2.1-section-result

February 14, 2025

1 Section 2 - Controlling for confounding factors

- A **confounding factor** is a variable that is associated with both a **feature** and an **outcome**
- The presence of confounding factors can distort the true relationship between features and outcomes, leading to incorrect conclusions
- In this section, we will explore the use of **logistic regression** to control for the influence of confounding factors in calculations of association

1.1 Example 2.1

Application 2.1: Determining if protein size explains the previously noted (**Example 1.1**) association of entanglements with misfolding in *E. coli*

1.1.1 Why do we suspect protein length may be a confounding factor?

- Small proteins will tend to have simple free-energy landscapes that favor fast & correct folding (**Figure 2.1.1**)

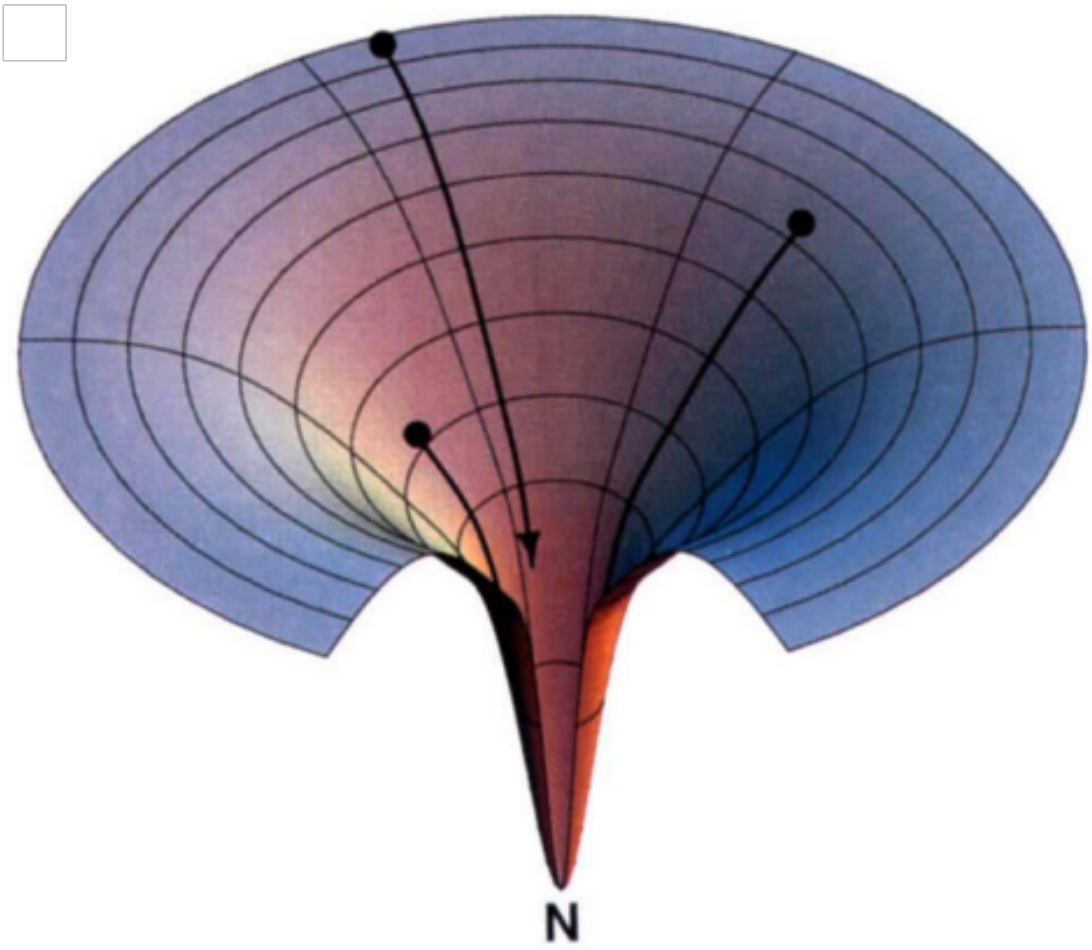

Figure 2.1.1. *Smaller proteins will tend to have simpler free-energy landscapes approaching the idealized case shown here, in which the native state (N) is at the global free-energy minimum and there are no local minima with similar stabilities. Reproduced from Dill & Chan. Nat. Struc. Biol. 1997.*

