پروژه‌های

یادگیری ماشین

تهیه کننده: رامتین عبدی

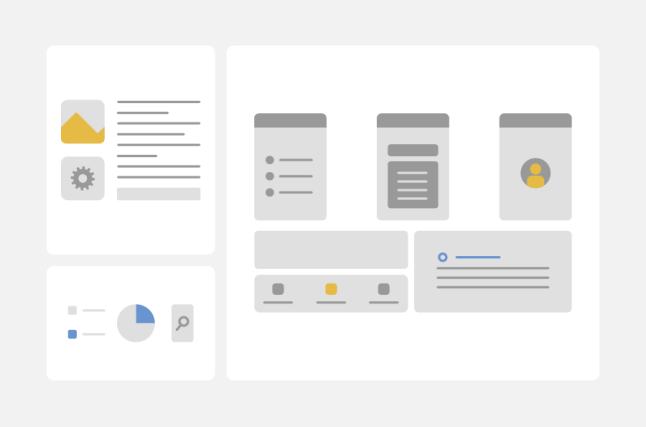
استاد راهنما: استاد زهرا ولدخانی

تابستان و پاییز ۱۴۰۳

فهرست

پروژه ۱: تحلیل اطلاعات مسافران

پیش‌بینی سفرهای بعدی، علت سفرهای کاربران، احتمال کنسلی بلیط و ریسک و تحلیل نظرات کاربران



پروژه۲: دسته‌بندی صفحات وب

تقسیم‌بندی وبسایت‌های مختلف به گروه‌های مختلف

پروژه ۱

بخش۱:

عنوان پروژه: پیش‌بینی دلیل سفر با استفاده از یادگیری ماشین

۱. مقدمه

- هدف: توسعه مدلی برای پیش‌بینی دلیل سفر (مثل "کار" یا "شخصی") براساس داده‌های کاربران.

- انگیزه: درک دلایل سفر می‌تواند به شرکت‌ها کمک کند خدمات خود را برای بخش‌های مختلف کاربران بهینه کنند و تجربه مشتری و کارایی عملیاتی را بهبود بخشند.

۲. مروری بر داده‌ها

- مجموعه داده آموزشی: شامل ستون‌های مختلفی از جمله اطلاعات دموگرافیک کاربران، جزئیات سفر و شناسه‌ها است.

- مجموعه داده آزمون: مشابه مجموعه داده آموزشی اما بدون برچسب‌های دلیل سفر (متغیر هدف).

- ویژگی‌های کلیدی: ستون‌هایی مانند DepartureTime, CancelTime و شناسه‌ها مانند UserID و TicketID.

۳. پیش‌پردازش داده‌ها

- تبدیل زمان: DepartureTime به فرمت تاریخ جلالی برای مرتبط بودن منطقه‌ای تبدیل شد.

- مهندسی ویژگی‌ها:

  - استخراج ماه (month) و نشانگر تعطیلات آخر هفته (weekend) از DepartureTime.

  - ایجاد یک ویژگی باینری weekend (در صورت بودن روز به عنوان آخر هفته، True).

- انتخاب ویژگی‌ها: حذف ستون‌های غیرمرتبط مانند شناسه‌ها (BillID, TicketID) برای کاهش نویز داده‌ها.

۴. تقسیم‌بندی داده‌ها و اعتبارسنجی

- تقسیم آموزش و اعتبارسنجی: تقسیم ۸۵٪ از داده‌ها برای آموزش و ۱۵٪ برای اعتبارسنجی.

- نمونه‌گیری لایه‌ای: حفظ توزیع کلاس‌ها در هر دو مجموعه آموزشی و اعتبارسنجی با استفاده از TripReason.

۵. انتخاب مدل و آموزش

- کتابخانه: استفاده از چارچوب توزیع‌شده H2O برای آموزش مدل.

- الگوریتم: طبقه‌بندی جنگل تصادفی (Random Forest) در H2O.

- پارامترها:

  - تعداد درخت‌ها: ۱۰۰

  - حداکثر عمق: ۲۰

  - تعداد حداقلی نمونه‌ها در هر برگ: ۵

  - نرخ نمونه‌گیری: ۰٫۴

  - cross validation: ۵ تایی

  - توازن کلاس‌ها: فعال برای مدیریت عدم تعادل در کلاس‌ها

۶. ارزیابی مدل

- شاخص‌های اعتبارسنجی:

  - F1 Score: اندازه‌گیری عملکرد برای توازن کلاس‌ها.

  - classification report: جزئیات accuracy، recall و F1 برای هر کلاس.

  - ماتریس خطا: بصری‌سازی دقت طبقه‌بندی.

۷. پیش‌بینی‌های نهایی روی داده‌های آزمون

- پردازش داده‌های آزمون: استفاده از همان مراحل پیش‌پردازش برای مجموعه داده آزمون.

- خروجی پیش‌بینی‌ها: تولید پیش‌بینی‌ها برای TripReason و ذخیره آن‌ها به‌عنوان submission.

۸. نتایج و یافته‌ها

- F1 Score: ٪۸۱

- classification report:

	precision	recall	f1-score	support
0	0.79	0.71	0.75	6686
1	0.79	0.85	0.82	8467
accuracy			0.79	15153
macro avg	0.79	0.78	0.78	15153
weighted avg	0.79	0.79	0.79	15153

توضیح گام به گام کد:

۱. ایمپورت کتابخانه‌ها

```
import pandas as pd
import numpy as np
import jdatetime
from sklearn.model_selection import train_test_split
import h2o
from h2o.estimators import H2ORandomForestEstimator
from sklearn.metrics import f1_score, classification_report
```

- Pandas و NumPy: برای کار با داده‌ها و تحلیل آن‌ها.
- Jalali Date (jdatetime): برای تبدیل تاریخ به تقویم جلالی.
- Scikit-learn: برای تقسیم داده‌ها و ارزیابی مدل.
- H2O: کتابخانه‌ای برای یادگیری ماشین توزیع شده که برای داده‌های بزرگ کاربرد دارد.

۲. بارگذاری و مشاهده داده‌ها

```
train_data = pd.read_csv('../data/train_data.csv')
test_data = pd.read_csv('../data/test_data.csv')
```

- بارگذاری داده‌های آموزشی و آزمون از فایل‌های CSV.

۳. پیش‌پردازش ستون تاریخ‌ها

```
train_data['DepartureTime'] = pd.to_datetime(train_data['DepartureTime'])
test_data['DepartureTime'] = pd.to_datetime(test_data['DepartureTime'])

train_data['DepartureTime'] = train_data['DepartureTime'].apply(lambda x: jdatetime.date.fromgregorian(date=x))
test_data['DepartureTime'] = test_data['DepartureTime'].apply(lambda x: jdatetime.date.fromgregorian(date=x))
```

- تبدیل تاریخ DepartureTime از تقویم میلادی به تقویم جلالی برای بومی‌سازی داده‌ها.

۴. مهندسی ویژگی‌ها

```
train_data['month'] = train_data['DepartureTime'].apply(lambda x: x.month)
test_data['month'] = test_data['DepartureTime'].apply(lambda x: x.month)

train_data['weekend'] = train_data['DepartureTime'].apply(lambda x: x.weekday())
test_data['weekend'] = test_data['DepartureTime'].apply(lambda x: x.weekday())

train_data['weekend'] = train_data['weekend'].apply(lambda x: x in [4, 5])
test_data['weekend'] = test_data['weekend'].apply(lambda x: x in [4, 5])
```

- استخراج ویژگی‌های month و weekend از DepartureTime:

- month: استخراج ماه سفر.

- weekend: بررسی اینکه آیا روز سفر آخر هفته (پنج‌شنبه یا جمعه) است یا خیر.

۵. حذف ستون‌های غیرضروری

```
train_data.drop(columns=['CancelTime', 'BillID', 'TicketID', 'UserID', 'HashPassportNumber_p
                         'HashEmail', 'BuyerMobile', 'NationalCode', 'VehicleType',
                         'Created', 'DepartureTime'], inplace=True)

test_data.drop(columns=['CancelTime', 'BillID', 'TicketID', 'UserID', 'HashPassportNumber_p'
                        'HashEmail', 'BuyerMobile', 'NationalCode', 'VehicleType',
                        'Created', 'DepartureTime'], inplace=True)
```

- حذف ستون‌های غیرضروری مانند شناسه‌ها و اطلاعات شخصی برای کاهش نویز داده‌ها.

۶. تقسیم داده‌ها به مجموعه‌های آموزشی و اعتبارسنجی

```
train, val = train_test_split(train_data, test_size=0.15, random_state=42, stratify=train_data['TripReason'])
X_val = val.drop(columns='TripReason')
Y_val = val['TripReason']
```

- تقسیم داده‌های train_data به train و val (۸۵٪ برای آموزش و ۱۵٪ برای اعتبارسنجی).

- استفاده از stratify برای حفظ توزیع کلاس‌ها در TripReason.

۷. راه‌اندازی H2O

```
h2o.init()
```

- شروع سرور H2O برای استفاده از قابلیت‌های یادگیری ماشین آن.

۸. آموزش مدل جنگل تصادفی با H2O

```
train_h2o = h2o.H2OFrame(train)
target = 'TripReason'
X = train.columns.to_list()
X.remove(target)

model = H2ORandomForestEstimator(
    ntrees=100,
    max_depth=20,
    min_rows=5,
    sample_rate=0.4,
    nfolds=5,
    balance_classes=True
)

model.train(x=X, y=target, training_frame=train_h2o)
```

- H2OFrame: تبدیل DataFrame پانداس train به یک H2OFrame برای آموزش مدل.
- پارامترهای مدل جنگل تصادفی:
- ntrees=100: استفاده از ۱۰۰ درخت تصمیم‌گیری.
- max_depth=20: محدود کردن عمق هر درخت.
- min_rows=5: اطمینان از اینکه هر برگ حداقل ۵ نمونه دارد.
- sample_rate=0.4: استفاده از ۴۰٪ داده‌ها در هر تقسیم.
- balance_classes=True: توازن کلاس‌ها در TripReason.

۹. ارزیابی عملکرد مدل

```
X_val_h2o = h2o.H2OFrame(X_val)
pred = model.predict(X_val_h2o).as_data_frame()
pred = pred['predict']

mapping = {'Int': 0, 'Work': 1}
pred = pred.map(mapping)
Y_val = Y_val.map(mapping)

print(f1_score(Y_val, pred))
print(classification_report(Y_val, pred))
```

- تبدیل داده‌های اعتبارسنجی: تبدیل X_val به H2OFrame و تولید پیش‌بینی.

