ポートフォリオ②

ベイズ定理を用いたレコメンドモデル

目次

・サービス概要
・データ分析、仮設、出口設計
・前回の課題
・ベイズ定理の採用理由
・データ準備、前処理
・特徴量エンジニアリング
・アルゴリズム、数式
・モデル実装、予測、評価
・レコメンド
・振り返り、課題

サービス概要 ※ポートフォリオ（機械学習①）からの引用

・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。

データ分析、仮説、出口設計

・機械学習①の結果分析
閲覧案件レコメンドメールを導入したが、応募数が本当に上がっているのか。誤差の範囲ではないのか。
→カイ二乗検定（統計的検定）→誤差の範囲ではない。

・仮説
機械学習①のレコメンドでは、仕事の属性、個人の属性を外してしまっているレコメンドが一定数あった。
よりパーソナライズされた、セレンディピティの高いレコメンドを行うため、説明変数は評価点だけでなく、新たな特徴量を追加する必要があるはずだ。
より複数の特徴量から精度の高いレコメンドを行えば、結果、応募数をより伸ばすことができるだろう。

・出口設計
実運用に乗っている閲覧案件レコメンドメールのレコメンドの質・幅ともに向上させる。
結果分析は、A/Bテストを実施し、施策効果を評価する。

前回の課題

①モデルの精度が低い
②処理精度が低い
③初期のレコメンド精度が低い
④パーソナライズ、セレンディピティの精度が低い

↓

①②③：Naive Bayesを採用
④：特徴量を設定（閲覧案件の属性：職種、業界、年収、契約形態、勤務地）

ベイズ定理(Naive Bayes)採用理由

・大きなデータセットにも有効

・処理速度が比較的高速

・少ない学習でも性能を期待できる

・重要性の低い特徴量の影響を受けにくい

データ準備(前処理)

①データ取得元:kaggleデータ「Job Recommendation Case Study」を使用
https://www.kaggle.com/jsrshivam/job-recommendation-case-study

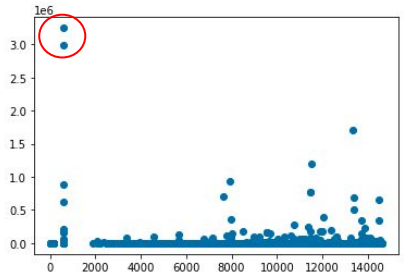
②前処理
・特徴量の抽出
→案件情報（案件名、企業、採用ポジション、所在地、稼働種別）、閲覧日時
・不要カラム削除（相関の高いもの（多重共線性対応）、欠損値の多いカラム）
・外れ値除去
・欠損値補完、除去
・時系列データ加工
・学習データ、テストデータへの分割

データ準備①

・外れ値除去

```
# 外れ値検出
data = job_data_time.values
job_data_time.shape
print(job_data_time.columns)
x = data[:, 0]
y = data[:, 7]
plt.scatter(x, y)
```

```
Index(['applicant_id', 'job_id', 'title', 'position', 'company', 'city',
       'state_code', 'view_duration'],
      dtype='object')

<matplotlib.collections.PathCollection at 0x7fc8ca7a8d00>
```




```
============
[2984851, 3248075]
```

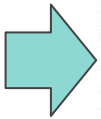
```
job_data_outlier = job_data_time.copy()
#条件にマッチしたIndexを削除
drop_index = job_data_outlier.index[job_data_outlier['view_duration'] > 2900000]
job_data_outlier = job_data_outlier.drop(drop_index)

print(job_data_time.shape)
print('============')
print(job_data_outlier.shape)
```

```
(12348, 8)
============
(12347, 8)
```

データ準備②

・欠損値補完、除去

```
job_data_lost = job_data_mold.copy()
# カテゴリ変数の欠損値処理
print(job_data_lost.isnull().sum())
print('===============')
job_data_lost['company'] = job_data_lost['company'].fillna('other')
job_data_lost.dropna(subset = ['state_code'], inplace = True) # nanの行のみ削除
job_data_lost['view_duration'] = job_data_lost['view_duration'].fillna(0)
print('<欠損値処理結果>')
print(job_data_lost.isnull().sum())
print('===============')
```



```
applicant_id      0
job_id            0
title             0
position          0
company         580
city              0
state_code       22
view_duration  1795
dtype: int64
===============
<欠損値処理結果>
applicant_id    0
job_id          0
title           0
position        0
company         0
city            0
state_code      0
view_duration   0
dtype: int64
===============
```

