fzfatv7oc

April 13, 2024

0.0.1 Part 1: Poisson regression models and hypothesis testing

Exercise 1.1

[1]:
```
import numpy as np

def crossings_count(sentence_length, alpha, beta):
    # Calculate lambda using the formula
    lambda_i = np.exp(alpha + beta * sentence_length)

    # Generate the number of crossings using a Poisson distribution
    crossings = np.random.poisson(lambda_i)

    return crossings

# Example usage:
sentence_length = 10
alpha = 1.5
beta = 0.1

crossings = crossings_count(sentence_length, alpha, beta)
print(f"Number of crossings in a sentence of length {sentence_length}: 
 ↪{crossings}")
```

```
Number of crossings in a sentence of length 10: 11
```

Exercise 1.2

[2]:
```
alpha_mean, alpha_std = 0.15, 0.1
beta_mean, beta_std = 0.25, 0.05

# Generate prior predictions for sentences of length 4
num_samples = 1000
sentence_length = 4

# Sample alpha and beta values from normal distributions
alphas = np.random.normal(alpha_mean, alpha_std, num_samples)
betas = np.random.normal(beta_mean, beta_std, num_samples)

# Calculate expected number of crossings for each sampled alpha and beta
```

```
expected_crossings = [crossings_count(sentence_length, alpha, beta) for alpha,␣
↪beta in zip(alphas, betas)]

print(expected_crossings[:20])

print(f"Mean expected number of crossings for sentences of length␣
↪{sentence_length}: {np.mean(expected_crossings)}")
print(f"Standard deviation of expected number of crossings: {np.
↪std(expected_crossings)}")
```

```
[1, 3, 1, 6, 4, 1, 6, 3, 1, 3, 1, 2, 1, 1, 2, 1, 3, 1, 5, 6]
Mean expected number of crossings for sentences of length 4: 3.302
Standard deviation of expected number of crossings: 2.006687818271691
```

Exercise 1.3 Model M1

[3]:
```
import pandas as pd
import statsmodels.api as sm

# Load the data from the CSV file
data = pd.read_csv('crossings.csv')

# Define the predictor variable (sentence length) and the response variable␣
↪(number of crossings)
X = data['s.length']
y = data['nCross']

# Add intercept term
X = sm.add_constant(X)

# Fit Poisson regression model
model = sm.GLM(y, X, family=sm.families.Poisson()).fit()

# Print model summary
print(model.summary())
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                 nCross   No. Observations:                 1900
Model:                            GLM   Df Residuals:                     1898
Model Family:                 Poisson   Df Model:                            1
Link Function:                    Log   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -2813.4
Date:                Sat, 13 Apr 2024   Deviance:                       2272.1
Time:                        04:48:02   Pearson chi2:                 2.08e+03
No. Iterations:                     5   Pseudo R-squ. (CS):             0.6070
Covariance Type:            nonrobust
==============================================================================
```

```
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -1.4429      0.061    -23.755      0.000      -1.562      -1.324
s.length       0.1494      0.004     38.505      0.000       0.142       0.157
==============================================================================
```

