دانشگاه صنعتي اميركبير
(پلی تکنیک تهران)
دانشکده ریاضی و علوم کامپیوتر

پروژه پنج درسی هوش مصنوعی

ایلیا راوند

اردیبهشت۱۴۰۲

چکیده

در این پروژه ما با یک دادەست از اطلاعات واقعی بانک سر و کار داریم.

به ما اطلاعات کامل چندین حساب بانکی داده شده و در انتها به ما تقلبی بودن یا نبودن آنها گفته شده.

هدف این پروژه پیادەســـازی الگوریتم‌های مختلف هوش مصـــنوعی برای یافتن تقلبی بودن یا نبودن این اطلاعات است.

مراحل اصلی اینگونه هستند:

ابتدا ما باید داده‌ها را تمیز کنیم که بتوان روی آنها پیادەسازی الگوریتم‌ها را انجام داد.

و بعد kmeans پیادەسازی می‌کنیم.

در ادامه هم با الگوریتم‌های درون کتابخانه lazy predict پیش‌بینی می‌کنیم بهترین الگوریتم چیست.

و بعد هم برای مثال‌های بیشتر الگوریتم‌های knn و mlp را پیادەسازی می‌کنیم.

واژه‌های کلیدی:

کشف تقلب در حساب بانکی

Predictlazy

Kmeans

خوشه‌بندی

تحلیل داده

فهرست مطالب **صفحه**

چکیده ..... أ
۱ مقدمه ..... **ERROR! BOOKMARK NOT DEFINED.**
۲ پیش‌پردازش داده 12
۳ خوشه‌بندی 12
۴ PREDICTLAZY 12
۵ KNN-MLP 1212
منابع و مراجع 12

مقدمه

مراحل اصلی کار عبارتند از:

۱ .تمیز کردن داده برای استفاده مناسب

۲ .پیاده‌سازی خوشه‌بندیkmeans

۳ .پیاده‌سازی پیش‌بینیlazy

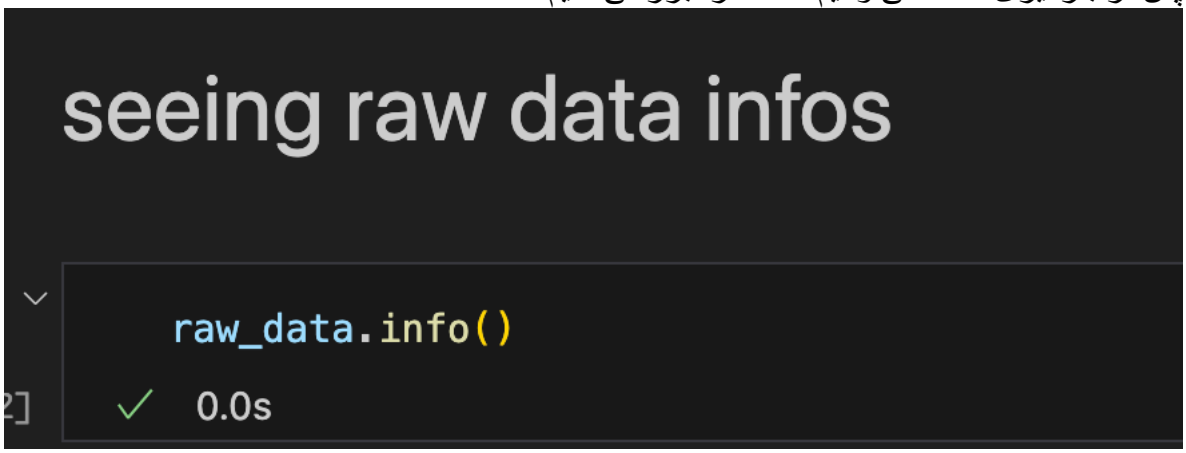
۴ .پیاده‌سازی الگوریتمknn

هر بخش به طور کامل توضیح داده خواهد شد.

پیش‌پردازش داده:

در ابتدا، فایل را از فرمت CSV بارگیری می‌کنیم.

