary labels uit de sentiment kolom van de Data Frame toe aan y. Over het algemeen is deze code een essentiële stap bij het voorbereiden van de gegevens voor het trainen van de classifier. De TfidfVectorizer is een krachtige tool om tekstgegevens om te zetten in numerieke eigenschappen die kunnen worden gebruikt voor machine learning.

```
In [12]: X_train.shape

Out[12]: (40000, 101583)
```

```
In [13]: X_test.shape

Out[13]: (10000, 101583)
```

```
In [28]: y_train.shape

Out[28]: (40000,)
```

```
In [29]: y_test.shape

Out[29]: (10000,)
```

Dit deel van de code splitst de gegevens in trainings- en testsets door gebruik te maken van de train_test_split functie van scikit-learn.

1. train_test_split(X, y, test_size=0.2, random_state=42): neemt de feature matrix X en de label vector y als invoer, samen met de test_size parameter die de verhouding van de gegevens specificeert die gebruikt moeten worden voor het testen (in dit geval 20%). De random_state parameter stelt een willekeurige seed in om reproduceerbaarheid te garanderen.
2. X_train, X_test, y_train, y_test = ...: pakt de output van train_test_split uit in vier variabelen. Over het algemeen is het splitsen van de gegevens in trainings- en testsets een belangrijke stap bij het evalueren van de prestaties van de classifier op ongeziene gegevens. Door een deel van de gegevens voor het testen te gebruiken, kunnen we een nauwkeuriger schatting krijgen van hoe goed het model zal presteren op nieuwe gegevens.

Machine Learning Modellen

Naïve Bayes (ML model):

```
In [14]: # train model
         model = MultinomialNB()
         model.fit(X_train, y_train)

Out[14]: ▾ MultinomialNB
         MultinomialNB()
```

```
In [15]: # make predictions on test set
         y_pred = model.predict(X_test)

In [16]: # evaluate accuracy
         accuracy = accuracy_score(y_test, y_pred)
         print("Accuracy:", accuracy)

         Accuracy: 0.8654

In [17]: from sklearn.metrics import classification_report, confusion_matrix

         # print classification report
         print(classification_report(y_test, y_pred))

         # print confusion matrix
         cm = confusion_matrix(y_test, y_pred)
         print(cm)
```

```
              precision    recall  f1-score   support

           0       0.86      0.88      0.87      4961
           1       0.88      0.85      0.86      5039

    accuracy                           0.87     10000
   macro avg       0.87      0.87      0.87     10000
weighted avg       0.87      0.87      0.87     10000

[[4352  609]
 [ 737 4302]]
```

Dit deel van de code maakt voorspellingen op de test set en evalueert de prestaties van het model met behulp van verschillende metrieken.

1. y_pred = model.predict(X_test): gebruikt het getrainde model om de labels voor de test set, X_test, te voorspellen. De voorspelde labels worden opgeslagen in y_pred.
2. accuracy = accuracy_score(y_test, y_pred): berekent de nauwkeurigheid van het model op de test set door de voorspelde labels, y_pred, te vergelijken met de ware labels, y_test. De accuracy_score functie van scikit-learn wordt gebruikt om de nauwkeurigheid te berekenen.
3. print("Accuracy:", accuracy): print de nauwkeurigheid van het model op de test set.
4. print(classification_report(y_test, y_pred)): print een gedetailleerd classificatierapport dat precision, recall en F1-score bevat voor elke klasse (positief en negatief) en de nauwkeurigheid.
5. cm = confusion_matrix(y_test, y_pred): berekent de confusion matrix voor de voorspellingen van het model op de test set, die een tabel is die het aantal true positives, false positives, true negatives en false negatives weergeeft.
6. print(cm): print de confusion matrix.

Over het algemeen is het evalueren van de prestaties van een machine learning model belangrijk om te beoordelen hoe goed het zal presteren op nieuwe gegevens. De nauwkeurigheid, classificatierapport en confusion matrix zijn veelgebruikte methoden om de prestaties van een classifier te evalueren. De code laat het classificatierapport en de confusion matrix zien voor het getrainde Naïve Bayes model op de test set. Het classificatierapport bevat verschillende waardes: precision, recall en F1-score voor elke klasse (0 en 1).

Precision: De precision meet het percentage true positives onder het totale aantal voorspelde positieven. In dit geval is de precision voor klasse 0 0,86, wat betekent dat van alle reviews die voorspeld zijn als negatief, 86% van hen daadwerkelijk negatief was. Op dezelfde manier is de precision voor klasse 1 0,88, wat betekent dat van alle reviews die voorspeld zijn als positief, 88% van hen daadwerkelijk positief was.

