1 导入库

```
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
from collections import Counter
import tensorflow as tf
from sklearn.cluster import KMeans
import os
import pickle
import re
from tensorflow.python.ops import math_ops

from urllib.request import urlretrieve
from os.path import isfile, isdir
from tqdm import tqdm
import zipfile
import hashlib

from gensim.models import word2vec
import gensim
import math
from math import sqrt
import matplotlib.pyplot as plt
from collections import defaultdict
```

2 下载数据

```
def _unzip(save_path, _, database_name, data_path):
    """
    Unzip wrapper with the same interface as _ungzip
    :param save_path: The path of the gzip files
    :param database_name: Name of database
    :param data_path: Path to extract to
    :param _: HACK - Used to have to same interface as _ungzip
    """
    print('Extracting {}...'.format(database_name))
    with zipfile.ZipFile(save_path) as zf:
        zf.extractall(data_path)

def download_extract(database_name, data_path):
    """
```

```
    Download and extract database
    :param database_name: Database name
    """
    DATASET_ML1M = 'ml-1m'

    if database_name == DATASET_ML1M:
        url = 'http://files.grouplens.org/datasets/movielens/ml-1m.zip'
        hash_code = 'c4d9eecfca2ab87c1945afe126590906'
        extract_path = os.path.join(data_path, 'ml-1m')
        save_path = os.path.join(data_path, 'ml-1m.zip')
        extract_fn = _unzip

    if os.path.exists(extract_path):
        print('Found {} Data'.format(database_name))
        return

    if not os.path.exists(data_path):
        os.makedirs(data_path)

    if not os.path.exists(save_path):
        with DLProgress(unit='B', unit_scale=True, miniters=1,
desc='Downloading {}'.format(database_name)) as pbar:
            urlretrieve(
                url,
                save_path,
                pbar.hook)

    assert hashlib.md5(open(save_path, 'rb').read()).hexdigest() ==
hash_code, \
        '{} file is corrupted.  Remove the file and try
again.'.format(save_path)

    os.makedirs(extract_path)
    try:
        extract_fn(save_path, extract_path, database_name, data_path)
    except Exception as err:
        shutil.rmtree(extract_path)  # Remove extraction folder if there
is an error
        raise err

    print('Done.')
    # Remove compressed data
#     os.remove(save_path)
class DLProgress(tqdm):
    """
    Handle Progress Bar while Downloading
    """
    last_block = 0

    def hook(self, block_num=1, block_size=1, total_size=None):
```

```
        """
        A hook function that will be called once on establishment of the
network connection and
        once after each block read thereafter.
        :param block_num: A count of blocks transferred so far
        :param block_size: Block size in bytes
        :param total_size: The total size of the file. This may be -1 on
older FTP servers which do not return
                                a file size in response to a retrieval
request.
        """
        self.total = total_size
        self.update((block_num - self.last_block) * block_size)
        self.last_block = block_num
```

```
data_dir = './'
download_extract('ml-1m', data_dir)
```

```
Found ml-1m Data
```

3 分析数据

3.1 分析用户数据

```
users_title = ['UserID', 'Gender', 'Age', 'OccupationID', 'Zip-code']
users = pd.read_csv('./ml-1m/users.dat', sep='::', header=None,
names=users_title, engine = 'python')
```

```
users.head(5)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	UserID	Gender	Age	OccupationID	Zip-code
0	1	F	1	10	48067

	UserID	Gender	Age	OccupationID	Zip-code
1	2	M	56	16	70072
2	3	M	25	15	55117
3	4	M	45	7	02460
4	5	M	25	20	55455

UserID 用户编号， Gender用户性别， Age用户年龄， OccupationID 职业编号

- 年龄数据集用分段表示：
  - 1: "Under 18"
  - 18: "18-24"
  - 25: "25-34"
  - 35: "35-44"
  - 45: "45-49"
  - 50: "50-55"
  - 56: "56+"
- 职业编号如下：
  - 0: "other" or not specified
  - 1: "academic/educator"
  - 2: "artist"
  - 3: "clerical/admin"
  - 4: "college/grad student"
  - 5: "customer service"
  - 6: "doctor/health care"
  - 7: "executive/managerial"
  - 8: "farmer"
  - 9: "homemaker"
  - 10: "K-12 student"
  - 11: "lawyer"
  - 12: "programmer"
  - 13: "retired"
  - 14: "sales/marketing"
  - 15: "scientist"
  - 16: "self-employed"
  - 17: "technician/engineer"
  - 18: "tradesman/craftsman"
  - 19: "unemployed"
  - 20: "writer"

