Logistic Regression Extensions

Justin Post

Logistic Regression

As with linear regression, we can include multiple predictors and interaction terms!

- Grab our data and fit a basic logistic regression model

```
import pandas as pd; import numpy as np
from sklearn.linear_model import LogisticRegression
#read data
water = pd.read_csv("data/water_potability.csv")
#fit model
log_reg = LogisticRegression(penalty = 'none')
log_reg.fit(X = water["Hardness"].values.reshape(-1,1), y = water["Potability"].values)
```

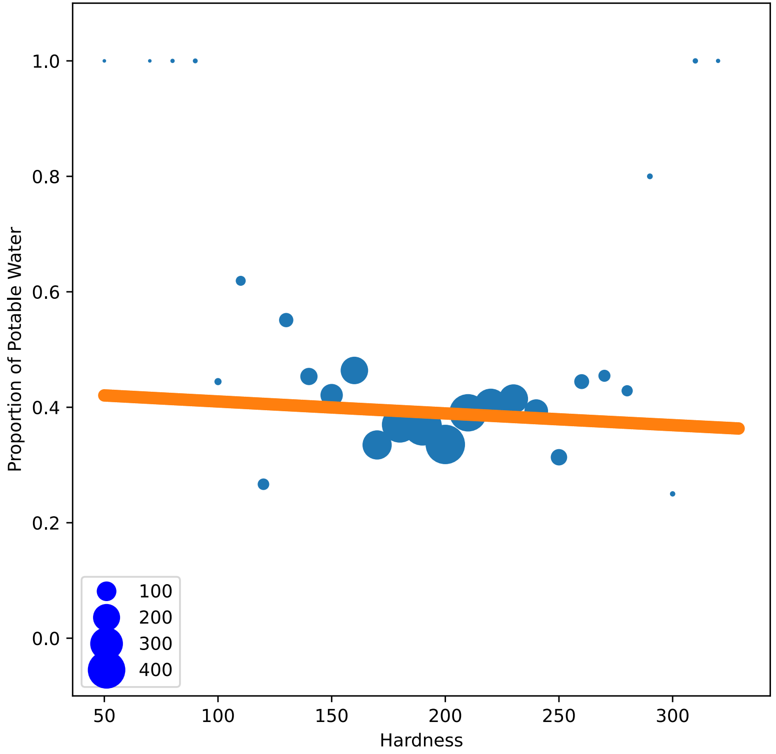
```
print(log_reg.intercept_, log_reg.coef_)
```

```
## [-0.27748213] [[-0.00086296]]
```

NC STATE UNIVERSITY

Visual

```
## (-0.1, 1.1)
```



1.0
0.8
0.6
0.4
0.2
0.0
Proportion of Potable Water
50
100
150
200
250
300
Hardness
100
200
300
400

Predictors

Can add a categorical variable as a predictor using dummy variables

- Create a high and low chloramines variable

```
water["Chlor_Cat"] = pd.cut(water["Chloramines"], [0.35, 9, 13.2], labels = ['low', 'high'])
water['highChl'] = pd.get_dummies(data = water['Chlor_Cat'])['high']
```

NC STATE UNIVERSITY

Predictors

Can add a categorical variable as a predictor using dummy variables

- Create a high and low chloramines variable

```
water["Chlor_Cat"] = pd.cut(water["Chloramines"], [0.35, 9, 13.2], labels = ['low', 'high'])
water['highChl'] = pd.get_dummies(data = water['Chlor_Cat'])['high']
```

- Adding a dummy variable just changes the intercept!

NC STATE UNIVERSITY

Visual of Models

`highChl` variable mostly just shifts the logistic curve over in the part we care about:

```
log_reg = LogisticRegression(penalty = 'none')
log_reg.fit(X = water[["Hardness", "highChl"]], y = water["Potability"])
```

```
print(log_reg.intercept_, log_reg.coef_)
```

```
## [-0.32288886] [[-0.00083126  0.33873976]]
```

NC STATE UNIVERSITY

Visual of Models

`highChl` variable mostly just shifts the logistic curve over in the part we care about:

```
to_pred = pd.DataFrame(np.array([[i, 1 if j == 1 else 0] for i in range(50, 330) for j in range(2)]),
                       columns = ["Hardness", "highChl"])
to_pred.head()
```