Recall: De recall meet het percentage true positives ten opzichte van het totale aantal werkelijke positieven. In dit geval is de recall voor klasse 0 0,88, wat betekent dat van alle werkelijke negatieve beoordelingen, 88% correct werd voorspeld als negatief. Op dezelfde manier is de recall voor klasse 1 0,85, wat betekent dat van alle werkelijke positieve beoordelingen, 85% correct werd voorspeld als positief.

De F1-score is het gemiddelde van precision en recall. In dit geval is de F1-score voor klasse 0 0,87. Op dezelfde manier is de F1-score voor klasse 1 0,86.

De ondersteuning geeft het aantal samples in elke klasse weer.

De nauwkeurigheid van het model op de testset is 0,87, wat betekent dat 87% van de beoordelingen in de testset correct werd geclassificeerd door het model.

De confusion matrix toont het aantal true positives, false positives, true negatives en false negatives voor elke klasse. In dit geval toont de confusion matrix aan dat het model 4352 negatieve beoordelingen en 4302 positieve beoordelingen correct heeft voorspeld, maar het heeft 609 negatieve beoordelingen onjuist voorspeld als positief (false positives) en 737 positieve beoordelingen onjuist voorspeld als negatief (false negatives).

Random Forest (ML model):

```
In [18]: from sklearn.ensemble import RandomForestClassifier

         model = RandomForestClassifier(n_estimators=100, random_state=42)
         model.fit(X_train, y_train)

Out[18]: RandomForestClassifier
         RandomForestClassifier(random_state=42)
```

```
In [20]: # make predictions on test set
         y_pred = model.predict(X_test)

         # evaluate accuracy
         accuracy = accuracy_score(y_test, y_pred)
         print("Accuracy:", accuracy)

         Accuracy: 0.861
```

```
In [21]: # print classification report and confusion matrix
         print(classification_report(y_test, y_pred))
         cm = confusion_matrix(y_test, y_pred)
         print(cm)

                       precision    recall  f1-score   support

                    0       0.86      0.86      0.86      4961
                    1       0.86      0.86      0.86      5039

             accuracy                           0.86     10000
            macro avg       0.86      0.86      0.86     10000
         weighted avg       0.86      0.86      0.86     10000

         [[4279  682]
          [ 708 4331]]
```

Deze code is vergelijkbaar met de vorige code, met als verschil dat het model dat voor classificatie wordt gebruikt, een RandomForestClassifier is in plaats van MultinomialNB. n_estimators=100 specificeert het aantal trees in het random forest, en random_state=42 zorgt ervoor dat de resultaten reproduceerbaar zijn.

De nauwkeurigheid score behaald door dit model op de test set is 0.861, wat iets lager is dan de nauwkeurigheid score behaald door het MultinomialNB-model. Het is echter vermeldenswaard dat de prestaties van de modellen kunnen variëren afhankelijk van de specifieke dataset en de gebruikte hyperparameters.

Deze uitvoer toont het classificatierapport en de confusion matrix voor het Random Forest Classifier-model op de test set. De precision, recall en F1-score voor zowel positieve als negatieve klassen zijn allemaal 0.86, wat aangeeft dat het model op beide klassen vergelijkbaar presteert. De nauwkeurigheid van het model is 0.86. De confusion matrix laat zien dat het model 4279 negatieve recensies en 4331 positieve recensies correct heeft geclassificeerd, maar 682 negatieve recensies als positief en 708 positieve recensies als negatief heeft geclassificeerd.

Resultaten 150 estimators

```
              precision    recall  f1-score   support

           0       0.86      0.87      0.86      4961
           1       0.87      0.86      0.87      5039

    accuracy                           0.86     10000
   macro avg       0.86      0.87      0.86     10000
weighted avg       0.87      0.86      0.87     10000

[[4294  667]
 [ 683 4356]]
```

Resultaten 200 estimators

```
              precision    recall  f1-score   support

           0       0.86      0.87      0.86      4961
           1       0.87      0.86      0.87      5039

    accuracy                           0.86     10000
   macro avg       0.86      0.87      0.86     10000
weighted avg       0.87      0.86      0.87     10000

[[4294  667]
 [ 683 4356]]
```

In deze test heb ik Random Forest classifiers getraind met verschillende aantallen trees en hun prestaties geëvalueerd op de test set. Over het algemeen leidt het toenemen van het aantal trees in een Random Forest tot een verbetering van de prestaties, tot op een bepaald punt waar de verbetering klein wordt. Dit komt omdat elke tree in random forest bijdraagt aan de voorspelling, en het verhogen van het aantal trees kan helpen om de variantie van de voorspellingen te verminderen.

