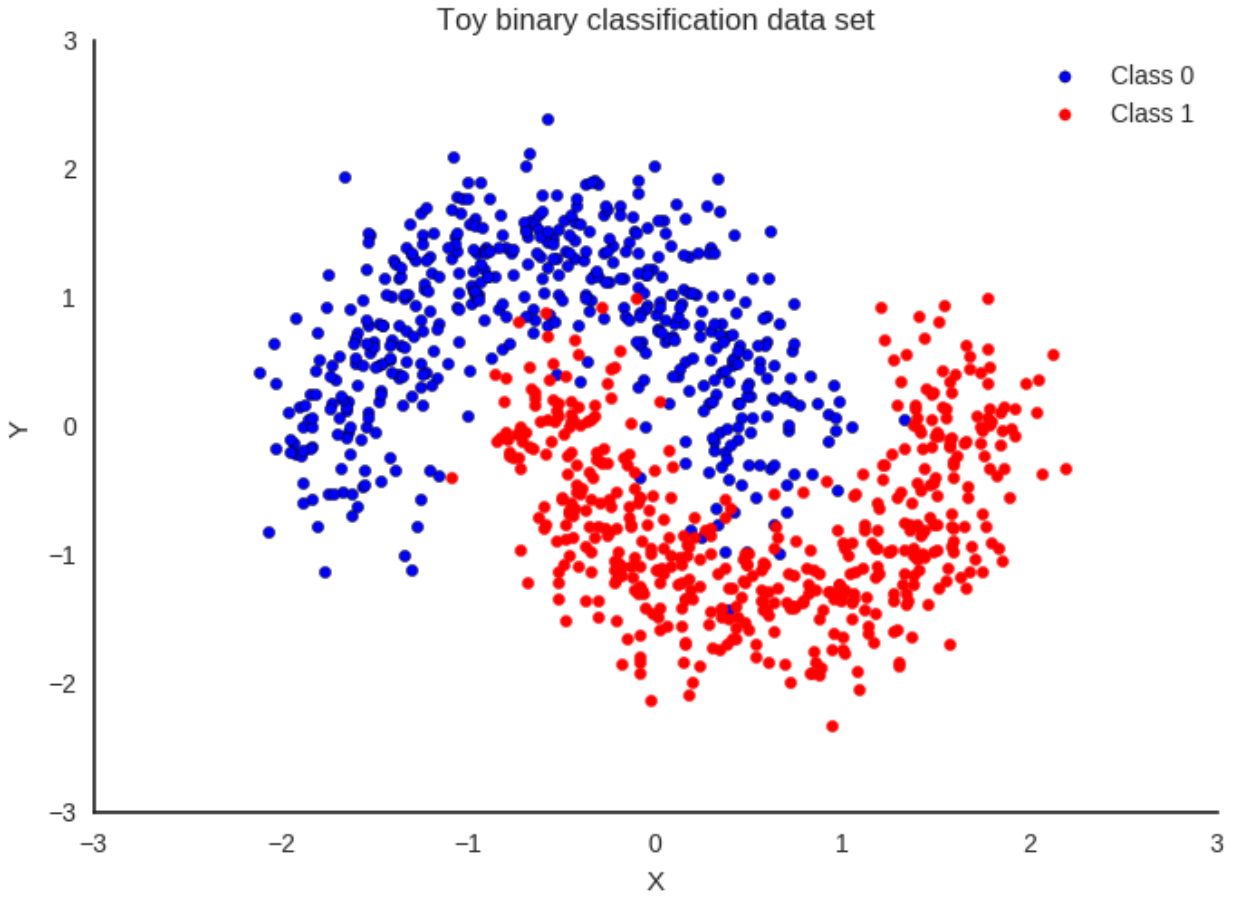
- İkili sınıflandırma (Binary Classification)
- Temel metin sınıflandırma örneği:Imdb

İkili sınıflandırma (Binary Classification)

- İkili sınıflandırma, verilerin iki sınıfa ayrılmasını içerir. Örneğin, bir müşterinin cinsiyet, yaş, yer vb. gibi bağımsız değişkenleri temel alarak belirli bir ürün alıp almadığı (Evet / Hayır) ikili sınıflandırma problemi olabilir.



https://www.kdnuggets.com/2017/04/must-know-evaluate-binary-classifier.html

İkili sınıflandırma örneği: IMDB film incelemeleri

- Derin öğrenmeyi kullanarak film incelemelerinden pozitif veya negatif görüşler tahmin edilebilir.
- Büyük Film İnceleme Veri Kümesi (genellikle IMDB veri kümesi olarak adlandırılır) eğitim için 25.000 kutuplu yorum (iyi veya kötü) ve test için 25.000 kutuplu yorum içerir.
- Problem, yapılan bir yorumun olumlu veya olumsuz bir anlama sahip olup olmadığını belirlemektir.