پس از بارگیری داده، می‌توانیم داده‌ها را بررسی کنیم.



```
Data columns (total 32 columns):
 #   Column                            Non-Null Count    Dtype
---  ------                            --------------    -----
 0   fraud_bool                        1000000 non-null  int64
 1   income                            1000000 non-null  float64
 2   name_email_similarity             1000000 non-null  float64
 3   prev_address_months_count         1000000 non-null  int64
 4   current_address_months_count      1000000 non-null  int64
 5   customer_age                      1000000 non-null  int64
 6   days_since_request                1000000 non-null  float64
 7   intended_balcon_amount            1000000 non-null  float64
 8   payment_type                      1000000 non-null  object
 9   zip_count_4w                      1000000 non-null  int64
 10  velocity_6h                       1000000 non-null  float64
 11  velocity_24h                      1000000 non-null  float64
 12  velocity_4w                       1000000 non-null  float64
 13  bank_branch_count_8w              1000000 non-null  int64
 14  date_of_birth_distinct_emails_4w  1000000 non-null  int64
 15  employment_status                 1000000 non-null  object
 16  credit_risk_score                 1000000 non-null  int64
 17  email_is_free                     1000000 non-null  int64
 18  housing_status                    1000000 non-null  object
 19  phone_home_valid                  1000000 non-null  int64
...
 30  device_fraud_count                1000000 non-null  int64
 31  month                             1000000 non-null  int64
dtypes: float64(9), int64(18), object(5)
memory usage: 244.1+ MB
```

متوجه می‌شویم که داده‌ها از وضعیت خوبی برخوردار هستند و مقادیر null ندارند. اما مشکل اصلی ما در پنج ستونی است که داده‌های آنها از نوع دسته‌ای (categorical) هستند. برای حل این مشکل، می‌توانیم روش‌هایی مانند رمزگذاری برچسب (label encoding) رای تبدیل داده‌های دسته‌ای به عددی استفاده کنیم.

switching them with numericals

```
# data[categorical_columns] = data[categorical_columns].apply(lambda x: x.astype('category').cat.code
# print(data[categorical_columns])


# or we can do this with labelencoder
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
data[categorical_columns] = data[categorical_columns].apply(lambda x: labelencoder.fit_transform(x))

data[categorical_columns]
```

✓ 0.3s

در این دادەتابع، هدف به وضوح تقلب بودن یا نبودن دادەها مربوط است. از آنجایی که تقلبی بودن تمامی دادەها یا بیشترین آنها با مقدار ۱ درصد کل دادەها مشخص شده است، ما نیازمند به تعادل دادەها هستیم. این کار را به صورت زیر انجام می‌دهیم.

making more samples

```
# using smote to balance the data
from imblearn.over_sampling import SMOTE
smote = SMOTE()

print(y.value_counts())


X_smote, y_smote = smote.fit_resample(X, y)

X = X_smote
y = y_smote

print(y.value_counts())
```

✓ 1.3s

```
fraud_bool
0    988971
1     11029
Name: count, dtype: int64
fraud_bool
0    988971
1    988971
Name: count, dtype: int64
```

در نهایت برای کاهش حجم پردازش‌ها، داده‌ها را مقیاس می‌دهیم.