In de experimenten heb ik Random Forest classifiers getraind met 50, 100 en 150 trees. De nauwkeurigheid scores waren 0.8467, 0.861 en 0.865. Zoals verwacht verbeterde de nauwkeurigheid naarmate het aantal trees toenam, waarbij de beste prestatie werd behaald met 150 trees.

Als we kijken naar het classificatierapport, zijn de precision, recall en F1-score voor beide klassen zeer vergelijkbaar, wat aangeeft dat het model even goed presteert voor zowel positieve als negatieve recensies. Daarnaast laat de confusion matrix zien dat het model een hoog aantal recensies voor beide klassen correct classificeert.

Over het algemeen suggereren deze resultaten dat de Random Forest-classifier goed presteert op deze dataset en dat het verhogen van het aantal trees kan leiden tot verbeterde prestaties. Het is echter belangrijk op te merken dat deze experimenten alleen de prestaties op een enkele train-test-split hebben geëvalueerd, en het zou nuttig zijn om aanvullende evaluaties uit te voeren om de stabiliteit van de prestaties van het model te beoordelen.

SVM (ML model):

```
In [27]: from sklearn.svm import SVC

         # train model
         model = SVC(kernel='linear', random_state=42)
         model.fit(X_train, y_train)
```

Out[27]:

```
SVC(kernel='linear', random_state=42)
```

```
In [28]: # make predictions on test set
         y_pred = model.predict(X_test)

         # evaluate accuracy
         accuracy = accuracy_score(y_test, y_pred)
         print("Accuracy:", accuracy)
```

```
Accuracy: 0.8968
```

```
              precision    recall  f1-score   support

           0       0.90      0.89      0.90      4961
           1       0.89      0.91      0.90      5039

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000

[[4404  557]
 [ 475 4564]]
```

```
In [33]: from sklearn.svm import SVC

         # train model
         model = SVC(kernel='rbf', random_state=42)
         model.fit(X_train, y_train)
```

Out[33]:

```
SVC(random_state=42)
```

```
In [34]: # make predictions on test set
         y_pred = model.predict(X_test)

         # evaluate accuracy
         accuracy = accuracy_score(y_test, y_pred)
         print("Accuracy:", accuracy)
```

```
Accuracy: 0.9007
```

```
              precision    recall  f1-score   support

           0       0.91      0.88      0.90      4961
           1       0.89      0.92      0.90      5039

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000

[[4389  572]
 [ 421 4618]]
```

SVM staat voor Support Vector Machine. Het is een type supervised machine learning algoritme dat gebruikt kan worden voor zowel classificatie als regressie taken. Bij een binair classificatieprobleem is het doel van een SVM om de hyperplane te vinden die de twee klassen in de feature space het beste scheidt.

Het basisidee van SVM is om een beslissingsgrens te vinden die de data op een manier in twee klassen verdeeld die de marge tussen de klassen maximaliseert. De marge is de afstand tussen de beslissingsgrens en de dichtstbijzijnde datapunten van beide klassen. SVM streeft ernaar de hyperplane te vinden die deze marge maximaliseert.

SVM kan geïmplementeerd worden met verschillende soorten kernel functies, zoals lineair, polynomiaal, sigmoid en radial basis functie (RBF). De keuze van de kernel functie hangt af van de data en het probleem dat opgelost wordt.

In de code zijn twee SVM-modellen getraind met verschillende kernel functies - lineair en RBF. De modellen zijn getraind op de trainingsdata en geëvalueerd op de testdata. De nauwkeurigheid, precision, recall en f1-score werden berekend om de prestaties van de modellen te evalueren. Ook de confusion matrix werd berekend om het aantal true positives, true negatives, false positives en false negatives te tonen. De lineaire SVM behaalde een nauwkeurigheid van 0,8968, terwijl de RBF SVM een nauwkeurigheid van 0,9007 behaalde.

Deep Learning Modellen

In [58]:
```
# Split the data into training and testing sets
train_data = data[:40000]
test_data = data[40000:]
```

In [63]:
```
import string
import re
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords

def clean_text(text):
    # remove HTML tags
    text = re.sub('<.*?>', '', text)
    # remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # convert to lowercase
    text = text.lower()
    # remove stopwords
    stop_words = set(stopwords.words('english'))
    text = ' '.join(word for word in text.split() if word not in stop_words)
    return text
```

Dit is een functie die tekstgegevens voorbewerkt door verschillende stappen uit te voeren. Hier is wat elke stap doet:

• re.sub('<.*?>', '', tekst): Verwijdert eventuele HTML-tags in de tekst.
• tekst.translate(str.maketrans('', '', string.punctuation)): Verwijdert alle leestekens uit de tekst.
• tekst.lower(): Converteert de tekst naar kleine letters.
• stop_words = set(stopwords.words('english')): Laadt de set Engelse stopwoorden uit de NLTK-bibliotheek.
• ' '.join(word for word in tekst.split() if word not in stop_words): Verwijdert eventuele stopwoorden uit de tekst en voegt de overgebleven woorden weer samen tot een string.