İkili sınıflandırma örneği: IMDB film incelemeleri

```
In [28]: from keras.datasets import imdb

In [29]: (train_data, train_labels), (test_data, test_labels)=imdb.load_data(num_words=10000)

In [30]: len(train_data)
Out[30]: 25000

In [31]: len(test_data)
Out[31]: 25000

In [32]: print(train_data[0])
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100,
43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172,
4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469,
4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 2,
5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38,
619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82,
2, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317,
46, 7, 4, 2, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18,
4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226,
65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]

In [33]: print(train_labels[0])
1

In [34]: print(train_labels[100])
0
```

https://www.tensorflow.org/tutorials/keras/basic_text_classification

İkili sınıflandırma örneği: IMDB film incelemeleri

```
In [120]: word_index = imdb.get_word_index()

In [121]: word_index = {k:(v+3) for k,v in word_index.items()}

In [122]: reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])

In [123]: str=[reverse_word_index.get(i, '?') for i in train_data[0]]

In [124]: print(str)
['?', 'this', 'film', 'was', 'just', 'brilliant', 'casting', 'location', 'scenery',
'story', 'direction', "everyone's", 'really', 'suited', 'the', 'part', 'they', 'played',
'and', 'you', 'could', 'just', 'imagine', 'being', 'there', 'robert', '?', 'is', 'an',
'amazing', 'actor', 'and', 'now', 'the', 'same', 'being', 'director', '?', 'father',
'came', 'from', 'the', 'same', 'scottish', 'island', 'as', 'myself', 'so', 'i', 'loved',
'the', 'fact', 'there', 'was', 'a', 'real', 'connection', 'with', 'this', 'film', 'the',
'witty', 'remarks', 'throughout', 'the', 'film', 'were', 'great', 'it', 'was', 'just',
'brilliant', 'so', 'much', 'that', 'i', 'bought', 'the', 'film', 'as', 'soon', 'as', 'it',
'was', 'released', 'for', '?', 'and', 'would', 'recommend', 'it', 'to', 'everyone', 'to',
'watch', 'and', 'the', 'fly', 'fishing', 'was', 'amazing', 'really', 'cried', 'at', 'the',
'end', 'it', 'was', 'so', 'sad', 'and', 'you', 'know', 'what', 'they', 'say', 'if', 'you',
'cry', 'at', 'a', 'film', 'it', 'must', 'have', 'been', 'good', 'and', 'this',
'definitely', 'was', 'also', '?', 'to', 'the', 'two', 'little', "boy's", 'that', 'played',
'the', '?', 'of', 'norman', 'and', 'paul', 'they', 'were', 'just', 'brilliant', 'children',
'are', 'often', 'left', 'out', 'of', 'the', '?', 'list', 'i', 'think', 'because', 'the',
'stars', 'that', 'play', 'them', 'all', 'grown', 'up', 'are', 'such', 'a', 'big',
'profile', 'for', 'the', 'whole', 'film', 'but', 'these', 'children', 'are', 'amazing',
'and', 'should', 'be', 'praised', 'for', 'what', 'they', 'have', 'done', "don't", 'you',
'think', 'the', 'whole', 'story', 'was', 'so', 'lovely', 'because', 'it', 'was', 'true',
'and', 'was', "someone's", 'life', 'after', 'all', 'that', 'was', 'shared', 'with', 'us',
'all']
```

İkili sınıflandırma örneği: IMDB film incelemeleri

```
16 def vectorize_sequences(sequences, dimension=10000):
17     # Sıfırlardan oluşan, (len(sequences), dimension)
18     results = np.zeros((len(sequences), dimension))
19     for i, sequence in enumerate(sequences):
20         results[i, sequence] = 1.
21     return results
22
23 (train_data, train_labels),(test_data, test_labels)\
24 =imdb.load_data(num_words=10000)
25
26 # Eğitim ve test verilerini vektöre dönüştür
27 x_train = vectorize_sequences(train_data)
28 x_test = vectorize_sequences(test_data)
29
30 # Etiketleri vektöre dönüştür
31 y_train = np.asarray(train_labels).astype('float32')
32 y_test = np.asarray(test_labels).astype('float32')
33
```