- تبدیل پیش‌بینی‌ها: نگاشت دسته‌بندی‌های TripReason (مانند 'Int' و 'Work') به اعداد برای ارزیابی.

- شاخص‌های ارزیابی:

- f1_score: اندازه‌گیری تعادل بین دقت و بازخوانی.

- classification_report: نمایش جزئیات دقت، بازخوانی، و F1.

۱۰. تولید پیش‌بینی‌ها روی داده‌های آزمون

```
test_h2o = h2o.H2OFrame(test_data)
test_pr = model.predict(test_h2o).as_data_frame()
test_pr['TripReason'] = test_pr['predict']

submission = pd.DataFrame(test_pr['TripReason'])
submission
```

- پیش‌بینی روی داده‌های آزمون: تبدیل test_data به H2OFrame و تولید پیش‌بینی.

- فرمت‌بندی خروجی: ذخیره پیش‌بینی‌ها در یک DataFrame پانداس برای ارسال نهایی.

پروژه ۱

بخش۲:

عنوان پروژه: پیش‌بینی لغو بلیط

۱. مرور کلی پروژه

- هدف: پیش‌بینی لغو بلیط با استفاده از داده‌های جمع‌آوری‌شده درباره خرید بلیط و اطلاعات مشتری.

- مجموعه داده‌ها: داده‌های آموزشی و آزمایشی جداگانه با ستون‌هایی مانند DepartureTime، Created، BillID، Cancel و غیره.

- هدف نهایی: آموزش یک مدل پیش‌بینی برای شناسایی بلیط‌هایی که احتمال لغو دارند.

۲. پیش‌پردازش داده‌ها

- بارگذاری داده‌ها:

- داده‌های آموزشی و آزمایشی با استفاده از pandas وارد شدند.

- مدیریت تاریخ و زمان:

- تبدیل ستون‌های DepartureTime و Created به اشیای زمانی در تاریخ.

- محاسبه یک ویژگی جدید به نام TimeInterval به عنوان تفاوت روزانه بین DepartureTime و Created.

- مهندسی ویژگی‌ها:

- ایجاد یک ستون باینری IsFamily برای نشان دادن اینکه آیا چند بلیط تحت یک BillID صادر شده‌اند یا خیر.

- تبدیل DepartureTime به تاریخ جلالی.

- استخراج اطلاعات مربوط به ماه و آخر هفته از DepartureTime.

- حذف ستون‌های غیرضروری مانند BillID، TicketID، UserID و VehicleType برای ساده‌سازی داده‌ها.

۳. تقسیم داده‌ها

- تقسیم داده‌های آموزشی به مجموعه‌های train و val (اعتبارسنجی) با اندازه اعتبارسنجی ۱۵٪ و استفاده از نمونه‌برداری طبقه‌بندی‌شده برای تعادل کلاس Cancel.

۴. مدل‌سازی با استفاده از جنگل تصادفی H2O

- کتابخانه: استفاده از کتابخانه h2o به دلیل قابلیت‌های مدل‌سازی کارآمد و مقیاس‌پذیر.

- پارامترهای مدل:

- نوع مدل: H2ORandomForestEstimator

- پارامترهای کلیدی:

- ntrees=1000

- max_depth=20

- min_rows=5

- sample_rate=0.4

- balance_classes=True (برای مقابله با عدم تعادل کلاس‌ها)

- آموزش: آموزش مدل بر روی نسخه H2OFrame از داده‌های آموزشی.

۵. ارزیابی مدل

- پیش‌بینی بر روی داده‌های اعتبارسنجی:

- اعمال مدل بر روی مجموعه داده اعتبارسنجی.

- تبدیل پیش‌بینی‌ها به فرمت باینری برای مقایسه با مقادیر واقعی.

- معیارها:

- محاسبه f1_score و classification report.

- نمایش confusion matrix با seaborn برای ارزیابی بصری عملکرد مدل.

۶. پیش‌بینی داده‌های آزمایشی

- تولید پیش‌بینی‌ها:

- تبدیل داده‌های آزمایشی به فرمت H2OFrame و پیش‌بینی احتمال لغو.

- تبدیل پیش‌بینی‌ها به برچسب‌های باینری.

- آماده‌سازی برای ارسال:

- تهیه ارسال نهایی با پیش‌بینی‌های Cancel به صورت DataFrame.

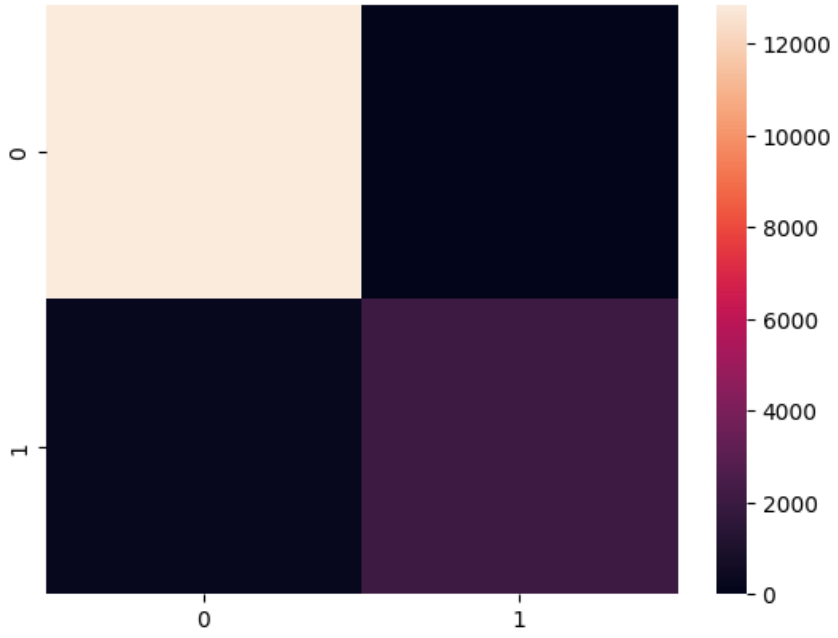
۷. نتایج و یافته‌ها

- F1 Score: ۹۴٪

- classification report:

	precision	recall	f1-score	support
0	0.98	1.00	0.99	12858
1	1.00	0.89	0.94	2295
accuracy			0.98	15153
macro avg	0.99	0.95	0.97	15153
weighted avg	0.98	0.98	0.98	15153

confusion matrix:



توضیح گام به گام کد:

۱. وارد کردن کتابخانه‌ها و بارگذاری داده‌ها

```
import pandas as pd
import numpy as np
train_data = pd.read_csv('../data/train_data.csv')
test_data = pd.read_csv('../data/test_data.csv')
```

- کتابخانه‌ها: pandas برای دستکاری داده‌ها و numpy برای عملیات عددی.

- بارگذاری داده‌ها: داده‌های آموزشی و آزمایشی از فایل‌های CSV بارگذاری می‌شوند که شامل اطلاعاتی درباره خرید بلیط، مشخصات مشتری و سایر فاکتورهای مربوط به لغو بلیط است.

۲. تبدیل تاریخ‌ها و مهندسی ویژگی‌ها

```
import jdatetime
train_data['DepartureTime'] = pd.to_datetime(train_data['DepartureTime'])
test_data['DepartureTime'] = pd.to_datetime(test_data['DepartureTime'])

train_data['Created'] = pd.to_datetime(train_data['Created'])
test_data['Created'] = pd.to_datetime(test_data['Created'])
```

- تبدیل تاریخ‌ها: ستون‌های DepartureTime و Created به فرمت زمانی تبدیل می‌شوند که امکان دستکاری تاریخ را فراهم می‌کند.

- این ستون‌ها برای محاسبه ویژگی‌های اضافی استفاده می‌شوند.

۳. محاسبه ویژگی TimeInterval

```
train_data['TimeInterval'] = (train_data['DepartureTime'] - train_data['Created']).dt.days
test_data['TimeInterval'] = (test_data['DepartureTime'] - test_data['Created']).dt.days
```

- محاسبه TimeInterval: تفاوت (بر حسب روز) بین DepartureTime و Created محاسبه می‌شود که یک ویژگی جدید به نام TimeInterval ایجاد می‌کند. این ویژگی می‌تواند به شناسایی الگوهای لغو بلیط کمک کند.

۴. ایجاد ویژگی IsFamily

```
train_data['IsFamily'] = train_data.groupby('BillID')['Male'].transform(lambda x : 1 if x.nunique() > 1 else 0)
test_data['IsFamily'] = test_data.groupby('BillID')['Male'].transform(lambda x : 1 if x.nunique() > 1 else 0)
```

- ویژگی IsFamily: برای هر گروه BillID بررسی می‌کند که آیا بیش از یک ورودی منحصربه‌فرد در ستون Male وجود دارد یا خیر، که نشان‌دهنده یک خانواده است. در صورت درست بودن، به IsFamily مقدار ۱ و در غیر این صورت ۰ اختصاص می‌دهد.

۵. تبدیل تاریخ‌های میلادی به جلالی و استخراج ویژگی‌های مرتبط با تاریخ

```
train_data['DepartureTime'] = train_data['DepartureTime'].apply(lambda x: jdatetime.date.fromgregorian(date=x))
test_data['DepartureTime'] = test_data['DepartureTime'].apply(lambda x: jdatetime.date.fromgregorian(date=x))

train_data['month'] = train_data['DepartureTime'].apply(lambda x: x.month)
test_data['month'] = test_data['DepartureTime'].apply(lambda x: x.month)

train_data['weekend'] = train_data['DepartureTime'].apply(lambda x: x.weekday())
test_data['weekend'] = test_data['DepartureTime'].apply(lambda x: x.weekday())
```

- تبدیل به تاریخ جلالی: DepartureTime به تقویم جلالی تبدیل می‌شود.

- استخراج ویژگی ماه و آخر هفته: ویژگی‌های جدید month (ماه جلالی) و weekend (روز هفته) ایجاد می‌شود تا الگوهای مرتبط با زمان مشخصی از سال یا هفته بررسی شوند.

۶. شناسایی آخر هفته

```
train_data['weekend'] = train_data['weekend'].apply(lambda x: x in [4, 5])
test_data['weekend'] = test_data['weekend'].apply(lambda x: x in [4, 5])
```

- نشانگر آخر هفته: ویژگی weekend به True یا False تبدیل می‌شود و نشان می‌دهد آیا روز در آخر هفته (پنجشنبه یا جمعه) قرار دارد یا خیر.