```
users.shape[0]
```

```
6040
```

共有6040个用户

3.2 分析电影数据

```
movies_title = ['MovieID', 'Title', 'Genres']
movies = pd.read_csv('./ml-1m/movies.dat', sep='::', header=None,
names=movies_title, engine = 'python')
```

```
movies
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	MovieID	Title	Genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
3878	3948	Meet the Parents (2000)	Comedy
3879	3949	Requiem for a Dream (2000)	Drama
3880	3950	Tigerland (2000)	Drama
3881	3951	Two Family House (2000)	Drama

	MovieID	Title	Genres
3882	3952	Contender, The (2000)	Drama|Thriller

3883 rows × 3 columns

MovieID 电影编号，Title 电影名称， Genres类别

- 电影类别有：
  - Action
  - Adventure
  - Animation
  - Children's
  - Comedy
  - Crime
  - Documentary
  - Drama
  - Fantasy
  - Film-Noir
  - Horror
  - Musical
  - Mystery
  - Romance
  - Sci-Fi
  - Thriller
  - War
  - Western

```
movies.shape[0]
```

```
3883
```

共收录了3883个电影

3.3 评分数据

```
ratings_title = ['UserID','MovieID', 'Rating', 'timestamps']
ratings = pd.read_csv('./ml-1m/ratings.dat', sep='::', header=None,
names=ratings_title, engine = 'python')
```

```
ratings.head(120)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	UserID	MovieID	Rating	timestamps
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291
...	...	...	...	...
115	2	480	5	978299809
116	2	1442	4	978299297
117	2	2067	5	978298625
118	2	1265	3	978299712
119	2	1370	5	978299889

120 rows × 4 columns

```
ratings.shape[0]
```

```
1000209
```

共有一百万条评分信息，对于每个用户来说，平均每个用户记录了100+的电影评价信息

4 数据预处理

4.1 处理用户信息

- 将用户的性别变为0，1(女性：0.男性：1)
- 年龄：分别赋予成7个类别，改为数字 0-7
- 职业信息不改变
- 舍弃邮政编码信息

```
users_title = ['UserID', 'Gender', 'Age', 'JobID', 'Zip-code']
users = pd.read_csv('./ml-1m/users.dat', sep='::', header=None,
names=users_title, engine = 'python')
users = users.filter(regex='UserID|Gender|Age|JobID')
gender_map = {'F':0, 'M':1}
users['Gender'] = users['Gender'].map(gender_map)
age_map = {val:ii for ii,val in enumerate(set(users['Age']))}
users['Age'] = users['Age'].map(age_map)
users
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	UserID	Gender	Age	JobID
0	1	0	0	10
1	2	1	5	16
2	3	1	6	15
3	4	1	2	7
4	5	1	6	20
...	...	...	...	...
6035	6036	0	6	15
6036	6037	0	2	1

	UserID	Gender	Age	JobID
6037	6038	0	5	1
6038	6039	0	2	0
6039	6040	1	6	6

6040 rows × 4 columns

4.2 处理电影信息

```
movies_title = ['MovieID', 'Title', 'Genres']
movies = pd.read_csv('./ml-1m/movies.dat', sep='::', header=None,
names=movies_title, engine = 'python')
movies.drop("Title",1,inplace=True)
```

```
L_all =
['Action','Adventure','Animation','Children\'s','Comedy','Crime','Documen
tary','Drama','Fantasy','Film-Noir','Horror',
     'Musical','Mystery','Romance','Sci-Fi','Thriller','War','Western']
genres_map=[]


for val in movies['Genres'].str.split('|'):
    temp=[]
    for i in range(len(L_all)):
        if(L_all[i] in val):
            temp.append(1)
        else:
            temp.append(0)
    genres_map.append(temp)

movies['Genres'] =genres_map
```

```
movies
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	MovieID	Genres
0	1	[0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1	2	[0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, ...
2	3	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, ...
3	4	[0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...
4	5	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
...	...	...
3878	3948	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3879	3949	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...
3880	3950	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...
3881	3951	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...
3882	3952	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...

3883 rows × 2 columns

4.2.1聚类

```
X=np.array(genres_map)
km = KMeans(n_clusters=200).fit(X)
# 标签结果
rs_labels = km.labels_
movies['Genres'] =rs_labels
```

```
movies
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	MovieID	Genres
0	1	75
1	2	22
2	3	5
3	4	0
4	5	3
...	...	...
3878	3948	3
3879	3949	1
3880	3950	1
3881	3951	1
3882	3952	25

3883 rows × 2 columns

4.2.2 主成分分析

```
X=np.array(genres_map)
X
```

```
array([[0, 0, 1, ..., 0, 0, 0],
       [0, 1, 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, ..., 1, 0, 0]])
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components='mle')
pca.fit(X)
pca.explained_variance_ratio_
```

```
array([0.23271188, 0.16895203, 0.08742511, 0.0822061 , 0.07767275,
       0.06302748, 0.04761725, 0.04061996, 0.03733275, 0.03523988,
       0.02596828, 0.02071835, 0.02007116, 0.01728318, 0.01282514,
       0.01208367, 0.01031404])
```