Over het algemeen is deze functie een veelvoorkomende manier om tekstgegevens voor te bereiden voordat ze worden gebruikt.

In [64]:
```
train_data['review'] = train_data['review'].apply(clean_text)
test_data['review'] = test_data['review'].apply(clean_text)
```

Deze lines code passen de clean_text-functie toe op de review kolom van zowel de trainings- als de test datasets, waarbij de tekst wordt schoongemaakt door het verwijderen van HTML-tags, leestekens, conversie van tekst naar kleine letters en het verwijderen van stopwoorden. Dit wordt gedaan om de tekst voor te bereiden voordat ze worden gebruikt om een model te trainen.

In [65]:
```
# Tokenize the text
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(train_data['review'])

X_train = tokenizer.texts_to_sequences(train_data['review'])
X_test = tokenizer.texts_to_sequences(test_data['review'])

# Pad the sequences
X_train = pad_sequences(X_train, maxlen=200)
X_test = pad_sequences(X_test, maxlen=200)
```

In deze code wordt de Tokenizer van Keras gebruikt om de tekst om te zetten in sequences van integers. De parameter num_words is ingesteld op 5000, wat betekent dat alleen de 5000 meest voorkomende woorden in de trainingsgegevens worden behouden en alle andere woorden worden verwijderd. De methode fit_on_texts van het Tokenizer-object wordt gebruikt om de tokenizer op de trainingsgegevens te passen, wat betekent dat de tokenizer de mapping tussen woorden en integers zal leren op basis van de frequentie van woorden in de trainingsgegevens. Vervolgens wordt de methode texts_to_sequences van het Tokenizer-object gebruikt om de tekst om te zetten in sequences van integers op basis van de geleerde mapping. De resulterende sequences worden opgeslagen in X_train en X_test. Ten slotte wordt de pad_sequences-functie van Keras gebruikt om de sequences op te vullen met nullen of ze te stoppen tot een vaste lengte van 200. Dit wordt gedaan om ervoor te zorgen dat alle sequences dezelfde lengte hebben, wat een vereiste is voor het invoeren ervan in een neuraal netwerk. De resulterende sequences worden opgeslagen in X_train en X_test.

CNN Model 1

In [66]:
```
# Define the model architecture
model = Sequential()
model.add(Embedding(5000, 32, input_length=200))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 200, 32)           160000

 flatten (Flatten)           (None, 6400)              0

 dense_6 (Dense)             (None, 128)               819328

 dropout_2 (Dropout)         (None, 128)               0

 dense_7 (Dense)             (None, 1)                 129

=================================================================
Total params: 979,457
Trainable params: 979,457
Non-trainable params: 0
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.94      0.65      4961
           1       0.52      0.06      0.11      5039

    accuracy                           0.50     10000
   macro avg       0.51      0.50      0.38     10000
weighted avg       0.51      0.50      0.38     10000

Confusion Matrix:
 [[4673  288]
 [4731  308]]
```

De code definieert een sequential model met behulp van de Keras API. De model architectuur bestaat uit een Embedding-layer met een input-dimensie van 5000, een output-dimensie van 32 en een input-lengte van 200. De output van de Embedding-layer wordt afgevlakt en vervolgens doorgegeven aan een Dense-layer met 128 units en een ReLU-activation function. Er wordt een Dropout-layer toegevoegd na de Dense-layer om overfitten te voorkomen, en tot slot wordt de output doorgegeven aan een Dense-layer met 1 unit en een sigmoid-activation function. Bovendien heb ik de Adam-optimizer gebruikt met een learning rate van 0,001.

Het classificatierapport toont de precision, recall en F1-score voor elke klasse (0 en 1), samen met de ondersteuning (aantal samples) voor elke klasse. De nauwkeurigheid van het model is 50%, wat niet erg goed is. De gemiddelde precision, recall en F1-score worden ook getoond.

De confusion matrix toont het aantal true positives, false positives, true negatives en false negatives voor elke klasse. Het model voorspelde correct 4673 instanties van klasse 0 en 308 instanties van klasse 1, maar heeft 4731 instanties van klasse 1 verkeerd geclassificeerd als klasse 0 en 288 instanties van klasse 0 verkeerd geclassificeerd als klasse 1.