۷. حذف ستون‌های غیرضروری

```
train_data.drop(columns= ['CancelTime', 'BillID', 'TicketID', 'UserID', 'HashPassportNumber_p', 'HashEmail',
                          'BuyerMobile', 'NationalCode', 'VehicleType', 'Created', 'DepartureTime'], inplace= True)
test_data.drop(columns= ['BillID', 'TicketID', 'UserID', 'HashPassportNumber_p', 'HashEmail',
                          'BuyerMobile', 'NationalCode', 'VehicleType', 'Created', 'DepartureTime'], inplace= True)
```

- حذف ستون‌ها: ستون‌هایی که ممکن است در مدل مؤثر نباشند یا به دلایل حفظ حریم خصوصی نیاز به حذف داشته باشند، حذف می‌شوند.

۸. تقسیم داده‌ها

```
train, val = train_test_split(train_data, test_size= 0.15, random_state= 42, stratify= train_data['Cancel'])
X_val = val.drop(columns= 'Cancel')
Y_val = val['Cancel']
```

- تقسیم داده‌ها: داده‌های آموزشی به مجموعه‌های train و val تقسیم می‌شود، به‌طوری که ۱۵٪ از داده‌ها به‌عنوان اعتبارسنجی استفاده می‌شوند. نمونه‌برداری طبقه‌بندی‌شده (stratified sampling) انجام شده تا توزیع متعادل کلاس Cancel تضمین شود.

۹. آموزش مدل با استفاده از جنگل تصادفی H2O

```
import h2o
from h2o.estimators import H2ORandomForestEstimator
h2o.init()

train_h2o = h2o.H2OFrame(train)
target = 'Cancel'
X = train.columns.to_list()
X.remove(target)
model = H2ORandomForestEstimator(
    ntrees=1000,
    max_depth=20,
    min_rows=5,
    sample_rate=0.4,
    nfolds=5,
    balance_classes=True
)
model.train(x=X, y=target, training_frame=train_h2o)
```

- راه‌اندازی H2O: یک نمونه از H2O راه‌اندازی و train به H2OFrame تبدیل می‌شود.

- پارامترهای جنگل تصادفی: یک مدل جنگل تصادفی با ۱۰۰۰ درخت، عمق حداکثری ۲۰، تعادل کلاس‌ها و cross validation ۵-بخشی آموزش داده می‌شود.

۱۰. ارزیابی مدل

```
from sklearn.metrics import f1_score, classification_report, confusion_matrix
import seaborn as sns
X_val_h2o = h2o.H2OFrame(X_val)
pred = model.predict(X_val_h2o).as_data_frame()
pred['predict'] = pred['predict'].apply(lambda x: 1 if x >= 0.5 else 0)
print(f1_score(Y_val, pred))
print(classification_report(Y_val, pred))
sns.heatmap(confusion_matrix(Y_val, pred))
```

- محاسبه معیارها: پیش‌بینی‌های مدل به فرمت باینری تبدیل می‌شود و معیارهایی مانند امتیاز F1، classification report و confusion matrix محاسبه می‌شود.

- مصور سازی: confusion matrix برای ارزیابی دقیق‌تر عملکرد مدل ترسیم می‌شود.

۱۱. پیش‌بینی داده‌های آزمایشی

```
test_h2o = h2o.H2OFrame(test_data)
test_pr = model.predict(test_h2o).as_data_frame()
test_pr['predict'] = test_pr['predict'].apply(lambda x: 1 if x >= 0.5 else 0)
test_pr['Cancel'] = test_pr['predict']
submission = pd.DataFrame(test_pr['Cancel'])
```

- پیش‌بینی بر روی داده‌های آزمایشی: مدل بر روی داده‌های آزمایشی اعمال و پیش‌بینی‌ها به برچسب‌های باینری تبدیل می‌شود تا برای ارسال آماده شوند.

پروژه ۲

بخش۱:

عنوان پروژه: پیش‌بینی جنسیت کاربر از طریق متن پروفایل با استفاده از یادگیری ماشین

امروزه شبکه‌های اجتماعی کاربردهای گسترده‌ای دارند. اولین استفاده‌ی آن، تفریح و اوقات فراغت است. اما از دید دیگر، می‌توان از شبکه‌های اجتماعی برای پیدا کردن الگوهای رفتاری استفاده کرد. به عنوان مثال با تحلیل نظرات کاربران شبکه‌های اجتماعی می‌توانیم ضعف‌های کسب و کار خود را پیدا کنیم. «جنسیت» یکی از پارامترهای تاثیرگذار در رفتار کاربران است. در مواجهه با یک موضوع، بانوان عمدتاً یک‌طور واکنش نشان خواهند داد و آقایان نیز طور دیگری! حال در این پروژه قصد داریم با اطلاعاتی که دیتاک از کاربران توییتر و اینستاگرم در اختیار ما قرار داده است، جنسیت آن‌ها را پیش‌بینی کنیم.

۱. مقدمه

- هدف: پیش‌بینی جنسیت کاربران بر اساس داده‌های متنی پروفایل‌های شبکه‌های اجتماعی، شامل نام کامل، نام کاربری و بیوگرافی.

- چالش: داده‌های متنی پروفایل‌ها معمولاً شامل نویز، غیررسمی و ترکیبی از زبان‌های مختلف (مثلاً فارسی و انگلیسی) است. بنابراین، پردازش اولیه دقیق و انتخاب مدل مناسب برای رسیدن به دقت بالا ضروری است.

- رویکرد: ما از تکنیک‌های پردازش زبان طبیعی برای پاک‌سازی و تبدیل داده‌های متنی، مدل‌های یادگیری ماشین برای استخراج ویژگی‌ها و طبقه‌بندی، و تکنیک‌های ترکیب مدل‌ها برای بهبود دقت استفاده کرده‌ایم.

۲. بارگذاری داده‌ها

ما داده‌ها را با استفاده از کتابخانه Pandas بارگذاری کرده‌ایم که امکان پردازش و مهندسی ویژگی‌ها را فراهم می‌کند.

```
import numpy as np
import pandas as pd

train = pd.read_csv('../data/train_data.csv')
test = pd.read_csv('../data/test_data.csv')
```

توضیح:

- train: داده‌های آموزشی که شامل برچسب‌های جنسیت است و برای آموزش مدل‌ها استفاده می‌شود.

- test: داده‌های بدون برچسب که برای پیش‌بینی‌های نهایی استفاده می‌شود.

۳. پیش‌پردازش داده‌ها

پیش‌پردازش داده‌ها یکی از مراحل مهم پروژه است. در این بخش، تمرکز بر پاک‌سازی و نرمال‌سازی داده‌های متنی است تا مدل‌ها بتوانند الگوهای مرتبط را شناسایی کنند.

۳,۱ پاک‌سازی و توکن‌سازی متن

۱. نرمال‌سازی:

- متن‌های فارسی را به یک شکل استاندارد تبدیل می‌کند (مثلاً نرمال‌سازی حروف فارسی).

- کتابخانه: از کتابخانه Hazm که برای نرمال‌سازی متون فارسی مناسب است استفاده شده است.

۲. توکن‌سازی:

- متن را به واحدهای جداگانه (کلمات) تقسیم می‌کند.

- هدف: جدا کردن واحدهای معنایی که برای استخراج ویژگی‌ها مؤثر است.

۳. حذف کلمات توقف:

- کلمات رایج و کم‌اهمیت (مثل "و"، "در") را حذف می‌کند.

- هدف: کاهش ابعاد داده و تمرکز روی کلمات اطلاعاتی.

۴. حذف علائم نگارشی و ایموجی‌ها:

- علائم نگارشی و ایموجی‌ها را که معمولاً تأثیر زیادی در پیش‌بینی جنسیت ندارند، حذف می‌کند.

کد برای تابع پیش‌پردازش:

تابع preprocessor این مراحل را پیاده‌سازی کرده و متن را به فرمتی مناسب برای یادگیری ماشین تبدیل می‌کند.

```
import re
from hazm import Normalizer, WordTokenizer, InformalLemmatizer, stopwords_list
norm = Normalizer(persian_numbers= False)
tokenizer = WordTokenizer()
lemma = InformalLemmatizer()
stopwords = stopwords_list()
punctuation = "،؛!؟.?_-()@/,"

for i in ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
 "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself',
 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',
 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these',
 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because',
 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where',
 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no',
 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just',
 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren',
 "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn',
 "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't",
 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren',
 "weren't", 'won', "won't", 'wouldn', "wouldn't"]:
    stopwords.append(i)
```

```
def preproceesor(text):

    text = text.lower()
    def remove_emoji(x):
        emoji_pattern = re.compile("["
        u"\U0001F600-\U0001F64F"  # emoticons
        u"\U0001F300-\U0001F5FF"  # symbols & pictographs
        u"\U0001F680-\U0001F6FF"  # transport & map symbols
        u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
        u"\U00002500-\U00002BEF"  # chinese characters
        u"\U00002702-\U000027B0"
        u"\U00002702-\U000027B0"
        u"\U000024C2-\U0001F251"
        u"\U0001f926-\U0001f937"
        u"\U00010000-\U0010ffff"
        u"\u200d"
        u"\u2640-\u2642"
        u"\u2600-\u2B55"
        u"\u23cf"
        u"\u23e9"
        u"\u231a"
        u"\u3030"
        u"\ufe0f"  # dingbats
        "]+", flags=re.UNICODE)
        return emoji_pattern.sub(r'', x)

    text = remove_emoji(text)
    normalized_text = norm.normalize(text)
    tokens = tokenizer.tokenize(normalized_text)
    removal =list()
```

```
text = remove_emoji(text)
normalized_text = norm.normalize(text)
tokens = tokenizer.tokenize(normalized_text)
removal =list()

for token in tokens:
    if token in punctuation or token.isdigit() or token in stopwords:
        removal.append(token)
for i in removal:
    tokens.remove(i)
return ' '.join(tokens)
```

۴. استخراج ویژگی‌ها با استفاده از TF-IDF

یکی از مشکلات امتیازدهی براساس فرکانس یا شمارش این است که مثلاً کلمه‌ی the ممکن است در یک سند بسیار پرتکرار باشد و امتیاز بالایی کسب کند، اما چنین کلمه‌ای واقعاً محتوای اطلاعاتی زیادی برای مدل فراهم نمی‌کند.

برای حل این مشکل می‌توان به کمک این‌که هر کلمه چه‌قدر در کل سندها ظاهر شده، فرکانس کلمه‌ها را تغییر مقیاس داد. به چنین روشی TF-IDF گفته می‌شود که متشکل از دو جزء زیر است:

فرکانس عبارت (Term Frequency): امتیازی بر اساس فرکانس رخداد کلمه در سند جاری. برای محاسبه‌ی آن کافیست تعداد رخداد کلمه در آن سند را تقسیم بر تعداد کل کلمه‌های آن سند کنیم.