- As proteins increase in size, their free-energy landscapes increase in complexity (**Figure 2.1.2**)

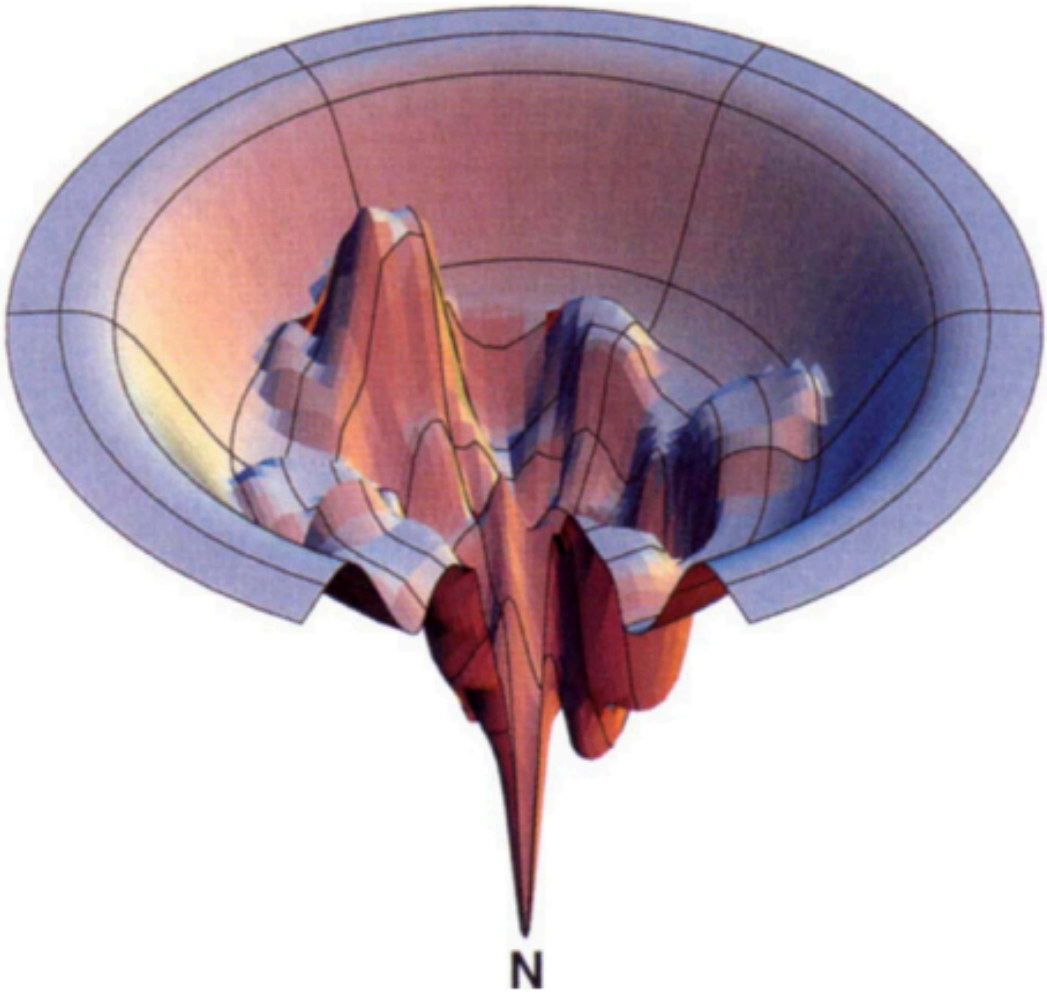

Figure 2.1.2. *Larger proteins will tend to have more complex free-energy landscapes characterized by a number of local minima that may be similar to the native state (N) in free energy. These local minima may correspond to misfolded states that are separated from the native state by high energy barriers. Reproduced from Dill & Chan. Nat. Struc. Biol. 1997.*

