Machine Learning - IMDB Example

Quelle

A Simple Guide to Scikit-Learn — Building a Machine Learning Model in Python
https://towardsdatascience.com/a-beginners-guide-to-text-classification-with-scikit-learn-632357e16f3a

Daten

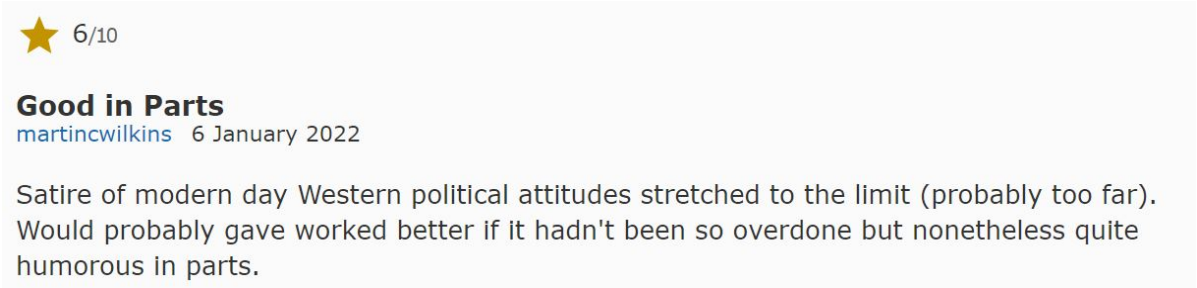
pandas

pandas ist eine Programmbibliothek zur Verarbeitung und Analyse von Daten in der Programmiersprache Python. Insbesondere enthält sie Datenstrukturen und Operatoren für den Zugriff auf numerische Tabellen und Zeitreihen.

In [1]:
```
import pandas as pd
```

IMDB-Datensatz von kaggle

https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews
IMDB-Datensatz mit 50.000 Filmbewertungen für die Verarbeitung natürlicher Sprache oder die Textanalyse mit Stimmungseinordnung.



In [2]:
```
# Data Gathering
"""
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
import os
os.chdir("/content/drive/My Drive/ML_WO")
!ls
"""
```

Out[2]:
```
'\nfrom google.colab import drive \ndrive.mount(\'/content/drive\',force_remount=True)\nimport os \nos.chdir("/content/drive/My Drive/ML_WO")\n!ls\n'
```

In [3]:
```
# Load data in Data Frame
df_review = pd.read_csv('IMDB Dataset.csv')
df_review
```

Out[3]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive

	review	sentiment
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

Wir nehmen eine Teilmenge von 2000 positiven und 2000 negativen Bewertungen, um unbalancierte Daten zu bekommen.

In [4]:
```
df_positive = df_review[df_review['sentiment']=='positive'][:2000]
df_negative = df_review[df_review['sentiment']=='negative'][:2000]
df_review_2 = pd.concat([df_positive, df_negative])
```

In [5]:
```
df_review_2
```

Out[5]:

	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
4	Petter Mattei's "Love in the Time of Money" is...	positive
5	Probably my all-time favorite movie, a story o...	positive
...	...	...
3953	Let's see: what are the advantages to watching...	negative
3954	For one thing, he produced this movie. It has ...	negative
3956	Contains spoilers The movie plot can be summar...	negative
3959	After high school Track & Field athelete, Laur...	negative
3961	The summary pretty much sums it all up. This i...	negative

4000 rows × 2 columns

In [6]:
```
print(sum(df_review_2['sentiment'] == 'positive'))
print(sum(df_review_2['sentiment'] == 'negative'))
```

```
2000
2000
```

Machine Learning

Scikit-learn

Scikit-learn ist eine freie Software-Bibliothek zum maschinellen Lernen für die Programmiersprache Python. Es bietet verschiedene Klassifikations-, Regressions- und Clustering-Algorithmen.
https://scikit-learn.org/stable/

Daten in Trainingsdaten und ein Testdaten aufspalten

Um ein Machine Learning Modell korrekt zu trainieren, wird ein Datensatz benötigt. Bei überwachten Lernverfahren wird dieser Datensatz in der Regel in mindestens zwei verschiedene Datensätze unterteilt: Training- und Testdaten. Oft gibt es noch einen weiteren Datensatz: Validierungsdaten.