5 模型构建

5.1 决策树模型

```
class DTreeID3(object):

    def __init__(self, epsilon=0.0001):
        self.tree = Node()
        self.epsilon = epsilon

    def fit(self, X_train, Y_train):
        A_recorder = np.arange(X_train.shape[1])
        self._train(X_train, Y_train, self.tree, A_recorder)

    def predict(self, X):
        n = X.shape[0]
        Y = np.zeros(n)
        for i in range(n):
            Y[i] = self.tree.predict_classification(X[i, :])
        return Y

    def visualization(self):
```

```
        return self._visualization_dfs(self.tree)

    def _train(self, A, D, node, AR):
        # 1. 结束条件：若 D 中所有实例属于同一类，决策树成单节点树，直接返回
        if np.any(np.bincount(D) == len(D)):
            node.y = D[0]
            return
        # 2. 结束条件：若 A 为空，则返回单结点树 T，标记类别为样本默认输出最多的类别
        if A.size == 0:
            node.y = np.argmax(np.bincount(D))
            return
        # 3. 计算特征集 A 中各特征对 D 的信息增益，选择信息增益最大的特征 A_g
        max_info_gain, g = self._feature_choose_standard(A, D)
        # 4. 结束条件：如果 A_g 的信息增益小于阈值 epsilon，决策树成单节点树，直接
返回
        if max_info_gain <= self.epsilon:
            node.y = np.argmax(np.bincount(D))
            return
        # 5. 对于 A_g 的每一可能值 a_i，依据 A_g = a_i 将 D 分割为若干非空子集
D_i，将当前结点的标记设为样本数最大的 D_i 对应
            # 的类别，即对第 i 个子节点，以 D_i 为训练集，以 A - {A_g} 为特征集，
递归调用以上步骤，得到子树 T_i，返回 T_i
        node.label = AR[g]
        a_cls = np.bincount(A[:, g])
        new_A, AR = np.hstack((A[:, 0:g], A[:, g+1:])),
np.hstack((AR[0:g], AR[g+1:]))
        for k in range(len(a_cls)):
            a_row_idxs = np.argwhere(A[:, g] == k).T[0].T
            child = Node(k)
            node.append(child)
            A_child, D_child = new_A[a_row_idxs, :], D[a_row_idxs]
            self._train(A_child, D_child, child, AR)

    def _feature_choose_standard(self, A, D):
        row, col = A.shape
        prob = self._cal_prob(D)
        prob = np.array([a if 0 < a <= 1 else 1 for a in prob])
        entropy = -np.sum(prob * np.log2(prob))
        max_info_gain_ratio = None
        g = None
        for j in range(col):
            a_cls = np.bincount(A[:, j])
            condition_entropy = 0
            for k in range(len(a_cls)):
                a_row_idxs = np.argwhere(A[:, j] == k)
                # H(D)
                prob = self._cal_prob(D[a_row_idxs].T[0])
                prob = np.array([a if 0 < a <= 1 else 1 for a in prob])
                H_D = -np.sum(prob * np.log2(prob))
                # H(D|A)=SUM(p_i * H(D|A=a_i))
```

```
                condition_entropy += a_cls[k] / np.sum(a_cls) * H_D
            feature_choose_std = entropy - condition_entropy
            if max_info_gain_ratio is None or max_info_gain_ratio <
feature_choose_std:
                max_info_gain_ratio = feature_choose_std
                g = j
        return max_info_gain_ratio, g

    def _cal_prob(self, D):
        statistic = np.bincount(D)
        prob = statistic / np.sum(statistic)
        return prob

    def _visualization_dfs(self, node, layer=0):
        prefix = '\n' if layer else ''
        output_str = [prefix + ' ' * 4 * layer, '%r+%r ' % (node.y,
node.label)]
        if not node.child:
            return ''.join(output_str)
        for child in node.child:
            output_str.append(self._visualization_dfs(child, layer=layer
+ 1))
        return ''.join(output_str)

class DTreeC45(DTreeID3):

    def _feature_choose_standard(self, A, D):
        row, col = A.shape
        prob = self._cal_prob(D)
        prob = np.array([a if 0 < a <= 1 else 1 for a in prob])
        entropy = -np.sum(prob * np.log2(prob))
        max_info_gain_ratio = None
        g = None
        for j in range(col):
            a_cls = np.bincount(A[:, j])
            condition_entropy = 0
            for k in range(len(a_cls)):
                a_row_idxs = np.argwhere(A[:, j] == k)
                # H(D) = -SUM(p_i * log(p_i))
                prob = self._cal_prob(D[a_row_idxs].T[0])
                prob = np.array([a if 0 < a <= 1 else 1 for a in prob])
                H_D = -np.sum(prob * np.log2(prob))
                # H(D|A)=SUM(p_i * H(D|A=a_i))
                condition_entropy += a_cls[k] / np.sum(a_cls) * H_D
            feature_choose_std = entropy / (condition_entropy + 0.0001)
            if max_info_gain_ratio is None or max_info_gain_ratio <
feature_choose_std:
                max_info_gain_ratio = feature_choose_std
                g = j
        return max_info_gain_ratio, g
```

```
class DTreeCART(DTreeID3):

    def _train(self, A, D, node, AR):
        self.visited_set = set()
        self._train_helper(A, D, node, AR)

    def _train_helper(self, A, D, node, AR):
        # 1. 结束条件：若 D 中所有实例属于同一类，决策树成单节点树，直接返回
        if np.any(np.bincount(D) == len(D)):
            node.y = D[0]
            return
        # 2. 与 ID3, C4.5 不一样, 不会直接去掉 A
        if A.size == 0:
            node.y = np.argmax(np.bincount(D))
            return
        # 3. 与 ID3, C4.5 不一样, 不仅要确定最优切分特征，还要确定最优切分值
