mini4

May 22, 2024

[15]:
```python
import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score, StratifiedKFold,␣
 ↪train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score
```

[16]:
```python
# 1. Load and Preprocess Data
df_vote = pd.read_csv("/Users/rouren/Desktop/24S ML/HW/mini4/vote.csv")
df_work = pd.read_csv("/Users/rouren/Desktop/24S ML/HW/mini4/work.csv")

print("Data types of variables in df_vote:")
print(df_vote.dtypes)

print("\nData types of variables in df_work:")
print(df_work.dtypes)
```

```
Data types of variables in df_vote:
prtage       int64
pesex       object
ptdtrace    object
pehspnon    object
prcitshp    object
peeduca     object
vote        object
dtype: object

Data types of variables in df_work:
prtage       int64
pesex       object
ptdtrace    object
pehspnon    object
prcitshp    object
peeduca     object
work        object
dtype: object
```

[17]:
```python
# 2(a) Convert 'vote' and 'work' variables to binary form
df_vote['vote'] = df_vote['vote'].map({'did not vote': 0, 'vote': 1})
df_work['work'] = df_work['work'].map({'not flexible': 0, 'flexible': 1})

df_work.dropna(subset=['work'], inplace=True)

print("Updated df_vote dataset:")
print(df_vote.head())

print("\nUpdated df_work dataset:")
print(df_work.head())
```

```
Updated df_vote dataset:
   prtage   pesex    ptdtrace      pehspnon                        prcitshp  \
0      19  FEMALE  White Only  NON-HISPANIC      NATIVE, BORN IN THE UNITED
1      35    MALE  White Only  NON-HISPANIC      NATIVE, BORN IN THE UNITED
2      48    MALE  White Only      HISPANIC  FOREIGN BORN, U.S. CITIZEN BY
3      55    MALE  White Only  NON-HISPANIC      NATIVE, BORN IN THE UNITED
4      25  FEMALE  White Only  NON-HISPANIC      NATIVE, BORN IN THE UNITED

                        peeduca  vote
0    SOME COLLEGE BUT NO DEGREE     1
1  MASTER'S DEGREE (EX: MA, MS,     1
2              5TH OR 6TH GRADE     1
3             BACHELOR'S DEGREE     0
4    SOME COLLEGE BUT NO DEGREE     1

Updated df_work dataset:
   prtage   pesex    ptdtrace      pehspnon                     prcitshp  \
0      35    MALE  White Only  NON-HISPANIC  NATIVE, BORN IN THE UNITED
1      41    MALE  White Only  NON-HISPANIC  NATIVE, BORN IN THE UNITED
2      53    MALE  White Only  NON-HISPANIC  NATIVE, BORN IN THE UNITED
3      21  FEMALE  White Only  NON-HISPANIC  NATIVE, BORN IN THE UNITED
4      45  FEMALE  Black Only  NON-HISPANIC  NATIVE, BORN IN THE UNITED

                       peeduca  work
0  HIGH SCHOOL GRAD-DIPLOMA OR     1
1  HIGH SCHOOL GRAD-DIPLOMA OR     1
2            BACHELOR'S DEGREE     0
3   SOME COLLEGE BUT NO DEGREE     1
4   SOME COLLEGE BUT NO DEGREE     1
```