معکوس فرکانس سند (Inverse Document Frequency): امتیازی بر اساس این‌که آن کلمه تا چه‌حد در میان سندها کم‌یاب است. برای محاسبه‌ی آن کافیست تعداد اسناد را بر تعداد اسنادی که آن کلمه در آن‌ها رخ داده تقسیم کنیم و در نهایت از آن لگاریتم بگیریم.

برای محاسبه‌ی مقدار نهایی TF-IDF کافیست حاصل دو جزء بالا را در هم ضرب کنیم.

فرض کنید یک سند شامل ۱۰۰۰ کلمه باشد و کلمه‌ی a در آن 50 بار تکرار شده باشد. در این‌صورت فرکانس عبارت آن با تقسیم ۵۰ بر ۱۰۰۰ به دست می‌آید که معادل ۰/۰۵ است. حال فرض کنید مجموعاً ۳۰۰۰ سند داشته باشیم و در ۱۰۰ سند، کلمه‌ی a ظاهر شده باشد. در این‌صورت معکوس فرکانس سند برابر خواهد شد با لگاریتم تقسیم ۳۰۰۰ بر ۱۰۰ یعنی معادل با ۱/۴۷. در نهایت ۰/۰۵ را ضرب در ۱/۴۷ می‌کنیم که معادل است با ۰/۰۷۳.

کد برای استخراج ویژگی‌های TF-IDF

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfTransformer, CountVectorizer

pipe = Pipeline([('count', CountVectorizer(preprocessor= preproceesor)),
                 ('tfidf', TfidfTransformer())])
```

```
vector_columns = ['fullname', 'username', 'biography']
vector_dict = dict()
for c in vector_columns:
    pipe.fit(train[c])
    vector_dict[f'{c}_vectors'] = pd.DataFrame(pipe.transform(train[c]).toarray(), columns= pipe.get_feature_names_out())
    vector_dict[f'{c}_testvectors'] = pd.DataFrame(pipe.transform(test[c]).toarray(), columns= pipe.get_feature_names_out())
```

۵. استانداردسازی و کدگذاری داده‌ها

توضیح:

- استانداردسازی: نرمال‌سازی follower_count و following_count برای هماهنگی مقدارها.

- کدگذاری: تبدیل متغیر هدف gender به صورت عددی (۰ برای زن و ۱ برای مرد).

```
from sklearn.preprocessing import StandardScaler

train['gender'] = train['gender'].map({'woman': 0, 'man': 1})
scaler = StandardScaler()

for column in ['follower_count', 'following_count']:
    train[column] = scaler.fit_transform(train[[column]])
    test[column] = scaler.transform(test[[column]])
```

۶. کاهش ابعاد با PCA

توضیح PCA:

- هدف: کاهش ابعاد داده‌ها با حفظ ۸۰٪ اطلاعات، که باعث بهبود کارایی و کاهش پیچیدگی مدل می‌شود.

- کاربرد: این تکنیک به دلیل ماهیت چندبعدی ویژگی‌های TF-IDF مفید است.

```
from sklearn.decomposition import PCA
pca = PCA(n_components= 0.8)
for i, j in zip(['fullname_vectors', 'username_vectors', 'biography_vectors'], ['fullname_testvectors', 'username_testvectors', 'biography_testvectors']):
    vector_dict[i] = pd.DataFrame(pca.fit_transform(vector_dict[i]))
    vector_dict[j] = pd.DataFrame(pca.transform(vector_dict[j]))
```

۷. مدل‌سازی

ما از ترکیب مدل‌ها برای بهره‌گیری از ویژگی‌های هر کدام و ایجاد یک مدل ترکیبی استفاده کرده‌ایم.

۷,۱ دسته‌بند SVC برای هر ستون متنی

برای هر ستون متنی (fullname، username، و biography) یک مدل SVC جداگانه آموزش داده شده است تا الگوهای منحصربه‌فرد هر منبع را شناسایی کند.

```
from sklearn.svm import SVC

# Initialize and train models
models = []
for column in ['fullname', 'username', 'biography']:
    model = SVC(C=1, kernel='poly')
    model.fit(vector_dict[f'{column}_vectors'].loc[train.index], train['gender'])
    models.append(model)
```

۷,۲ مدل ترکیبی با XGBoost

ما با استفاده از مدل XGBoost یک مدل ترکیبی می‌سازیم. این مدل ترکیبی پیش‌بینی‌های مدل‌های SVC برای هر منبع متنی (نام کامل، نام کاربری و بیوگرافی) و دیگر ویژگی‌های عددی (مثل تعداد دنبال‌کننده‌ها و دنبال‌شوندگان) را با هم ترکیب کرده و جنسیت کاربر را پیش‌بینی می‌کند.

چرا XGBoost؟

XGBoost یک الگوریتم تقویت‌شده گرادیان است که در مسابقات علمی و چالش‌های داده محبوبیت بالایی دارد، زیرا:

- سرعت و کارایی: XGBoost به طور بهینه و با سرعت بالا عمل می‌کند، و همچنین در تنظیمات مختلفی مثل طبقه‌بندی و رگرسیون قابل استفاده است.

- دقت بالا: این الگوریتم از رویکرد تقویت‌شده گرادیان استفاده می‌کند که به مدل کمک می‌کند به دقت بالایی برسد.

- توانایی در مقابله با داده‌های نامتوازن: XGBoost توانایی خوبی در مدیریت داده‌های نامتوازن دارد، که در این پروژه، داده‌های جنسیتی ممکن است نامتوازن باشند.

مراحل آموزش مدل XGBoost به عنوان یک مدل ترکیبی

۱. ویژگی‌های ورودی: ما داده‌ها را به مدل XGBoost به دو دسته‌ی اصلی تقسیم می‌کنیم:

- ویژگی‌های متنی: بردارهای متنی که از پردازش نام کامل، نام کاربری، و بیوگرافی استخراج شده و توسط مدل‌های SVC هرکدام پیش‌بینی شده‌اند.

- ویژگی‌های عددی: ویژگی‌های عددی مانند follower_count و following_count که به کمک استانداردسازی تغییر مقیاس داده‌اند.

۲. ساختن مدل XGBoost: ما یک مدل XGBoost را با پارامترهای پیش‌فرض تنظیم کرده و آن را بر روی ویژگی‌های ورودی آموزش می‌دهیم. به این ترتیب، مدل از پیش‌بینی‌های هر مدل متنی (SVC) و همچنین ویژگی‌های عددی برای پیش‌بینی جنسیت نهایی کاربر استفاده می‌کند.

۳. آموزش مدل با داده‌های آموزشی: مدل XGBoost با داده‌های آموزشی آموزش می‌بیند و یاد می‌گیرد که چگونه وزن مناسبی به هر یک از منابع مختلف ویژگی‌ها اختصاص دهد تا به پیش‌بینی‌های دقیق‌تر برسد.

```
from xgboost import XGBClassifier

# ساخت مدل XGBoost
model = XGBClassifier()

# آموزش مدل با داده‌های آموزشی که شامل همه ویژگی‌ها است (و ویژگی‌های عددی SVC پیش‌بینی‌های)
model.fit(train.drop(columns='gender'), train['gender'])
```

مدل XGBoost به عنوان یک مدل ترکیبی عمل می‌کند و پیش‌بینی‌های مدل‌های SVC و ویژگی‌های عددی را با هم ترکیب می‌کند. این کار با بهره‌گیری از توانایی XGBoost در شناسایی الگوهای پیچیده و مهم از ویژگی‌های مختلف، منجر به دقت بالاتری در پیش‌بینی جنسیت می‌شود.

۸. ارزیابی

برای ارزیابی عملکرد مدل، از classification_report و f1_score استفاده می‌کنیم.

```
from sklearn.metrics import classification_report, f1_score

pred = model.predict(val.drop(columns='gender'))
print(f1_score(val['gender'], pred))
print(classification_report(val['gender'], pred))
```

توضیح:

- F1 Score: ترکیبی از دقت و یادآوری را فراهم می‌کند، که برای کلاس‌های نامتوازن بسیار مفید است.
- classification report: متریک‌هایی مانند accuracy، recall و F1 برای هر کلاس را ارائه می‌دهد.

۹. پیش‌بینی نهایی روی داده‌های تست

مدل آموزش‌دیده را برای پیش‌بینی روی داده‌های تست اعمال می‌کنیم.

```
test_pred = model.predict(test)
submission = pd.DataFrame(test_pred, columns=['gender'])
submission['gender'] = submission['gender'].map({0: 'woman', 1: 'man'})
submission
```

۱۰. نتایج و یافته‌ها

- F1 Score: ٪۷۹

- classification report:

	precision	recall	f1-score	support
0	0.79	0.82	0.80	400
1	0.81	0.78	0.80	400
accuracy			0.80	800
macro avg	0.80	0.80	0.80	800
weighted avg	0.80	0.80	0.80	800

پروژه ۲

بخش۲:

عنوان پروژه: موضوع‌بندی

در این پروژه داده‌های واقعی وب فارسی که توسط پلتفرم یکتانت پالایش و جمع‌آوری شده در اختیار ما قرار گرفته است. هدف پروژه؛ ساخت یک مدل یادگیری ماشین است که با توجه متن‌های موجود در یک پیوند (Link) نظیر «عنوان»، «توضیحات»، «محتوای متنی [کامل]» و غیره بتواند دسته‌ی موضوعی آن سند را پیش‌بینی کند. به‌عنوان مثال اگر پیوندی از یک سایت خبری با عنوان «کیهان کلهر جایزه مرد سال موسیقی جهان را دریافت کرد» داشته باشیم، مدل ما باید پیش‌بینی کند که این مطلب مرتبط با موضوع «موسیقی» است. البته در این مثال ما تنها از ویژگی «عنوان» یاد کردیم، در حالی‌که می‌توان از متن «توضیحات» یا «محتوای متنی» هم کمک گرفت.

هدف پروژه

هدف این پروژه، دسته‌بندی داده‌های متنی به دسته‌های مشخص با استفاده از تکنیک‌های پیش‌پردازش متن و یادگیری ماشین پیشرفته است. این پروژه هر مرحله از فرایند، شامل آماده‌سازی داده‌ها، مهندسی ویژگی‌ها، آموزش مدل و ارزیابی را پوشش می‌دهد و در نهایت به تولید پیش‌بینی برای داده‌های آزمایشی منجر می‌شود.

۱. کتابخانه‌ها و آماده‌سازی داده‌ها

۱,۱ وارد کردن کتابخانه‌ها

برای انجام مراحل مختلف فرایند یادگیری ماشین، کتابخانه‌های زیر وارد می‌شوند:

- پردازش داده‌ها: numpy و pandas برای کار با داده‌ها و انجام تغییرات به کار می‌روند.

- پردازش متن: کتابخانه re برای استفاده از الگوهای منظم در پاکسازی متن و hazm برای توکنیزاسیون و نرمال‌سازی متن فارسی.

- یادگیری ماشین: بسته sklearn ابزارهایی برای برداری‌سازی، تبدیل، کدگذاری و ارزیابی مدل‌ها فراهم می‌کند و xgboost برای آموزش مدل دسته‌بندی به دلیل عملکرد قوی‌اش در داده‌های ساختار یافته استفاده می‌شود.