Sıfırlardan oluşan 10000 elemanlı bir vektör oluştur. 10000 kelimeden yorum içerisinde kullanılanları 1 diğerlerini 0 yap

Kaynak: https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/3.5-classifying-movie-reviews.ipynb

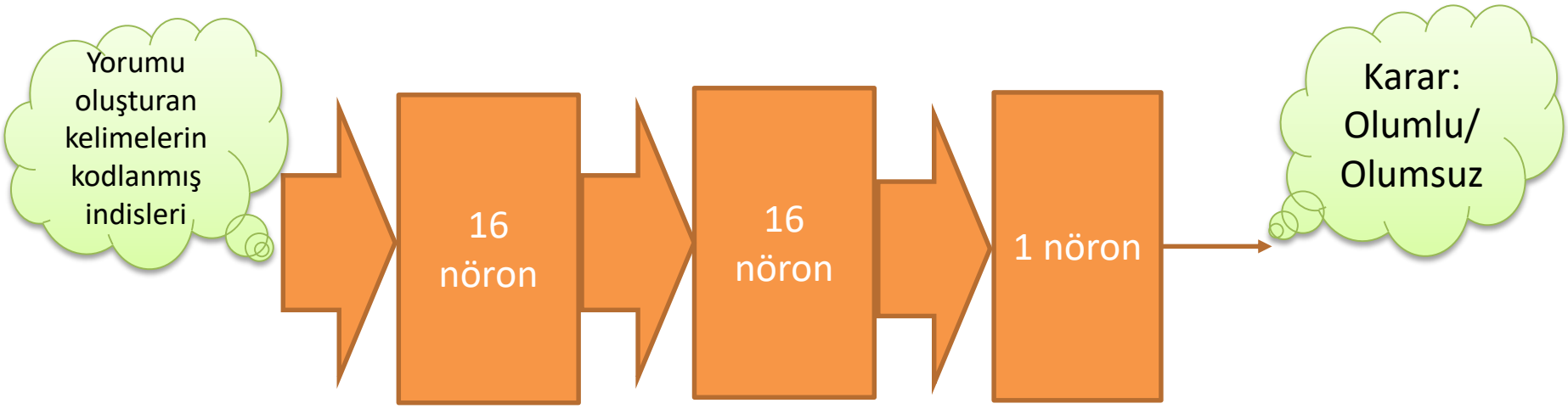
Veri setinin uygunlaştırılması

```
In [214]: train_data[0]
Out[214]:
[1,
 14,
 22,
 16,
 43,
 530,
 973,
 1622,
 1385,
 65,
 458,
 4468,
 66,
 3941,
 4,
 173,
 36,
 256,
 5,
 25,
 100,
 43,
 838,
 112,
 50,
 670,
 2,
 9,
 35,
```

```
In [262]: y=np.sort(train_data[0]).reshape(len(train_data[0]),1)

In [263]: y
Out[263]:
array([[   1],
       [   2],
       [   2],
       [   2],
       [   2],
       [   2],
       [   2],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   4],
       [   5],
       [   5],
       [   5],
       [   5],
       [   5],
       [   5],
       [   5],
       [   5],
       [   5],
       [   5],
       [   6],
       [   6],
       [   6],
       [   7],
       [   7],
       [   8],
       [   8],
       [   8],
```

```
In [269]: y=x_train[0].reshape(len(x_train[0]),1)

In [270]: y[0:50]
Out[270]:
array([[0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
```

Ağ modelinin tanımlanması




```
36 model=models.Sequential()
37
38 model.add(layers.Dense(16,
39                          activation='relu',
40                          input_shape=(10000,)))
41
42 model.add(layers.Dense(16,
43                          activation='relu'))
44
45 model.add(layers.Dense(1,
46                          activation='sigmoid'))
47
```

