kNNでおすすめの映画を取得するモデルを実装【①アイテムベースレコメンド】省メモリでスパース行列を扱うテクニックも説明します

2020.03.24 2020.02.22

Python

今回はレコメンドエンジンの代表的なアルゴリズムであるkNNを実装していきます。

kNN（kNearestNeighber、k近傍）とは

kNNとは、空間（平面）上にマップされたサンプル郡から相対距離を求めるものです。

分類の対象になる新たなサンプルが投入されたとき、新たなサンプルと距離が近いものを投票制で決定します。その性質から、レコメンドエンジンの最も単純な実装として知られています。

下記に良いまとめがありましたので引用させていただきます。

> ・通称 K-NN（K-Nearest Neighbor Algorithm の略称）
> ・特徴空間上において、近くにある K個 オブジェクトのうち、最も一般的なクラスに分類する。

・距離の算出には、一般的にユークリッド距離が使われる。（他にマンハッタン距離などがある
・次元の呪いのため、高次元データには向かない。
・トレーニングデータ数・特徴量が増えると予測が遅くなる。
・クラス分類や回帰分析に利用可能

https://qiita.com/fujin/items/128ed7188f7e7df74f2c

今回は映画のレーティングデータから、ある映画を見た人に別の映画をオススメするモデルを実装したいとおもいます。『この映画を見ている人はこんな映画も見ています』という例のやつです。