```
import numpy as np
import pandas as pd
import re
from hazm import Normalizer, WordTokenizer, InformalLemmatizer, stopwords_list
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfTransformer, CountVectorizer
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, f1_score
from xgboost import XGBClassifier
```

۱,۲ خواندن و پیش‌نمایش داده‌ها

داده‌های آموزش و آزمایش با استفاده از pd.read_csv() خوانده می‌شوند و اولین چند سطر برای مشاهده ساختار و انواع ستون‌ها نمایش داده می‌شوند.

```
train = pd.read_csv('../data/yektanet_train.csv')
test = pd.read_csv('../data/yektanet_test.csv')
train.head()
test.head()
```

ستون‌های داده:

ستون	توضیحات
`category`	موضوع (متغیر هدف)
`description`	توضیحات
`text_content`	محتوای متنی
`title`	عنوان
`h1`	محتوای تگ `h1` صفحه
`h2`	محتوای تگ `h2` صفحه
`url`	آدرس پیوند
`domain`	دامنه‌ی وب‌سایت
`id`	آیدی پیوند

۲. پیش‌پردازش داده‌ها

۲,۱ مدیریت مقادیر گم‌شده

برای مدیریت مقادیر گم‌شده، ستون description در صورت نبودن داده از text_content پر می‌شود. به همین صورت، domain در صورت نبود داده از url پر می‌شود.

```
train['description'].fillna(train['text_content'], inplace=True)
test['description'].fillna(train['text_content'], inplace=True)
train['domain'].fillna(train['url'], inplace=True)
test['domain'].fillna(train['url'], inplace=True)
```

۲,۲ استخراج نام دامنه

برای استخراج دامنه اصلی از URL، یک تابع کمکی domain_name تعریف می‌شود که بخش ریشه دامنه را استخراج می‌کند.

```
def domain_name(text):
    return text.rsplit('.')[0]

train['domain'] = train['domain'].apply(domain_name)
test['domain'] = test['domain'].apply(domain_name)
```

۲,۳ حذف ستون‌های غیرضروری

ستون‌های غیرمربوط مانند h1، h2، id و url حذف می‌شوند تا داده‌ها ساده‌تر شده و نویز کاهش یابد.

```
train.drop(columns=['h1', 'h2', 'id', 'url'], inplace=True)
test.drop(columns=['h1', 'h2', 'id', 'url'], inplace=True)
```

۳. پاکسازی و توکنیزاسیون متن

۳,۱ تابع توکنیزاسیون

تابع tokenizer مراحل پردازش متن را انجام می‌دهد:

۱. کوچک‌سازی حروف برای یکنواخت‌سازی.

۲. حذف اموجی‌ها با استفاده از الگوهای regex.

۳. نرمال‌سازی با استفاده از hazm برای استانداردسازی متن فارسی.

۴. توکنیزاسیون و حذف کلمات بی‌معنی و نشانه‌گذاری‌ها.

```python
def tokenizer(text):
    text = text.lower()
    emoji_pattern = re.compile("["..."]+", flags=re.UNICODE)  # برای حذف اموجی‌ها regex الگوی
    text = emoji_pattern.sub(r'', text)

    norm = Normalizer(persian_numbers=False)
    normalized_text = norm.normalize(text)

    wordtokenizer = WordTokenizer()
    tokens = wordtokenizer.tokenize(normalized_text)

    punctuation = "،.؛!؟.?_-()@/,"
    stopwords = stopwords_list()
    tokens = [token for token in tokens if token not in punctuation and token not in stopwords]

    return ' '.join(tokens)
```

۳,۲ برداری‌سازی متن با TF-IDF

ما با استفاده از ترکیب CountVectorizer و TfidfTransformer داده‌های متنی را به بردارهای عددی تبدیل می‌کنیم.

```python
pipe = Pipeline([
    ('count', CountVectorizer(preprocessor=tokenizer)),
    ('tfidf', TfidfTransformer())
])
```

۴. مهندسی ویژگی‌ها

۴,۱ تولید بردارهای TF-IDF برای ویژگی‌های متنی

هر ستون متنی (description، text_content، و title) برای مجموعه داده‌های آموزش و آزمایش برداری‌سازی می‌شود.
بردارهای تولید شده در دیکشنری vector_dict ذخیره می‌شوند.

```python
vector_columns = ['description', 'text_content', 'title']
vector_dict = dict()
for c in vector_columns:
    pipe.fit(train[c])
    vector_dict[f'{c}_vectors'] = pd.DataFrame(pipe.transform(train[c]).toarray(), columns= pipe.get_feature_names_out())
    vector_dict[f'{c}_testvectors'] = pd.DataFrame(pipe.transform(test[c]).toarray(), columns= pipe.get_feature_names_out())
```

۴,۲ برداری‌سازی ویژگی دامنه

نام دامنه‌ها نیز برای نمایش اطلاعات به صورت ویژگی عددی برداری‌سازی می‌شوند.

```
count_vect = CountVectorizer()
count_vect.fit(train['domain'])
final_counts = count_vect.transform(train['domain'])
vector_dict['domain_vectors'] = pd.DataFrame(pipe.transform(train['domain']).toarray(), columns= pipe.get_feature_names_out())
vector_dict['domain_testvectors'] = pd.DataFrame(pipe.transform(test['domain']).toarray(), columns= pipe.get_feature_names_out())
```

۵. کاهش ابعاد با PCA

استفاده از تحلیل مولفه‌های اصلی (PCA) به کاهش ابعاد ویژگی‌ها کمک می‌کند و تنها ۸۰٪ از واریانس را حفظ می‌کند تا خطر بیش‌برازش کاهش یابد.

```
from sklearn.decomposition import PCA
pca = PCA(n_components= 0.8)
for i, j in zip(['description_vectors', 'text_content_vectors', 'title_vectors'],
                ['description_testvectors', 'text_content_testvectors', 'title_testvectors']):
    vector_dict[i] = pd.DataFrame(pca.fit_transform(vector_dict[i]))
    vector_dict[j] = pd.DataFrame(pca.transform(vector_dict[j]))
```

و در انتها سه ستون را حذف می‌کنیم.

```
train.drop(columns= ['description', 'text_content', 'title', 'domain'], inplace=True)
test.drop(columns= ['description', 'text_content', 'title', 'domain'], inplace=True)
```

۶. تقسیم داده‌ها به مجموعه‌های آموزشی و اعتبارسنجی

```
from sklearn.model_selection import train_test_split
train, val = train_test_split(train, random_state= 42, test_size= 0.1, stratify= train['category'])
```

- تقسیم داده‌های train_data به train و val (۹۰٪ برای آموزش و ۱۰٪ برای اعتبارسنجی).
- استفاده از stratify برای حفظ توزیع کلاس‌ها در category.

۷. کدگذاری برچسب‌ها

برای تبدیل مقادیر دسته‌ای در ستون هدف category به مقادیر عددی از LabelEncoder استفاده می‌شود.

```
label_encoder = LabelEncoder()
train['category'] = label_encoder.fit_transform(train['category'])
```

۸. آموزش و ارزیابی مدل

۸.۱ آموزش مدل‌ها برای هر ویژگی

با استفاده از XGBClassifier، مدل‌های جداگانه‌ای برای هر مجموعه بردار ویژگی آموزش داده می‌شوند تا دسته‌ها را براساس ستون‌های متنی مختلف پیش‌بینی کنند.

```
from xgboost import XGBClassifier
description_model = XGBClassifier()
description_vectors = vector_dict['description_vectors']
description_model.fit(description_vectors.loc[train.index], train['category'])
```

```
text_content_model = XGBClassifier()
text_content_vectors = vector_dict['text_content_vectors']
text_content_model.fit(text_content_vectors.loc[train.index], train['category'])
```

```
title_model = XGBClassifier()
title_vectors = vector_dict['title_vectors']
title_model.fit(title_vectors.loc[train.index], train['category'])
```

```
domain_model = XGBClassifier()
domain_vectors = vector_dict['domain_vectors']
domain_model.fit(domain_vectors.loc[train.index], train['category'])
```

۸.۲ ترکیب پیش‌بینی‌ها برای مدل نهایی

پیش‌بینی‌های مدل‌های جداگانه به عنوان ورودی مدل نهایی استفاده می‌شوند که دسته نهایی را پیش‌بینی می‌کند.

```
for m, v, d in zip([description_model, text_content_model, title_model, domain_model],
                   [description_vectors, text_content_vectors, title_vectors, domain_vectors],
                   ['description', 'text_content', 'title', 'domain']):
    train[d] = m.predict(v.loc[train.index])
    val[d] = m.predict(v.loc[val.index])
```

۸,۳ مدل نهایی

در انتها با استفاده از نتایج مدل‌های قبلی مدل نهایی قرار می‌گیرد

```
model = XGBClassifier()

model.fit(train.drop(columns= 'category'), train['category'])
```

۹. ارزیابی مدل

۹.۱ معیارهای ارزیابی

برای ارزیابی مدل روی مجموعه اعتبارسنجی، از f1_score و classification_report استفاده می‌شود که دقت دسته‌بندی‌ها را در دسته‌های مختلف نشان می‌دهد.

```
pred = model.predict(val.drop(columns='category'))
print(f1_score(val['category'], pred, average='weighted'))
print(classification_report(val['category'], pred))
```

۱۰. پیش‌بینی نهایی و ارسال نتایج

۱۰.۱ استفاده از مدل‌ها برای پیش‌بینی داده‌های آزمایشی

هر مدل برای بردارهای مربوطه در مجموعه آزمایشی پیش‌بینی‌هایی ایجاد می‌کند.