[18]:
```python
# 2(b) Compare categories in categorical variables
categorical_variables = ['pesex', 'ptdtrace', 'pehspnon', 'prcitshp', 'peeduca']

for var in categorical_variables:
    unique_categories_vote = set(df_vote[var].unique())
    unique_categories_work = set(df_work[var].unique())
```

```
    if unique_categories_vote != unique_categories_work:
        print(f"Discrepancy found in variable '{var}':")
        print(f"Categories in df_vote: {unique_categories_vote}")
        print(f"Categories in df_work: {unique_categories_work}")
    else:
        print(f"No discrepancy found in variable '{var}'.")
```

```
No discrepancy found in variable 'pesex'.
Discrepancy found in variable 'ptdtrace':
Categories in df_vote: {'W-B-AI', '2 or 3 Races', 'White Only', 'Black-AI',
'W-A-HP', 'Black-Asian', 'White-AI', 'Black Only', 'Asian Only',
'Hawaiian/Pacific Islander Only', 'American Indian, Alaskan', 'White-Hawaiian',
'White-Asian', 'Asian-HP', 'White-Black'}
Categories in df_work: {'2 or 3 Races', 'White Only', 'Black-AI', '4 or 5
Races', 'White-AI', 'Black Only', 'Asian Only', 'Hawaiian/Pacific Islander
Only', 'American Indian, Alaskan', 'White-Hawaiian', 'White-Asian', 'Asian-HP',
'White-Black'}
No discrepancy found in variable 'pehspnon'.
Discrepancy found in variable 'prcitshp':
Categories in df_vote: {'NATIVE, BORN IN THE UNITED', 'NATIVE, BORN IN PUERTO
RICO OR', 'FOREIGN BORN, U.S. CITIZEN BY', 'NATIVE, BORN ABROAD OF'}
Categories in df_work: {'NATIVE, BORN IN THE UNITED', 'FOREIGN BORN, U.S.
CITIZEN BY', 'NATIVE, BORN ABROAD OF', 'FOREIGN BORN, NOT A CITIZEN OF',
'NATIVE, BORN IN PUERTO RICO OR'}
No discrepancy found in variable 'peeduca'.
```

[19]:

```
# 2(c) Convert categorical variables using one-hot encoding
df_vote_encoded = pd.get_dummies(df_vote, columns=categorical_variables,␣
 ↪drop_first=True)
df_work_encoded = pd.get_dummies(df_work, columns=categorical_variables,␣
 ↪drop_first=True)
```

[20]:

```
# 2(d) Ensure the same structure by adding missing columns
for col in df_work_encoded.columns:
    if col not in df_vote_encoded.columns:
        df_vote_encoded[col] = 0

for col in df_vote_encoded.columns:
    if col not in df_work_encoded.columns:
        df_work_encoded[col] = 0

df_vote_encoded = df_vote_encoded.reindex(columns=sorted(df_vote_encoded.
 ↪columns), fill_value=0)
df_work_encoded = df_work_encoded.reindex(columns=sorted(df_work_encoded.
 ↪columns), fill_value=0)
```

```
scaler = StandardScaler()
X_vote_scaled = scaler.fit_transform(df_vote_encoded.drop(columns=['vote']))
X_work_scaled = scaler.fit_transform(df_work_encoded.drop(columns=['work']))
```

[21]:

```
# 2(e)
# Scaling the data before fitting an SVM classifier is essential because SVMs␣
↪are sensitive to the scale of features. Features with larger scales can␣
↪dominate the optimization process, leading to biased results. Scaling␣
↪ensures that all features contribute equally, facilitates faster␣
↪convergence, and maintains accurate distance metrics calculation, ultimately␣
↪improving model performance and generalization.
```

