При выборе технологий для решния задачи я искал инстументы, позволяющие наиболее быстро и удобно экспериментировать с изменениями. Под эти критерии подходят:
- язык **Python** как немногословный и интерпретируемый
- высокоуровневая бибиотека машинного обучения **Keras**
- **Pandas** как средство парсинга csv и гибкой трансформации таблиц

Поставленная задача – задача **бинарной классификации** по смешанным входным данным (признаковое описание + изображения). Её можно решать, например, следующими методами машинного обучения:

k ближайших соседей. Вполне правдоподобно, что похожие фильмы собираются в кластеры с небольшим расстоянием (например, по евклидовой метрике) между объеками классификации в каждом кластере. Мне кажется, этот метод может дать хорошее качество классификации, плюс понятность и визуализируемость результатов.

Решающее дерево может дать отличную интерпретируемость модели на естественном языке.

Нейронная сеть. На мой взгляд, самый интересный для экспериментов (и в то же время дающий хорошие результаты) вариант, т.к. можно варьировать количество скрытых слоёв, количество нейронов в слое, функции активации, топологию сети и многие другие параметры. Я выбрал именно эту модель.

Был быстро собран первый рабочий прототип. Чтобы понять, как я пришёл к дальнейшим решениям, рассмотрим этот прототип и его проблемы.

Для начала я отобрал только количественные признаки чтобы сразу загнать их в нейронную сеть.

```
features_names = [
    'Duration',
    # 'Language',
    # 'Country',
    # 'Rating',
    'Action',
    'Adventure',
    'Animation',
    'Biography',
    'Crime',
    'Documentary',
    'Drama',
    'Family',
    'Fantasy',
    'History',
    'Horror',
    'Music',
    'Musical',
    'Mystery',
    'News',
    'Romance',
    'Sci-Fi',
    'Sport',
    'Thriller',
    'War',
    'Western',
    # 'Poster'
]
```

Теперь пару слов о самой нейронке. Я выбрал полносвязный многослойный перцептрон, или deep feedforward сеть.

```
def construct_neural_network():
    nn = Sequential()
    nn.add(Dense(8, input_dim=22, activation='relu'))
    nn.add(Dense(12, activation='relu'))
    nn.add(Dense(1, activation='sigmoid'))

    nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return nn
```

22 входа по количеству признаков, **два скрытых** слоя с **ReLU**-активацией на **8 и 12 нейронов** и **выходной** нейрон с **сигмоидальной** функцией. Выбор функций активации определяется видом задачи (бинарная классификация): в частности, сигмоид выдаёт на выход значение от 0 до 1 и симметричен относительно 0.5, а значит с помощью этой функции можно получать **вероятность**, что объект относится к классу 1 – то, что нам нужно. Будем минимизировать **перекрёстную энтропию** – типичный подход для классификации, как и использование метода оптимизации **adaptive moment estimation**.

Поскольку сеть довольно сложная (два скрытых слоя), я сразу задумался о проблеме **переобучения**. Натренировать модель на всей обучающей выборке, а затем применить её к данным с неизвестным значением target – недостаточно хороший подход. Я решил тренировать нейронку не на всём наборе из train.csv, а выделить отдельно чуть более ста объектов, не скармливая их модели при обучении (далее я **условно называю** именно их **тестовой подвыборкой** (лучше было назвать её контрольной или cross-validation, но это я понял к концу работы)).

```
def read_data_from_csv(train_size: int) -> tuple:
    train_data = pd.read_csv('train.csv', index_col='Id')

    features_names = [...]
    train_features = train_data[features_names][:train_size].values
    train_labels = train_data['Target'][:train_size].values

    test_features = train_data[features_names][train_size:].values
    test_labels = train_data['Target'][train_size:].values

    return (train_features,
            train_labels,
            test_features,
            test_labels)


def start():
    np.random.seed(33)

    train_features, train_labels, test_features, test_labels = read_data_from_csv(train_size=3500)
```

Когда обучение завершено, эти объекты используются для проверки модели “в боевых условиях” - если на них модель предсказывает хуже, значит, нейронка просто заучила ответы.

Кстати, random seed здесь задаётся ради воспроизводимости результатов. Как потом оказалось, при разном семени точность предсказания прототипа варьируется приблизительно ±10%. Это повод считать, что модель неустойчива и требует улучшений.