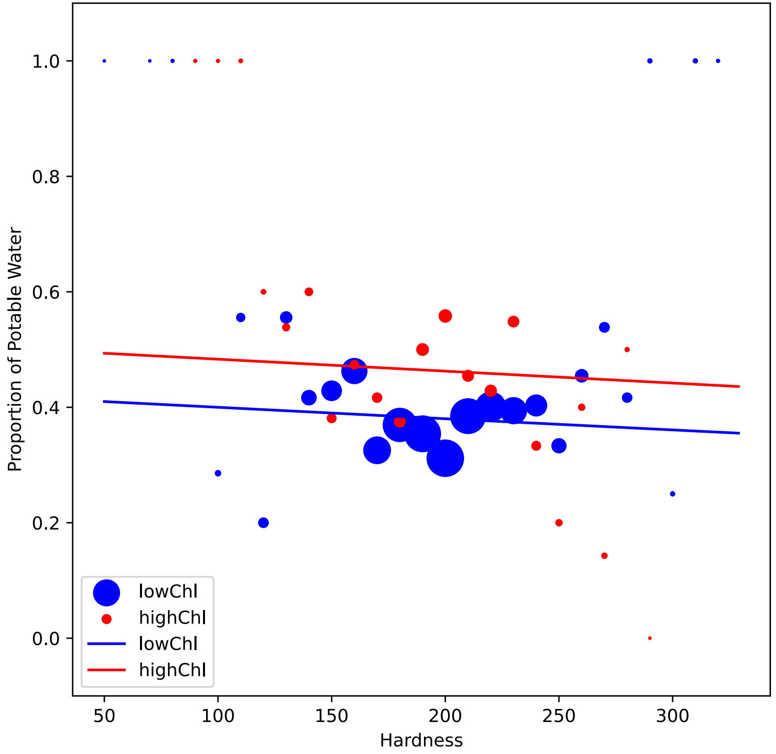
```
##    Hardness  highChl
## 0        50        0
## 1        50        1
## 2        51        0
## 3        51        1
## 4        52        0
```

```
pred_probs = pd.DataFrame(log_reg.predict_proba(to_pred))
pred_probs.head()
```

```
##           0         1
## 0  0.590118  0.409882
## 1  0.506428  0.493572
## 2  0.590319  0.409681
## 3  0.506635  0.493365
## 4  0.590520  0.409480
```

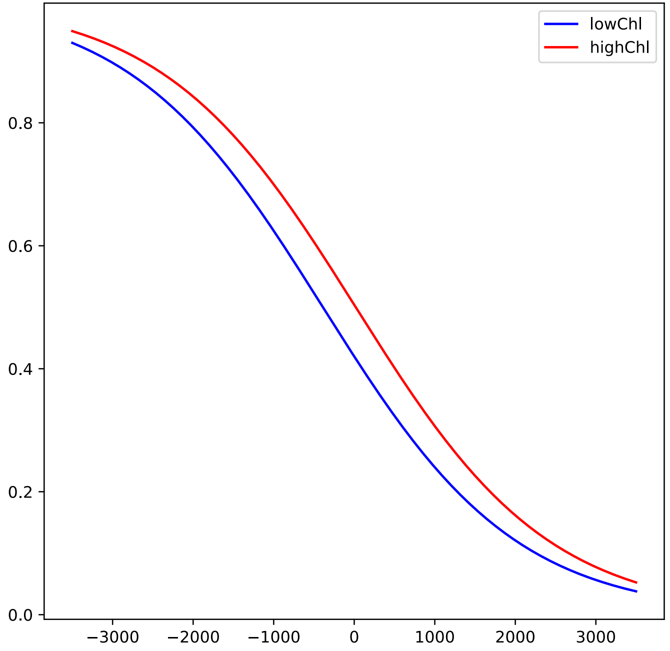
Visual of Models

```
## (-0.1, 1.1)
```



NC STATE UNIVERSITY

Not a Constant Difference



Interaction Terms Can Be Included

- If we fit an interaction term with our dummy variable, we essentially fit two separate logistic regression models

Fitting an Interaction Model

- To include interaction terms, create with `sklearn`

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(interaction_only=True, include_bias = False)
design = poly.fit_transform(water[["Hardness", "highChl"]])
design
```

```
## array([[204.89045547,   0.        ,   0.        ],
##        [129.42292051,   0.        ,   0.        ],
##        [224.23625939,   1.        , 224.23625939],
##        ...,
##        [175.7626463 ,   0.        ,   0.        ],
##        [230.60375751,   0.        ,   0.        ],
##        [195.10229859,   0.        ,   0.        ]])
```

Fitting an Interaction Model

- To include interaction terms, create with `sklearn`