        max_info_gain, g, v, a_idx, other_idx =
self._feature_choose_standard(A, D)
        if (g, v) in self.visited_set:
            node.y = np.argmax(np.bincount(D))
            return
        self.visited_set.add((g, v))
        # 4. 结束条件：如果 A_g 的信息增益小于阈值 epsilon，决策树成单节点树，直接
返回
        if max_info_gain <= self.epsilon:
            node.y = np.argmax(np.bincount(D))
            return
        # 5. 与 ID3, C4.5 不一样, 不是 len(a_cls) 叉树，而是二叉树
        node.label = AR[g]
        idx_list = a_idx, other_idx
        for k, row_idx in enumerate(idx_list):
            row_idx = row_idx.T[0].T
            child = Node(k)
            node.append(child)
            A_child, D_child = A[row_idx, :], D[row_idx]
            self._train_helper(A_child, D_child, child, AR)

    def _feature_choose_standard(self, A, D):
        row, col = A.shape
        min_gini, g, v, a_idx, other_idx = None, None, None, None, None
        for j in range(col):
            a_cls = np.bincount(A[:, j])
            # 与 ID3, C4.5 不一样,不仅要确定最优切分特征，还要确定最优切分值
            for k in range(len(a_cls)):
                # 根据切分值划为两类
                a_row_idxs, other_row_idxs = np.argwhere(A[:, j] == k),
np.argwhere(A[:, j] != k)
                # H(D) = -SUM(p_i * log(p_i))
```

```
                a_prob, other_prob = self._cal_prob(D[a_row_idxs].T[0]),
self._cal_prob(D[other_row_idxs].T[0])
                a_gini_D, other_gini = 1 - np.sum(a_prob * a_prob), 1 -
np.sum(other_prob * other_prob)
                # H(D|A)=SUM(p_i * H(D|A=a_i))
                gini_DA = a_cls[k] / np.sum(a_cls) * a_gini_D + (1 -
a_cls[k] / np.sum(a_cls)) * other_gini
                if min_gini is None or min_gini > gini_DA:
                    min_gini, g, v, a_idx, other_idx = gini_DA, j, k,
a_row_idxs, other_row_idxs

        return min_gini, g, v, a_idx, other_idx

class DTreeRegressionCART(object):

    def __init__(self, max_depth=1):
        self.tree = Node()
        self.max_depth = max_depth

    def fit(self, X_train, Y_train):
        A_recorder = np.arange(X_train.shape[1])
        self._train(X_train, Y_train, self.tree, A_recorder)

    def predict(self, X):
        n = X.shape[0]
        Y = np.zeros(n)
        for i in range(n):
            Y[i] = self.tree.predict_regression(X[i, :])
        return Y

    def _train(self, A, D, node, AR, depth=0):
        # 1. 结束条件：到最后一层 | A 或 D 一样
        if depth == self.max_depth or np.all(D == D[0]) or np.all(A ==
A[0]):
            node.y = np.mean(D)
            return
        # 2. 选择第j个变量A_j（切分变量splitting variable）和 切分点
s（splitting point）
        min_f, min_j, min_s, min_idx1, min_idx2 = None, None, None, None,
None
        row, col = A.shape
        for j in range(col):
            a_col = A[:, j]
            # 这里实现比较简化，s 就直接取最值的平均数
            s = (np.max(a_col) + np.min(a_col)) * 0.5
            R1_idx, R2_idx = np.argwhere(a_col <= s).T[0],
np.argwhere(a_col > s).T[0]
            if R1_idx.size == 0 or R2_idx.size == 0:
                continue
            c1, c2 = np.mean(D[R1_idx]), np.mean(D[R2_idx])
```

```
            f1, f2 = np.sum(np.square(D[R1_idx] - c1)),
np.sum(np.square(D[R2_idx] - c2))
            if min_f is None or min_f > f1 + f2:
                min_f, min_j, min_s, min_idx1, min_idx2 = f1 + f2, j, s,
R1_idx, R2_idx
        if min_f is None:
            node.y = np.mean(D)
            return
        # 3. 向下一层展开
        node.label, node.s = AR[min_j], min_s
        for i, idx_list in enumerate((min_idx1, min_idx2)):
            child = Node(i)
            node.append(child)
            self._train(A[idx_list, :], D[idx_list], child, AR, depth+1)

    def visualization(self):
        return self._visualization_dfs(self.tree)

    def _visualization_dfs(self, node, layer=0):
        prefix = '\n' if layer else ''
        output_str = [prefix + ' ' * 4 * layer, '%r+%r+%r' % (node.y,
node.label, node.s)]
        if not node.child:
            return ''.join(output_str)
        for child in node.child:
            output_str.append(self._visualization_dfs(child, layer=layer
+ 1))
        return ''.join(output_str)

class Node(object):

    def __init__(self, x=None):
        self.label = None
        self.x = x
        self.s = None  # Number
        self.child = []
        self.y = None
        self.data = None

    def append(self, child):
        self.child.append(child)

    def predict_classification(self, features):
        if self.y is not None:
            return self.y
        for child in self.child:
            if child.x == features[self.label]:
                return child.predict_classification(features)
        return self.child[1].predict_classification(features)
```

```
    def predict_regression(self, features):
        if self.y is not None:
            return self.y
        child_idx = 0 if features[self.label] <= self.s else 1
        return self.child[child_idx].predict_regression(features)
```

