Logistic Regression

Data Mining for Business Analytics in Python

Shmueli, Bruce, Gedeck & Patel

 rev 11/26/19

Logistic Regression

- Extends idea of linear regression to situation where outcome variable is categorical
- Widely used, particularly where a structured model is useful to explain (=*profiling*) or to predict
- We will do both binary and multi-class classification

Video-1

Video-2

Python Functionality Needed

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression,
    LogisticRegressionCV
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from mord import LogisticIT
import matplotlib.pylab as plt
import seaborn as sns
from dmba import classificationSummary, gainsChart, liftChart
from dmba.metric import AIC_score
```

The Logit

Goal: Find a function of the predictor variables that relates them to a 0/1 outcome

- Instead of *Y* as outcome variable (like in linear regression), we use a function of Y called the ***logit***
- Logit can be modeled as a linear function of the predictors
- The logit can be mapped back to a probability, which, in turn, can be mapped to a class

Logistic Regression Predictor Types

- **Binary Variables**: These are suitable and commonly used, representing two categories (e.g., 0 and 1).
- **Categorical Variables**: If you have categorical predictors with more than two categories, you can convert them into binary dummy variables using pd.get_dummies(). This allows you to represent each category as a binary feature.
- **Continuous Variables**: Logistic regression can also include continuous predictors (e.g., age, income).

Step 1: Logistic Response Function

p = Probability of belonging to class 1

Need to relate p to predictors with a function that guarantees $0 \leq p \leq 1$

Standard linear function (as shown below) does not:

$$p = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots \beta_q x_q$$

q = number of predictors

The Fix:
use ***logistic response function***

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots \beta_q x_q)}}$$

The logistic function is an S-shaped curve that transforms a linear combination of input features into a value between 0 and 1.

Step 2: The Odds*

The odds represent the likelihood or probability of an event occurring relative to the likelihood of it not occurring. The odds are often expressed as a ratio of the probability of success to the probability of failure.
The odds of an event are defined as:

$$Odds = \frac{p}{1-p}$$

p = Probability of event

Or, given the odds of an event, the probability of the event can be computed by:

$$p = \frac{Odds}{1+Odds}$$

