Chieu

December 16, 2025

[48]:
```
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
```

Câu 1. (a)

Xóa vùng định danh (ID)

[11]:
```
df = pd.read_csv('bank-data.csv')
df = df.drop(columns=['id'])
```

Tiền xử lý dữ liệu.

[12]:
```
y = df['pep']
X = df.drop('pep', axis=1)

X = pd.get_dummies(X)

print(X.columns)
print(X.shape)
```

```
Index(['age', 'income', 'children', 'sex_FEMALE', 'sex_MALE',
       'region_INNER_CITY', 'region_RURAL', 'region_SUBURBAN', 'region_TOWN',
       'married_NO', 'married_YES', 'car_NO', 'car_YES', 'save_act_NO',
       'save_act_YES', 'current_act_NO', 'current_act_YES', 'mortgage_NO',
       'mortgage_YES'],
      dtype='object')
(600, 19)
```

Chuẩn hóa dữ liệu.

[13]:
```
scaler = StandardScaler()

cols = X.columns
```

```
X_scaled_array = scaler.fit_transform(X)

X_scaled = pd.DataFrame(X_scaled_array, columns=cols)

print(X_scaled.head())
```

```
        age    income  children  sex_FEMALE  sex_MALE  region_INNER_CITY  \
0  0.388887 -0.774168 -0.011049         1.0      -1.0           1.109272
1 -0.166170  0.198706  1.883121        -1.0       1.0          -0.901493
2  0.597034 -0.849474 -0.958135         1.0      -1.0           1.109272
3 -1.345667 -0.554643  1.883121         1.0      -1.0          -0.901493
4  1.013327  1.788562 -0.958135         1.0      -1.0          -0.901493

   region_RURAL  region_SUBURBAN  region_TOWN  married_NO  married_YES  \
0     -0.436436        -0.339473    -0.636516    1.393261    -1.393261
1     -0.436436        -0.339473     1.571053   -0.717741     0.717741
2     -0.436436        -0.339473    -0.636516   -0.717741     0.717741
3     -0.436436        -0.339473     1.571053   -0.717741     0.717741
4      2.291288        -0.339473    -0.636516   -0.717741     0.717741

     car_NO   car_YES  save_act_NO  save_act_YES  current_act_NO  \
0  0.986754 -0.986754     1.491914     -1.491914        1.771421
1 -1.013423  1.013423     1.491914     -1.491914       -0.564519
2 -1.013423  1.013423    -0.670280      0.670280       -0.564519
3  0.986754 -0.986754     1.491914     -1.491914       -0.564519
4  0.986754 -0.986754    -0.670280      0.670280        1.771421

   current_act_YES  mortgage_NO  mortgage_YES
0        -1.771421     0.731113     -0.731113
1         0.564519    -1.367777      1.367777
2         0.564519     0.731113     -0.731113
3         0.564519     0.731113     -0.731113
4        -1.771421     0.731113     -0.731113
```

Chia data train/test

[16]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,␣
↪random_state=42)
```

Train model KNN

[17]:
```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

[17]:
```
KNeighborsClassifier()
```

[18]:
```
y_pred = knn.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

          NO       0.64      0.70      0.67        93
         YES       0.65      0.59      0.61        87

    accuracy                           0.64       180
   macro avg       0.64      0.64      0.64       180
weighted avg       0.64      0.64      0.64       180
```

Test nhiều k để tìm tối ưu.

[19]:
```
for k in range(1, 20, 2):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    acc = (y_pred == y_test).mean()
    print(f"k={k}: Accuracy = {acc:.4f}")
```

```
k=1: Accuracy = 0.6000
k=3: Accuracy = 0.6056
k=5: Accuracy = 0.6444
k=7: Accuracy = 0.6611
k=9: Accuracy = 0.6056
k=11: Accuracy = 0.6444
k=13: Accuracy = 0.6500
k=15: Accuracy = 0.6611
k=17: Accuracy = 0.6444
k=19: Accuracy = 0.6556
```

$k = 7$ có độ chính xác cao nhất, thử lấy chi tiết cho $k = 7$

[21]:
```
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Confusion Matrix:
[[69 24]
 [37 50]]
```

```
Classification Report:
              precision    recall  f1-score   support

          NO       0.65      0.74      0.69        93
         YES       0.68      0.57      0.62        87

    accuracy                           0.66       180
   macro avg       0.66      0.66      0.66       180
weighted avg       0.66      0.66      0.66       180
```