Итак, обучим модель, затем посмотрим на достоверность предсказаний:

```
model = construct_neural_network()
model.fit(train_features, train_labels, epochs=100, batch_size=512)

predicted_test_labels = model.predict_classes(test_features)

matches = [predicted_test_labels[i] == test_labels[i] for i in range(len(test_labels))].count(True)
print('\n\n{} out of {}. Accuracy = {:.2f}%'.format(matches, len(test_labels), matches/len(test_labels)*100))
```

В результате имеем около восьмидесяти процентов точности как на собственно обучащей выборке, так и на тестовой подвыборке — следовательно, переобучения нет.

```
Epoch 97/100
 512/3500 [===>..........................] - ETA: 0s - loss: 0.4528 - acc: 0.7871
3500/3500 [==============================] - 0s - loss: 0.4520 - acc: 0.7883
Epoch 98/100
 512/3500 [===>..........................] - ETA: 0s - loss: 0.4444 - acc: 0.7812
3500/3500 [==============================] - 0s - loss: 0.4537 - acc: 0.7849
Epoch 99/100
 512/3500 [===>..........................] - ETA: 0s - loss: 0.4927 - acc: 0.7676
3500/3500 [==============================] - 0s - loss: 0.4508 - acc: 0.7897
Epoch 100/100
 512/3500 [===>..........................] - ETA: 0s - loss: 0.4488 - acc: 0.7852
3500/3500 [==============================] - 0s - loss: 0.4534 - acc: 0.7877
 32/135 [======>.......................] - ETA: 0s

109 out of 135. Accuracy = 80.74%

Process finished with exit code 0
```

Прототип функционирует, теперь настало время поговорить о его проблемах:
1. На вход модели подаются не все признаки. Следует **преобразвать категориальные признаки** (рейтинг, страна-производитель) в несколько числовых (one hot encoding) и добавить в нейронную сеть входные нейроны для них.
2. Один из признаков — продолжительность — **не нормирован**, что может вызывать слишком большой перекос в значениях весов и неустойчивость модели. Следует преобразовать длительность из минут в некоторые значения от 0 до 1
3. Тестовая подвыборка бралась как 135 последних значений обучащей выборки. Если данные были собраны так, что существует закономерность, коррелирущая с id объекта, это вносит неточность в оценку качества предсказания модели. Данные следует **перемешать**.

Быстро устраним вторую и третью проблемы.
Нормировка методом minmax:

```
def normalize_duration(df):
    result = df.copy()
    max_value = df['Duration'].max()
    min_value = df['Duration'].min()
    result['Duration'] = (df['Duration'] - min_value) / (max_value - min_value)
    return result
```

Перемешивание:

```
train_data = train_data.iloc[np.random.permutation(len(train_data))]
```

Первая же проблема – поле для эксперимента. Например, колонка «язык» имеет следующий возможный набор значений:

```
print(train_data['Language'].unique())
```

```
['English' 'Norwegian' 'Spanish' nan 'French' 'German' 'Japanese' 'Dutch'
 'Hindi' 'Hebrew' 'Russian' 'Mandarin' 'Italian' 'Aramaic' 'Persian'
 'Filipino' 'Arabic' 'Dari' 'Thai' 'Portuguese' 'Chinese' 'Zulu' 'Danish'
 'Cantonese' 'None' 'Icelandic' 'Polish' 'Korean' 'Indonesian' 'Romanian'
 'Swedish' 'Greek' 'Bosnian' 'Urdu' 'Czech' 'Telugu' 'Dzongkha'
 'Aboriginal' 'Maya' 'Hungarian' 'Mongolian' 'Kazakh']
```