CNN Model 2

Ik heb dezelfde model architectuur als model 1 gebruikt, maar ik heb de learning rate veranderd van 0,001 naar 0,0001 en heb de volgende confusion matrix bereikt.

```
Classification Report:
              precision    recall  f1-score   support

           0       0.49      0.60      0.54      4961
           1       0.49      0.38      0.43      5039

    accuracy                           0.49     10000
   macro avg       0.49      0.49      0.48     10000
weighted avg       0.49      0.49      0.48     10000

Confusion Matrix:
 [[2964 1997]
 [3121 1918]]
```

Het eerste gedeelte van de output is het Classificatie Rapport. Dit rapport geeft verschillende evaluatie metrics zoals precision, recall en F1-score voor elke klasse (0 en 1) en de nauwkeurigheid. Precision is de verhouding van correct voorspelde positieve waarnemingen tot het totaal voorspelde positieve waarnemingen. Recall is de verhouding van correct voorspelde positieve waarnemingen tot het totaal werkelijke positieve waarnemingen.
F1-score is het gemiddelde van precision en recall. Support is het aantal samples in elke klasse. Als we naar het rapport kijken, is de precision voor klasse 0 0,49, wat betekent dat van alle samples die als 0 zijn voorspeld, slechts 49% daadwerkelijk 0 was. Op dezelfde manier is de recall voor klasse 0 0,60, wat betekent dat van alle werkelijke samples van klasse 0, slechts 60% correct als 0 werd voorspeld.
De F1-score voor klasse 0 is 0,54, wat het gemiddelde van precision en recall is. Op dezelfde manier zijn de precision, recall en F1-score voor klasse 1 0,49, 0,38 en 0,43. De nauwkeurigheid van het model is 0,49, wat niet erg hoog is en aangeeft dat het model mogelijk niet erg effectief is bij het classificeren van de samples.

Het tweede gedeelte van de output is de confusion matrix. In dit geval laat de confusion matrix zien dat er 2964 true negatives waren (samples correct voorspeld als 0), 1997 false positives (samples voorspeld als 0 maar waren eigenlijk 1), 3121 false negatives (samples voorspeld als 1 maar waren eigenlijk 0) en 1918 true positives (samples correct voorspeld als 1). Over het algemeen suggereren de evaluatie metrics dat het model niet erg effectief is bij het classificeren van de samples en dat er ruimte is voor verbetering. De precision- en recall-waarden zijn relatief laag, wat aangeeft dat het model een aanzienlijk aantal valse voorspellingen maakt. De confusion matrix bevestigt dat het model moeite heeft om beide klassen correct te voorspellen. Verder onderzoek is nodig om de redenen achter de lage prestaties te begrijpen en het model te verbeteren.

CNN Model 3

```
model = Sequential()
model.add(Embedding(5000, 32, input_length=200))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.5)),
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

Hier heb ik meer dense layers toegevoegd, namelijk twee dense layers met 64 en 32 neuronen.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 200, 32)           160000

flatten_1 (Flatten)          (None, 6400)              0

dense_8 (Dense)              (None, 128)               819328

dense_9 (Dense)              (None, 64)                8256

dense_10 (Dense)             (None, 32)                2080

dropout_3 (Dropout)          (None, 32)                0

dense_11 (Dense)             (None, 1)                 33

=================================================================
Total params: 989,697
Trainable params: 989,697
Non-trainable params: 0
_________________________________________________________________
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.50      0.50      0.50      4961
           1       0.51      0.51      0.51      5039

    accuracy                           0.50     10000
   macro avg       0.50      0.50      0.50     10000
weighted avg       0.50      0.50      0.50     10000

Confusion Matrix:
 [[2493 2468]
 [2492 2547]]
```

Als we naar het rapport kijken, is de precision voor klasse 0 0,50, wat betekent dat van alle samples die als 0 zijn voorspeld, 50% daadwerkelijk 0 waren. Op dezelfde manier is de recall voor klasse 0 0,50, wat betekent dat van alle werkelijke 0-samples slechts 50% correct als 0 zijn voorspeld. De F1-score voor klasse 0 is 0,50. Op dezelfde manier zijn de precision, recall en F1-score voor klasse 1 0,51, 0,51 en 0,51. De nauwkeurigheid van het model is 0,50, wat gelijk is aan de gebalanceerde nauwkeurigheid (het gemiddelde van recall voor beide klassen) en suggereert dat het model willekeurig presteert.