We can also relate the Odds to the predictors:

$$Odds = e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\square+\beta_q x_q}$$

- This metric is very popular in horse races, sports, gambling, epidemiology, and other areas. Instead of talking about the probability of winning or contacting a disease, people talk about the odds of winning or contacting a disease. How are these two different? If, for example, the probability of winning is 0.5, the odds of winning are 0.5/0.5 = 1. If the probability of winning is 0.2, the odds of winning are 0.2/0.8 = 0.25.
- Odds takes values from 0 to + infinity.
- Video-1
- Video-2

Step 3: Take log on both sides

This gives us the logit:

$$\log(\text{odds}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_q x_q$$

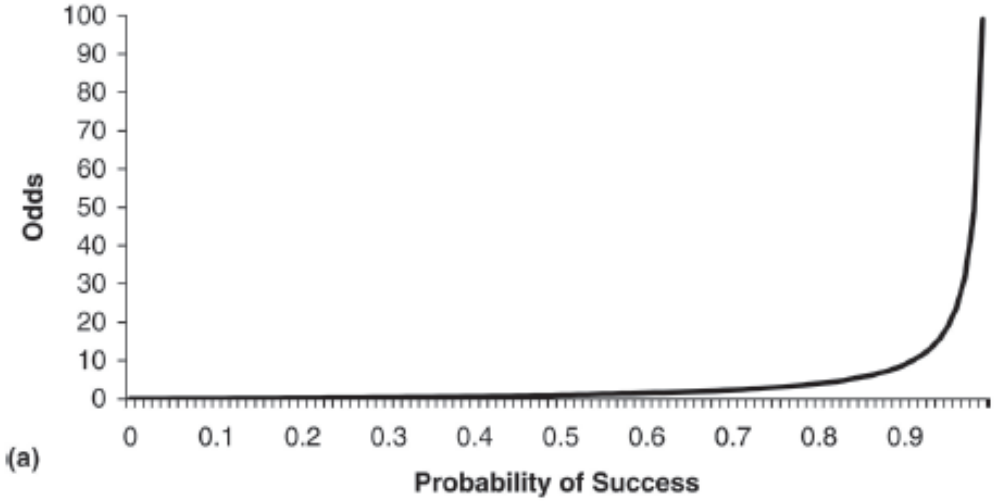
So, the logit is a linear function of predictors x_1, x_2, ...

- Takes values from -infinity to +infinity

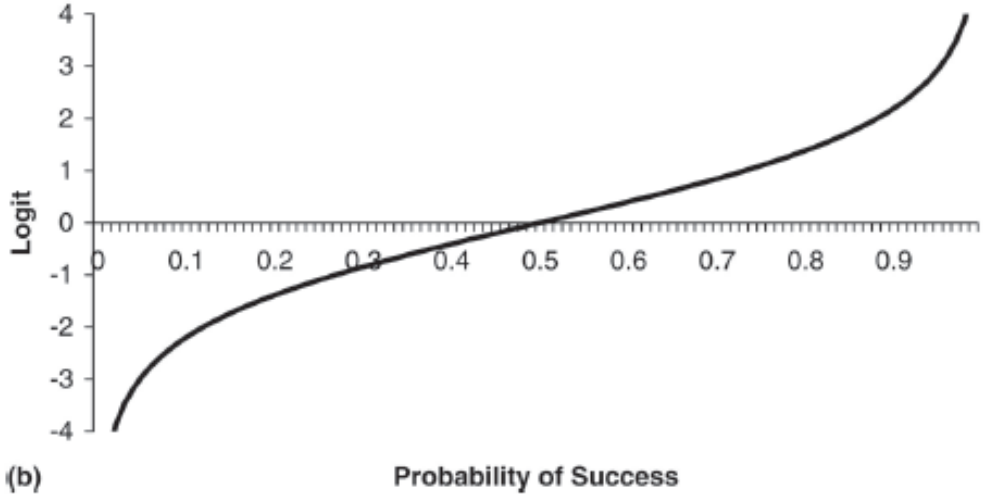
Review the relationship between logit, odds and probability

Odds (a) and Logit (b) as a function of *P*

Odds can take any non negative value.



Logit can take any real value.



Example: Personal Loan Offer

(UniversalBank.csv)

Outcome variable: Accept bank loan (0/1)

Predictors: Demographic info, and info about their bank relationship

Single Predictor Model

Modeling loan acceptance on income (x)

$$\text{Prob}(Personal\ Loan = Yes \mid Income = x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

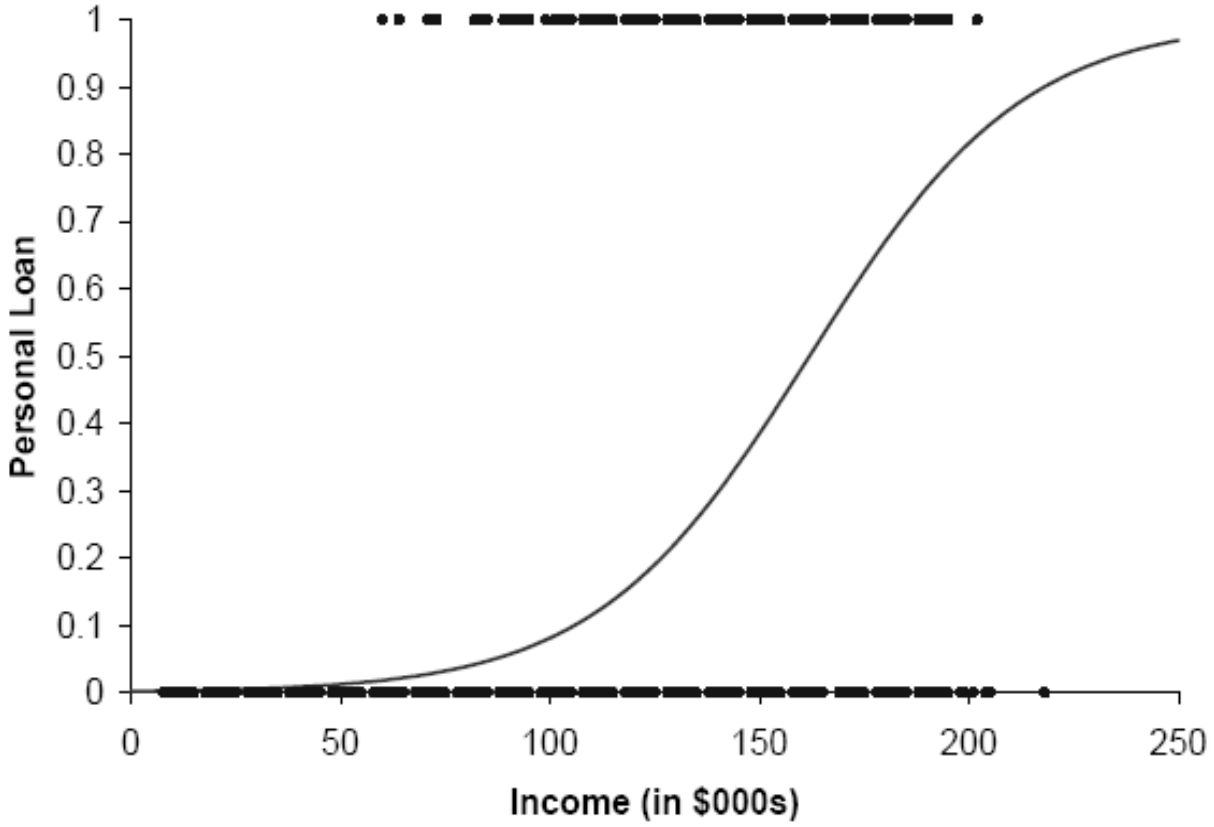
In terms of odds: Odds(Personal Loan = Yes | Income = x) = $e^{\beta_0 + \beta_1 x}$

Fitted coefficients (more later): b_0 = -6.3525,
b_1 = -0.0392

$$P(Personal\ Loan = Yes \mid Income = x) = \frac{1}{1 + e^{6.3525 - 0.0392x}}$$

Plot of data points
(Personal Loan as a function of Income)
and the fitted logistic curve

$$P(Personal\ Loan = Yes \mid Income = x) = \frac{1}{1 + e^{6.3525 - 0.0392x}}$$



Last step - Classify

Model produces an estimated probability of being a “1”

- Convert to a classification by establishing cutoff level
- If estimated prob. > cutoff, classify as “1”

Ways to Determine Cutoff

- 0.50 is popular initial choice
- Additional considerations
  - Maximize classification accuracy
  - Maximize sensitivity (subject to min. level of specificity)
  - Minimize false positives (subject to max. false negative rate)
  - Minimize expected cost of misclassification (need to specify costs)

Example, cont.

- Estimates of β's are derived through an iterative process called *maximum likelihood estimation*
- Let's include all 12 predictors in the model now

Video

Data Preparation