```
description_testvectors = vector_dict['description_testvectors']
text_content_testvectors = vector_dict['text_content_testvectors']
title_testvectors = vector_dict['title_testvectors']
domain_testvectors = vector_dict['domain_testvectors']

for m, v, d in zip([description_model, text_content_model, title_model, domain_model],
                   [description_testvectors, text_content_testvectors, title_testvectors, domain_testvectors],
                   ['description', 'text_content', 'title', 'domain']):
    test[d] = m.predict(v)

test_pred = model.predict(test)
```

۱۰.۲ تولید فایل خروجی

پیش‌بینی‌ها در یک DataFrame ذخیره شده و ستون category به برچسب‌های اصلی بازگردانده می‌شود.

```
submission = pd.DataFrame(test_pred, columns=['category'])
submission['category'] = label_encoder.inverse_transform(submission['category'])
```

۱۱. نتایج و یافته‌ها

- F1 Score: ۷۴٪

- classification report:

	precision	recall	f1-score	support
0	0.35	0.43	0.38	21
1	0.00	0.00	0.00	5
2	0.71	0.54	0.61	28
3	0.50	0.50	0.50	8
4	0.69	0.83	0.75	29
5	0.64	0.73	0.69	49
6	0.00	0.00	0.00	3
7	0.67	0.62	0.64	13
8	1.00	0.33	0.50	3
9	0.71	0.71	0.71	24
10	0.83	0.83	0.83	18
11	0.82	0.82	0.82	62
12	0.00	0.00	0.00	3
13	0.79	0.65	0.71	17
14	0.91	0.88	0.89	24
15	1.00	0.83	0.91	12
16	0.83	0.62	0.71	16
17	0.93	1.00	0.96	13
18	0.94	0.94	0.94	31
19	0.57	0.76	0.65	41
20	0.94	0.94	0.94	51
21	0.86	0.75	0.80	8
...				
accuracy			0.75	479
macro avg	0.67	0.62	0.63	479
weighted avg	0.75	0.75	0.74	479

پروژه ۲

بخش۳:

عنوان پروژه: تحلیل جست‌وجو

این پروژه به تحلیل الگوهای استفاده از خدمات حمل و نقل و اقامتی در ایران با استفاده از داده‌های جستجوی کاربران می‌پردازد. ما انواع مختلف حمل و نقل و اقامت (اتوبوس، پرواز، قطار، هتل، تاکسی، و کشتی) را دسته‌بندی و فراوانی هر نوع خدمت را براساس فعالیت‌های جستجوی کاربران بررسی می‌کنیم. با ترکیب این داده‌ها با اطلاعات جمعیتی و شهری، محبوبیت شهرها از نظر حجم جستجو برای خدمات مختلف مورد بررسی قرار می‌گیرد. اطلاعات‌های کلیدی از طریق نمودارهای دایره‌ای و هیستوگرام‌ها به تصویر کشیده می‌شود که ترجیحات کاربران در خدمات حمل و نقل و محبوب‌ترین شهرها را نشان دهد. این تحلیل به درک تقاضای سفر، محبوبیت شهرها و توزیع خدمات حمل و نقل در ایران کمک می‌کند.

۱. مقدمه

با افزایش استفاده از پلتفرم‌های آنلاین برای خدمات سفر و حمل و نقل، تحلیل داده‌های جستجوی کاربران به یک روش موثر برای درک ترجیحات سفر تبدیل شده است. این پروژه از داده‌های جستجوی کاربران از سرویس «مستربلیت» و اطلاعات جمعیتی شهرهای ایران برای تحلیل موارد زیر استفاده می‌کند:

۱. درصد هر نوع خدمات حمل و نقل و اقامت.

۲. محبوبیت شهرهای خاص به عنوان مقصد خدمات یا سفر.

۳. ترکیب داده‌های شهری با داده‌های جمعیتی برای بررسی الگوهای تقاضا.

۲. آماده‌سازی داده‌ها

۲,۱. کتابخانه‌ها

ما از کتابخانه‌های زیر استفاده می‌کنیم:

- numpy و pandas برای پردازش داده‌ها.
- plotly.express برای مصورسازی داده‌ها.

۲,۲. بارگذاری داده‌ها

۱. داده‌های جستجوی کاربران: داده‌های جستجوی کاربران از یک فایل JSON (mrbilit_search.json) بارگذاری می‌شود که شامل جزئیاتی مانند نوع خدمات جستجو شده (مثلاً اتوبوس، پرواز) و نام شهرها است.

ستون	توضیحات
`ServiceType`	نوع سرویس
`TypedStrings`	لیستی از رشته‌ی تایپ‌شده‌ی کاربر به ترتیب زمانی
`AcceptString`	رشته‌ای که در نهایت انتخاب شده است

۲. داده‌های شهر: داده‌های اضافی شهرها از یک فایل CSV (iran_cities.csv) بارگذاری می‌شود. این فایل شامل اطلاعاتی از قبیل نام هر شهر، استان و جمعیت براساس سرشماری سال ۲۰۱۶ است.

ستون	توضیحات
City EN	نام انگلیسی شهر
City FA	نام فارسی شهر
Province	استان
Countise	شهرستان
District	بخش
Latitude	عرض جغرافیایی
Longitude	طول جغرافیایی
Area	مساحت (کیلومتر مربع)
Elevation	ارتفاع (متر)
Census 2016	جمعیت سال ۲۰۱۶
Census 2011	جمعیت سال ۲۰۱۱
Wikipedia EN	لینک ویکی‌پدیا انگلیسی
Wikipedia FA	لینک ویکی‌پدیا فارسی
GeoHack	لینک سایت GeoHack

```
import numpy as np
import pandas as pd
import plotly.express as px

# بارگذاری فایل‌های داده
data = pd.read_json('../data/mrbilit_search.json')
cities = pd.read_csv('../data/iran_cities.csv')
```

۳. تحلیل نوع خدمات

۳,۱. محاسبه درصد انواع خدمات

ستون ServiceType در دیتافریم data شامل نوع خدماتی است که کاربران جستجو کرده‌اند. ما درصد هر نوع خدمات نسبت به کل جستجوها را محاسبه می‌کنیم.

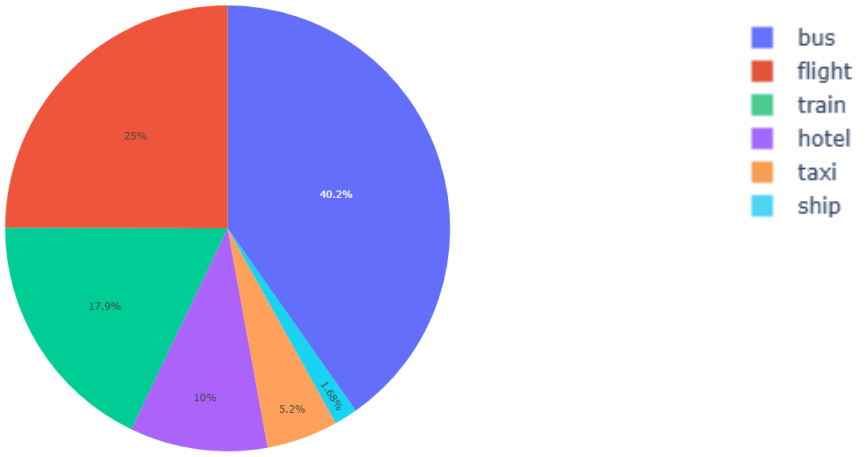
```
percentages = {'bus': 0, 'flight': 0, 'train': 0, 'hotel': 0, 'taxi': 0, 'ship': 0}

# محاسبه درصد برای هر نوع خدمات
for s in percentages.keys():
    percentages[s] = (data['ServiceType'][data['ServiceType'] == s].count() / data.shape[0])
```

۳,۲. نمودار توزیع نوع خدمات

از نمودار دایره‌ای برای نمایش درصد هر نوع خدمات استفاده می‌کنیم، که نمایی شفاف از ترجیحات کاربران را نشان می‌دهد.

```
import plotly.io as pio
pio.renderers.default = 'browser'
fig = px.pie(values=percentages.values(),
             names=list(percentages.keys()),
             title='نمودار دایره‌ای از انواع خدمات')
fig.show()
```

نمودار:



۴. پیش‌پردازش داده‌ها

۴,۱. نرمال‌سازی نام شهرها

برای استانداردسازی نام شهرها، ما در ستون AcceptString در data، هر گونه پسوند (مانند "- پایانه") را حذف می‌کنیم.

```
preprocessed_data = data.copy()

def only_city(text):
    if '- پایانه' in text:
        return text.split(' -')[0]
    else:
        return text

# اعمال تابع پاکسازی
preprocessed_data['AcceptString'] = preprocessed_data['AcceptString'].apply(only_city)
preprocessed_data.head()
```

۴,۲. فیلتر کردن خدمات حمل و نقل

از آنجایی که پروژه بر داده‌های حمل و نقل متمرکز است، ورودی‌هایی که ServiceType آنها "هتل" است را حذف می‌کنیم.

```
data_transport = preprocessed_data[preprocessed_data['ServiceType'] != 'hotel']
```

۵. تحلیل محبوبیت شهرها

۵,۱. شناسایی شهرهای پرجستجو

۲۰ شهر برتر را بر اساس تعداد جستجوها برای خدمات حمل و نقل شناسایی می‌کنیم.

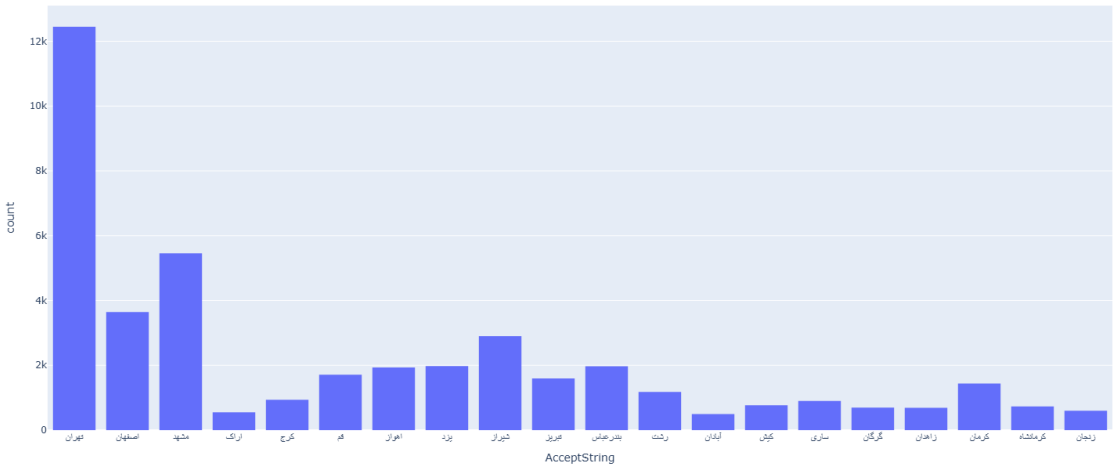
```
top_cities = data_transport['AcceptString'].value_counts()[:20].index.tolist()
```

۵,۲. نمودار: ۲۰ شهر برتر بر اساس حجم جستجو

از هیستوگرام برای نمایش ۲۰ شهر برتر از نظر حجم جستجو استفاده می‌کنیم که اطلاعات‌هایی درباره‌ی محبوب‌ترین شهرها را در میان کاربران ارائه می‌دهد.