10-Fold Cross Validation

[23]:
```
for k in range(1, 20, 2):
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_scaled, y, cv=10)
    print(f"k={k}: Accuracy = {scores.mean():.4f} (+/- {scores.std():.4f})")
```

```
k=1: Accuracy = 0.6283 (+/- 0.0500)
k=3: Accuracy = 0.6250 (+/- 0.0318)
k=5: Accuracy = 0.6200 (+/- 0.0488)
k=7: Accuracy = 0.6217 (+/- 0.0582)
k=9: Accuracy = 0.6317 (+/- 0.0474)
k=11: Accuracy = 0.6400 (+/- 0.0291)
k=13: Accuracy = 0.6433 (+/- 0.0429)
k=15: Accuracy = 0.6500 (+/- 0.0316)
k=17: Accuracy = 0.6717 (+/- 0.0582)
k=19: Accuracy = 0.6617 (+/- 0.0563)
```

Model tốt nhất: KNN với k=17, đánh giá bằng 10-Fold CV, đạt Accuracy = 67.17%

Có thể cân nhắc k=15 (Accuracy=65%, std=0.0316) nếu muốn ổn định hơn.

Câu 1. (b)

[26]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.1, random_state=42, stratify=y
)
```

[27]:
```
print("Tỷ lệ trong Train:")
print(y_train.value_counts(normalize=True))

print("\nTỷ lệ trong Test:")
print(y_test.value_counts(normalize=True))
```

```
Tỷ lệ trong Train:
pep
NO     0.542593
YES    0.457407
Name: proportion, dtype: float64
```

```
Tỷ lệ trong Test:
pep
NO     0.55
YES    0.45
Name: proportion, dtype: float64
```

OK

TÌM K TỐT NHẤT TRÊN TRAIN SET (10-Fold CV)

[29]:
```python
for k in range(1, 30, 2):
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10)
    print(f"k={k}: Accuracy = {scores.mean():.4f}")
```

```
k=1: Accuracy = 0.6593
k=3: Accuracy = 0.6241
k=5: Accuracy = 0.6148
k=7: Accuracy = 0.6278
k=9: Accuracy = 0.6278
k=11: Accuracy = 0.6352
k=13: Accuracy = 0.6241
k=15: Accuracy = 0.6444
k=17: Accuracy = 0.6685
k=19: Accuracy = 0.6704
k=21: Accuracy = 0.6556
k=23: Accuracy = 0.6648
k=25: Accuracy = 0.6630
k=27: Accuracy = 0.6611
k=29: Accuracy = 0.6519
```

Câu b - Model tốt nhất: KNN với k=19, Accuracy = 67.04% (đánh giá bằng 10-Fold CV trên Train set)

Câu 1. (c)

[32]:
```python
knn_best = KNeighborsClassifier(n_neighbors=19)
knn_best.fit(X_train, y_train)


#Predict trên test set
y_pred = knn_best.predict(X_test)

# Đánh giá
print("=== KẾT QUẢ TRÊN TEST SET ===")
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
=== KẾT QUẢ TRÊN TEST SET ===

Confusion Matrix:
[[25  8]
 [14 13]]

Classification Report:
              precision    recall  f1-score   support

          NO       0.64      0.76      0.69        33
         YES       0.62      0.48      0.54        27

    accuracy                           0.63        60
   macro avg       0.63      0.62      0.62        60
weighted avg       0.63      0.63      0.63        60
```

0.1 Nhận xét:

1. **Accuracy = 63%** - thấp hơn lúc đánh giá trên Train (67%)
   - Điều này bình thường, vì Test set là dữ liệu model chưa thấy
2. **Model dự đoán NO tốt hơn YES:**
   - Recall NO = 76% (tìm được 25/33 người NO)
   - Recall YES = 48% (chỉ tìm được 13/27 người YES)
3. **14 người YES bị dự đoán sai thành NO** → Model có xu hướng dự đoán NO nhiều hơn

Kết luận

Model KNN với **k=19** đạt **Accuracy 63%** trên Test set. Model phân lớp **NO** tốt hơn **YES**. Kết quả chấp nhận được nhưng chưa thật sự tốt, có thể do dữ liệu ít (600 mẫu) hoặc các features chưa đủ mạnh để phân biệt 2 lớp.

Câu 2.

[36]:
```
df = pd.read_excel('Collected_Hr_data_performances.xls',␣
  ↪sheet_name='Full_Employee_Per')
print(f"Shape: {df.shape}")
```