Trainingsdaten
Ein Trainingsdatensatz ist ein Datensatz mit Beispielen (oder auch Zielvariablen genannt), die für das Lernen der Muster und Zusammenhänge in den Daten verwendet wird. Die Anpassung der Gewichte des Algorithmus wird über den Trainingsdatensatz antrainiert d.h. der Algorithmus lernt aus diesen Daten. Trainingsdaten mit Beispielen werden für Klassifikations- und Regressionsprobleme benötigt.

Testdaten
Die Testdaten werden bei dem Training nicht genutzt d.h. der Algorithmus kennt die Daten nicht und kann diese nicht zum Lernen nutzen. Auch hier sind Beispiele bzw. Zielvariablen vorhanden, woran im Anschluss die Qualität des Modells gemessen werden kann.

In [7]:
```
from sklearn.model_selection import train_test_split
train, test = train_test_split(df_review_2, test_size=0.33, random_state=42)
```

In [8]:
```
train_x, train_y = train['review'], train['sentiment']
test_x, test_y = test['review'], test['sentiment']
```

In [9]:
```
train_x
```

Out[9]:
```
750     I was surprised that " Forgiving the Franklins...
2910    The movie appeals to public due to charisma of...
2035    I have a question for the writers and producer...
1695    (This review is based on the English language ...
2280    The Second Renaissance, part 1 let's us show h...
                              ...
2235    This was one of my favorite series when I was ...
2565    I love horses and admire hand drawn animation,...
1685    It's unbelievable but the fourth is better tha...
3026    Well, I generally like Iranian movies, and aft...
2393    "Fungicide" is quite possibly the most incompe...
Name: review, Length: 2680, dtype: object
```

In [10]:
```
test_x
```

Out[10]:
```
1110    Brilliant thriller, deserving far more fame, M...
2999    I hired this movie expecting a few laughs, hop...
1048    A proof that it's not necessary for a movie to...
3831    Some amusing humor, some that falls flat, some...
1990    - A newlywed couple move into the home of the ...
                            ...
682     Created in 1928, and originally named Mortimer...
3312    I'm sick of people whining about Ewoks! True, ...
490     Relentless like one of those loud action movie...
1412    San Francisco is a big city with great acting ...
569     I watched this movie on TV last night, hoping ...
Name: review, Length: 1320, dtype: object
```

Textrepräsentation (Bag of Words + CountVectorizer)

Klassifizierer und Lernalgorithmen erwarten eher numerische Merkmalsvektoren als rohe Textdokumente. Aus diesem Grund müssen wir unseren Filmrezensionstext in numerische Vektoren umwandeln. Es gibt viele Textdarstellungstechniken.

In diesem einfachen Fall werden wir **Bag of Words (BOW)** verwenden, da wir uns für die Häufigkeit der Wörter in den Filmkritiken interessieren; die Reihenfolge der Wörter ist jedoch irrelevant. Eine gängige Methode zur Darstellung von Bag of Words sind CountVectorizer.

Der CountVectorizer gibt die Häufigkeit des Auftretens von Wörtern in einem Dokument an. Betrachten wir die folgenden Sätze:

```
review = [“I love writing code in Python. I love Python code”,
          “I hate writing code in Java. I hate Java code”]
```

Die Representation mit CountVectorizer ist:

	code	hate	java	love	python	writing
review[1]	2	0	0	2	2	1
review[2]	2	2	2	0	0	1

In [11]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [12]:
```
tfidf = TfidfVectorizer(stop_words='english')
train_x_vector = tfidf.fit_transform(train_x)
```

In [13]:
```
train_x_vector
```

Out[13]:
```
<2680x29142 sparse matrix of type '<class 'numpy.float64'>'
        with 239505 stored elements in Compressed Sparse Row format>
```

Sparse Matrix

Man bezeichnet eine Matrix, bei der so viele Einträge aus Nullen bestehen, als **dünnbesetzte oder schwachbesetzte Matrix** (englisch **sparse matrix**). Man versucht, dies insbesondere hinsichtlich Algorithmen sowie Speicherung auszunutzen.