```
temp = data_transport[data_transport['AcceptString'].isin(top_cities)]
fig = px.histogram(temp, x="AcceptString", title='هیستوگرام از ۲۰ شهر برتر')
fig.show()
```

نمودار:



۶. تحلیل جمعیت شهرها

۶,۱. ترکیب با داده‌های شهری

ما داده‌های فیلتر شده خود را با داده‌های جمعیتی شهر ترکیب می‌کنیم تا اطلاعاتی همچون جمعیت و استان را نیز اضافه کنیم. این کار به ما کمک می‌کند تا محبوبیت شهرها را با توجه به اندازه و جمعیت آنها بسنجیم.

```
temp = pd.merge(data_transport, cities[['City FA', 'Province', '2016 Census']],
                left_on='AcceptString', right_on='City FA')
```

۶,۲. شناسایی شهرهای پرجمعیت

ما روی شهرهایی با جمعیت بالای ۵۰۰،۰۰۰ نفر تمرکز می‌کنیم تا شهرهایی که جمعیت بالایی دارند اما در جستجوها کمتر دیده می‌شوند را شناسایی کنیم.

```
top_provinces = temp['Province'].value_counts()[:20].index.tolist()
temp = temp['City FA'][temp['2016 Census'] >= 500000]
not_in_top = list(set(temp) - set(top_cities))
```

۷. نتایج

۱. ترجیحات خدمات: نمودار دایره‌ای، شکلی از ترجیحات کاربران را بر اساس نوع خدمات ارائه می‌دهد. از این طریق می‌توان نتیجه گرفت که کدام خدمات بیشتر و کدام کمتر محبوبیت دارند. به عنوان مثال، اگر پروازها بخش عمده‌ای از نمودار را تشکیل دهند، به علاقه بیشتر کاربران به سفرهای هوایی اشاره دارد.

۲. محبوبیت شهرها: هیستوگرام از ۲۰ شهر برتر، نشان می‌دهد که کاربران به کدام شهرها بیشتر علاقه دارند و این می‌تواند مقاصد محبوب یا مراکز پرتردد را شناسایی کند.

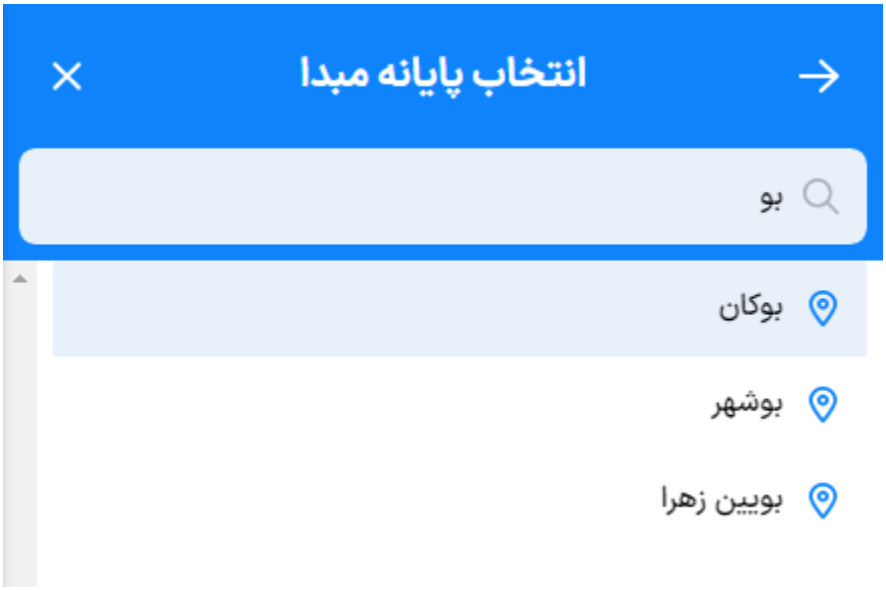
۳. تحلیل جمعیت: با مقایسه شهرهای پر جستجو با داده‌های جمعیتی، می‌توانیم شهرهای پرجمعیتی که جستجوهای کمتری دارند را شناسایی کنیم؛ این مورد ممکن است به عدم حضور خدمات کافی در این مناطق یا علاقه کمتر به سفر به این مناطق اشاره کند.

پروژه ۲

بخش ۴:

عنوان پروژه: پیشنهاد خودکار

در این پروژه به سراغ یک مسئله‌ی کاملاً واقعی و چالش‌برانگیز صنعت خواهیم رفت. مشاهده کرده‌اید که در وب‌سایت‌ها هنگامی‌که در حال تایپ داخل یک فیلد متنی هستید معمولاً لیستی از متن‌های مشابه به شما پیشنهاد می‌شود. نمونه‌ای از این قابلیت که در وب سایت مستربلیط به کار گرفته شده را در تصویر زیر مشاهده می‌کنید.



با این حال معمولاً چنین لیست‌های پیشنهادی‌ای بسیار ساده هستند و تنها عباراتی را که با رشته‌ی تایپ‌شده‌ی کاربر شروع می‌شوند پیشنهاد می‌دهند تا کاربر نیازی به تکمیل تایپ خود نداشته باشد و زودتر به نتیجه برسد. اما در این پروژه قصد داریم پیشنهاددهنده‌ای بسازیم که هوشمندتر و منعطف‌تر عمل کند. به‌عنوان مثال شاید کاربر عبارت «بابل» را تایپ کرده اما در ابتدا غلط املایی داشته و منظورش «زابل» بوده باشد. یا شاید همان‌طور که حتماً برای شما هم پیش آمده و روی اعصاب‌تان رفته یادش رفته باشد که کیبورد خود را روی زبان فارسی تنظیم کند و عبارتی مثل «fhfg» را تایپ کرده باشد. همچنین چرا اسم‌های انگلیسی شهرها را هم در نظر نگیریم و وقتی شخص مثلاً «Zahedan» را تایپ کرده باشد متوجه نشویم که منظورش «زاهدان» بوده است؟ با این حال، این‌ها تنها تعدادی از الگوهای ممکن هستند که منجر به بهبود پیشنهادهای سیستم خواهند شد. ما می‌توانیم به کمک داده‌های جمع‌آوری‌شده از آن‌چه که کاربران مرحله به مرحله تایپ کرده‌اند و آن‌چه که در نهایت انتخاب کرده‌اند مدلی طراحی کنیم که با توجه به ورودی کاربر، لیستی از محتمل‌ترین انتخاب‌ها را به وی پیشنهاد دهد.

۱. مقدمه

این سیستم به منظور بهبود تجربه کاربر در پیدا کردن مکان‌های مقصد خود در سیستم‌های حمل‌ونقل (مانند تاکسی‌های آنلاین و اتوبوس) طراحی شده است. چالش‌های این مدل شامل:

- شناسایی غلط‌های املایی و تایپی

- تشخیص کاراکترهای انگلیسی که باید به کاراکترهای فارسی تبدیل شوند.

- ایجاد رتبه‌بندی مناسب براساس تاریخچه جستجوهای قبلی کاربران و میزان محبوبیت مکان‌هاست.

این مدل باید قادر باشد با اطلاعات حداقلی و گاهی ناقص، مکان‌های مناسب را با دقت بالا به کاربر پیشنهاد دهد.

۲. کتابخانه‌ها

برای پیاده‌سازی این مدل، از کتابخانه‌های زیر استفاده شده است:

- numpy: برای انجام محاسبات عددی کارآمد.

- pandas: برای پردازش و مدیریت داده‌ها، از جمله فیلتر کردن و گروه‌بندی اطلاعات.

- Levenshtein: برای محاسبه فاصله بین رشته‌ها که در محاسبه شباهت بین ورودی کاربر و مکان‌های معتبر استفاده می‌شود.

- sklearn.preprocessing.MinMaxScaler: برای نرمال‌سازی فاصله‌ها و تبدیل آن‌ها به امتیازهای قابل مقایسه.

- re: برای شناسایی الگوهای متنی مانند وجود حروف انگلیسی.

- rbo: برای محاسبه شباهت بین دو لیست رتبه‌بندی شده، جهت ارزیابی دقت پیشنهادات مدل.

۳. بارگذاری داده‌ها

داده‌های مورد استفاده از فایل‌های JSON و CSV بارگذاری می‌شوند که شامل اطلاعات زیر است:

ستون	توضیحات
`ServiceType`	نوع سرویس
`TypedStrings`	لیستی از رشته‌ی تایپ‌شده‌ی کاربر به ترتیب زمانی
`AcceptString`	رشته‌ای که در نهایت انتخاب شده است

```
import numpy as np
import pandas as pd

data = pd.read_json('../data/mrbilit_search.json')
test_data = pd.read_json('../data/test_data.json')
cities = pd.read_csv('../data/iran_cities.csv')
typo = pd.read_csv('../data/typo_char.csv')
```

- data: داده‌های جستجوی کاربر شامل نوع خدمات و رشته‌های تایپ‌شده و پذیرفته‌شده است.
- test_data: شامل اطلاعات تست است که برای ارزیابی عملکرد مدل استفاده می‌شود.
- cities: شامل نام شهرهای ایران برای استفاده در مدل.
- typo: شامل جدول جایگزینی حروف فارسی و انگلیسی برای تصحیح غلط‌های کاراکتری است.

۴. فیلتر کردن داده‌ها

در این قسمت، تنها رکوردهایی که مربوط به خدمات «اتوبوس» و «تاکسی» هستند، از میان داده‌های data انتخاب می‌شوند:

```
data = data[(data['ServiceType'] == 'bus') | (data['ServiceType'] == 'taxi')]
```

۵. انتخاب رشته طولانی‌ترین کلمه تایپ‌شده

برای افزایش دقت، از طولانی‌ترین رشته وارد شده توسط کاربر استفاده می‌شود:

```
def select_string(lst):
    return max(lst, key=len)

data['TypedStrings'] = data['TypedStrings'].apply(select_string)
```

تابع select_string طولانی‌ترین رشته در لیست ورودی‌ها را بازمی‌گرداند. این تابع بر روی ستون TypedStrings اعمال می‌شود و طولانی‌ترین رشته به عنوان متن اصلی جستجو ذخیره می‌شود.

۶. محاسبه فراوانی رشته‌های پذیرفته‌شده

محاسبه فراوانی نسبی رشته‌های پذیرفته‌شده برای هر عبارت تایپ‌شده:

```
accpet_str = data.groupby('TypedStrings')['AcceptString'].value_counts(normalize= True).unstack()
accpet_str.fillna(1e-10, inplace= True)
```

این قسمت داده‌ها را بر اساس TypedStrings گروه‌بندی کرده و سپس فراوانی نسبی هر رشته AcceptString را به صورت یک ماتریس ذخیره می‌کند. این اطلاعات برای ارزیابی احتمال تطابق رشته‌های تایپ‌شده با رشته‌های پذیرفته‌شده به کار می‌رود. مقادیر NaN با مقدار بسیار کوچکی پر شده‌اند تا مانع از بروز خطا شوند.

۷. آماده‌سازی داده مکان و محاسبه محبوبیت

این قسمت یک مجموعه منحصر به فرد از مکان‌ها و فراوانی نسبی آن‌ها را تولید می‌کند تا برای رتبه‌بندی نهایی استفاده شود.