5.2随机森林

```
class RandomForest(object):

    def __init__(self, tree_count=10):
        self.tree_list = []
        self.tree_count = tree_count

    def fit(self, X_train, Y_train):
        # Generate decision tree
        for i in range(self.tree_count):
            dt_CART = DTreeRegressionCART()
            # Bagging data
            n, m = X_train.shape
            sample_idx = np.random.permutation(n)
            feature_idx = np.random.permutation(m)[:int(np.sqrt(m))]
            X_t_ = X_train[:, feature_idx]
            X_t_, Y_t_ = X_t_[sample_idx, :], Y_train[sample_idx]
            # Train
            dt_CART.fit(X_t_, Y_t_)
            self.tree_list.append((dt_CART, feature_idx))
            print('=' * 10 + ' %r/%r tree trained ' % (i + 1,
self.tree_count) + '=' * 10)
            # print(dt_CART.visualization())

    def predict(self, X):
        output_matrix = np.zeros((self.tree_count, X.shape[0]))
        output_label = np.zeros(X.shape[0])
        for i, (tree, feature_idx) in enumerate(self.tree_list):
            output_matrix[i, :] = tree.predict(X[:, feature_idx])
        for col in range(output_matrix.shape[1]):
            output_label[col] = np.argmax(np.bincount(output_matrix[:,
col].astype(int)))
        return output_label.astype(int)
```

