Summary algorithm

SVD 기반의 Collaborate Filtering 을 이용한 Recommend 시스템
Simple SVD는 undefined value에 대해 추정이 불가능하기 때문에 Funk SVD 방식으로 값이 없는 값들을 추정하는 방식.
Class를 만들어 값들을 저장

Detail Each function

프로그램 실행 시 시작되는 메인 함수

```
def main():
    argv = sys.argv

    if len(argv) == 3:
        training_file = argv[1]
        test_file = argv[2]
    else:
        print("usage :", argv[0].split('/')[-1], "<training file> <test file>")
        exit(0)

    output_file = test_file.split('.')[0] + '.base_prediction.txt'

    training = pd.read_csv(training_file, sep='\t', names=['user_id', 'item_id', 'rating', 'timestamp'])
    test = pd.read_csv(test_file, sep='\t', names=['user_id', 'item_id', 'rating', 'timestamp'])
    training = training.drop(columns='timestamp')

    test = test.drop(columns='timestamp')

    rec = Recommend()
    rec.fit(training.values[:, :2], training.values[:, 2])

    test['rating'] = rec.predict(test)
    test.to_csv(output_file, sep='\t', index=False, header=None)
```

불필요한 데이터를 제거한 후 추천 시스템을 이용해 점수를 측정하고 결과를 파일로 저장한다.

Recommend class 를 초기화하는 method

```
def __init__(self):
    self.mean = 2
    self.user = None
    self.item = None
    self.user_mat = None
    self.item_mat = None
    self.bias_user = None
    self.bias_item = None

    self.lr = 0.0001
    self.reg = 0.02
    self.factors = 100
    self.epochs = 20
```

초기값들은 surprise lib의 값들을 참고하였습니다.

Data를 학습시키는 method

```
def fit(self, X, y):
    unique_user = np.unique(X[:, 0])
    unique_item = np.unique(X[:, 1])

    bias_user = np.zeros(unique_user.size, np.double)
    bias_item = np.zeros(unique_item.size, np.double)

    user_mat = np.random.rand(len(unique_user), self.factors)
    item_mat = np.random.rand(self.factors, len(unique_item))

    mean = np.mean(y)

    for _ in range(self.epochs):
        for (u, i), r in zip(X, y):
            u = np.where(unique_user == u)[0][0]
            i = np.where(unique_item == i)[0][0]
            predict_r = mean + bias_user[u] + bias_item[i] + np.dot(user_mat[u, :], item_mat[:, i])
            err = r - predict_r

            bias_user[u] += self.lr * (err - self.reg * bias_user[u])
            bias_item[i] += self.lr * (err - self.reg * bias_item[i])
            for k in range(self.factors):
                user_mat[u, k] += self.lr * (err * item_mat[k, i] - self.reg * user_mat[u, k])
                item_mat[k, i] += self.lr * (err * user_mat[u, k] - self.reg * item_mat[k, i])

    self.user = unique_user
    self.item = unique_item
    self.bias_user = bias_user
    self.bias_item = bias_item
    self.user_mat = user_mat
    self.item_mat = item_mat
    self.mean = mean
```

$$\hat{r}_{u,i} = \bar{r} + bu_u + bi_i + \sum_{f=1}^{F} P_{u,f} * Q_{i,f}$$

$$err = r - \hat{r}$$

$$bu_u = bu_u + \alpha * (err - \lambda * bu_u)$$

$$bi_i = bi_i + \alpha * (err - \lambda * bi_i)$$

$$P_{u,f} = P_{u,f} + \alpha * (err * Q_{i,f} - \lambda * P_{u,f})$$

$$Q_{i,f} = Q_{i,f} + \alpha * (err * P_{u,f} - \lambda * Q_{i,f})$$

위와 같은 funk SVD update 방식에 따라 코드를 구성하여. 사용되는 Parameter들을 업데이트 하였습니다.

학습한 결과를 바탕으로 점수를 예측하는 method

```
def predict(self, test):
    return [
        self.predict_pair(u, i) for u, i in zip(test['user_id'], test['item_id'])
    ]

def predict_pair(self, u, i):
    user_known, item_known = False, False
    pred = self.mean

    if u in self.user:
        user_known = True
        u = np.where(self.user == u)[0][0]
        pred += self.bias_user[u]

    if i in self.item:
        item_known = True
        i = np.where(self.item == i)[0][0]
        pred += self.bias_item[i]

    if user_known and item_known:
        pred += np.dot(self.user_mat[u, :], self.item_mat[:, i])

    pred = min(5, pred)
    pred = max(1, pred)

    return pred
```

학습에 사용된 bias 와 matrix 를 이용해 점수를 예측합니다.

현재 점의 이웃을 구하고 이웃의 수가 minpts보다 적으면 False를 반환합니다.
이웃들을 같은 클러스터로 표시해주고
이웃의 이웃을 돌면서 클러스터를 확장해갑니다.

Instruction for compile

제출한 source code는 .py 형태이므로 python3 명령어를 이용해 실행하면 됩니다.
Ex) python3 recommender.py u1.base u1.text