[22]:

```
# 3. Train SVM Classifier

X = df_work_encoded.drop(columns=['work'])
y = df_work_encoded['work']

# Scale the features
X_scaled = scaler.fit_transform(X)

C_values = [0.1, 1, 5, 10]
kernels = ['linear', 'poly', 'sigmoid']

cv = StratifiedKFold(n_splits=5)

results = {}

# Train and evaluate models
for C in C_values:
    for kernel in kernels:
        svm = SVC(C=C, kernel=kernel)
        cv_scores = cross_val_score(svm, X_scaled, y, cv=cv, scoring='accuracy')
        error_rate = 1 - cv_scores.mean()
        results[(C, kernel)] = error_rate
        print(f"C: {C}, Kernel: {kernel}, Cross-Validation Error Rate:␣
↪{error_rate:.4f}")

print("\nCross-Validation Error Rates for all models:")
for params, error_rate in results.items():
    print(f"C: {params[0]}, Kernel: {params[1]} -> Error Rate: {error_rate:.
↪4f}")
```

```
C: 0.1, Kernel: linear, Cross-Validation Error Rate: 0.1408
C: 0.1, Kernel: poly, Cross-Validation Error Rate: 0.3616
C: 0.1, Kernel: sigmoid, Cross-Validation Error Rate: 0.1424
C: 1, Kernel: linear, Cross-Validation Error Rate: 0.1424
C: 1, Kernel: poly, Cross-Validation Error Rate: 0.1728
C: 1, Kernel: sigmoid, Cross-Validation Error Rate: 0.1616
```

```
C: 5, Kernel: linear, Cross-Validation Error Rate: 0.1424
C: 5, Kernel: poly, Cross-Validation Error Rate: 0.1502
C: 5, Kernel: sigmoid, Cross-Validation Error Rate: 0.1772
C: 10, Kernel: linear, Cross-Validation Error Rate: 0.1424
C: 10, Kernel: poly, Cross-Validation Error Rate: 0.1500
C: 10, Kernel: sigmoid, Cross-Validation Error Rate: 0.1772

Cross-Validation Error Rates for all models:
C: 0.1, Kernel: linear -> Error Rate: 0.1408
C: 0.1, Kernel: poly -> Error Rate: 0.3616
C: 0.1, Kernel: sigmoid -> Error Rate: 0.1424
C: 1, Kernel: linear -> Error Rate: 0.1424
C: 1, Kernel: poly -> Error Rate: 0.1728
C: 1, Kernel: sigmoid -> Error Rate: 0.1616
C: 5, Kernel: linear -> Error Rate: 0.1424
C: 5, Kernel: poly -> Error Rate: 0.1502
C: 5, Kernel: sigmoid -> Error Rate: 0.1772
C: 10, Kernel: linear -> Error Rate: 0.1424
C: 10, Kernel: poly -> Error Rate: 0.1500
C: 10, Kernel: sigmoid -> Error Rate: 0.1772
```

[23]:
```
# 4. Pick and report the value of C and kernel that minimize the 5FCV error rate
best_C = 0.1
best_kernel = 'linear'

print(f"Best Model: C={best_C}, Kernel={best_kernel}")

# Train the best SVM model
best_svm = SVC(C=best_C, kernel=best_kernel)
best_svm.fit(X_scaled, y)
```

```
Best Model: C=0.1, Kernel=linear
```

[23]:
```
SVC(C=0.1, kernel='linear')
```

[24]:
```
# 5. Accuracy score of the model

X_work = df_work_encoded.drop('work', axis=1)
y_work = df_work_encoded['work']

scaler = StandardScaler()
X_work_scaled = scaler.fit_transform(X_work)

svm_model = SVC(C=0.1, kernel='linear', random_state=26)
svm_model.fit(X_work_scaled, y_work)

X_train, X_test, y_train, y_test = train_test_split(X_work_scaled, y_work,␣
 ↪test_size=0.2, random_state=42)
```

```
svm_model.fit(X_train, y_train)

y_pred = svm_model.predict(X_test)

print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Accuracy Score:", accuracy_score(y_test, y_pred))
```

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

           0       0.84      0.75      0.79       423
           1       0.83      0.89      0.86       577

    accuracy                           0.83      1000
   macro avg       0.83      0.82      0.83      1000
weighted avg       0.83      0.83      0.83      1000

Accuracy Score: 0.834
```

[25]:

```
# 6. With the SVM model that you fit on df_work, impute the work schedules␣
↪using the core variables from df_vote.
if 'prcitshp_FOREIGN BORN, NOT A CITIZEN OF' not in df_vote_encoded.columns:
    df_vote_encoded['prcitshp_FOREIGN BORN, NOT A CITIZEN OF'] = 0

if 'ptdtrace_4 or 5 Races' not in df_vote_encoded.columns:
    df_vote_encoded['ptdtrace_4 or 5 Races'] = 0

X_vote = df_vote_encoded[X_work.columns]
X_vote_scaled = scaler.transform(X_vote)

predicted_work_flexibility = svm_model.predict(X_vote_scaled)

predicted_df = pd.DataFrame(predicted_work_flexibility,␣
↪columns=['Imputed_Work_Flexibility'])

summary_statistics = predicted_df['Imputed_Work_Flexibility'].describe()
print("Descriptive Statistics for the Imputed Work Flexibility Measure:")
print(summary_statistics)