Model M2

[12]:
```python
import pandas as pd
import numpy as np
from scipy.stats import poisson

# Load the data from the CSV file
data = pd.read_csv('crossings.csv')

# Define the predictor variable (sentence length) and the response variable␣
↪(number of crossings)
X = data['s.length'].values
y = data['nCross'].values

# Add indicator variable for language (0 for English, 1 for German)
data['Language_Indicator'] = (data['Language'] == 'German').astype(int)
R = data['Language_Indicator'].values

# Define likelihood function for Model M2
def likelihood_M2(alpha, beta, beta_language, beta_interact):
    lambda_i = np.exp(alpha + beta * X + beta_language * R + beta_interact * X␣
↪* R)
    log_likelihood = np.sum(poisson.logpmf(y, lambda_i))
    return log_likelihood

# Define prior log probability for parameters
def prior_log_prob(alpha, beta, beta_language, beta_interact):
    prior_alpha = -0.5 * ((alpha - 0.15) / 0.1)**2
    prior_beta = -0.5 * (beta / 0.15)**2
    prior_beta_language = -0.5 * (beta_language / 0.15)**2
    prior_beta_interact = -0.5 * (beta_interact / 0.15)**2
    return prior_alpha + prior_beta + prior_beta_language + prior_beta_interact

# Metropolis-Hastings algorithm for Model M2
def metropolis_hastings_M2(n_iterations, initial_alpha, initial_beta,␣
↪initial_beta_language, initial_beta_interact):
    alpha_current = initial_alpha
    beta_current = initial_beta
    beta_language_current = initial_beta_language
    beta_interact_current = initial_beta_interact
    accepted_samples = []
```

```python
    for _ in range(n_iterations):
        # Propose new parameters
        alpha_proposed = np.random.normal(alpha_current, 0.1)
        beta_proposed = np.random.normal(beta_current, 0.1)
        beta_language_proposed = np.random.normal(beta_language_current, 0.1)
        beta_interact_proposed = np.random.normal(beta_interact_current, 0.1)

        # Compute acceptance probability
        log_alpha = likelihood_M2(alpha_proposed, beta_proposed, 
 ↪beta_language_proposed, beta_interact_proposed)
        + prior_log_prob(alpha_proposed, beta_proposed, beta_language_proposed, 
 ↪beta_interact_proposed)
        log_alpha_current = likelihood_M2(alpha_current, beta_current, 
 ↪beta_language_current, beta_interact_current)
        + prior_log_prob(alpha_current, beta_current, beta_language_current, 
 ↪beta_interact_current)
        log_acceptance_prob = log_alpha - log_alpha_current

        # Accept or reject proposal
        if np.log(np.random.uniform()) < log_acceptance_prob:
            alpha_current = alpha_proposed
            beta_current = beta_proposed
            beta_language_current = beta_language_proposed
            beta_interact_current = beta_interact_proposed
            accepted_samples.append((alpha_current, beta_current, 
 ↪beta_language_current, beta_interact_current))

    return accepted_samples

# Set initial values for parameters
initial_alpha = 0.15
initial_beta = 0
initial_beta_language = 0
initial_beta_interact = 0

# Run Metropolis-Hastings for Model M2
samples_M2 = metropolis_hastings_M2(10000, initial_alpha, initial_beta, 
 ↪initial_beta_language, initial_beta_interact)

# Calculate posterior means and standard deviations
posterior_mean_M2 = np.mean(samples_M2, axis=0)
posterior_std_M2 = np.std(samples_M2, axis=0)

# Print posterior means and standard deviations
print("\nModel M2:")
print("alpha:", posterior_mean_M2[0], "(Std:", posterior_std_M2[0], ")")
```

```python
print("beta:", posterior_mean_M2[1], "(Std:", posterior_std_M2[1], ")")
print("beta_language:", posterior_mean_M2[2], "(Std:", posterior_std_M2[2], ")")
print("beta_interact:", posterior_mean_M2[3], "(Std:", posterior_std_M2[3], ")")
```

```
Model M2:
alpha: -0.21381754331157182 (Std: 0.41324540116212455 )
beta: 0.050347321071550925 (Std: 0.034783742757181005 )
beta_language: -0.654455839716083 (Std: 0.26699144296761995 )
beta_interact: 0.07512914116392039 (Std: 0.019869504330144467 )
```

Exercise 1.4

[14]:
```python
import pandas as pd
import numpy as np
from scipy.stats import poisson
from sklearn.model_selection import KFold
import statsmodels.api as sm

# Load the data from the CSV file
observed = pd.read_csv("crossings.csv")

# Center the predictors
observed['s.length'] -= observed['s.length'].mean()
observed['lang'] = (observed['Language'] == 'German').astype(int)

# Create the interaction term between s.length and lang
observed['s.length_lang_interaction'] = observed['s.length'] * observed['lang']

# Prepare the predictors and response variables
X = observed[['s.length', 'lang', 's.length_lang_interaction']].values
y = observed['nCross'].values

# K-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
lpds_m1 = []
lpds_m2 = []

for train_index, test_index in kf.split(observed):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Fit model M1
    X_train_m1 = sm.add_constant(X_train[:, 0])  # Adding constant for intercept
    model_m1 = sm.GLM(y_train, X_train_m1, family=sm.families.Poisson()).fit()

    # Fit model M2
    X_train_m2 = sm.add_constant(X_train)  # Adding constant for intercept
```

```
    model_m2 = sm.GLM(y_train, X_train_m2, family=sm.families.Poisson()).fit()

    # Calculate log pointwise predictive density using test data for model M1
    lambda_m1 = model_m1.predict(sm.add_constant(X_test[:, 0]))
    lppd_m1 = np.sum(np.log(poisson.pmf(y_test, mu=lambda_m1)))

    # Calculate log pointwise predictive density using test data for model M2
    lambda_m2 = model_m2.predict(sm.add_constant(X_test))
    lppd_m2 = np.sum(np.log(poisson.pmf(y_test, mu=lambda_m2)))

    lpds_m1.append(lppd_m1)
    lpds_m2.append(lppd_m2)

# Predictive accuracy of model M1
elpd_m1 = sum(lpds_m1)

# Predictive accuracy of model M2
elpd_m2 = sum(lpds_m2)

# Evidence in favor of M2 over M1
difference_elpd = elpd_m2 - elpd_m1

print("Evidence in favor of M2 over M1:", difference_elpd)
```

```
Evidence in favor of M2 over M1: 133.9533544006572
```

Such a high ELPD value suggests that model M2 is better for making predictions on new data.

[ ]:

```

```