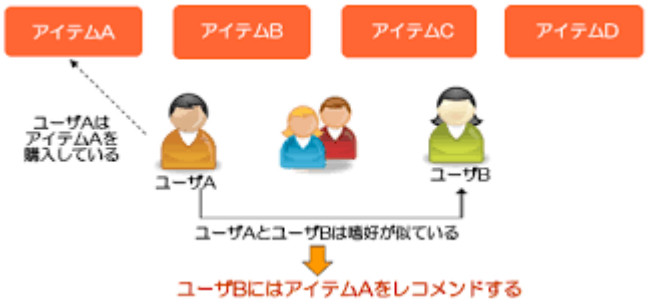
PythonでkNNを実装

kNNを実装するには、データの持ち方によって2つの方法が考えられます。

それは、アイテム同士の距離を測り、一人のユーザに対して距離の近い別のアイテムを提案する手法と、

https://www.activecore.jp/column/rwh_2/

ユーザ同士の距離をはかり、別のユーザに同じアイテムを提案する手法です。



https://www.activecore.jp/column/rwh_2/

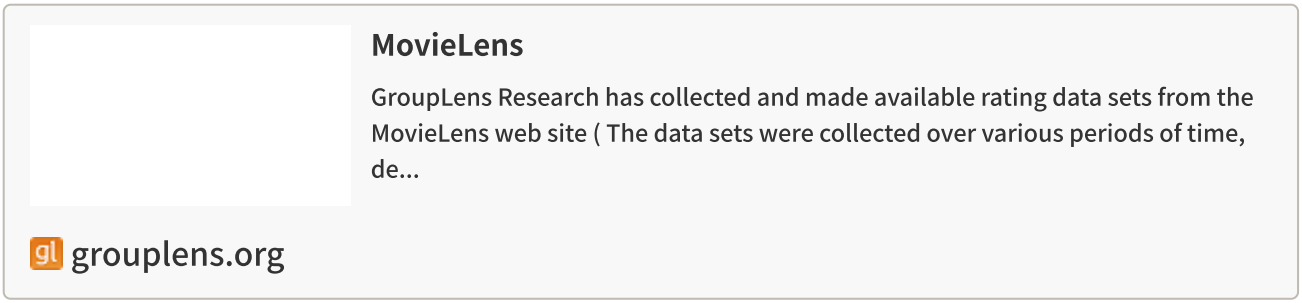
同じデータから両方のパターンが考えられますが、実装としては前者がより簡単です。

ということで、今回は前者の「あるユーザに別のおすすめアイテムを提案」できるモデルを目指します。

使ったデータ

今回はMovielensという映画評論のサイトより、ユーザが映画につけたレーティングデータを利用したいと思います。

下記のページにて無料で公開されています。全部で200万件もあるため、今回は途中でランダムにサンプリングしつつ利用していきます。



Zipの中にいくつかのデータが入っていますが、今回は”rating.csv”を使います。

実装開始

では、書いて行きましょう。まずはいつもどおりimportしてデータをロードします。

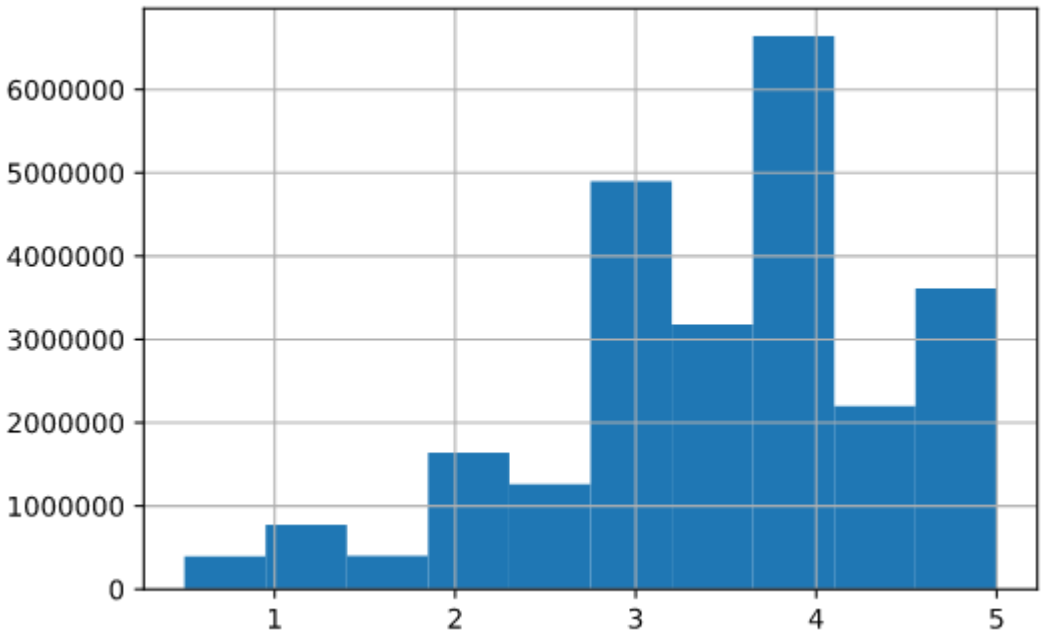
```
import pandas as pd
import numpy as np
from sklearn.neighbors import NearestNeighbors
from scipy.sparse import csr_matrix
import time
import matplotlib.pyplot as plt

#データを読み込む
print('Loading dataset...')
train = pd.read_csv("/Users/XXXXXX/ratings.csv")
print('Finished')
```

では次にdescribeとhistでデータをざっと見てみます。

```
train["rating"].describe()
train["rating"].hist()
```

```
count    2.500010e+07
mean     3.533854e+00
std      1.060744e+00
min      5.000000e-01
25%      3.000000e+00
50%      3.500000e+00
```

```
75%     4.000000e+00
max     5.000000e+00
```



（ラベルをつけ忘れましたが）ヒストグラムの横軸はRatingの点数、縦軸は該当する人数です。0~5の0.5点刻みで投票する仕組みのようです。

また、事前に述べましたがデータ数が極めて膨大なので次節でランダムサンプリングするとともに、データの縦横を変える作業、いわゆるピボットをできるだけ省メモリで行う工夫をしてみましょう。

ちなみにこのデータには欠損値がないため、そのまま進めます。

スパース行列をメモリ消費量を抑えつつ実装

この節では、データをサンプリングし、全量の1%のみを抽出→ピボットして縦軸を映画ID、横軸をユーザID、交点にRatingの点数を、というふうにデータの構造を変えていきます。