In [14]:
```
pd.DataFrame.sparse.from_spmatrix(train_x_vector,
                                  index=train_x.index,
                                  columns=tfidf.get_feature_names_out())
```

Out[14]:

	00	000	007	01	01pm	04	06	07	08	09	...	zzzzzzzzzzzzzzzzz	æon	ça	émigrés	ê
750	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
2910	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
2035	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
1695	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
2280	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2235	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
2565	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
1685	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
3026	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	
2393	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	

2680 rows × 29142 columns

Auswahl des Modells

Entscheidungsbaum

https://de.wikipedia.org/wiki/Entscheidungsbaum

In [15]:
```
from sklearn.tree import DecisionTreeClassifier

dec_tree = DecisionTreeClassifier()
dec_tree.fit(train_x_vector, train_y)
```

Out[15]:
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

In [16]:
```
print(dec_tree.predict(tfidf.transform(['A good movie'])))
```

```
['positive']
```

In [17]:
```
print(dec_tree.predict(tfidf.transform(['A bad movie'])))
```

```
['negative']
```

In [18]:
```
print(dec_tree.predict(tfidf.transform(['The movie sucks'])))
```

```
['positive']
```

Support Vector Machine

Eine **Support Vector Machine** (**SVM**) unterteilt eine Menge von Objekten so in Klassen, dass um die Klassengrenzen herum ein möglichst breiter Bereich frei von Objekten bleibt.
https://de.wikipedia.org/wiki/Support_Vector_Machine

In [19]:
```
from sklearn.svm import SVC

svc = SVC(kernel='linear')
svc.fit(train_x_vector, train_y)
```

Out[19]:
```
SVC(kernel='linear')
```

In [20]:
```
print(svc.predict(tfidf.transform(['A good movie'])))
```

```
['positive']
```

In [21]:
```
print(svc.predict(tfidf.transform(['A bad movie'])))
```

```
['negative']
```

In [22]:
```
print(svc.predict(tfidf.transform(['The movie sucks'])))
```

```
['negative']
```

In [23]:
```
print(svc.predict(tfidf.transform(['A terrible movie'])))
```

```
['negative']
```

In [24]:
```
print(svc.predict(tfidf.transform(['I like this movie'])))
```

```
['negative']
```

Evaluation der Modelle

Messung der Genauigkeit der Modelle.

Mittlere Genauigkeit der Modelle

In [25]:
```
test_x_vector = tfidf.transform(test_x)
```

In [26]:
```
dec_tree.score(test_x_vector, test_y)
```

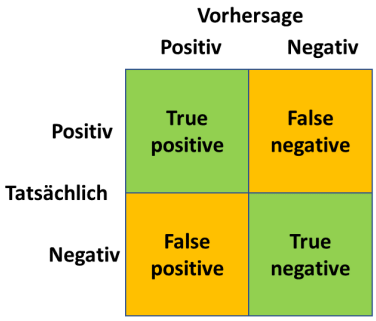
Out[26]:
```
0.6727272727272727
```

In [27]:
```
svc.score(test_x_vector, test_y)
```

Out[27]:
```
0.8560606060606061
```

Confusion Matrix

Eine **Konfusionsmatrix** ist eine Tabelle, die eine Visualisierung der Leistung eines Algorithmus ermöglicht. Diese Tabelle besteht oft aus zwei Zeilen und zwei Spalten, in denen die Anzahl der falsch-positiven, falsch-negativen, wahr-positiven und wahr-negativen Ergebnisse angegeben wird.



In [28]:
```
from sklearn.metrics import confusion_matrix
```

In [29]:
```
conf_mat = confusion_matrix(test_y,
                            dec_tree.predict(test_x_vector),
                            labels=['positive', 'negative'])
print("Confusion Matrix Decision Tree")
conf_mat
```

Out[29]:
```
Confusion Matrix Decision Tree
array([[452, 236],
       [196, 436]], dtype=int64)
```

In [30]:
```
conf_mat = confusion_matrix(test_y,
                            svc.predict(test_x_vector),
                            labels=['positive', 'negative'])
print("Confusion Matrix SVC")
conf_mat
```

Out[30]:
```
Confusion Matrix SVC
array([[600,  88],
       [102, 530]], dtype=int64)
```

Qualität: Abhängig vom Modell und den Parametern

Beispiel-Datensatz mit klinischen Messungen von Brustkrebs-Tumoren mit 30 Feature und Klassen "harmlos" (benign) und "gefährlich" (malignant).