```
from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(interaction_only=True, include_bias = False)
design = poly.fit_transform(water[["Hardness", "highChl"]])
```

```
log_reg = LogisticRegression(penalty = 'none', solver = "newton-cg")
log_reg.fit(X = design, y = water["Potability"])
```

```
print(log_reg.intercept_, log_reg.coef_)
```

```
## [-0.53109022] [[ 2.28776554e-04  1.65799556e+00 -6.75416560e-03]]
```

NC STATE UNIVERSITY

Visualizing the Interaction Model Fit

```
to_pred = pd.DataFrame(np.array([[i, 1 if j == 1 else 0] for i in range(50, 330) for j in range(2)]),
                       columns = ["Hardness", "highChl"])
to_pred.head()
```

```
##    Hardness  highChl
## 0        50        0
## 1        50        1
## 2        51        0
## 3        51        1
## 4        52        0
```

```
to_pred_int = poly.fit_transform(to_pred)
to_pred_int
```

```
## array([[ 50.,   0.,   0.],
##        [ 50.,   1.,  50.],
##        [ 51.,   0.,   0.],
##        ...,
##        [328.,   1., 328.],
##        [329.,   0.,   0.],
##        [329.,   1., 329.]])
```

NC STATE UNIVERSITY

Visualizing the Interaction Model Fit

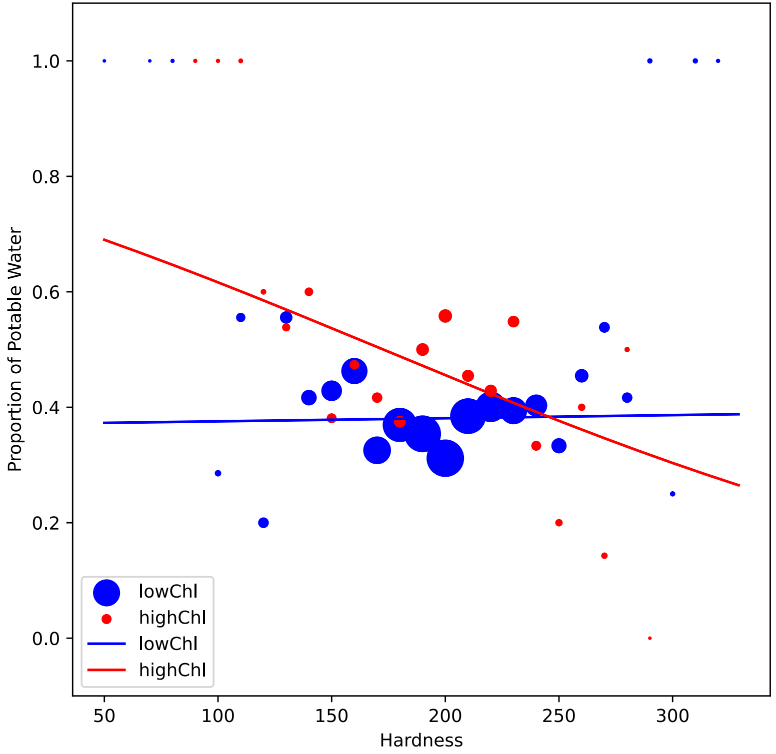
```
to_pred = pd.DataFrame(np.array([[i, 1 if j == 1 else 0] for i in range(50, 330) for j in range(2)]),
                       columns = ["Hardness", "highChl"])
to_pred.head()
to_pred_int = poly.fit_transform(to_pred)
```