Optimizasyon parametreleri

- Kayıp (loss) fonksiyonu olarak mean_squared_error gibi bir fonksiyon kullanılabilir. Ağın çıkışı iki sınıfa ait bir olasılık belirteceği için binary_crossentropy tercih edilmiştir.
- Burada binary_accuracy ile eğitim sırasında doğruluk gözlenecektir.

```
52 model.compile(optimizer=optimizers.RMSprop(lr=0.001),
53                 loss='binary_crossentropy',
54                 metrics=['binary_accuracy'])
55
56 #model.compile(optimizer=optimizers.RMSprop(lr=0.001),
57 #                 loss=losses.binary_crossentropy,
58 #                 metrics=[metrics.binary_accuracy])
```

Optimizasyon parametreleri

```
model.fit(x_train,
          y_train,
          epochs=20,
          batch_size=500)
```

Doğrulama (Validation) yapılmadan eğitim.

```
x_validation = x_train[:5000]
partial_x_train = x_train[5000:]

y_validation = y_train[:5000]
partial_y_train = y_train[5000:]

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=500,
                    validation_data=(x_validation, y_validation))
```

Eğitim sırasında her veri setini dolaşımda doğruluk ölçülebilir. Bu işlem eğitimde kullanılmayan örneklerle yapılacağı için eğitim setinden belirtilen sayıda örneğin doğrulama için ayrılması gerekir.

Eğitimin geçekleştirilmesi

```
20000/20000 [==============================] - 4s 193us/step - loss: 0.0753 -
binary_accuracy: 0.9750 - val_loss: 0.3715 - val_binary_accuracy: 0.8798
Epoch 10/20
20000/20000 [==============================] - 4s 198us/step - loss: 0.0648 -
binary_accuracy: 0.9794 - val_loss: 0.4183 - val_binary_accuracy: 0.8748
Epoch 11/20
20000/20000 [==============================] - 5s 234us/step - loss: 0.0517 -
binary_accuracy: 0.9856 - val_loss: 0.4532 - val_binary_accuracy: 0.8710
Epoch 12/20
20000/20000 [==============================] - 4s 215us/step - loss: 0.0486 -
binary_accuracy: 0.9857 - val_loss: 0.4601 - val_binary_accuracy: 0.8708
Epoch 13/20
20000/20000 [==============================] - 4s 206us/step - loss: 0.0371 -
binary_accuracy: 0.9900 - val_loss: 0.4971 - val_binary_accuracy: 0.8700
Epoch 14/20
20000/20000 [==============================] - 4s 194us/step - loss: 0.0346 -
binary_accuracy: 0.9901 - val_loss: 0.5306 - val_binary_accuracy: 0.8700
Epoch 15/20
20000/20000 [==============================] - 4s 196us/step - loss: 0.0265 -
binary_accuracy: 0.9935 - val_loss: 0.5627 - val_binary_accuracy: 0.8688
Epoch 16/20
20000/20000 [==============================] - 4s 198us/step - loss: 0.0266 -
binary_accuracy: 0.9930 - val_loss: 0.5886 - val_binary_accuracy: 0.8668
Epoch 17/20
20000/20000 [==============================] - 4s 192us/step - loss: 0.0195 -
binary_accuracy: 0.9953 - val_loss: 0.6267 - val_binary_accuracy: 0.8676
Epoch 18/20
20000/20000 [==============================] - 4s 205us/step - loss: 0.0171 -
binary_accuracy: 0.9962 - val_loss: 0.6483 - val_binary_accuracy: 0.8704
Epoch 19/20
20000/20000 [==============================] - 4s 208us/step - loss: 0.0136 -
binary_accuracy: 0.9972 - val_loss: 0.6812 - val_binary_accuracy: 0.8648
Epoch 20/20
20000/20000 [==============================] - 4s 193us/step - loss: 0.0114 -
binary_accuracy: 0.9972 - val_loss: 0.7083 - val_binary_accuracy: 0.8674

In [299]:
```

- fit() fonksiyonu ile eğitim gerçekleştirildiğinde yandaki gibi her bir iterasyonda optimizasyonun durumu hakkında bilgiler yazdırılmaktadır.