5.3支持向量机

```
class SVMModel(object):
    """
    SVM model
    """
```

```
    def __init__(self, max_iter=10000, kernel_type='linear', C=1.0,
epsilon=0.00001):
        self.max_iter = max_iter
        self.kernel_type = kernel_type
        self.kernel_func_list = {
            'linear': self._kernel_linear,
            'quadratic': self._kernel_quadratic,
        }
        self.kernel_func = self.kernel_func_list[kernel_type]
        self.C = C
        self.epsilon = epsilon
        self.alpha = None

    def fit(self, X_train, Y_train):
        """
        Training model
        :param X_train: shape = num_train, dim_feature
        :param Y_train: shape = num_train, 1
        :return: loss_history
        """
        n, d = X_train.shape[0], X_train.shape[1]
        self.alpha = np.zeros(n)
        # Iteration
        for i in range(self.max_iter):
            diff = self._iteration(X_train, Y_train)
            if i % 100 == 0:
                print('Iter %r / %r, Diff %r' % (i, self.max_iter, diff))
            if diff < self.epsilon:
                break

    def predict_raw(self, X):
        return np.dot(self.w.T, X.T) + self.b

    def predict(self, X):
        #temp = np.sign(np.dot(self.w.T, X.T) + self.b).astype(int)
        #l = len(temp)
        #for i in range(l):
        #    if temp[i]==-1:
        #        temp[i] = 0
        #return temp
        return np.sign(np.dot(self.w.T, X.T) + self.b).astype(int)

    def _iteration(self, X_train, Y_train):
        alpha = self.alpha
        alpha_prev = np.copy(alpha)
        n = alpha.shape[0]
        for j in range(n):
            # Find i not equal to j randomly
            i = j
            for _ in range(1000):
```

```
                if i != j:
                    break
                i = random.randint(0, n - 1)
            x_i, x_j, y_i, y_j = X_train[i, :], X_train[j, :],
Y_train[i], Y_train[j]
            # Define the similarity of instances. K11 + K22 - 2K12
            k_ij = self.kernel_func(x_i, x_i) + self.kernel_func(x_j,
x_j) - 2 * self.kernel_func(x_i, x_j)
            if k_ij == 0:
                continue
            a_i, a_j = alpha[i], alpha[j]
            # Calculate the boundary of alpha
            L, H = self._cal_L_H(self.C, a_j, a_i, y_j, y_i)
            # Calculate model parameters
            self.w = np.dot(X_train.T, np.multiply(alpha, Y_train))
            self.b = np.mean(Y_train - np.dot(self.w.T, X_train.T))
            # Iterate alpha_j and alpha_i according to 'Delta W(a_j)'
            E_i = self.predict(x_i) - y_i
            E_j = self.predict(x_j) - y_j
            alpha[j] = a_j + (y_j * (E_i - E_j) * 1.0) / k_ij
            alpha[j] = min(H, max(L, alpha[j]))
            alpha[i] = a_i + y_i * y_j * (a_j - alpha[j])
        diff = np.linalg.norm(alpha - alpha_prev)
        return diff

    def _kernel_linear(self, x1, x2):
        return np.dot(x1, x2.T)

    def _kernel_quadratic(self, x1, x2):
        return np.dot(x1, x2.T) ** 2

    def _cal_L_H(self, C, a_j, a_i, y_j, y_i):
        if y_i != y_j:
            L = max(0, a_j - a_i)
            H = min(C, C - a_i + a_j)
        else:
            L = max(0, a_i + a_j - C)
            H = min(C, a_i + a_j)
        return L, H
```

```
def getans(res):
    for i in range(len(res)):
        if res[i]== -1:
            res[i]=0
    return res
```

6 生成训练集和测试集

```
movies_dic = {}
for i in range(len(movies)):
    movies_dic[movies['MovieID'][i]] = movies['Genres'][i]
```

```
users.drop(['UserID'],axis=1,inplace =True)
```

```
X_train = []
Y_train = []
temp = []
score = 3
for i in range(len(ratings)//20):
    #print(i)
    temp = list(users.iloc[ratings['UserID'][i]-1])
    temp.append(movies_dic[ratings['MovieID'][i]])
    X_train.append(temp)
    #if ratings['Rating'][i]>score:
    #    Y_t.append(1)
    #else:
     #   Y_t.append(0)
    Y_train.append(ratings['Rating'][i])
X_train = np.array(X_t)
Y_train = np.array(Y_t)
```

```
X_train
```

```
array([[ 0,  0, 10,  1],
       [ 0,  0, 10, 18],
       [ 0,  0, 10, 85],
       ...,
       [ 1,  6,  7,  5],
       [ 1,  6,  7, 36],
       [ 1,  6,  7,  5]])
```

```
len(X_train)
```

```
50010
```

```
Y_train
```

```
array([5, 3, 3, ..., 4, 5, 4], dtype=int64)
```

```
len(Y_train)
```

```
50010
```

```
X_test = []
Y_test = []
temp = []
score = 3
for i in range(len(ratings)//20,len(ratings)//20+len(ratings)//100):
    #print(i)
    temp = list(users.iloc[ratings['UserID'][i]-1])
    temp.append(movies_dic[ratings['MovieID'][i]])
    X_test.append(temp)
    #if ratings['Rating'][i]>score:
    #    Y_t.append(1)
    #else:
     #   Y_t.append(0)
    Y_test.append(ratings['Rating'][i])
X_test = np.array(X_t)
Y_test = np.array(Y_t)
```