- This increased complexity in the free-energy landscape may lead to misfolding
- Protein size is thus a key confounding factor for our analysis in **Application 1.1** of the association of entanglement with misfolding

1.1.2 Testing for the influence of confounding factors with logistic regression

- Now that we have framed the problem, let's dive into using **logistic regression** to test the influence of confounding factors on association

1.1.3 Step 0 - Load libraries

- As always, we begin by loading libraries to set up our environment

[1]:
```
import pandas as pd
import statsmodels.api as sm
import numpy as np
```

1.1.4 Step 1 - Load and explore the data

- We need to do a little data manipulation to prepare for analysis; we will load one set of data with information on misfolding (LiP-MS) and entanglement and then use a second set of data to add protein length information

[2]:
```
# data4_p1 contains information about misfolding and entanglement status for␣
 ↪each protein
data_path = "/home/jovyan/data-store/data/iplant/home/shared/NCEMS/
 ↪BPS-training-2025/"
data4_p1    = pd.read_csv(data_path +␣
 ↪"NativeEntanglements_and_SigCuts_EXP_buffC.csv")

# data4_p2 contains information on protein length; we will only load the␣
 ↪columns "gene" and "uniprot_length"
data4_p2    = pd.read_csv(data_path + "Ecoli_entanglement_data.csv", usecols =␣
 ↪["gene", "uniprot_length"])

# remove duplicate rows based on gene identifier from data4_p2
data4_p2.drop_duplicates(subset = "gene", keep = "first", inplace = True)

# perform a merge of data4_p1 & data4_p2 to insert uniprot_length information␣
 ↪for each protein in data4_p1
data4_final = pd.merge(data4_p1, data4_p2, on = "gene", how = "left")

# print summary information
print ("Create a quick summary of the DataFrame:\n")
data4_final.info()

print ("\nPrint the first 10 rows of the DataFrame:\n")
data4_final.head(10)
```

```
Create a quick summary of the DataFrame:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 345 entries, 0 to 344
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   buff            345 non-null    object
 1   gene            345 non-null    object
 2   NativeEnt       345 non-null    bool
 3   NonRefoldable   345 non-null    bool
```

```
 4   uniprot_length  345 non-null    int64
dtypes: bool(2), int64(1), object(2)
memory usage: 8.9+ KB

Print the first 10 rows of the DataFrame:
```

[2]:

	buff	gene	NativeEnt	NonRefoldable	uniprot_length
0	C	P00350	True	True	468
1	C	P00370	True	True	447
2	C	P00448	True	True	206
3	C	P00509	True	True	396
4	C	P00561	True	False	820
5	C	P00579	False	False	613
6	C	P00864	True	True	883
7	C	P00934	True	True	428
8	C	P00954	False	True	334
9	C	P00957	True	True	876

1.1.5 Step 2 - Prepare for analysis

- We now have *almost* all of the information needed for our planned analysis in a single DataFrame, `data4_final`
- We need to do a few final data manipulations and add a single new column, and then we are ready to go

[3]:
```
# the values of NativeEnt & NonRefoldable are currently booleans (True & False)
# we need to convert True to 1 and False to 0 for logistic regression
recode_map                   = {True: 1, False: 0}
data4_final['NativeEnt']     = data4_final['NativeEnt'].map(recode_map)
data4_final['NonRefoldable'] = data4_final['NonRefoldable'].map(recode_map)

# finally, we need to add a column representing the intercept of the logistic␣
 ↪regression model
# we will use this in Step 3 when running the regression

# add column of 1's corresponding to the intercept
# the intercept represents the log-odds of the outcome when all features are␣
 ↪zero
data4_final['intercept'] = 1

# print a summary of this DataFrame
data4_final.info()
data4_final.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 345 entries, 0 to 344
Data columns (total 6 columns):
```

```
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   buff            345 non-null    object
 1   gene            345 non-null    object
 2   NativeEnt       345 non-null    int64
 3   NonRefoldable   345 non-null    int64
 4   uniprot_length  345 non-null    int64
 5   intercept       345 non-null    int64
dtypes: int64(4), object(2)
memory usage: 16.3+ KB
```