データ準備③

・時系列データ加工

```
# タイムゾーン(JST/UTC)変換
job_data_time['view_start'] = pd.to_datetime(job_data_time['view_start'], utc=False)
job_data_time['view_start'] = job_data_time['view_start'].astype('datetime64[ns]')
job_data_time['view_start'] = job_data_time['view_start'].astype('int')

job_data_time['view_end'] = pd.to_datetime(job_data_time['view_end'], utc=False)
job_data_time['view_end'] = job_data_time['view_end'].astype('datetime64[ns]')
job_data_time['view_end'] = job_data_time['view_end'].fillna(np.min(job_data_time['view_end']))
job_data_time['view_end'] = job_data_time['view_end'].astype('int')
```

特徴量エンジニアリング

・ラベルエンコーディング（ダミー値処理）

```
# カテゴリ変数処理
#ラベル・エンコーディング
# title, position, company, city, state_code
le = LabelEncoder()
data = ['title', 'position', 'company', 'city', 'state_code']
job_data_lost[data[0]] = le.fit_transform(job_data_lost[data[0]].values)
job_data_lost[data[1]] = le.fit_transform(job_data_lost[data[1]].values)
job_data_lost[data[2]] = le.fit_transform(job_data_lost[data[2]].values)
job_data_lost[data[3]] = le.fit_transform(job_data_lost[data[3]].values)
job_data_lost[data[4]] = le.fit_transform(job_data_lost[data[4]].values)
```

データ準備④

・データ分割

```
# 学習データとテストデータに分割
train_set, test_set = train_test_split(job_data_time, test_size=0.2, random_state=1)
print(train_set.shape)
print(test_set.shape)
print('========================')

# 説明変数と目的変数に分割
# 学習データを説明変数データと目的変数データに分割
train_X = train_set.drop('job_id', axis=1)
train_y = train_set['job_id']
print(train_X.shape)
print(train_y.shape)
print('========================')

# テスト用データを説明変数データと目的変数データに分割
test_X = test_set.drop('job_id', axis=1)
test_y = test_set['job_id']
print(test_X.shape)
print(test_y.shape)
```

アルゴリズム、手法

アルゴリズム：ベイズ定理（Naive Bayes）

ライブラリ：sklearn（GaussianNB、LabelEncoder、accuracy_score）

数式（ベイズ定理）

$$P(B_i|A) = \frac{P(A \cap B_i)}{P(A)}$$

A案件閲覧時のときレコメンド案件B（条件付き確率）

検証基準：適合率（Precision）

モデル実装、予測、評価

・ベイズモデル実装、レコメンドの結果予測、適合率での評価

```
# 学習
model = GaussianNB()
model.fit(train_X, train_y)
```

```
GaussianNB()
```

```
# 機械学習モデルmodelに対する出力結果
pred = model.predict(test_X)
print(pred)
```

```
[ 97544  73957 221890 ... 134292  79848 235787]
```

```
# 適合率算出
print(accuracy_score(pred, test_y))
```

```
0.4376518218623482
```

レコメンド

レコメンド10件を算出

```
# job_idとの紐づけ
pred_jobs = pd.Series(pred, index = test_X.index)

# 指定のjob_idとの関連の高い順にソート
jobs = pred_jobs.sort_values(ascending=False).index

# 上位10件を出力
output = list(jobs)[:10]
print(output)
```

```
[8718, 8709, 8639, 8659, 8648, 2181, 8427, 6025, 8463, 8426]
```

振り返り、課題

・まだまだ精度が高く出ていない。

・新案件や成長段階の案件が、ユーザーにレコメンドされない可能性がある。

・短期的には効果が出ているように見えても、長期的に課題の解決になっているのか検証できていない。※短期的、長期的なメールマーケティング、両方のアプローチが必要