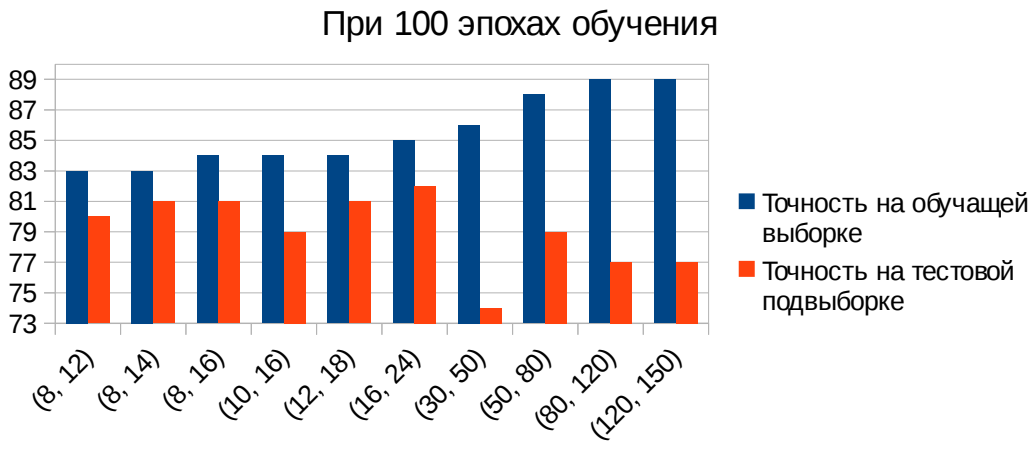
Понятно, что набор значений в выборке из test.csv может отличаться, однако многие значения встречаются и там, и там. Вопрос вот в чём: улучит ли качество предсказания добаваление таких данных в модель, или же лишь необоснованно переусложнит модель (число нейронов входного слоя возрастёт в несколько раз).

Итак, следующая незамысловатая функция выполняет one hot encoding любого категориального признака. В модель были добавлены данные о стране, языке и рейтинге.

```
def add_numerical_cols_from_categorical(df: 'pd.DataFrame', col_name: str) -> 'pd.DataFrame':
    languages = pd.get_dummies(df[col_name])
    result = pd.concat([df, languages], axis=1)
    global features_names
    features_names += list(languages)
    return result
```

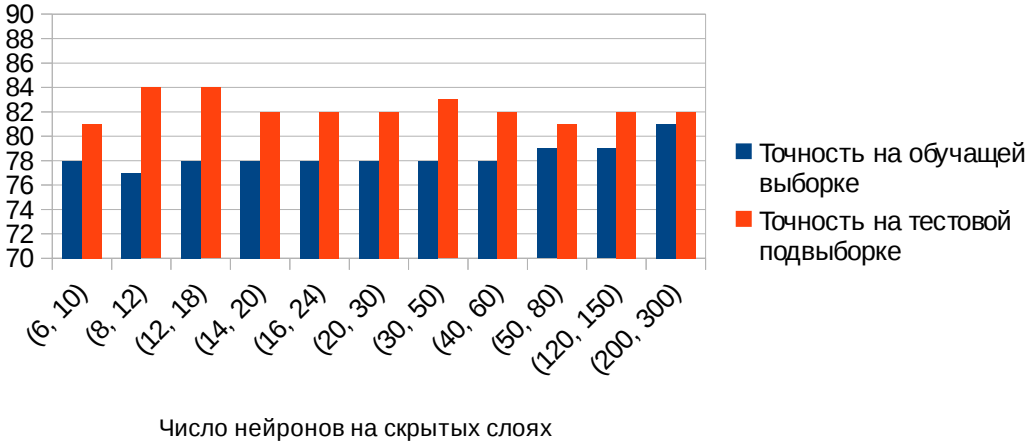
```
for feature in ['Language', 'Country', 'Rating']:
    train_data = add_numerical_cols_from_categorical(train_data, feature)
```

Возьмём за отправную точку исходную нейронку и пронаблюдаем зависимость точности модели от некоторых параметров. (Ещё я пробовал инициализировать веса с помощью Glorot, но статистически значимой разницы не увидел.)