```
pred_probs = pd.DataFrame(log_reg.predict_proba(to_pred_int))
pred_probs
```

```
##             0         1
## 0    0.627066  0.372934
## 1    0.309890  0.690110
## 2    0.627013  0.372987
## 3    0.311287  0.688713
## 4    0.626959  0.373041
## ..        ...       ...
## 555  0.732412  0.267588
## 556  0.612077  0.387923
## 557  0.733689  0.266311
## 558  0.612023  0.387977
## 559  0.734962  0.265038
##
## [560 rows x 2 columns]
```

Visualizing the Interaction Model Fit

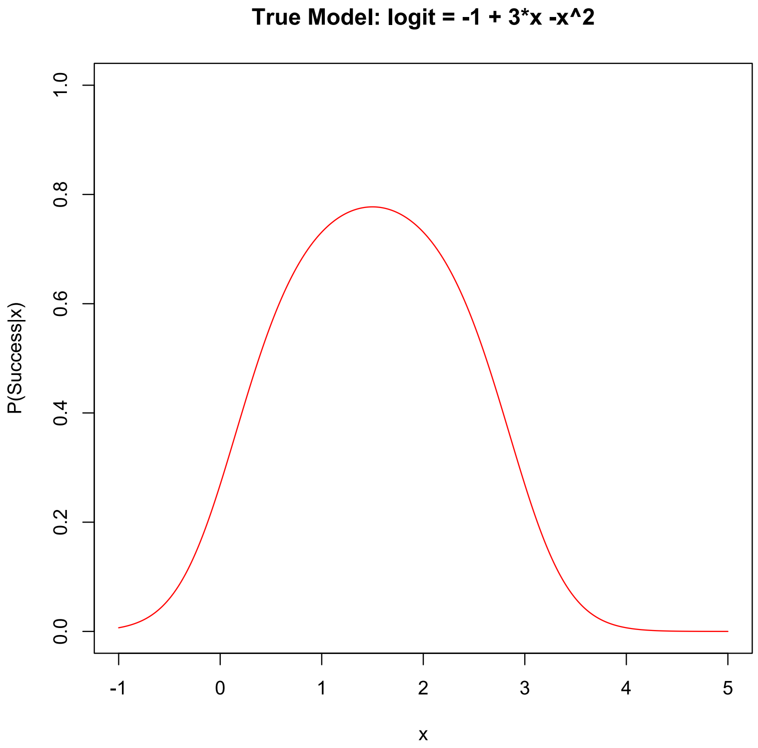
```
## (-0.1, 1.1)
```



NC STATE UNIVERSITY

Logistic Regression with Polynomial Term

- Adding in polynomial terms increases flexibility as well!



Selecting a Model

- Recall we can use k-fold CV as a proxy for **test set** error if we don't want to split the data
- Metric to quantify prediction quality? Basic measures:
    - Accuracy:

$$\frac{\#\text{ of correct classifications}}{\text{Total }\#\text{ of classifications}}$$

    - Misclassification Rate:

$$\frac{\#\text{ of incorrect classifications}}{\text{Total }\#\text{ of classifications}}$$

Selecting a Model

- Recall we can use k-fold CV as a proxy for **test set** error if we don't want to split the data
- Metric to quantify prediction quality? Basic measures:
    - Accuracy:

$$\frac{\#\text{ of correct classifications}}{\text{Total }\#\text{ of classifications}}$$

    - Misclassification Rate:

$$\frac{\#\text{ of incorrect classifications}}{\text{Total }\#\text{ of classifications}}$$

    - Log-loss: For each observation (y = 0 or 1), $-(ylog(\hat{p}) + (1-y)log(1-\hat{p}))$

Selecting a Model

- Accuracy is used by default here

```
from sklearn.model_selection import cross_validate
log_reg1 = LogisticRegression(penalty = 'none')
cv1 = cross_validate(log_reg1,
      X = water[["Hardness", "highChl"]],
      y = water["Potability"].values,
      cv = 5)
cv1['test_score']
```

```
## array([0.6097561 , 0.61068702, 0.61068702, 0.60916031, 0.61679389])
```

Selecting a Model

- Fit a couple more models and compare CV accuracy

```
cv2 = cross_validate(log_reg1,
      water[["Hardness", "Solids", "Chloramines", "Conductivity", "Organic_carbon"]].values,
      y = water["Potability"].values,
      cv = 5)
cv2['test_score']
```

```
## array([0.6097561 , 0.61068702, 0.61068702, 0.60916031, 0.60916031])
```

NC STATE UNIVERSITY

Selecting a Model

- Likely want to do some scaling when using polynomials...

```
log_reg2 = LogisticRegression(penalty = 'none', solver = "lbfgs", max_iter = 5000)
poly = PolynomialFeatures(interaction_only=True, include_bias = False)
poly.fit_transform(water[["Hardness", "Solids", "Chloramines"]])
```