```
Shape: (1200, 21)
```

Check phân bố class hiện tại

[37]:
```
print("PerformanceRating hiện tại:")
print(df['PerformanceRating'].value_counts().sort_index())
```

```
PerformanceRating hiện tại:
PerformanceRating
1     11
2    183
3    874
4    122
```

```
5     10
Name: count, dtype: int64
```

Gộp class

[ ]:
```
# Gộp rating 1 -> 2
df.loc[df['PerformanceRating'] == 1, 'PerformanceRating'] = 2
df.loc[df['PerformanceResult'] == 'Does not Meet Minimum', 'PerformanceResult']␣
 ↪= 'Meet Expectation'

# Gộp rating 5-> 4
df.loc[df['PerformanceRating'] == 5, 'PerformanceRating'] = 4
df.loc[df['PerformanceResult'] == 'Outstanding', 'PerformanceResult'] = 'Exceed␣
 ↪Expectation'

# Recheck
print("Sau khi gộp:")
print(df['PerformanceRating'].value_counts().sort_index())
```

```
Sau khi gộp:
PerformanceRating
2    194
3    874
4    132
Name: count, dtype: int64
```

[39]:
```
# Xem các cột
print(df.columns.tolist())
```

```
['Number', 'Age', 'Age.1', 'Gender', 'MaritalStatus', 'EducationLevel',
'EducationLevel.1', 'EducationBackground', 'JobRole', 'EnvironmentSatisfaction',
'EnvironmentSatisfaction.1', 'RelationshipSatisfaction',
'RelationshipSatisfaction.1', 'WorkLifeBalance', 'WorkLifeBalance.1',
'TotalWorkExperienceInYears', 'TotalWorkExperienceInYears.1',
'ExperienceYearsInCurrentRole', 'ExperienceYearsInCurrentRole.1',
'PerformanceRating', 'PerformanceResult']
```

[40]:
```
# Bỏ cột ID và cột trùng (.1)
# Giữ các cột số gốc + categorical không có phiên bản số

features = ['Age', 'Gender', 'MaritalStatus', 'EducationLevel',
            'EducationBackground', 'JobRole', 'EnvironmentSatisfaction',
            'RelationshipSatisfaction', 'WorkLifeBalance',
            'TotalWorkExperienceInYears', 'ExperienceYearsInCurrentRole']

X = df[features]
y = df['PerformanceRating']

print(f"X shape: {X.shape}")
```

```
print(f"Features: {list(X.columns)}")
```

```
X shape: (1200, 11)
Features: ['Age', 'Gender', 'MaritalStatus', 'EducationLevel',
'EducationBackground', 'JobRole', 'EnvironmentSatisfaction',
'RelationshipSatisfaction', 'WorkLifeBalance', 'TotalWorkExperienceInYears',
'ExperienceYearsInCurrentRole']
```

One Hot Encoding

[42]:
```
X = pd.get_dummies(X)
print(f"Sau encode: {X.shape}")
print(f"Các cột: {list(X.columns)}")
```

```
Sau encode: (1200, 37)
Các cột: ['Age', 'EducationLevel', 'EnvironmentSatisfaction',
'RelationshipSatisfaction', 'WorkLifeBalance', 'TotalWorkExperienceInYears',
'ExperienceYearsInCurrentRole', 'Gender_Female', 'Gender_Male',
'MaritalStatus_Divorced', 'MaritalStatus_Married', 'MaritalStatus_Single',
'EducationBackground_Human Resources', 'EducationBackground_Life Sciences',
'EducationBackground_Marketing', 'EducationBackground_Medical',
'EducationBackground_Other', 'EducationBackground_Technical Degree',
'JobRole_Business Analyst', 'JobRole_Data Scientist', 'JobRole_Delivery
Manager', 'JobRole_Developer', 'JobRole_Finance Manager', 'JobRole_Healthcare
Representative', 'JobRole_Human Resources', 'JobRole_Laboratory Technician',
'JobRole_Manager', 'JobRole_Manager R&D', 'JobRole_Manufacturing Director',
'JobRole_Research Director', 'JobRole_Research Scientist', 'JobRole_Sales
Executive', 'JobRole_Sales Representative', 'JobRole_Senior Developer',
'JobRole_Senior Manager R&D', 'JobRole_Technical Architect', 'JobRole_Technical
Lead']
```

[43]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Tách 1000 train, 200 test

[44]:
```
# 200/1200  0.167 (khoảng 16.7%)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=200, random_state=42, stratify=y
)
print(f"Train: {len(X_train)} mẫu")
print(f"Test: {len(X_test)} mẫu")
print(f"\nTỷ lệ class trong Train:")
print(y_train.value_counts(normalize=True))
print(f"\nTỷ lệ class trong Test:")
print(y_test.value_counts(normalize=True))
```