scaling x

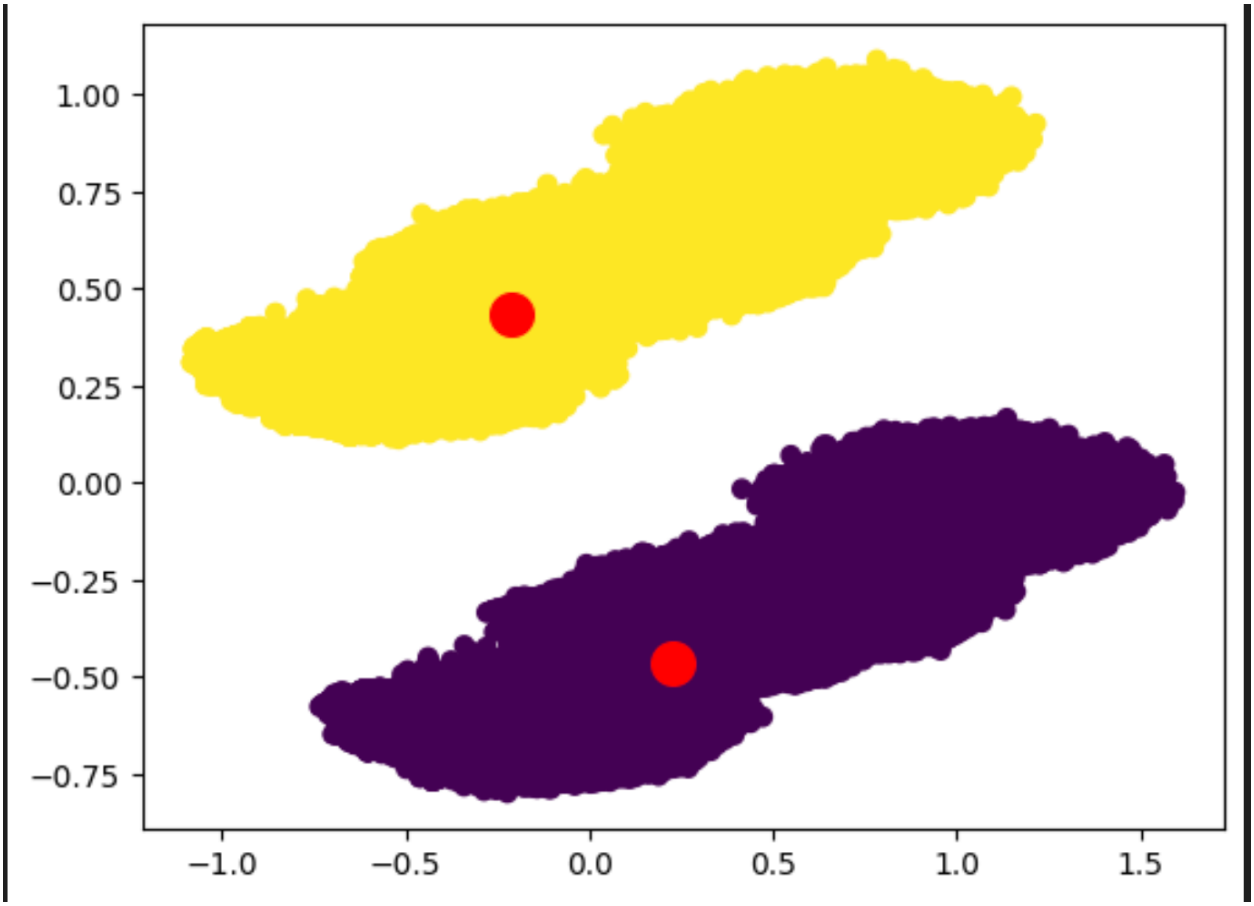
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

✓ 0.1s

خوشه‌بندی:

در این بخش، الگوریتم Kmeans را پیاده‌سازی می‌کنیم. برای این کار، ابتدا داده‌ها را به دو بخش تقسیم می‌کنیم و سپس خود الگوریتم Kmeans را پیاده‌سازی می‌کنیم.

پس از اجرای الگوریتم Kmeans، مشاهده می‌شود که این الگوریتم برای این مجموعه داده بسیار موثر نیست زیرا با دقت ۵۰٪ یا حتی با همپوشانی برچسب‌ها هم هماهنگ نیست.



```python
# measure k-means performance f1 and accuracy and precision and recall
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score

f1 = f1_score(y, Kmeans.labels_, average='weighted')
accuracy = accuracy_score(y, Kmeans.labels_)
precision = precision_score(y, Kmeans.labels_, average='weighted')
recall = recall_score(y, Kmeans.labels_, average='weighted')

print(f1, accuracy, precision, recall)
```

✓ 1.0s

```
0.5439628818487401 0.5440745987496094 0.5441178295042048 0.5440745987496094
```

Predictlazy:

چون داده بسیار زیاد است، این کد برای اجرا زیاد زمان می‌برد و برای رفع این مشکل بخشی از داده‌ها فقط بر روی 2.5 درصد از داده‌ها اجرا شده است. ابتدا داده‌ها را به دو بخش آزمون و آموزش تقسیم می‌کنیم و سپس الگوریتم PCA را برای کاهش ابعاد اجرا می‌کنیم. در نهایت، الگوریتم پیش‌بینی lazy را اجرا می‌کنیم.

```
from sklearn.model_selection import train_test_split

Xs, _, ys, _ = train_test_split(X, y, test_size=0.975)
```