```
## array([[2.04890455e+02, 2.07913190e+04, 7.30021187e+00, 4.25994282e+06,
##          1.49574374e+03, 1.51781034e+05],
##         [1.29422921e+02, 1.86300579e+04, 6.63524588e+00, 2.41115650e+06,
##          8.58752901e+02, 1.23615015e+05],
##         [2.24236259e+02, 1.99095417e+04, 9.27588360e+00, 4.46444116e+06,
##          2.07998944e+03, 1.84678592e+05],
##         ...,
##         [1.75762646e+02, 3.31555782e+04, 7.35023323e+00, 5.82751217e+06,
##          1.29189644e+03, 2.43701233e+05],
##         [2.30603758e+02, 1.19838694e+04, 6.30335653e+00, 2.76352531e+06,
##          1.45357770e+03, 7.55386013e+04],
##         [1.95102299e+02, 1.74041771e+04, 7.50930586e+00, 3.39559495e+06,
##          1.46508283e+03, 1.30693289e+05]])
```

```
cv3 = cross_validate(log_reg2,
      poly.fit_transform(water[["Hardness", "Solids", "Chloramines"]]),
      y = water["Potability"].values, cv = 5)
```

Selecting a Model

- Compare models
    - Can average accuracy measures here since we have basically the same number of observations in each fold

```
[round(cv1['test_score'].mean(),4), round(cv2['test_score'].mean(),4), round(cv3['test_score'].mean(),4)]
```

```
## [0.6114, 0.6099, 0.6084]
```

Selecting a Model

- Compare models
    - Can average accuracy measures here since we have basically the same number of observations in each fold

```
[round(cv1['test_score'].mean(),4), round(cv2['test_score'].mean(),4), round(cv3['test_score'].mean(),4)]
```

```
## [0.6114, 0.6099, 0.6084]
```

- Note: Proportion of non-potable water samples is `1998/(1998+1278) = 0.6099`
    - Our best model is just barely better than always guessing non-potable!

Selecting a Model

- Redo with `neg-log-loss` metric!
- Takes into account probability being modeled, not just binary classification
- Returns 'mean loss' by default

```
cv1 = cross_validate(log_reg1,
      water[["Hardness", "highChl"]],
      y = water["Potability"].values,
      cv = 5,
      scoring = "neg_log_loss")
cv1['test_score']
```

```
## array([-0.66620539, -0.66721921, -0.67092199, -0.66706474, -0.67022251])
```

Selecting a Model

```
cv2 = cross_validate(log_reg1,
      water[["Hardness", "Solids", "Chloramines", "Conductivity", "Organic_carbon"]],
      y = water["Potability"].values,
      cv = 5,
      scoring = "neg_log_loss")
cv2['test_score']
```

```
## array([-0.66856635, -0.66667087, -0.66694074, -0.66831083, -0.6707505 ])
```

```
cv3 = cross_validate(log_reg2,
      poly.fit_transform(water[["Hardness", "Solids", "Chloramines"]]),
      y = water["Potability"].values,
      cv = 5,
      scoring = "neg_log_loss")
cv3['test_score']
```

```
## array([-0.6726622 , -0.69241936, -0.68022034, -0.66760934, -0.6788677 ])
```

Selecting a Model

- Compare models
    - Can average metrics here since each fold has same number of values (roughly)

```
[round(cv1['test_score'].mean(),4), round(cv2['test_score'].mean(),4), round(cv3['test_score'].mean(),4)]
```

```
## [-0.6683, -0.6682, -0.6784]
```

Selecting a Model

- Compare models
    - Can average metrics here since each fold has same number of values (roughly)

```
[round(cv1['test_score'].sum(),4), round(cv2['test_score'].sum(),4), round(cv3['test_score'].sum(),4)]
```

```
## [-3.3416, -3.3412, -3.3918]
```

- Compare to `neg_log_loss` applied to always predicting non-potable with probability 1

```
from sklearn.metrics import log_loss
#returns 'mean loss per sample' by default
-log_loss(water["Potability"].values, np.array([[1,0] for _ in range(len(water["Potability"]))]))
```

```
## -13.473918263948669
```

- We do much better here!

Recap

- With a binary response variable, logistic regression can be used
- Model probability using a non-linear function
  - Can include polynomial terms, categorical variables via dummy variables, interactions, …
- Fit model with `LogisticRegression()`
- Can still use `cross_validate()` to select model
  - Commonly use accuracy/missclassification or log-loss as the loss function

Recap

- With a binary response variable, logistic regression can be used
- Model probability using a non-linear function
  - Can include polynomial terms, categorical variables via dummy variables, interactions, ...
- Fit model with `LogisticRegression()`
- Can still use `cross_validate()` to select model
  - Commonly use accuracy/missclassification or log-loss as the loss function

Note: Logistic Regression falls into a family of Generalized Linear Models (GLMs):

- Allows for responses from non-normal distributions