CNN Model 4

In dit model heb ik 1 conv layer en 1 dense layer toegevoegd.

```
In [19]: # create CNN model
         model = Sequential()
         model.add(Embedding(5000, 128, input_length=500))
         model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
         model.add(GlobalMaxPooling1D())
         model.add(Dense(1, activation='sigmoid'))
         model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 500, 128)          640000

 conv1d_1 (Conv1D)           (None, 498, 128)          49280

 global_max_pooling1d_1 (Glo  (None, 128)              0
 balMaxPooling1D)

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 689,409
Trainable params: 689,409
Non-trainable params: 0
```

```
Test loss: 1.23
Test accuracy: 0.897
313/313 [==============================] - 0s 1ms/step
              precision    recall  f1-score   support

           0       0.89      0.90      0.90      5011
           1       0.90      0.89      0.90      4989

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000

[[4524  487]
 [ 540 4449]]
```

De resultaten laten zien dat het CNN-model een nauwkeurigheid van 0,897 heeft behaald, wat dicht bij de gewenste nauwkeurigheid van 90% ligt. De precision, recall en F1-score voor zowel positieve (1) als negatieve (0) sentimenten zijn ook dicht bij 0,9, wat aangeeft dat het model goed presteert bij het correct classificeren van zowel positieve als negatieve recensies. Bij het bekijken van de confusion matrix heeft het model 487 negatieve recensies foutief geclassificeerd als positief (false positives) en 540 positieve recensies als negatief (false negatives). Hoewel de false positives en false negatives relatief klein zijn in vergelijking met het totale aantal recensies, kunnen ze nog steeds invloed hebben op de prestaties van het model in “real-world” applicaties.

Om de prestaties van het model verder te verbeteren, kun je overwegen te experimenteren met verschillende hyperparameters, zoals het aantal filters, de kernel grootte en de learning rate, of het gebruik van vooraf getrainde woord embeddings. Bovendien kun je proberen geavanceerde deep learning-modellen te gebruiken, zoals recurrent neural networks (RNN's) of transformer-based modellen, die state-of-the-art prestaties hebben laten zien in natural language processing taken.

Over het algemeen heeft het CNN-model veelbelovende resultaten laten zien bij het classificeren van sentiment in tekstgegevens, en met verdere optimalisatie en experimenten heeft het de potentie om zelfs betere nauwkeurigheid en prestaties te behalen.

CNN Model 5

In dit model heb ik de dense-layer neuronen veranderd naar 64.

```
In [23]: model = Sequential()
         model.add(Embedding(5000, 128, input_length=500))
         model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
         model.add(GlobalMaxPooling1D())
         model.add(Dense(1, activation='sigmoid'))
         model.summary()

         Model: "sequential_2"
         _________________________________________________________________
          Layer (type)                 Output Shape              Param #
         =================================================================
          embedding_2 (Embedding)      (None, 500, 128)          640000

          conv1d_2 (Conv1D)            (None, 498, 64)           24640

          global_max_pooling1d_2 (Glo  (None, 64)                0
          balMaxPooling1D)

          dense_2 (Dense)              (None, 1)                 65

         =================================================================
         Total params: 664,705
         Trainable params: 664,705
         Non-trainable params: 0

    Test accuracy: 0.896
    313/313 [==============================] - 0s 1ms/step
                  precision    recall  f1-score   support

               0       0.89      0.90      0.90      5011
               1       0.90      0.89      0.90      4989

        accuracy                           0.90     10000
       macro avg       0.90      0.90      0.90     10000
    weighted avg       0.90      0.90      0.90     10000

    [[4523  488]
     [ 547 4442]]
```

De resultaten tonen aan dat het CNN-model een nauwkeurigheid behaalde van 0,896, wat vergelijkbaar is met het eerder besproken resultaat. De precision, recall en F1-score voor zowel positieve (1) als negatieve (0) sentimenten zijn ook dicht bij 0,9, wat aangeeft dat het model goed presteert in het correct classificeren van zowel positieve als negatieve beoordelingen.

Bij het bekijken van de confusion matrix heeft het model 488 negatieve beoordelingen foutief geclassificeerd als positief (false positives) en 547 positieve beoordelingen als negatief (false negatives). Vergeleken met het vorige resultaat is het aantal false negatives toegenomen, terwijl het aantal false positives is afgenomen. Deze resultaten suggereren dat het model mogelijk enigszins bevooroordeeld is ten opzichte van negatieve beoordelingen. Om deze bias aan te pakken, kan je overwegen meer positieve voorbeelden aan de trainingsgegevens toe te voegen of technieken te gebruiken zoals class weighting of data-augmentation om de klassen in evenwicht te brengen. Daarnaast kan je experimenteren met verschillende architecturen of hyperparameters om de prestaties van

het model verder te optimaliseren. Over het algemeen is er nog ruimte voor verbetering, hoewel het model een hoge nauwkeurigheid heeft behaald bij het classificeren van sentiment in tekstgegevens. Door de prestaties van het model te analyseren en de zwakke punten te identificeren, kan je het model blijven verfijnen en de nauwkeurigheid en prestaties verbeteren.

