Переобучение

 Необходимым условием использования нейросетей является их возможность к обобщению – делать правильные предск5азания не только для обучающего набора данных, но и для других, ранее не используемых данных.

 Переобучение заключается в том, что нейросеть хорошо работает на тренировочном наборе данных, но теряет способность к обобщению.

С технической точки зрения, переобучение происходит, когда нейросеть имеет много параметров относительно объёма и разнообразия тренировочных данных. Нейросеть начинает подстраивается под случайные отклонения в данных, вместо того чтобы уловить истинные закономерности.

**Признаки переобучения:**

* **Разница в значениях функций потерь:**

С увеличением числа итераций возрастает разница в значениях функций потерь для обучающего и тестового наборов данных.

* **Разница в производительности:**

Модель демонстрирует высокую точность на обучающем наборе данных, но её производительность резко падает при работе с новыми, тестовыми данными.

* **Сложные границы принятия решений:**

Визуально можно увидеть, что границы, разделяющие классы, становятся очень извилистыми и избыточными, что свидетельствует о попытке "запомнить" каждый отдельный тренировочный пример.

**Методы борьбы с переобучением:**

Существует ряд стратегий, которые помогают снизить вероятность переобучения и улучшить способность модели к обобщению:

* [**Ранняя остановка**](https://www.google.com/search?newwindow=1&sca_esv=193255194e7c12f9&cs=0&sxsrf=AE3TifOxDiWGXusV9h-IGbAP0lsI14qnRQ%3A1759829899085&q=%D0%A0%D0%B0%D0%BD%D0%BD%D1%8F%D1%8F+%D0%BE%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0&sa=X&ved=2ahUKEwjhrqWf5ZGQAxUABhAIHbJ7M_QQxccNegQIKBAB&mstk=AUtExfAb3vOMMX8fE_3gEe_cwF1fJ8lkClk_8xzNsOKLJeGbMbzKE7A7k7tYOwQK2ZcDi40c7pBOnN573f4nvClZ1LkTS9zHYwFxFacmsRvCnhlePlPMDP3zVl_cACdtqhFm7EHzf0TJCueZv7Rf9DBch_eJScflaC2D-wN4mqre2iaea_0&csui=3)**:**

Процесс обучения прекращается до того, как модель начнет переобучаться, обычно, когда точность на проверочном наборе данных перестает улучшаться.

* [**Увеличение набора данных**](https://www.google.com/search?newwindow=1&sca_esv=193255194e7c12f9&cs=0&sxsrf=AE3TifOxDiWGXusV9h-IGbAP0lsI14qnRQ%3A1759829899085&q=%D0%A3%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B5%D0%BD%D0%B8%D0%B5+%D0%BD%D0%B0%D0%B1%D0%BE%D1%80%D0%B0+%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85&sa=X&ved=2ahUKEwjhrqWf5ZGQAxUABhAIHbJ7M_QQxccNegQIKRAB&mstk=AUtExfAb3vOMMX8fE_3gEe_cwF1fJ8lkClk_8xzNsOKLJeGbMbzKE7A7k7tYOwQK2ZcDi40c7pBOnN573f4nvClZ1LkTS9zHYwFxFacmsRvCnhlePlPMDP3zVl_cACdtqhFm7EHzf0TJCueZv7Rf9DBch_eJScflaC2D-wN4mqre2iaea_0&csui=3)**:**

Предоставление модели большего количества данных для обучения помогает ей выявлять более общие закономерности, а не запоминать шум в конкретных примерах.

* [**Регуляризация**](https://www.google.com/search?newwindow=1&sca_esv=193255194e7c12f9&cs=0&sxsrf=AE3TifOxDiWGXusV9h-IGbAP0lsI14qnRQ%3A1759829899085&q=%D0%A0%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F&sa=X&ved=2ahUKEwjhrqWf5ZGQAxUABhAIHbJ7M_QQxccNegQIKhAB&mstk=AUtExfAb3vOMMX8fE_3gEe_cwF1fJ8lkClk_8xzNsOKLJeGbMbzKE7A7k7tYOwQK2ZcDi40c7pBOnN573f4nvClZ1LkTS9zHYwFxFacmsRvCnhlePlPMDP3zVl_cACdtqhFm7EHzf0TJCueZv7Rf9DBch_eJScflaC2D-wN4mqre2iaea_0&csui=3)**:**

К основной функции потерь добавляется "штраф" за сложность модели, что стимулирует модель использовать только наиболее важные признаки и предотвращает избыточную подгонку.

* [**Прореживание (Dropout)**](https://www.google.com/search?newwindow=1&sca_esv=193255194e7c12f9&cs=0&sxsrf=AE3TifOxDiWGXusV9h-IGbAP0lsI14qnRQ%3A1759829899085&q=%D0%9F%D1%80%D0%BE%D1%80%D0%B5%D0%B6%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5+%28Dropout%29&sa=X&ved=2ahUKEwjhrqWf5ZGQAxUABhAIHbJ7M_QQxccNegQILBAB&mstk=AUtExfAb3vOMMX8fE_3gEe_cwF1fJ8lkClk_8xzNsOKLJeGbMbzKE7A7k7tYOwQK2ZcDi40c7pBOnN573f4nvClZ1LkTS9zHYwFxFacmsRvCnhlePlPMDP3zVl_cACdtqhFm7EHzf0TJCueZv7Rf9DBch_eJScflaC2D-wN4mqre2iaea_0&csui=3)**:**

Во время обучения случайным образом "отключается" часть нейронов, что предотвращает сильную зависимость модели от отдельных нейронов и способствует созданию более робастной модели.