```
Train: 1000 mẫu
Test: 200 mẫu
```

```
Tỷ lệ class trong Train:
PerformanceRating
3    0.728
2    0.162
4    0.110
Name: proportion, dtype: float64

Tỷ lệ class trong Test:
PerformanceRating
3    0.73
2    0.16
4    0.11
Name: proportion, dtype: float64
```

Train KNN và tìm k tối ưu nhất.

[45]:
```python
for k in range(1, 20, 2):
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10)
    print(f"k={k}: Accuracy = {scores.mean():.4f}")
```

```
k=1: Accuracy = 0.6440
k=3: Accuracy = 0.6430
k=5: Accuracy = 0.6940
k=7: Accuracy = 0.7160
k=9: Accuracy = 0.7240
k=11: Accuracy = 0.7340
k=13: Accuracy = 0.7340
k=15: Accuracy = 0.7320
k=17: Accuracy = 0.7360
k=19: Accuracy = 0.7320
```

$k = 17$ là giá trị tối ưu nhất. Đánh giá $k = 17$.

[46]:
```python
knn = KNeighborsClassifier(n_neighbors=17)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("=== KNN (k=17) TRÊN TEST SET ===")
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
=== KNN (k=17) TRÊN TEST SET ===

Confusion Matrix:
[[  3  28   1]
 [  0 146   0]
 [  0  22   0]]
```

```
Classification Report:
              precision    recall  f1-score   support

           2       1.00      0.09      0.17        32
           3       0.74      1.00      0.85       146
           4       0.00      0.00      0.00        22

    accuracy                           0.74       200
   macro avg       0.58      0.36      0.34       200
weighted avg       0.70      0.74      0.65       200
```

Nhận xét: Model gần như chỉ dự đoán class 3 (vì class 3 chiếm 73% dữ liệu). Đây là vấn đề class imbalance - model "lười", cứ đoán class đông nhất là xong.

Thử Naive Bayes

[49]:
```
nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)
print("=== NAÏVE BAYES TRÊN TEST SET ===")
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred_nb))
print("\nClassification Report:")
print(classification_report(y_test, y_pred_nb))
```

```
=== NAÏVE BAYES TRÊN TEST SET ===

Confusion Matrix:
[[29  0  3]
 [48  4 94]
 [12  0 10]]

Classification Report:
              precision    recall  f1-score   support

           2       0.33      0.91      0.48        32
           3       1.00      0.03      0.05       146
           4       0.09      0.45      0.16        22

    accuracy                           0.21       200
   macro avg       0.47      0.46      0.23       200
weighted avg       0.79      0.21      0.13       200
```

0.2 Kết luận Câu 2:

0.2.1 KNN (k=17):

- **Accuracy: 74%** trên Test set

- Model có xu hướng dự đoán class 3 (Meet Expectation) cho hầu hết các trường hợp
- Recall class 2 và class 4 rất thấp (0.09 và 0.00)

0.2.2 Naïve Bayes:

- **Accuracy: 21%** trên Test set

- Model phân biệt được cả 3 class nhưng accuracy thấp hơn nhiều
- Recall class 2 (0.91) và class 4 (0.45) cao hơn KNN

0.2.3 Nhận xét chung:

- **KNN hoạt động tốt hơn** về mặt accuracy tổng thể
- Cả 2 thuật toán đều gặp khó khăn do dữ liệu **mất cân bằng** (class 3 chiếm 73%)
- Để cải thiện, có thể áp dụng các kỹ thuật xử lý class imbalance như SMOTE, undersampling, hoặc thay đổi class weights