Chapters 4 and 5: Python Workbook

Ahmad Tahmid

Contents

1 Chapter 4 Solutions

1.1 Exercise 4.7.1

Goal

Use logistic regression to predict a binary outcome (e.g., `Direction` in the `Smarket` data), examine confusion matrix and accuracy.

Solution

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from sklearn.metrics import confusion_matrix

# Assume 'Smarket.csv' is in the working directory
Smarket = pd.read_csv("Smarket.csv", index_col=0)

# Convert Direction to 0/1
Smarket['Direction_binary'] = (Smarket['Direction'] == 'Up').astype(
    int)

# Split the data: train on years < 2005, test on 2005
train_mask = Smarket['Year'] < 2005
test_mask = ~train_mask

X_train = Smarket.loc[train_mask, ['Lag1','Lag2','Lag3','Lag4','Lag5
    ','Volume']]
y_train = Smarket.loc[train_mask, 'Direction_binary']
X_test  = Smarket.loc[test_mask, ['Lag1','Lag2','Lag3','Lag4','Lag5'
    ,'Volume']]
y_test  = Smarket.loc[test_mask, 'Direction_binary']

# Add constant for intercept
X_train_const = sm.add_constant(X_train)
X_test_const = sm.add_constant(X_test)

# Fit logistic regression via statsmodels
logit_model = sm.Logit(y_train, X_train_const).fit()
print(logit_model.summary())

# Predictions
y_pred_prob = logit_model.predict(X_test_const)
y_pred_class = (y_pred_prob >= 0.5).astype(int)

# Confusion matrix and accuracy
cm = confusion_matrix(y_test, y_pred_class)
```

```
print("Confusion Matrix:\n", cm)
accuracy = np.mean(y_test.values == y_pred_class)
print("Accuracy:", accuracy)
```

1.2 Exercise 4.7.2

Goal

Apply LDA, QDA, and KNN to the same or similar dataset, compare classification results.

Solution

```
import pandas as pd
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
    , QuadraticDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix

Smarket = pd.read_csv("Smarket.csv", index_col=0)
Smarket['Direction_binary'] = (Smarket['Direction'] == 'Up').astype(
    int)

train_mask = Smarket['Year'] < 2005
test_mask = ~train_mask

X_train = Smarket.loc[train_mask, ['Lag1','Lag2','Lag3','Lag4','Lag5
    ','Volume']]
y_train = Smarket.loc[train_mask, 'Direction_binary']
X_test  = Smarket.loc[test_mask, ['Lag1','Lag2','Lag3','Lag4','Lag5'
    ,'Volume']]
y_test  = Smarket.loc[test_mask, 'Direction_binary']

# LDA
lda = LinearDiscriminantAnalysis()
lda.fit(X_train, y_train)
lda_preds = lda.predict(X_test)
cm_lda = confusion_matrix(y_test, lda_preds)
acc_lda = np.mean(lda_preds == y_test)
print("LDA Confusion Matrix:\n", cm_lda)
print("LDA Accuracy:", acc_lda)

# QDA
qda = QuadraticDiscriminantAnalysis()
qda.fit(X_train, y_train)
qda_preds = qda.predict(X_test)
cm_qda = confusion_matrix(y_test, qda_preds)
```

```
acc_qda = np.mean(qda_preds == y_test)
print("QDA Confusion Matrix:\n", cm_qda)
print("QDA Accuracy:", acc_qda)

# KNN (k=3 example)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
knn_preds = knn.predict(X_test)
cm_knn = confusion_matrix(y_test, knn_preds)
acc_knn = np.mean(knn_preds == y_test)
print("KNN Confusion Matrix:\n", cm_knn)
print("KNN Accuracy:", acc_knn)
```

1.3 Exercise 4.7.3

Goal

Experiment with different K values in KNN, or different subsets of predictors, and observe how accuracy changes.

Solution