```
bank_df = pd.read_csv('UniversalBank.csv')
bank_df.drop(columns=['ID', 'ZIP Code'], inplace=True)
bank_df.columns = [c.replace(' ', '_') for c in bank_df.columns]

# Treat education as categorical, convert to dummy variables
bank_df['Education'] = bank_df['Education'].astype('category')
new_categories = {1: 'Undergrad', 2: 'Graduate', 3:
   'Advanced/Professional'} # This is a dictionary
bank_df.Education =
bank_df.Education.cat.rename_categories(new_categories)
bank_df = pd.get_dummies(bank_df, prefix_sep='_', drop_first=True)

X = bank_df.drop(columns=['Personal_Loan'])
y = bank_df['Personal_Loan']
```

Fitting Model

```
# partition data
train_X, valid_X, train_y, valid_y = train_test_split(X, y,
   test_size=0.4, random_state=1)

# fit a logistic regression (set penalty=l2 and C=1e42 to avoid regularization)
logit_reg = LogisticRegression(penalty="l2", C=1e42, solver='liblinear')
logit_reg.fit(train_X, train_y)
print('intercept ', logit_reg.intercept_[0])
print(pd.DataFrame({'coeff': logit_reg.coef_[0]},
   index=X.columns).transpose())
print('AIC', AIC_score(valid_y, logit_reg.predict(valid_X), df =
   len(train_X.columns) + 1))