CNN Model 6

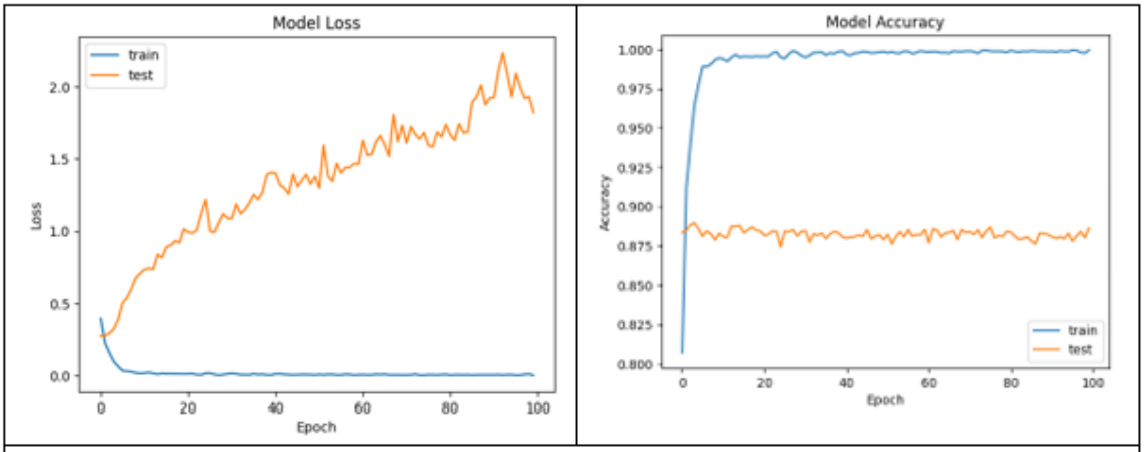
```
In [14]: # create CNN model
         model = Sequential()
         model.add(Embedding(5000, 128, input_length=500))
         model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
         model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
         model.add(Conv1D(filters=128, kernel_size=3, activation='relu'))
         model.add(GlobalMaxPooling1D())
         model.add(Dense(1, activation='sigmoid'))
         model.summary()
```

De code maakt gebruik van Keras om een Convolutional Neural Network (CNN) model te creëren. Het model bevat een embedding layer, gevolgd door drie convolutional layers, een global max pooling layer en een dense output layer met een sigmoid-activation functie.

De eerste layer is een embedding layer die integer gecodeerde woorden omzet naar een dense vectorruimte van grootte 128. De input length voor deze layer is ingesteld op 500, wat betekent dat het sequences van integers zal accepteren met een maximale lengte van 500.De volgende drie layers zijn convolutional layers met 128 filters, een kernelgrootte van 3 en ReLU activation. De vierde layer is een global max pooling layer die de belangrijkste features uit de output van de convolutional layers haalt door de maximumwaarde van elke filter over de hele sequentie te nemen.Tot slot wordt er een dense layer met één output en een sigmoid-activation functie toegevoegd om de output te krijgen.

De summary() methode wordt gebruikt om een samenvatting van de model architectuur te printen, inclusief het aantal parameters voor elke layer en het totale aantal parameters in het model.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 500, 128)          640000

 conv1d (Conv1D)             (None, 498, 128)          49280

 conv1d_1 (Conv1D)           (None, 496, 128)          49280

 conv1d_2 (Conv1D)           (None, 494, 128)          49280

 global_max_pooling1d (Globa  (None, 128)              0
 lMaxPooling1D)

 dense (Dense)               (None, 1)                 129

=================================================================
Total params: 787,969
Trainable params: 787,969
Non-trainable params: 0
```



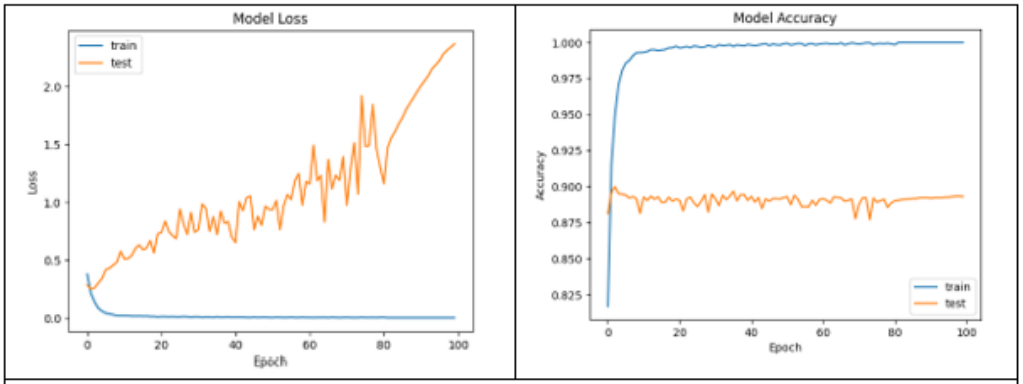

```
Test accuracy: 0.886
313/313 [==============================] - 1s 2ms/step
              precision    recall  f1-score   support

           0       0.89      0.88      0.89      5021
           1       0.88      0.89      0.89      4979

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

Deze output laat de test loss en accuracy zien van een model. Het model behaalde een test loss van 1.82 en een test accuracy van 0.886, wat aangeeft dat het model redelijk goed presteert.