7 训练模型

```
model_rf = RandomForest()
model_rf.fit(X_train,Y_train)
```

```
========== 1/10 tree trained ==========
========== 2/10 tree trained ==========
========== 3/10 tree trained ==========
========== 4/10 tree trained ==========
========== 5/10 tree trained ==========
========== 6/10 tree trained ==========
========== 7/10 tree trained ==========
========== 8/10 tree trained ==========
========== 9/10 tree trained ==========
========== 10/10 tree trained ==========
```

```
model_SVM = SVMModel()
model_SVM.fit(X_train,Y_train)
```

```
ans = model_rf.predict(X_test)
rmse(ans,Y_test)
```

```
0.5730736272520542
```

```
ans = model_SVM.predict(X_test)
rmse(ans,Y_test)
```

```
1.0672966670822537
```

8 利用sklearn与自己写的算法比较

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_wine
```

```
clf = DecisionTreeClassifier(random_state=0)
rfc = RandomForestClassifier(random_state=0)
```

```
clf = clf.fit(X_train,Y_train)
rfc = rfc.fit(X_train,Y_train)
```

```
ans = rfc.predict(X_test)
```

```
def rmse(Pre,Rea):
    di2 = 0
    for i in range(len(Pre)):
        di2 += (Pre[i]-Rea[i])**2
    return (di2/len(Pre))**0.5
```

```
rmse(ans,Y_test)
```

```
0.465071927759054
```

```
ans = clf.predict(X_t)
rmse(ans,Y_t)
```

```
0.7287626761034435
```

```
from sklearn.svm import SVR
```

```
svm_poly_reg1 = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
svm_poly_reg1.fit(X_t, Y_t)
ans = svm_poly_reg1.predict(X_t)
rmse(ans,Y_t)
```

```
1.1437202675980986
```

```
from sklearn.ensemble import GradientBoostingRegressor
```

```
GBR = GradientBoostingRegressor()
GBR.fit(X_t,Y_t)
ans = GBR.predict(X_t)
rmse(ans,Y_t)
```

```
0.4821576021847739
```

9 隐变量模型的效果

```
U_id  = {}
id_U  = {}
B_id  = {}
id_B  = {}
def grade(df):
    global U_id,id_U,B_id,id_B
    for i in range(0,len(df)):
        if df['user_id'][i] not in U_id:
            U_id[df['user_id'][i]]=i
            id_U[i]=df['user_id'][i]
            df['user_id'][i]=i
        else:
            df['user_id'][i] = U_id[df['user_id'][i]]
        if df['business_id'][i] not in B_id:
            B_id[df['business_id'][i]]=i
            id_B[i]=df['business_id'][i]
            df['business_id'][i]=i
        else:
            df['business_id'][i]=B_id[df['business_id'][i]]
    return df
def grade_te(df):
    for i in range(0,len(df)):
        df['user_id'][i]=U_id[df['user_id'][i]]
        df['business_id'][i]=B_id[df['business_id'][i]]
    return df
```

```
ratings.head(10)
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	UserID	MovieID	Rating	timestamps
0	1	1193	5	978300760
1	1	661	3	978302109

	UserID	MovieID	Rating	timestamps
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291
5	1	1197	3	978302268
6	1	1287	5	978302039
7	1	2804	5	978300719
8	1	594	4	978302268
9	1	919	4	978301368

```
df_train = ratings.head(30010)
```

```
df_train.drop('timestamps',axis=1,inplace=True)
```

```
F:\Anaconda3\lib\site-packages\pandas\core\frame.py:4308:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

```
df_train.rename(columns=
{'UserID':'user_id',"MovieID":'business_id',"Rating":"stars"},inplace=Tru
e)
df_train
```

```
F:\Anaconda3\lib\site-packages\pandas\core\frame.py:4441:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().rename(
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	user_id	business_id	stars
0	1	1193	5
1	1	661	3
2	1	914	3
3	1	3408	4
4	1	2355	5
...	...	...	...
30005	202	2918	3
30006	202	1036	5
30007	202	430	3
30008	202	3578	5
30009	202	1974	4