```
location = pd.DataFrame(set(accpet_str.columns), columns= ['location'])
location.dropna(inplace= True)
accept_string_counts = data['AcceptString'].value_counts(normalize=True).reset_index()
accept_string_counts.columns = ['AcceptString', 'popularity']
location = location.merge(accept_string_counts, how='left', left_on='location', right_on='AcceptString')
location.drop(columns= 'AcceptString', inplace= True)
location['popularity'].fillna(location['popularity'].min(), inplace= True)
```

- location: شامل مکان‌های منحصر به فرد به عنوان مکان‌های قابل‌پذیرش است.
- محبوبیت: محبوبیت هر رشته AcceptString را بر اساس تعداد دفعات جستجو و پذیرش آن محاسبه می‌کند.

۸. محاسبه فاصله بین متن‌ها

این تابع فاصله Levenshtein بین text ورودی و هر مکان را در location محاسبه کرده و نتایج را نرمال‌سازی می‌کند.

```
from Levenshtein import distance
from sklearn.preprocessing import MinMaxScaler

def calculate_distance(text):
    distance_df = pd.DataFrame(location)
    scaler = MinMaxScaler()
    distance_df['distance'] = distance_df['location'].apply(lambda x: distance(x, text))
    distance_df['norm_distance'] = scaler.fit_transform(distance_df[['distance']])
    distance_df['norm_distance'] = 1 - distance_df['norm_distance']

    return distance_df
```

- distance_df: یک DataFrame برای نگهداری فاصله‌ها و فاصله‌های نرمال شده بین ورودی و هر مکان.

- نرمال‌سازی فاصله‌ها: مقادیر فاصله با استفاده از MinMaxScaler نرمال شده و سپس معکوس می‌شوند تا فاصله‌های کوتاه‌تر اهمیت بیشتری داشته باشند.

۹. تشخیص حروف انگلیسی و اصلاح غلط‌های تایپی

در این قسمت از کد، در صورت وجود حروف انگلیسی، این حروف به فارسی تبدیل می‌شوند.

```
import re

def contains_english(text):
    if bool(re.search(r'[A-Za-z]', text)):
        temp = []
        for i in text:
            temp.append(typo['FA'][typo['EN'] == i].iloc[0])
        return ''.join(temp)
    else:
        return text
```

تابع contains_english بررسی می‌کند که آیا متن شامل حروف انگلیسی است و در صورت وجود، آن‌ها را با کاراکترهای معادل فارسی جایگزین می‌کند.

۱۰. مدل برای پیشنهاد مکان‌ها

این تابع برای هر متن ورودی، فاصله‌های متنی را محاسبه کرده و بر اساس اطلاعات تاریخی و محبوبیت، بهترین مکان‌ها را پیشنهاد می‌دهد.

```
def model(text):
    text = contains_english(text)
    temp_df = calculate_distance(text)

    if text in accpet_str.index:
        history_df = pd.DataFrame(accpet_str.loc[text]).reset_index()
        history_df.columns = ['AcceptString', 'history']
        temp_df = temp_df.merge(history_df, how='left', left_on='location', right_on='AcceptString')
        temp_df.drop(columns= 'AcceptString', inplace= True)
        temp_df['score'] = np.log(temp_df['history']) + np.log(temp_df['norm_distance'] ** 2)
    else:
        temp_df['score'] = np.log(temp_df['norm_distance'] ** 2) + (temp_df['popularity'] ** 10)

    temp_df.sort_values(by= 'score', ascending= False, inplace= True)
    result = temp_df['location'][:5].tolist()
    return result
```

- فاصله‌های نرمال‌شده و تاریخچه جستجو: در صورتی که متن ورودی در داده‌های تاریخی وجود داشته باشد، امتیازات براساس سابقه جستجوها و فاصله نرمال‌شده محاسبه می‌شوند.

- محبوبیت: در صورتی که متن ورودی در تاریخچه موجود نباشد، امتیاز براساس فاصله نرمال‌شده و محبوبیت مکان‌ها محاسبه می‌شود.

- مرتب‌سازی و انتخاب: مکان‌ها براساس امتیاز مرتب شده و ۵ مکان برتر پیشنهاد می‌شوند.

۱۱. ارزیابی مدل با شباهت رتبه‌بندی (RBO)

در برخی از مسائل خصوصاً در هنگام پیاده‌سازی یک سیستم توصیه‌گر (Recommender system)نیاز خواهیم داشت تا دو لیست دارای اولویت را با همدیگر مقایسه کنیم. برای این‌که بحث خود را شهودی‌تر ادامه دهیم بگذارید مثالی بزنیم. فرض کنیم در جمع دوستان؛ از هرکدام آن‌ها خواسته‌ایم نام ۵ کاراکتر محبوب خود در فیلم‌های ارباب حلقه‌ها را به ترتیب علاقه بیان کنند. از طرفی به‌عنوان مثال نام ۵ کاراکتر محبوب خودمان نیز به شرح جدول زیر باشد:

رتبه	لیست من
1	Aragorn
2	Treebeard
3	Legolas
4	Gandalf the White
5	Sam

حال قصد داریم با مقایسه‌ی لیست خودمان با لیست هرکدام از دوستانمان بفهمیم که سلیقه‌ی کدام یک از دوستان به ما شبیه‌تر است؟ به‌عنوان مثال اگر لیست دوست ۱ نیز مطابق با جدول زیر باشد بتوانیم شباهت این دو لیست را به همدیگر پیدا کرده و با یک عدد بیان کنیم.

رتبه	لیست من	لیست دوست ۱
1	Aragorn	Gandalf the White
2	Treebeard	Aragorn
3	Legolas	Boromir
4	Gandalf the White	Legolas
5	Sam	Sam

چه‌گونه می‌توانیم این شباهت را بسنجیم؟ توجه داشته باشید که در چنین حالت‌هایی باید از معیاری استفاده کنیم که از «رتبه» نیز تاثیر پذیرد. به‌عنوان مثال اگر در لیست دوستمان «Aragorn» در رتبه‌ی سوم قرار بگیرد باید مقدار شباهت کمتری نسبت به حالت کنونی که در رتبه‌ی دوم است داشته باشد.

یکی از بهترین روش‌های محاسبه‌ی شباهت‌های این‌چنینی Rank-Biased Overlapیا به اختصار RBOاست. اگر مقاله‌ی معرفی‌کننده‌ی این روش را مطالعه کنید شاید به نظرتان خیلی پیچیده به نظر برسد اما در واقع تنها مراحلی که نیاز است برای محاسبه‌ی این معیار طی کنید به شرح زیر است:

به ازای هر رتبه، اشتراک (Intersection) بین دو لیست تا آن رتبه را پیدا کنید.

به تعداد این اشتراک، هم‌پوشانی (Overlap) گوییم.

اکنون هم‌پوشانی را تقسیم بر آن رتبه کنید تا مطابقت (Agreement) به دست آید.

حال مجموع مطابقت تمام رتبه‌ها را محاسبه کرده و بر طول لیست ارزیابی خود تقسیم کنید تا مقدار RBOبه دست آید.

اگر این مراحل را برای مثال پیشین خود اجرا کنیم، محاسبات آن به شرح زیر خواهد بود:

رتبه	لیست من	لیست دوست ۱	اشتراک تا این رتبه	هم‌پوشانی تا این رتبه	مطابقت تا این رتبه
1	Aragorn	Gandalf the White	{}	0	0
2	Treebeard	Aragorn	{Aragorn}	1	$\frac{1}{2} = 0.5$
3	Legolas	Boromir	{Aragorn}	1	$\frac{1}{3} = 0.33$
4	Gandalf the White	Legolas	{Aragorn, Legolas, Gandalf the White}	3	$\frac{3}{4} = 0.75$
5	Sam	Sam	{Aragorn, Legolas, Gandalf the White, Sam}	4	$\frac{4}{5} = 0.8$
				مجموع	2.3833
				شباهت (RBO)	$\frac{1}{5} \times 2.3833 = 0.4766$

بنابراین میزان شباهت بین لیست ما با دوست ۱ طبق این معیار برابر ۰,۴۷۶۶ شده است. حال می‌توانیم این مقدار را برای دوستان دیگر نیز محاسبه کنیم و آن دوستی که بیشترین مقدار RBO را داشته در واقع هم‌سلیقه‌ترین فرد با ماست.

```
from rbo import RankingSimilarity
act = ['A', 'B', 'C']
pred = ['B', 'A', 'D']
rbo = RankingSimilarity(act, pred).rbo()
print(rbo)
```

این امتیاز همپوشانی بین دو لیست رتبه‌بندی شده، act (واقعی) و pred (پیش‌بینی) را اندازه‌گیری می‌کند. مقدار نزدیک به ۱ به معنای شباهت زیاد است.

۱۲. تولید پیشنهادات برای داده‌های تست

تابع زیر، model را بر روی test_data اعمال می‌کند و پیشنهادات مدل برای هر عبارت تایپ‌شده را به‌عنوان یک خروجی ذخیره می‌کند:

```
submission = pd.DataFrame(test_data['Typed'].apply(model).tolist(), columns= ['Suggestion0', 'Suggestion1', 'Suggestion2', 'Suggestion3',
       'Suggestion4'])
submission.head()
```

۱۳. نتایج

rbo: ٪ ۵۵/۵۴