ポートフォリオ

協調フィルタリングによるレコメンドモデル

サービス概要

・お仕事紹介のメールマガジンの配信サービス

・お仕事は、ユーザーごとにオススメのお仕事を抽出・レコメンドする。

・既存商品ラインナップは、ユーザの直近応募案件を起点とする「応募案件レコメンドメール」などが存在する。

課題分析

・応募数の母数が伸び悩んでいる

・サービス・ローンチ後、閲覧数は順調に伸びてきている

・お仕事応募は個人情報が直接届くため、応募までのハードルが高いのではないか。

・気になるお仕事への応募を後押しする新機能の開発を行う必要があるのではないか。

新機能（案）
閲覧検知機能、閲覧案件レコメンドメール

・メルマガ上のリンクのクリックを検知する機能を実装し、閲覧数の多い案件と類似する案件をレコメンドし、応募数の母数アップをはかる。

・応募への継げる媒体として、閲覧レコメンドメールを開発する。その際の効果検証は、閲覧レコメンド経由の流入者の応募数を計測し、効果検証を行う。

・閲覧レコメンドメール配信の開発初期は学習データ不足によりレコメンデーションの精度が低くなることが予想される。配信初回はキープ案件の登録履歴情報も合わせて、レコメンデーションの精度を向上させる。

・レコメンド案件数が規定数（10件）に満たない場合は、閲覧数の少ない案件と関連性の高い案件を別途抽出し、レコメンド案件数を15件に充足させる。

・期待する効果は、応募総数の103%増加。過去の応募数の増加率と売上金額の増加率の割合から300万円/月の増加を期待値とする。

アルゴリズム設計～データ分析～

閲覧数、閲覧者数実績

閲覧実績のあるユーザー数:3000/1月（ユニークユーザー数）

年代別利用者:30-40代女性が多い（全体の50%）

閲覧結果推移:5%/1週間の増加傾向

閲覧案件種別:派遣（事務)25%、契約社員（事務)20%・・・事務職が多い

アルゴリズム設計～アプローチ・手法①～

①ルールベースレコメンド

ユーザーの行動を起点として、事前に規定したルールベースで商品をレコメンドする。

②コンテンツベースレコメンド

属性情報を元に、類似性の高い商品をレコメンドする。

③協調フィルタリングレコメンド

アイテムベース：行動分析を基に、類似度の高い商品を算出しレコメンドする。

ユーザーベース：行動分析を基に、類似度の高い別ユーザーの購入商品をレコメンドする。

アルゴリズム設計～アプローチ・手法②～

手法	レコメンドの精度	実装コスト	検証コスト
ルールベースレコメンド	低	低	低
コンテンツベースレコメンド	低～中	中	高
協調フィルタリングレコメンド	中	中	中～高

アルゴリズム設計～アプローチ・手法③～

①ルールベースレコメンド

メリット：実装が用意

デメリット：柔軟性が低い、パーソナライゼイションが弱い

②コンテンツベースレコメンド

メリット：ルールベースよりはユーザーファースト、ユーザーの嗜好を捉えられる

デメリット：既存のユーザの嗜好に基づいてレコメンドされるため、レコメンドされる商品がユーザの既存の嗜好に制限される

③協調フィルタリングレコメンド

メリット：レコメンドの幅が広くなる（セレンディピティ）

デメリット：会員登録初期のユーザーに対してはデータ不足によりレコメンドの精度が低くなる。新商品がレコメンドされづらい。

アルゴリズム設計～理論～

セレンディピティの向上を期待できる「協調フィルタリング」のアイテムベースモデルを採用

ラベルあり、教師あり学習

類似のお仕事を分類、分類問題、 KNN(K近傍法)

メリット：シンプルで直感的

デメリット：予測時の計算コストが高い（前日分データを元に夜間処理で実施）

アルゴリズム設計～数式～

●類似度の計算（コサイン類似度）

$$sim_{cosine}(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| \times |\vec{b}|}$$

データ量（可変）の増減を考慮せず評価
※参考：https://jp.mathworks.com/help/stats/pdist.html（観測値ペア間のペアワイズ距離）

●予測スコアの算出

$$r_{u,i} = \sum_{j \in Q_t(u)} sim(i,j) * r_{u,j}$$

類似度×閲覧（評価）値からの算出

アルゴリズム設計～技術スタック～

■レコメンドアルゴリズム開発

ライブラリ:pandas, numpy, scipy, sklearn

言語:Python

IDE:Pycharm

■メルマガ開発

言語:Python

サーバー:GCP、AWS→heroku

データベース:(MongoDB)→Heroku:PostgreSql、ローカル:SQLite3

コンテナ/リソース管理:Docker, kubernetes(想定)