# Additional statistics
count_flexible = (predicted_df['Imputed_Work_Flexibility'] == 1).sum()
count_not_flexible = (predicted_df['Imputed_Work_Flexibility'] == 0).sum()
print(f"Count of Flexible Work Schedules: {count_flexible}")
print(f"Count of Not Flexible Work Schedules: {count_not_flexible}")
```

```
Descriptive Statistics for the Imputed Work Flexibility Measure:
```

```
count    5000.000000
mean        0.551000
std         0.497442
min         0.000000
25%         0.000000
50%         1.000000
75%         1.000000
max         1.000000
Name: Imputed_Work_Flexibility, dtype: float64
Count of Flexible Work Schedules: 2755
Count of Not Flexible Work Schedules: 2245
```

[26]:
```
# 7. Regress voting status on the imputed work schedule. Use age, age squared,␣
↪and sex as predictors in addition to the imputed work schedule. Report,␣
↪briefly interpret, and discuss the results.

if 'imputed_work' not in df_vote.columns:
    X_vote = df_vote_encoded[X_work.columns]
    X_vote_scaled = scaler.transform(X_vote)
    df_vote['imputed_work'] = svm_model.predict(X_vote_scaled)

df_vote['prtage'] = pd.to_numeric(df_vote['prtage'], errors='coerce')
df_vote['imputed_work'] = pd.to_numeric(df_vote['imputed_work'],␣
↪errors='coerce')

print("Checking for missing values:")
print(df_vote[['prtage', 'imputed_work']].isnull().sum())

df_vote.dropna(subset=['prtage', 'imputed_work'], inplace=True)

df_vote['age_squared'] = df_vote['prtage'] ** 2
X_regression = df_vote[['imputed_work', 'prtage', 'age_squared', 'pesex']]
X_regression = pd.get_dummies(X_regression, columns=['pesex'], drop_first=True)
y_regression = df_vote['vote']

X_regression = X_regression.apply(pd.to_numeric, errors='coerce')
X_regression['pesex_MALE'] = X_regression['pesex_MALE'].astype(int)

print("Checking for missing values after processing:")
print(X_regression.isnull().sum())

X_regression.dropna(inplace=True)
y_regression = y_regression[X_regression.index]

print("Final dataset for regression:")
print(X_regression.head())
print(y_regression.head())
```

```
X_regression = sm.add_constant(X_regression)

regression_model = sm.Logit(y_regression, X_regression).fit()

print(regression_model.summary())

coefficients = regression_model.params
print("\nCoefficients:")
print(coefficients)

print("\nDiscussion:")
print("The coefficient for 'imputed_work' represents the effect of the imputed␣
↪work schedule on the probability of voting.")
print("Positive coefficients increase the probability of voting, while negative␣
↪coefficients decrease it.")
print("The significance of coefficients (p-values) indicates whether these␣
↪predictors are statistically significant.")
print("Age and age squared help capture the non-linear effect of age on voting␣
↪probability.")
print("The coefficient for 'pesex_MALE' indicates the effect of being male on␣
↪the probability of voting compared to the reference category (female).")
```

```
Checking for missing values:
prtage          0
imputed_work    0
dtype: int64
Checking for missing values after processing:
imputed_work    0
prtage          0
age_squared     0
pesex_MALE      0
dtype: int64
Final dataset for regression:
   imputed_work  prtage  age_squared  pesex_MALE
0             1      19          361           0
1             1      35         1225           1
2             0      48         2304           1
3             0      55         3025           1
4             1      25          625           0
0    1
1    1
2    1
3    0
4    1
Name: vote, dtype: int64
Optimization terminated successfully.
```

```
         Current function value: 0.330385
         Iterations 9
                           Logit Regression Results
==============================================================================
Dep. Variable:                   vote   No. Observations:                 5000
Model:                          Logit   Df Residuals:                     4995
Method:                           MLE   Df Model:                            4
Date:                Wed, 22 May 2024   Pseudo R-squ.:                  0.5224
Time:                        17:58:22   Log-Likelihood:                -1651.9
converged:                       True   LL-Null:                       -3459.0
Covariance Type:            nonrobust   LLR p-value:                     0.000
================================================================================
                   coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
const            4.9463      0.680      7.272      0.000       3.613       6.280
imputed_work     0.0166      0.146      0.114      0.909      -0.270       0.303
prtage          -0.0344      0.033     -1.055      0.291      -0.098       0.029
age_squared     -0.0016      0.000     -4.030      0.000      -0.002      -0.001
pesex_MALE       0.1496      0.088      1.701      0.089      -0.023       0.322
================================================================================