[3]:

	buff	gene	NativeEnt	NonRefoldable	uniprot_length	intercept
0	C	P00350	1	1	468	1
1	C	P00370	1	1	447	1
2	C	P00448	1	1	206	1
3	C	P00509	1	1	396	1
4	C	P00561	1	0	820	1
5	C	P00579	0	0	613	1
6	C	P00864	1	1	883	1
7	C	P00934	1	1	428	1
8	C	P00954	0	1	334	1
9	C	P00957	1	1	876	1

1.1.6 Step 3 - Run the analysis

[4]:
```
# make two X datasets X1 & X2, one including the confounder and one excluding it

# X1 includes only the feature
X1 = data4_final[['intercept', 'NativeEnt']]

# X2 includes both the feature and the confounder
X2 = data4_final[['intercept', 'NativeEnt', 'uniprot_length']]

# define the dependent variable (i.e., the outcome)
y = data4_final['NonRefoldable']

# create two LogisticRegression() objects, fit the models, get the
# coefficients, and compute the odds ratios

# model1 will not include the confounder
model1  = sm.Logit(y, X1)
result1 = model1.fit(disp = 0)

# print a summary of result1
print ("\nResults when confounding factor IS NOT included:\n")

# get a summary of the results
```

```
odds_ratios = pd.DataFrame({"Coefficient": result1.params,
                            "OR"         : np.exp(result1.params),
                            "Lower CI"   : np.exp(result1.conf_int()[0]),
                            "Upper CI"   : np.exp(result1.conf_int()[1]),
                            "p-value"    : result1.pvalues}).
 ↪drop(index="intercept", errors="ignore")

# print the odds ratio
print (odds_ratios.round(3), "\n")

# model2 includes the confounder
model2 = sm.Logit(y, X2)
result2 = model2.fit(disp = 0)

# print a summary of result2
print ("\nResults when confounding factor IS included:\n")

# get a summary of the results
odds_ratios = pd.DataFrame({"Coefficient": result2.params,
                            "OR"         : np.exp(result2.params),
                            "Lower CI"   : np.exp(result2.conf_int()[0]),
                            "Upper CI"   : np.exp(result2.conf_int()[1]),
                            "p-value"    : result2.pvalues}).
 ↪drop(index="intercept", errors="ignore")

# print the odds ratio
print (odds_ratios.round(3), "\n")
```

```
Results when confounding factor IS NOT included:

           Coefficient     OR  Lower CI  Upper CI  p-value
NativeEnt        1.433  4.192     2.326     7.553      0.0


Results when confounding factor IS included:

                Coefficient     OR  Lower CI  Upper CI  p-value
NativeEnt             1.119  3.062     1.555     6.028    0.001
uniprot_length        0.002  1.002     1.000     1.004    0.099
```

1.1.7 Step 4 - Interpret the results

- When we **do not** include the confounding factor of protein length, we get the **same odds ratio of 4.19** as we found in **Example 1.1** using Fisher's Exact Test and again find a p-value «0.05
- When we **do** include protein length as a confounding factor, we observe that **the odds ratio**

decreases to 3.06 for the association of entanglement with misfolding; the odds ratio for the association of protein length with misfolding is 1.00

- The association of `uniprot_length` with disease association is not significant (p-value = 0.099 is >0.05)
- We conclude that while protein length is not associated with misfolding (odds ratio = 1.00), it indirectly influences the odds ratio of the feature, likely due to its association with the feature itself
  - Protein length is thus a partial confounding factor that explains some of the relationship between entanglement and misfolding
  - Entanglement is still associated with misfolding, indicating that this feature has a meaningful independent association with the outcome