モデル評価手法

評価手法：オフライン評価法

評価指標：Precision（適合率）

$$Precision = \frac{\text{抽出結果に含まれる適合アイテムの数}}{\text{抽出結果に含まれる全てのアイテム数}}$$

$$Precision = \frac{|A \cap B|}{|B|}$$

※実運用時はオンライン評価法（A/Bテストを想定）

フィジビリティスタディ（実現可能性検証）

●要求事項

応募数の103%増加（300件/月）目標

●調査要項

閲覧レコメンドメール配信対象者の2週間(経過観察期間)の応募率を取得）、閲覧者は、3000人/1月

●達成可能性

応募レコメンドメールの応募到達率は20%。応募実績なしのユーザーを対象者として多く含むため、閲覧レコメンドメールの応募到達率は10%を見込むと予想。

●代替案

目標達成率に満たない場合、PDCAを回し、A/Bテストにて商品改善を行う。

データ整備

並び替え、行列変換

```
# データ整備
job_list = data_base.sort_values('job_id').job_id.unique() # 並び替え
user_list = data_base.user_id.unique() # 並び替え
view_matrix_job = np.zeros([len(job_list), len(user_list)])
```

```
# 学習データ整形
for job_id in tqdm(range(1, len(job_list))): # 進捗表示
    user_list_job = data_train[data_train['job_id'] == job_id].sort_values('user_id').user_id.unique()
    for user_id in user_list_job:
        try:
            user_rate = data_train[(data_train['job_id'] == job_id) & (data_train['user_id'] == user_id)].loc[:, 'view_count']
        except None as e:
            user_rate = 0 # 該当なしの場合は0をセット
        view_matrix_job[job_id-1, user_id-1] = user_rate # 評価をセット
```

モデル開発

類似度と評価値より、上位　10件のレコメンド商品を算出する

```
# 閲覧数（閲覧数は評価点の見立てとする）
view_matrix_calc = view_matrix_job.copy() #
view_matrix_calc[view_matrix_calc != 0] = 1
view_matrix_train = np.abs(view_matrix_calc - 1) # 絶対値変換
```

```
# 類似度
similarity_matrix = 1 - pairwise_distances(view_matrix_job, metric='cosine') # コサイン類似度
np.fill_diagonal(similarity_matrix, 0)
```

```
# レコメンド算出
user_id = 100
hits = 0
```

```
# 閲覧(評価)点の算出
view_matrix_user = view_matrix_job[:, user_id - 1]
pre_view_user = similarity_matrix * view_matrix_user # 類似度×閲覧数
pre_view_user = pre_view_user.sum(axis=1) # 各評価の合算
pre_view_user_job = pre_view_user * view_matrix_train[:,user_id - 1]
# 該当率計算
recommend_list = np.argsort(pre_view_user_job)[::-1][:10] + 1
purchase_list_user = data_test[data_test.user_id == user_id].loc[:, 'job_id'].unique()
for job_id in recommend_list:
    if job_id in purchase_list_user:
        hits += 1
pre = hits / 10
```

※参考：

https://www.jstage.jst.go.jp/article/pjsai/JSAI2013/0/JSAI2013_1F51/_pdf/-char/ja
（A Method for Searching Question Sentences in Mathematics by String Patterns and the Structure of MathML）
https://qiita.com/fuuki/items/09295e458d15bc61d692（総和のシグマの範囲）
https://att-astrec.nict.go.jp/member/mutiyama/corpmt/12.pdf（対訳コーパスの自動作成）

結果検証

●結果

```
#レコメンド結果出力
print('Recommend list:', recommend_list)
print('Test Rated list:', purchase_list_user)
print('Precision:', str(pre))
```

```
Recommend list: [313 269 272 316 300 347 328 301 288 100]
Test Rated list: [ 269  272  301  313  326  328  340  887  896 1024]
Precision: 0.5
```

●精度評価

```
# 結果出力
precision = sum(precision_list) / len(precision_list)
print('Precision:', precision)
```

```
Precision: 0.5
```

振り返り

●アプローチ・手法の問題、課題

会員登録初期、データ不足により精度が低くなる。

●実装の問題、課題

処理精度が低い。類似度・スコアの計算方法の改善が必要。

処理性能が低い。行列計算を改善すべき。大量データの処理に耐えられよう分散処理の実装を検討する。

●改善案、学び

同じ商品を起点とした場合、同じ商品をレコメンドしてしまい、個人に紐づくパーソナライズされたレコメンドを行えない。ベイズ理論やその他のアルゴリズムの適用を検討する。