Coefficients:
const           4.946336
imputed_work    0.016622
prtage         -0.034384
age_squared    -0.001623
pesex_MALE      0.149623
dtype: float64

Discussion:
The coefficient for 'imputed_work' represents the effect of the imputed work
schedule on the probability of voting.
Positive coefficients increase the probability of voting, while negative
coefficients decrease it.
The significance of coefficients (p-values) indicates whether these predictors
are statistically significant.
Age and age squared help capture the non-linear effect of age on voting
probability.
The coefficient for 'pesex_MALE' indicates the effect of being male on the
probability of voting compared to the reference category (female).
```

[27]:

```python
# 8. Define the scaling function M(a, b) and compute values of a, b, and M(a,␣
↪b).

def scaling_function_M(a, b):
    term1 = 1 / (1 - 2 * b)
    term2 = 1 - ((1 - b) * b / a) - ((1 - b) * b / (1 - a))
```

```
    return term1 * term2

a = df_vote['imputed_work'].mean()
print(f"Proportion of imputed work schedules that are flexible (a): {a}")

b = 0.1408
print(f"Cross-validation error rate (b): {b}")

M_ab = scaling_function_M(a, b)

print(f"Scaling function M(a, b): {M_ab}")
print(f"Proportion a: {a}")
print(f"Error rate b: {b}")
```

```
Proportion of imputed work schedules that are flexible (a): 0.551
Cross-validation error rate (b): 0.1408
Scaling function M(a, b): 0.7113183737322479
Proportion a: 0.551
Error rate b: 0.1408
```

[28]:
```
# 9. Correct for the attenuation bias in your results from Question 7. Is the
 ↪bias-corrected version larger or smaller? Does the bias-correction change
 ↪your previous result? Briefly explain.

original_coefficient = regression_model.params['imputed_work']
print(f"Original coefficient for 'imputed_work': {original_coefficient}")

corrected_coefficient = original_coefficient / M_ab
print(f"Corrected coefficient for 'imputed_work': {corrected_coefficient}")

print("\nDiscussion:")
print(f"Original coefficient for 'imputed_work': {original_coefficient}")
print(f"Corrected coefficient for 'imputed_work': {corrected_coefficient}")

if corrected_coefficient > original_coefficient:
    print("The bias-corrected coefficient is larger than the original
 ↪coefficient, indicating that the original result underestimated the effect
 ↪of the imputed work schedule on the probability of voting.")
else:
    print("The bias-corrected coefficient is smaller than the original
 ↪coefficient, indicating that the original result overestimated the effect of
 ↪the imputed work schedule on the probability of voting.")

print("Does the bias-correction change your previous result?")
```

```
print("The bias correction helps to provide a more accurate estimate of the␣
↪effect of the imputed work schedule on the probability of voting. This␣
↪adjustment accounts for the measurement error and gives a more reliable␣
↪coefficient, which could either increase or decrease the estimated effect␣
↪depending on the direction of the bias.")
```

```
Original coefficient for 'imputed_work': 0.016621655914117177
Corrected coefficient for 'imputed_work': 0.02336739289736644

Discussion:
Original coefficient for 'imputed_work': 0.016621655914117177
Corrected coefficient for 'imputed_work': 0.02336739289736644
The bias-corrected coefficient is larger than the original coefficient,
indicating that the original result underestimated the effect of the imputed
work schedule on the probability of voting.
Does the bias-correction change your previous result?
The bias correction helps to provide a more accurate estimate of the effect of
the imputed work schedule on the probability of voting. This adjustment accounts
for the measurement error and gives a more reliable coefficient, which could
either increase or decrease the estimated effect depending on the direction of
the bias.
```