In [31]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

cancer = load_breast_cancer()

print("Sample counts per class:\n",
```

```
      {n: v for n, v in zip(cancer.target_names, np.bincount(cancer.target))})
print("\nFeature names:\n", cancer.feature_names)
print("\nFirst rows of data:\n", cancer['data'][:3])
```

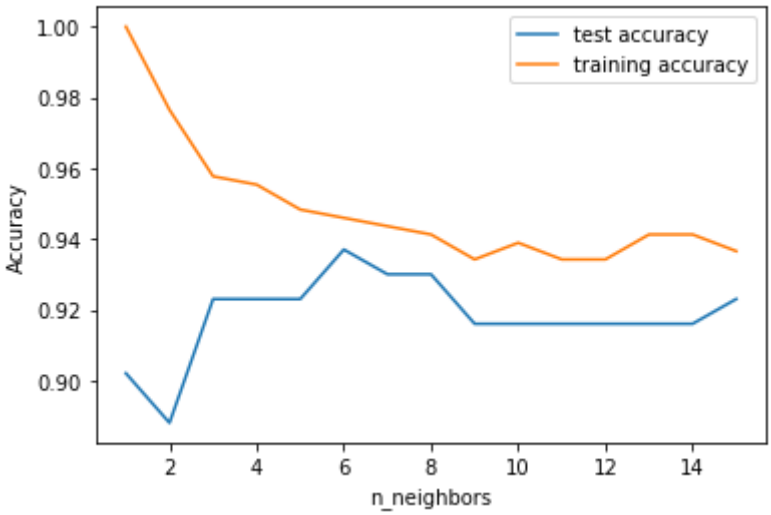
```
Sample counts per class:
 {'malignant': 212, 'benign': 357}

Feature names:
 ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']

First rows of data:
 [[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
  2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.974e-01
  1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00 9.403e+01
  6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03 2.357e+01
  2.553e+01 1.525e+02 1.709e+03 1.444e-01 4.245e-01 4.504e-01 2.430e-01
  3.613e-01 8.758e-02]]
```

In [32]:

```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=66)

training_accuracy = []
test_accuracy = []
# try n_neighbors
max_n = 15

neighbors_settings = range (1, max_n + 1)
for i in neighbors_settings:
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(X_train, y_train)

    training_accuracy.append(clf.score(X_train, y_train))
    test_accuracy.append(clf.score(X_test, y_test))

plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
```

Out[32]:

```
<matplotlib.legend.Legend at 0x263bdcece20>
```



Wenn man nur einen einzigen nächsten Nachbarn berücksichtigt, ist die Vorhersage für die Trainingsdaten natürlich perfekt. Hier sieht man, dass es wichtig ist, eine Unterteilung in Trainings- und Testdaten zu machen.

Die Anzahl von Nachbarn, für die die Genauigkeit am höchsten ist, ist abhängig vom Testdatensatz, aber in den Beispielen immer unter 15, oft unter 10. Danach sinkt sowohl die Genauigkeit als auch die Performanz des Modells. Also mehr ist nicht unbedingt besser. Bei einem nächsten Nachbarn ist die Genauigkeit immer noch bei 90% - vielleicht reicht das schon.

In [33]:
```
names = ["Nearest_Neighbors", "Linear_SVM", "Decision_Tree"]
classifiers = [
    KNeighborsClassifier(n_neighbors=3),
    SVC(kernel="linear", C=0.025),
    DecisionTreeClassifier(max_depth=5)]
```

In [34]:
```
scores = []
for name, clf in zip(names, classifiers):
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    scores.append(score)
```

In [35]:
```
scores_df = pd.DataFrame()
scores_df['name'] = names
scores_df['score'] = scores
scores_df.sort_values('score', ascending= False)
```

Out[35]:

	name	score
1	Linear_SVM	0.937063
0	Nearest_Neighbors	0.923077
2	Decision_Tree	0.909091