方法については下記を大いに参考にさせていただきました。


```
# trainをランダムサンプリング
```

```
train = train.sample(frac=0.01)

# カラムをカテゴリ変数化
userId_categorical = pd.api.types.CategoricalDtype(categories=sorted(train.userId.unique()), ordere
movieId_categorical = pd.api.types.CategoricalDtype(categories=sorted(train.movieId.unique()), ord
# カテゴリインスタンスを利用して新しいカラムを作成
row = train.userId.astype(userId_categorical).cat.codes
col = train.movieId.astype(movieId_categorical).cat.codes
# マトリックスにRatingの数値を当てはめる
sparse_matrix = csr_matrix((train["rating"], (row, col)), shape=(userId_categorical.categories.size, m
# スパース行列をDataframe化する
train_pivot_sparce = pd.SparseDataFrame(sparse_matrix, index = userId_categorical.categories, colu
```

やっていることとしては、pandanの.api.types.CategoricalDtypeメソッドで重複するカラムからユニークな値を取り出しています。これが次のDataframeのカラムになります。

次にcsr_matrixで交点に該当するRatingを検索し、当てはめています。

最後に、得られたマトリックスをpandasの関数でSparceDataFrame化しています。

この、ユーザと何かしらのアイテムが対応するようなタイプのデータはスパース（=疎）な行列になりがちです。というのも、全ユーザが全アイテムについて評価しているわけではないので、該当なしのデータは0を代入するより他にありません。

そうすると、n*mの行列の大部分が0になるような集合になりがちです。中身が0であったとしても、処理上は同じだけのリソースを食うため、スパース行列の扱いはメモリを多く消費しがちです。

疎行列

ja.wikipedia.org

pandasではこのスパース行列をうまく処理して省メモリで保持するためのSparceDataFrameというものが存在します。

しかし、もとのDataFrameからピボットさせる際は一度通常のDataFrameを経由してSparceDataFrameにキャストする必要があるため、前者の段階でメモリを大量に食います。環境によってはメモリエラーで処理自体が止まる可能性もあります。

※余談ですが、この処理をSageMakerのml.t2.mediumという最小のノートブックインスタンスで実行しようとしたところメモリエラーで処理が止まってしまい、今上限解除の申請を出しています。

学習と実行結果

さて、工夫してデータセットを揃えましたので、学習させましょう。これは例によりScikit Learnを使うため楽ちんです。

```
# n_neibors やalgorithm、metricなど重要なアーギュメントを設定
knn = NearestNeighbors(n_neighbors=9,algorithm= 'brute', metric= 'cosine')
# 前処理したデータセットでモデルを訓練
model_knn = knn.fit(train_pivot)
```

また、便宜上学習という言葉を使ってはいますが、kNNは怠惰学習といって、Fitの段階では既知のデータを取り込むだけです。つまり、次のデータを予測するためのモデルは作られていません。

データが投入された段階で改めて計算を行いますので、その他の機械学習でよくある「学習は重いが、推論は軽い」というあるあるが通用しません。推論のほうが重くなったりします。

次に、推論するための関数を定義しておきます。

```
def movie_prediction(movie):
    distance, indice = model_knn.kneighbors(train_pivot.iloc[train_pivot.index== ¥
movie].values.reshape(1,-1),n_neighbors=4)
    for i in range(0, len(distance.flatten())):
        if  i == 0:
            print('Recommendations if you like the movie ¥
{0}:\n'.format(train_pivot[train_pivot.index== movie].index[0]))
        else:
            print('{0}: {1} with distance: {2}'.format(i,train_pivot.index[indice.flatten() ¥[i]],distance.flat
```

準備が整ったところで推論をしてみます。

```
movie_prediction(2455)
```

```
Recommendations if you like the anime 2455:
1: 145370 with distance: 0.0
```

```
2: 60112 with distance: 0.0
3: 132877 with distance: 0.0
```

と、映画2455を見た人に対しておすすめの映画が3つ提案されました。元データからしてタイトルが無くUIDのみだったので感覚的にはわからないのが残念ですが、モデルとしては動いているようです。

おわりに

以上、knnを使っておすすめの映画を提案するモデルを作りました。

レコメンドの仕組みはECなどでよく目にするためなんとなくは認知していましたが、実装してみて初めてデータの持ち方などのポイントに気づけた感じです。

今回は以上です。お読みいただきありがとうございました。