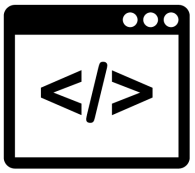


Project AI (INFPRJ01)

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(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]]
```

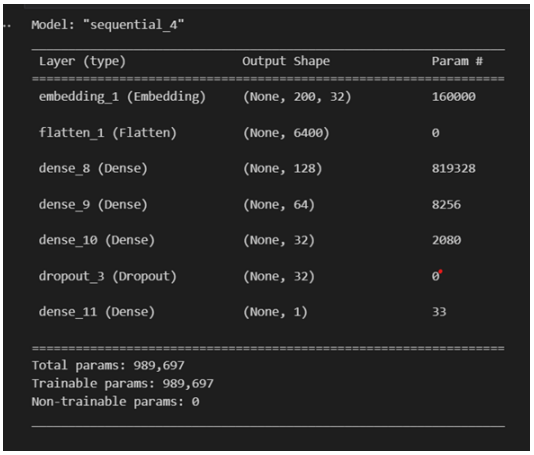
Het eerste gedeelte van de output is het Classificatie Rapport. Dit rapport geeft verschillende evaluatie metrics zoals precision, recall en F1-score voor elke klasse (0 en 1) en de nauwkeurigheid. Precision is de verhouding van correct voorspelde positieve waarnemingen tot het totaal voorspelde positieve 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.

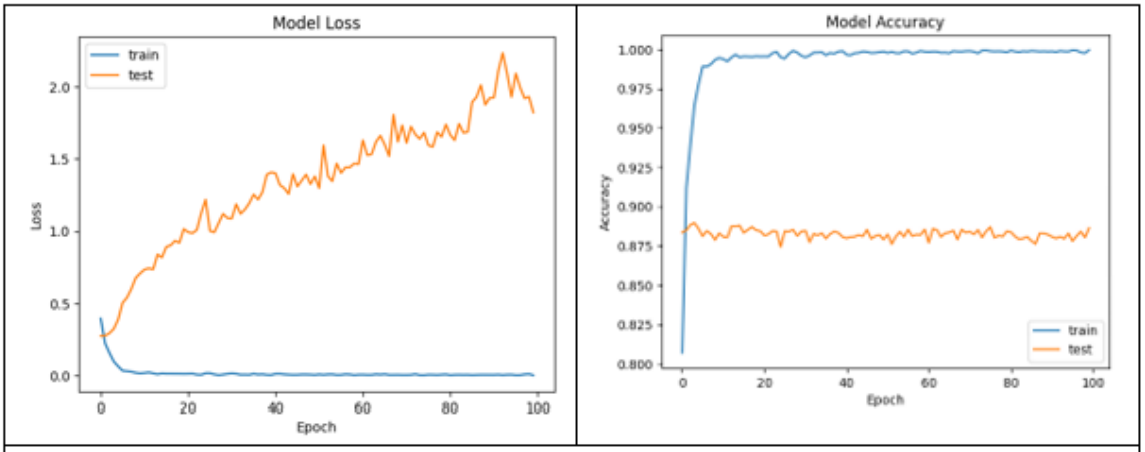
De eerste layer is een embedding layer die integer gecodeerde woorden omzet naar een dense vectorruimte van grootte 128. De input length voor deze layer is ingesteld op 500, wat betekent dat het sequences van integers zal accepteren met een maximale lengte van 500.

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
```



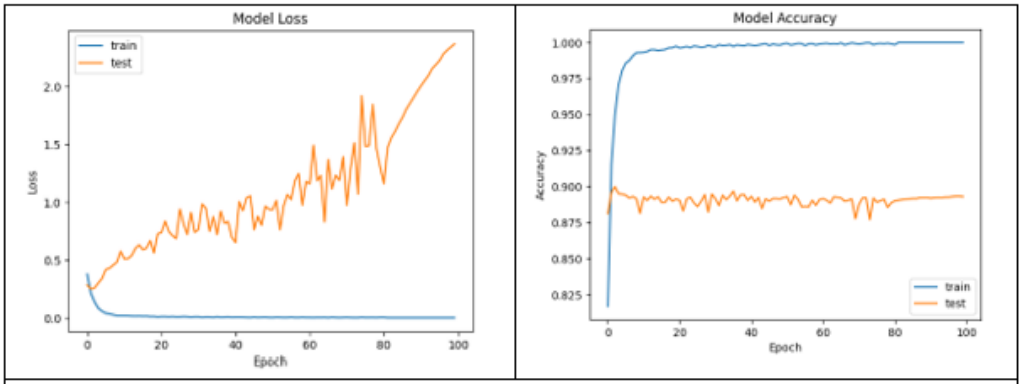

```
Test accuracy: 0.886
313/313 [==============================] - 1s 2ms/step
              precision    recall  f1-score   support

           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.