# After fitting the model
log_odds = logit_reg.intercept_[0] + np.dot(valid_X, logit_reg.coef_[0])
odds = np.exp(log_odds)
print('Odds for validation set:', odds)
```

Fitting Model

- Penalty: {'l1', 'l2', 'elasticnet', None}, default='l2': Add a L2 penalty term which corresponds to Ridge regularization (and L1 is Lasso regularization). This adds a penalty term proportional to the square of the magnitude of the coefficients to the logistic regression objective function. 'elasticnet': Both L1 and L2 penalty terms are added.
- The C parameter is the inverse of the regularization strength. Larger values of C correspond to weaker regularization. Setting C to a very large value (like 1e42) essentially turns off regularization.
- 'solver' parameter specifies the algorithm to use in the optimization problem. 'liblinear' is suitable for small to medium-sized datasets and supports both L1 and L2 regularization. For larger datasets, consider using 'saga'. Other solvers: newton-cg, lbfgs, sag.
- Here is why a high C with an L2 penalty can be used:

  1. Data Complexity: If the data is complex and the relationships between the features and the target variable are intricate, a high C value can help the model capture those complexities more effectively.

  2. Large Sample Size: In situations where you have a large amount of training data, a high C value may be appropriate. With more data, the model has a better chance of generalizing well even with weak regularization.

  3. Underfitting: If you find that your model is underfitting the training data, meaning it is too simple and not capturing important patterns, a high C can be used to relax the regularization and allow the model to become more flexible.

  4. Emphasis on Training Data: Sometimes, you want the model to fit the training data closely and prioritize minimizing training error over preventing overfitting.

  In such cases, you might question whether it is necessary to use both a high C and L2 regularization if they effectively cancel each other out. Here are some considerations:

  1. Model Flexibility: The choice of C and regularization depends on the flexibility you want in your model. If you remove both C and L2 regularization, the logistic regression model becomes unregularized and can fit the training data closely. This may lead to overfitting, especially if your dataset is small or noisy.

  2. Interpretability: Regularization, even when applied with a relatively high C, can still help improve model stability and generalization without sacrificing the model's interpretability. Removing all regularization can result in a model that is overly complex and harder to interpret.

  3. Data Complexity: In some cases, even with high C, a small amount of regularization (such as L2) can help prevent the model from becoming overly complex and overfitting to the training data, particularly when dealing with complex datasets.

  4. Cross-Validation: It is essential to determine the optimal value of C through techniques such as cross-validation. Instead of setting it to an extremely high value, you can choose a moderate C that provides a good balance between model complexity and generalization.
- AIC and BIC may serve different purposes in different contexts. AIC might be more appropriate when you want to balance model fit and complexity, while BIC might be more appropriate when you prioritize model simplicity.

Results

intercept -12.6189552131403 5

	Age	**Experience**	**Income**	**Family**	**CCAvg**	**Mortgage**
coeff	-0.032549	0.03416	0.058824	0.614095	0.240534	0.001012

	Securities_Account	**CD_Account**	**Online**	**CreditCard**
coeff	-1.026191	3.647933	-0.677862	-0.95598

	Education_Graduate	**Education_Advanced/Professional**
coeff	4.192204	4.341697

AIC -709.1524769205962

In the context of the Akaike Information Criterion (AIC), the goal is to find a balance between model goodness of fit and model complexity. The AIC is a measure used for model selection, and lower AIC values are preferred. Note that AIC score itself does not have an absolute interpretation. Rather, it is useful for comparing different models, built based on various regularization methods, C values, etc.

1. Lower AIC values indicate a better balance between goodness of fit and model complexity.
2. Models with lower AIC are considered more parsimonious, meaning they achieve a good fit to the data with fewer parameters.
3. When comparing models, the model with the lowest AIC is preferred, as it suggests a better trade-off between explaining the data and avoiding overfitting.

Converting from logit to probabilities