train history

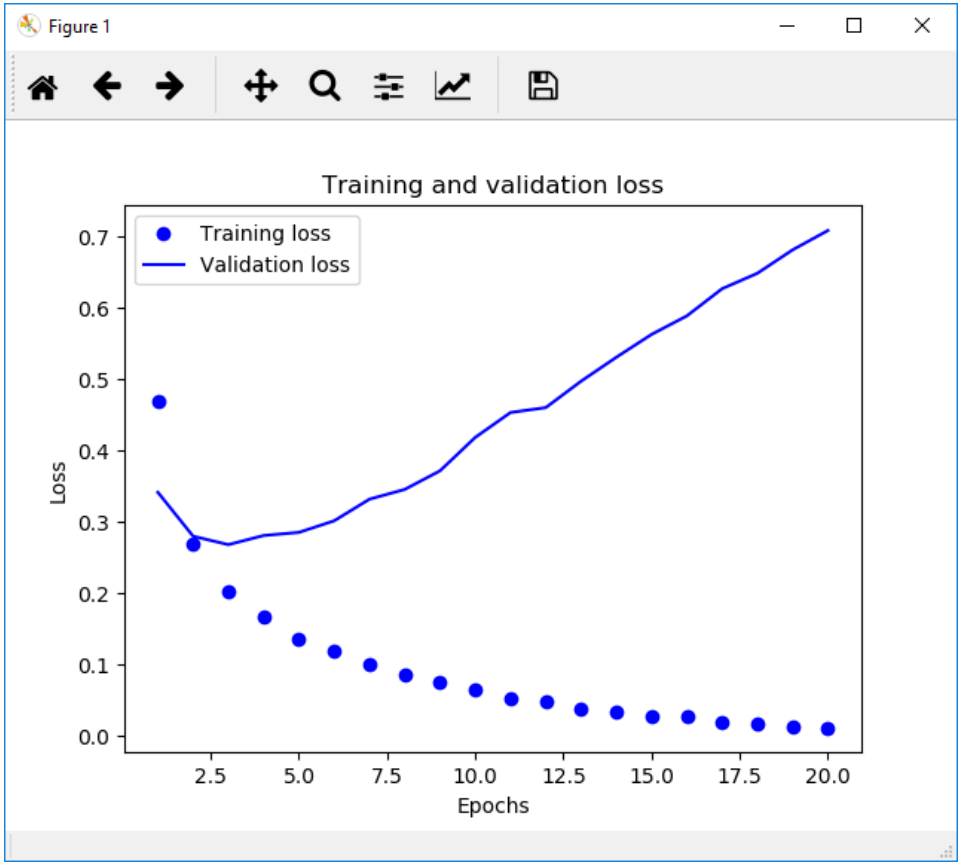
```
In [304]: history.params
Out[304]:
{'batch_size': 500,
 'epochs': 20,
 'steps': None,
 'samples': 20000,
 'verbose': 1,
 'do_validation': True,
 'metrics': ['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy']}

In [305]: history.history['loss']
Out[305]:
[0.4691212125122547,
 0.2696381252259016,
 0.20318141616880894,
 0.16699168421328067,
 0.13623715490102767,
 0.118472003005445,
 0.10028673037886619,
 0.08507895935326815,
 0.07529340004548431,
 0.06475454457104206,
 0.05171188111416995,
 0.04857624201104045,
 0.037075738934800026,
 0.034565999545156954,
 0.026503147394396364,
 0.026570425694808365,
 0.019534876430407168,
 0.017106045153923333,
 0.013624661194626242,
 0.011400983622297644]

In [306]:
```

Eğtim sonunda fit() ile döndürülen history nesnesinden optimizayon parametrelerinin iterasyona (epoch) bağlı değişimi incelenebilir.

Yanda, kayıp (loss) fonksiyonunun iterasyona bağlı değerleri listelenmiştir.

Grafik çizimleri (History)

```
history_dict = history.history
loss_values = history_dict['loss']#grafik 1
val_loss_values = history_dict['val_loss']#grafik 2
epochs = range(1,len(loss_values ) + 1) # yatay eksen: 1..20
plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Test

```
In [320]: test_loss, test_acc = model.evaluate(x_test,y_test)
25000/25000 [==============================] - 4s 161us/step

In [321]: print(test_acc)
0.84808

In [322]: print(test_loss)
0.8149055649638176
```

- Eğitim setinde %86 civarında elde edilen başarımın test setinde %84 civarına düştüğü görülüyor.

Deneyler

- Katman sayısını azalt veya artır: 1 katman veya 3 katman
- Katmanlarda kullanılan birim sayısını artır:32, 48 veya 64
- Loss fonksiyonunu değiştir: mse
- Activasyon fonksiyonlarını değiştir: Relu yerine tanh kullan
- Optimizasyon yöntemini değiştir: Adam

https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/3.5-classifying-movie-reviews.ipynb