✓ 0.2s

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(Xs, ys, test_size=0.3)
```

✓ 0.0s

```
pca = PCA(n_components=2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

✓ 3.6s

applying lazypredict

+ Code + Markdown

```
# Use lazypredict to train and evaluate various models
from lazypredict.Supervised import LazyClassifier


clf = LazyClassifier(verbose=0, ignore_warnings=True, custom_metric=None)

models, predictions = clf.fit(X_train, X_test, y_train, y_test)

print(models)
```

✓ 2m 42.3s

```
100%|          | 29/29 [02:42<00:00,  5.59s/it]
                            Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
XGBClassifier                   0.83              0.83     0.83      0.83
LGBMClassifier                  0.83              0.83     0.83      0.83
LabelPropagation                0.82              0.82     0.82      0.82
AdaBoostClassifier              0.82              0.82     0.82      0.82
LabelSpreading                  0.82              0.82     0.82      0.82
SVC                             0.82              0.82     0.82      0.82
RandomForestClassifier          0.82              0.82     0.82      0.82
KNeighborsClassifier            0.81              0.81     0.81      0.81
ExtraTreesClassifier            0.81              0.81     0.81      0.81
SGDClassifier                   0.80              0.80     0.80      0.80
BaggingClassifier               0.80              0.80     0.80      0.80
BernoulliNB                     0.80              0.80     0.80      0.80
LogisticRegression              0.80              0.80     0.80      0.80
NuSVC                           0.80              0.80     0.80      0.80
CalibratedClassifierCV          0.80              0.80     0.80      0.80
GaussianNB                      0.79              0.79     0.79      0.79
LinearSVC                       0.79              0.79     0.79      0.79
Perceptron                      0.79              0.79     0.79      0.79
NearestCentroid                 0.79              0.79     0.79      0.79
RidgeClassifier                 0.79              0.79     0.79      0.79
RidgeClassifierCV               0.79              0.79     0.79      0.79
LinearDiscriminantAnalysis      0.79              0.79     0.79      0.79
QuadraticDiscriminantAnalysis   0.79              0.79     0.79      0.79
...
ExtraTreeClassifier             0.02
DecisionTreeClassifier          0.12
PassiveAggressiveClassifier     0.02
DummyClassifier                 0.01
```

نتیجه می‌گیریم که روش کار ما با داده‌ها درســـت بوده و الگوریتم‌های توجه به خوبی با دقت بالا عمل کرده‌اند.

Knn-mlp:

در ادامه، الگوریتم‌های KNN و MLP را پیاده‌سازی می‌کنیم.

applying knn

```
# applying knn
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report

knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print(classification_report(y_test, y_pred))
```

6] ✓ 7.6s

	precision	recall	f1-score	support
0	0.83	0.81	0.82	296541
1	0.81	0.84	0.82	296842
accuracy			0.82	593383
macro avg	0.82	0.82	0.82	593383
weighted avg	0.82	0.82	0.82	593383

Crossvalidation using knn:

cross validation

```
# cross validation with Nearest centroid
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(n_neighbors=5)
scores = cross_val_score(clf, X2d, y, cv=5)
print(scores)
print(scores.mean())
```

✓ 22.9s

```
[0.80477465 0.82345566 0.80172806 0.81437758 0.79298411]
0.8074640139588108
```

با توجه به اینکه ما در این بخش از داده‌ها از ارقام یکســـانی برخورداریم، می‌توانیم نتیجه بگیریم که الگوریتم KNN برای این مجموعه داده مناسب است.

Mlp:

applying neuralnetwork in seperate format

```
# applying noralnetwork
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
mlp.fit(X_train, y_train)
y_pred = mlp.predict(X_test)
print(classification_report(y_test, y_pred))
```

✓ 46.5s

```
              precision    recall  f1-score   support

           0       0.83      0.83      0.83    296541
           1       0.83      0.83      0.83    296842

    accuracy                           0.83    593383
   macro avg       0.83      0.83      0.83    593383
weighted avg       0.83      0.83      0.83    593383
```

منابع و مراجع

https://www.kaggle.com/datasets/sgpjesus/bank-account-fraud-dataset-neurips-2022

There are no sources in the current document.