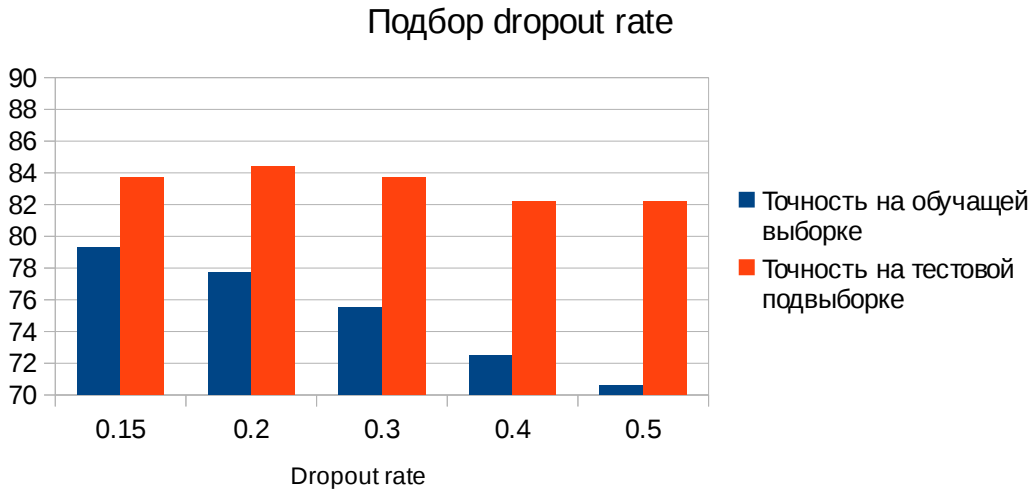
Точность на обучающей выборке почти монотонно растёт с ростом числа нейронов, но это не важно, так как после точки (16, 24) истинная точность модели падает. Самое важное наблюдение: при использовании такого большого набора признаков **имеет место переобучение.** Для борьбы с этим используют L1 и L2 регуляризацию, а также временное случайное исключение нейронов при обучении — **dropout**. Попробуем применить последнюю технику.

```
nn = Sequential()
nn.add(Dropout(0.2, input_shape=(input_size,)))
nn.add(Dense(8, activation='relu'))
nn.add(Dropout(0.2))
```

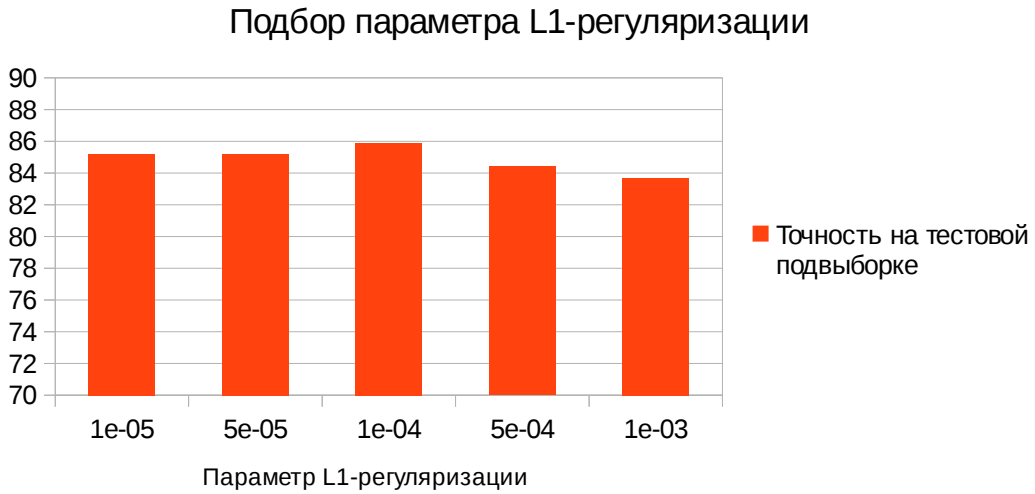


Отлично! Выглядит, как будто колонки на графике переставились местами. На деле же точность при обучении ниже, так как включены не все нейроны, а при дальнейшем применении модели выключенных нейронов нет. Dropout действительно решил проблему переобучения.

В дальнейшем используется сеть со слоями на 8 и 12 скрытых нейронов и 100 эпох обучения.



Виден экстремум при значении 0.2, его и будем использовать.
Добавим L1-регуляризацию для зануления несущественных признаков:



Точность ещё выше: теперь 85.9% при параметре 0.0001.

Итак, я устранил все основные проблемы. Осталось добавить работу с test.csv и записью результатов в файл. Основная трудность здесь в том, что категориальные признаки в двух таблицах имеют не одинаковое множество значений, соответственно в результате они содержат разный набор one hot признаков. Удобно, что во второй таблице индексы продолжают значения с

последнего из первой. Это даёт возможность слить таблицы в одну и заполнить несоответствия признаков нулями. Pandas имеет для этого удобный API.

Итак, мне удалось пройти путь от запуска простейшего прототипа до ощутимого улучшения качества предсказания с использованием различных методик.