* [**Батч-нормализация**](https://www.google.com/search?newwindow=1&sca_esv=193255194e7c12f9&cs=0&sxsrf=AE3TifOxDiWGXusV9h-IGbAP0lsI14qnRQ%3A1759829899085&q=%D0%91%D0%B0%D1%82%D1%87-%D0%BD%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F&sa=X&ved=2ahUKEwjhrqWf5ZGQAxUABhAIHbJ7M_QQxccNegQIKxAB&mstk=AUtExfAb3vOMMX8fE_3gEe_cwF1fJ8lkClk_8xzNsOKLJeGbMbzKE7A7k7tYOwQK2ZcDi40c7pBOnN573f4nvClZ1LkTS9zHYwFxFacmsRvCnhlePlPMDP3zVl_cACdtqhFm7EHzf0TJCueZv7Rf9DBch_eJScflaC2D-wN4mqre2iaea_0&csui=3)**:**

Нормализация данных на входе каждого слоя помогает стабилизировать процесс обучения и снижает риск переобучения.

Тестовый пример

 Рассмотрим работу нейросети в зависимости от отношения числа нейронов к числу объектов в тренировочном наборе данных.

 Аппроксимация функции $y(x)=0.5∙x+G\left(μ,σ\right)$

$G\left(μ,σ\right)$ – случайная величина с нормальным распределением.

Сеть – 2 слоя, скрытый слой 3 нейрона.

***Обучающая и тестовая последовательности – 300 точек:***



чёрные точки – обучающий набор, σ=0.52

красные точки – тестовый набор, σ=0.55

синяя линия – результат аппроксимации

***Обучающая и тестовая последовательности – 100 точек:***



обучающий набор - σ=0.45

тестовый набор - σ=0.45

***Обучающая и тестовая последовательности – 30 точек:***



обучающий набор - σ=0.52

тестовый набор - σ=0.51

***Обучающая и тестовая последовательности – 10 точек:***



обучающий набор - σ=0.45

тестовый набор - σ=0.78

***Обучающая и тестовая последовательности – 5 точек:***



обучающий набор - σ=0.02

тестовый набор - σ=0.58

Эффект переобучения!

Реальный датасет

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import make\_column\_transformer

from sklearn.model\_selection import GroupShuffleSplit

spotify = pd.read\_csv('./spotify.csv')

spotify.info()

X = spotify.copy().dropna()

y = X.pop('track\_popularity')

artists = X['track\_artist']

features\_num = ['danceability', 'energy', 'key', 'loudness', 'mode',

 'speechiness', 'acousticness', 'instrumentalness',

 'liveness', 'valence', 'tempo', 'duration\_ms']

features\_cat = ['playlist\_genre']

preprocessor = make\_column\_transformer(

 (StandardScaler(), features\_num),

 (OneHotEncoder(), features\_cat),

)

def group\_split(X, y, group, train\_size=0.75):

 splitter = GroupShuffleSplit(train\_size=train\_size)

 train, test = next(splitter.split(X, y, groups=group))

 return (X.iloc[train], X.iloc[test], y.iloc[train], y.iloc[test])

X\_train, X\_test, y\_train, y\_test = group\_split(X, y, artists)

X\_train = preprocessor.fit\_transform(X\_train)

X\_test = preprocessor.transform(X\_test)

y\_train = y\_train / 100 # popularity is on a scale 0-100, so this rescales to 0-1.

y\_test = y\_test / 100

input\_shape = [X\_train.shape[1]]

print("Input shape: {}".format(input\_shape))

#---------------------------------------------------------------

import keras

from keras.models import Sequential

from keras.layers import Dense, Activation

#model = keras.Sequential([layers.Dense(1, input\_shape=input\_shape),])

model = Sequential()

model.add(Dense(1, input\_shape=input\_shape))

model.compile(optimizer='adam',loss='mse')

history = model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), batch\_size=512, epochs=50, verbose=1)

history\_df = pd.DataFrame(history.history)

history\_df.loc[0:, ['loss', 'val\_loss']].plot()

plt.show()

print("Minimum Validation Loss: {:0.4f}".format(history\_df['val\_loss'].min()));



#-----------------------------------------------------------------

model = keras.Sequential()

model.add (Dense(128, activation='relu', input\_shape=input\_shape))

model.add (Dense(64, activation='relu'))

model.add (Dense(1))

model.compile(optimizer='adam', loss='mse')

history = model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), batch\_size=512, epochs=50)

history\_df = pd.DataFrame(history.history)

history\_df.loc[:, ['loss', 'val\_loss']].plot()

plt.show()



#--------------------------------------------------------------------

model = keras.Sequential()

model.add (Dense(128, activation='relu', input\_shape=input\_shape))

model.add (Dense(64, activation='relu'))

model.add (Dense(1))

model.compile(optimizer='adam', loss='mse')

callback = keras.callbacks.EarlyStopping(monitor="loss", mode="min", verbose=1, patience=100, min\_delta=0.01)

history = model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), batch\_size=512, epochs=500, callbacks=[callback])

history\_df = pd.DataFrame(history.history)

history\_df.loc[:, ['loss', 'val\_loss']].plot()

plt.show()

print("Minimum Validation Loss: {:0.4f}".format(history\_df['val\_loss'].min()));

print('end')