30010 rows × 3 columns

```
tr_grade = grade(df_train)
```

```
<ipython-input-87-8f7474cf6e9e>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  tr_grade = grade(df_train)
```

```
class LF:
    def __init__(self,df,k,norm):
        #先生成一个User_Business评分矩阵，这里先构建一个全0矩阵，稍后填充
        self.UB =
np.mat(np.zeros((int(df[['user_id']].max())+1,int(df[['business_id']].max
())+1)))
        #找到对B评论的所有U，和U评论的所有B
        self.B_U = defaultdict(set)
        self.U_B = defaultdict(set)
        for i in range(0,len(df)):
            user,business,stars =int(df['user_id']
[i]),int(df['business_id'][i]),df['stars'][i]
            self.UB[user,business]=stars
            self.B_U[business].add(user)
            self.U_B[user].add(business)
        self.k= k  #选取的k
        self.norm = norm
        #构建预测评分矩阵
        self.User =  np.mat(np.random.uniform(sqrt(1/k),sqrt(5/k),
(self.UB.shape[0],k)))
        self.Business =  np.mat(np.random.uniform(sqrt(1/k),sqrt(5/k),
(self.UB.shape[1],k)))

    #定义损失函数
    def loss(self):
        ret = self.norm * (np.sum(np.square(self.User)) +
np.sum(np.square(self.Business)))
        #User * Business 的转置
        pred = self.User * self.Business.T
        for i in range(self.UB.shape[0]):
            for j in range(self.UB.shape[1]):
                if self.UB[i,j] != 0:
                    ret += (self.UB[i,j] - pred[i,j]) ** 2
        return ret

    #梯度下降
    #lr学习率，maxd最大迭代深度，th阈值
    def grad_fit(self,lr = 0.01,maxd = 15,th = 100):
        d = 0
        x = []
        loss_val = []
        train_score = []
        val_score = []
        while d < maxd and self.loss() > th:
            for uid in range(1,self.UB.shape[0]):
                grad = 2 * self.norm * self.User[uid]
                for bid in self.U_B[uid]:
```

```
                    grad = grad - 2 * (self.UB[uid,bid] - self.User[uid]
* self.Business[bid].T) * self.Business[bid]
                self.User[uid] = self.User[uid] - lr * grad
            for bid in range(1,self.UB.shape[1]):
                grad = 2 * self.norm * self.Business[bid]
                for uid in self.B_U[bid]:
                    grad = grad - 2 * (self.UB[uid,bid] - self.User[uid]
* self.Business[bid].T) * self.User[uid]
                self.Business[bid] = self.Business[bid] - lr * grad
            x.append(d)
            loss_val.append(self.loss())
            train_score.append(self.RMSE_score(tr_grade))
            val_score.append(self.RMSE_score(tr_grade))
            d += 1
        return x,loss_val,train_score,val_score

    #交替最小二乘法
    #maxd最大迭代深度，th阈值
    def als_fit(self,maxd = 25,th = 100):
        d = 0
        x = []
        loss_val = []
        train_score = []
        val_score = []
        while d < maxd and self.loss() > th:
            for uid in range(1,self.UB.shape[0]):
                left = np.mat(np.zeros((1,self.k)))
                right = np.mat(np.zeros((self.k,self.k)))
                for bid in self.U_B[uid]:
                    right += self.Business[bid].T * self.Business[bid]
                    left += self.UB[uid,bid] * self.Business[bid]
                right += self.norm * np.identity(self.k)
                if abs(np.linalg.det(right)) < 1e-6:
                    self.User[uid] = left * np.linalg.pinv(right +
self.norm * np.identity(self.k))
                else:
                    self.User[uid] = left * np.linalg.inv(right +
self.norm * np.identity(self.k))
                #采用moore-penrose伪逆
            for bid in range(1,self.UB.shape[1]):
                left = np.mat(np.zeros((1,self.k)))
                right = np.mat(np.zeros((self.k,self.k)))
                for uid in self.B_U[bid]:
                    right += self.User[uid].T * self.User[uid]
                    left += self.UB[uid,bid] * self.User[uid]
                right += self.norm * np.identity(self.k)
                if abs(np.linalg.det(right)) < 1e-6:
                    self.Business[bid] = left * np.linalg.pinv(right +
self.norm * np.identity(self.k))
                else:
```

```
                    self.Business[bid] = left * np.linalg.inv(right +
self.norm * np.identity(self.k))
                    #同上，采用moore-penrose伪逆
            x.append(d)
            loss_val.append(self.loss())
            train_score.append(self.RMSE_score(tr_grade))
            val_score.append(self.RMSE_score(tr_grade))
            d += 1
        return x,loss_val,train_score,val_score

    #计算评价指标RMSE
    def RMSE_score(self,df):
        r = 0
        n = 0
        pred = self.User * self.Business.T
        for i in range(0,len(df)):
            uid,bid,stars =int(df['user_id'][i]),int(df['business_id']
[i]),df['stars'][i]
            if uid < pred.shape[0] and bid < pred.shape[1]:
                r += (pred[uid,bid] - stars) ** 2
                n += 1
        return sqrt(r/n)

    #预测结果
    def pred(self,df_test):
        ans = []
        pred = self.User* self.Business.T
        for idx,row in df_test.iterrows():
            uid,bid = int(row['user_id']),int(row['business_id'])
            if uid < pred.shape[0] and bid < pred.shape[1]:
                ans.append(pred[uid,bid])
            else:
                ans.append(3)
        return ans
```

```
model = LF(df=tr_grade,k=5,norm=0.01)
```

```
x,loss_val,train_score,val_score = model.grad_fit()
```

```
model.RMSE_score(tr_grade)
```

```
0.262451647883004
```