```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix

Smarket = pd.read_csv("Smarket.csv", index_col=0)
Smarket['Direction_binary'] = (Smarket['Direction'] == 'Up').astype(
    int)

train_mask = Smarket['Year'] < 2005
test_mask = ~train_mask

X_train = Smarket.loc[train_mask, ['Lag1','Lag2']]
y_train = Smarket.loc[train_mask, 'Direction_binary']
X_test  = Smarket.loc[test_mask, ['Lag1','Lag2']]
y_test  = Smarket.loc[test_mask, 'Direction_binary']

# Evaluate K = 1..10
for k in range(1, 11):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    preds = knn.predict(X_test)
    accuracy = np.mean(preds == y_test)
    print(f"K={k}, Accuracy={accuracy:.3f}")
```

2 Chapter 5 Solutions

2.1 Exercise 5.3.1

Goal

Use cross-validation (e.g., LOOCV or K-fold) to estimate test error for polynomial regression (predicting `mpg` from `horsepower` in `Auto`).

Solution

```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

Auto = pd.read_csv("Auto.csv", na_values='?').dropna()
X_hp = Auto[['horsepower']]
y_mpg = Auto['mpg']

degrees = [1, 2, 3, 4, 5]
kf = KFold(n_splits=10, shuffle=True, random_state=42)

for d in degrees:
    fold_mses = []
    poly = PolynomialFeatures(degree=d, include_bias=False)
    for train_idx, test_idx in kf.split(X_hp):
        X_train, X_test = X_hp.iloc[train_idx], X_hp.iloc[test_idx]
        y_train, y_test = y_mpg.iloc[train_idx], y_mpg.iloc[test_idx
           ]

        # Transform
        X_train_poly = poly.fit_transform(X_train)
        X_test_poly = poly.transform(X_test)

        # Fit
        lm = LinearRegression().fit(X_train_poly, y_train)
        preds = lm.predict(X_test_poly)
        mse = np.mean((y_test - preds)**2)
        fold_mses.append(mse)

    print(f"Degree={d}, Mean CV MSE={np.mean(fold_mses):.2f}")
```

2.2 Exercise 5.3.2

Goal

Use the bootstrap to estimate the standard error of a statistic (e.g., the median of `mpg` or the coefficient in a regression model).

Solution

```
import numpy as np
import pandas as pd

Auto = pd.read_csv("Auto.csv", na_values='?').dropna()

mpg_values = Auto['mpg'].values

def bootstrap_statistic(data, func, B=1000):
    n = len(data)
    stats = []
    for _ in range(B):
        sample_idx = np.random.randint(0, n, n)
        sample = data[sample_idx]
        stats.append(func(sample))
    return np.std(stats)

# Example: Standard error of the median
median_mpg = np.median(mpg_values)
boot_se = bootstrap_statistic(mpg_values, np.median, B=1000)

print("Sample median MPG:", median_mpg)
print("Bootstrap SE for median MPG:", boot_se)
```

2.3 Exercise 5.3.3

Goal

Use cross-validation to choose the best model (or set of predictors) in a multiple regression, or to compare linear vs. polynomial fits.

Solution

```
import pandas as pd
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression

Auto = pd.read_csv("Auto.csv", na_values='?').dropna()
```

```
# Suppose we compare mpg ~ cylinders + weight vs. mpg ~ cylinders +
    weight + year
X1 = Auto[['cylinders','weight']]
X2 = Auto[['cylinders','weight','year']]
y = Auto['mpg']

lm1 = LinearRegression()
lm2 = LinearRegression()

scores1 = cross_val_score(lm1, X1, y, cv=10, scoring='
    neg_mean_squared_error')
scores2 = cross_val_score(lm2, X2, y, cv=10, scoring='
    neg_mean_squared_error')

mse1 = -scores1.mean()
mse2 = -scores2.mean()

print("Model 1 MSE:", mse1)
print("Model 2 MSE:", mse2)
```

3 My Learnings

- experimented with classification methods (logistic regression, LDA, QDA, KNN) to handle binary outcomes
- used confusion matrices and accuracy metrics to evaluate model performance
- performed cross-validation (k-fold) to compare models like polynomial vs. linear regression
- explored bootstrap to estimate variability (e.g., standard errors of median or regression coefficients)
- gained practice with `numpy`, `pandas`, `statsmodels`, and `scikit-learn`