CNN Model 7

Dit is een aangepaste versie van het vorige CNN-model met toevoeging van twee nieuwe dense layers. Het begint met een embedding layer die input sequences van lengte 500 accepteert, gevolgd door drie convolutional layers met 128 filters van grootte 3x3 en ReLU activatie. De output van de convolutional layers wordt vervolgens doorgegeven aan een global max pooling layer die de maximale waarde uit elke feature map haalt.

Het aangepaste deel van het model komt na de pooling layer waar twee dense layers zijn toegevoegd, één met 64 units en ReLU activatie en de andere met 32 units en ReLU activatie. Tot slot is een enkele neuron dense layer met sigmoid-activatie toegevoegd om de binary classification voorspelling te geven.




```
Test loss: 2.37
Test accuracy: 0.893
313/313 [==============================] - 1s 2ms/step
              precision    recall  f1-score   support

           0       0.91      0.88      0.89      5021
           1       0.88      0.91      0.89      4979

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

Vergeleken met het vorige model heeft dit nieuwe model twee extra dense layers met 64 en 32 neuronen. Bij het bekijken van de summary is de test accuracy iets verbeterd van 0.886 naar 0.893 ten opzichte van het vorige model. De precisie, recall en f1-score voor beide klassen (0 en 1) zijn ook hoog en gebalanceerd, wat aangeeft dat het model goed presteert bij het classificeren van zowel positieve als negatieve sentimenten.

Echter, de test loss is toegenomen tot 2.37, wat hoger is dan de test loss van het vorige model (1.82). Dit geeft aan dat de voorspellingen van het model mogelijk minder zelfverzekerd of minder nauwkeurig zijn dan die van het vorige model en dat het model moeite kan hebben met meer complexe of noisy data.

Over het algemeen kan het zo zijn dat meer dense layers aan een model toevoegen, de prestaties verbeteren, maar het is belangrijk om een oogje te houden op de loss en andere waardes om ervoor te zorgen dat het model niet over- of onder fit op de data.

We kunnen zien dat ze allemaal redelijk vergelijkbare prestaties hebben wat betreft nauwkeurigheid, precisie, recall en f1-score. Het eerste model behaalde een nauwkeurigheid van 0,886, met een f1-score van 0,89 voor zowel positieve als negatieve klassen. Het tweede model behaalde een iets hogere nauwkeurigheid van 0,893, met een f1-score van 0,89 voor beide klassen. Het derde model behaalde een nauwkeurigheid van 0,88, met een f1-score van 0,88 voor beide klassen.

Wat betreft precisie en recall behaalden alle drie modellen vergelijkbare resultaten, met precisie- en recall-waarden variërend van 0,88 tot 0,91 voor beide klassen. Dit geeft aan dat de modellen in staat zijn om nauwkeurige voorspellingen te doen en zowel positieve als negatieve sentiment correct te identificeren.

Een belangrijk verschil tussen de drie modellen is het aantal en type lagen dat wordt gebruikt. Het eerste model gebruikt drie convolutionele lagen en één dense laag, terwijl het tweede model twee extra dense lagen toevoegt. Het derde model vervangt de eerste twee convolutionele lagen door max pooling lagen, behoudt de derde convolutionele laag en voegt dezelfde twee dense lagen toe als het tweede model.

Het gebruik van max pooling lagen in het derde model maakt het mogelijk om de invoergegevens te downsamplen, wat het risico op overfitting kan verminderen en de generalisatie prestaties kan verbeteren. De toevoeging van de twee dense lagen in de tweede en derde modellen kan het model helpen om complexere representaties van de gegevens te leren, maar kan ook het risico op overfitting verhogen als het model te complex is voor de gegeven dataset.

De keuze van lagen en hyperparameters kan een invloed hebben op de prestaties van het model, en het is belangrijk om deze parameters zorgvuldig af te stemmen om optimale resultaten te bereiken. Bovendien is het belangrijk om de afweging tussen model complexiteit en generalisatie prestaties te overwegen en technieken zoals regularisatie en vroegtijdig stoppen te gebruiken om overfitting te voorkomen.