```
odds_result = pd.DataFrame({'actual': valid_y, 'odds': odds})
interestingCases = [2764, 932, 2721, 702] # Four random cases to see
print(odds_result.loc[interestingCases])

logit_reg_pred = logit_reg.predict(valid_X)
logit_reg_proba = logit_reg.predict_proba(valid_X)
logit_result = pd.DataFrame({'actual': valid_y,
   'p(0)': [p[0] for p in logit_reg_proba],
   'p(1)': [p[1] for p in logit_reg_proba],
   'predicted': logit_reg_pred })
print(logit_result.loc[interestingCases])
```

	actual	odds
2764	0	0.024493
932	0	2.021145
2721	1	30.816350
702	1	0.014309

Use: $p = \frac{Odds}{1+Odds}$

For example, $\frac{0.024493}{1+0.024493}$ = 0.024

	actual	p(0)	p(1)	predicted
2764	0	0.976	0.024	0
932	0	0.335	0.665	1
2721	1	0.032	0.968	1
702	1	0.986	0.014	0

Interpreting Odds, Probability

- For predictive classification, we typically use probability with a cutoff value

- For explanatory purposes, odds have a useful interpretation:

  - If we increase x_1 by one unit, holding x_2, x_3 … x_q constant, then
  - b_1 is the factor by which the odds of belonging to class 1 increases

Suppose we want to know how increasing family income by one unit will affect the probability of loan acceptance. This can be found straightforwardly if we consider not probabilities, but odds.

Loan Example: Evaluating Classification Performance

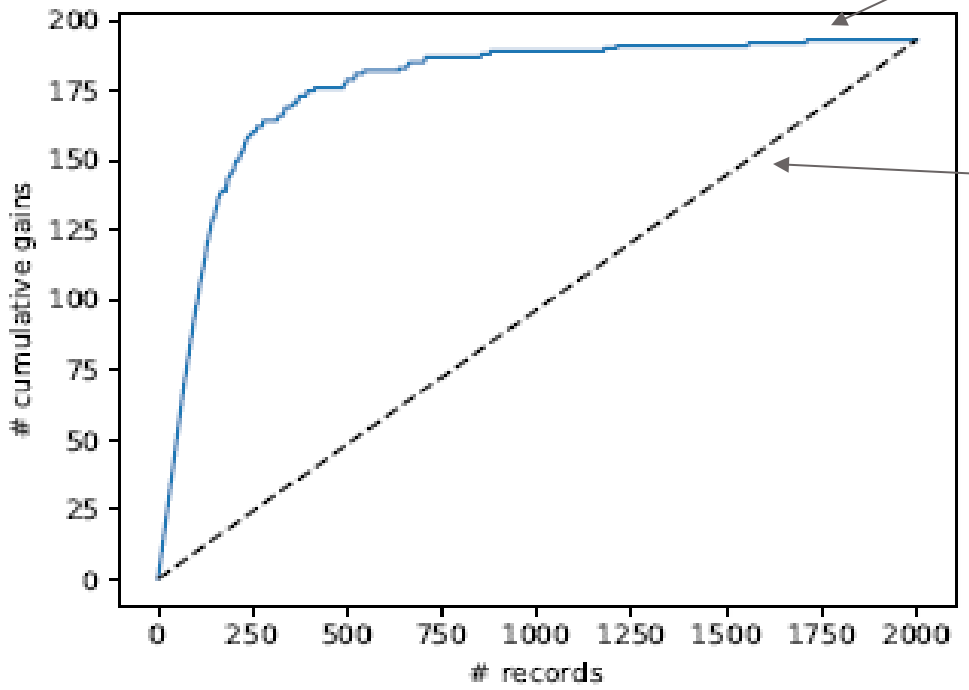
Performance measures: Confusion matrix and % of misclassifications

```
# training confusion matrix
classificationSummary(train_y, logit_reg.predict(train_X))
# validation confusion matrix
classificationSummary(valid_y, logit_reg.predict(valid_X))
Confusion Matrix (Accuracy 0.9603)
       Prediction
Actual    0    1
     0 2684   29
     1   90  197
Confusion Matrix (Accuracy 0.9595)
       Prediction
Actual    0    1
     0 1791   16
     1   65  128
```

More useful in this example: **Gains (Lift)** (terms sometimes used interchangeably)

Python's Gains Chart

```
df = logit_result.sort_values(by=['p(1)'], ascending=False)
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))
gainsChart(df.actual, ax=axes[0])
liftChart(df['p(1)'], title=False, ax=axes[1])
plt.show()
```



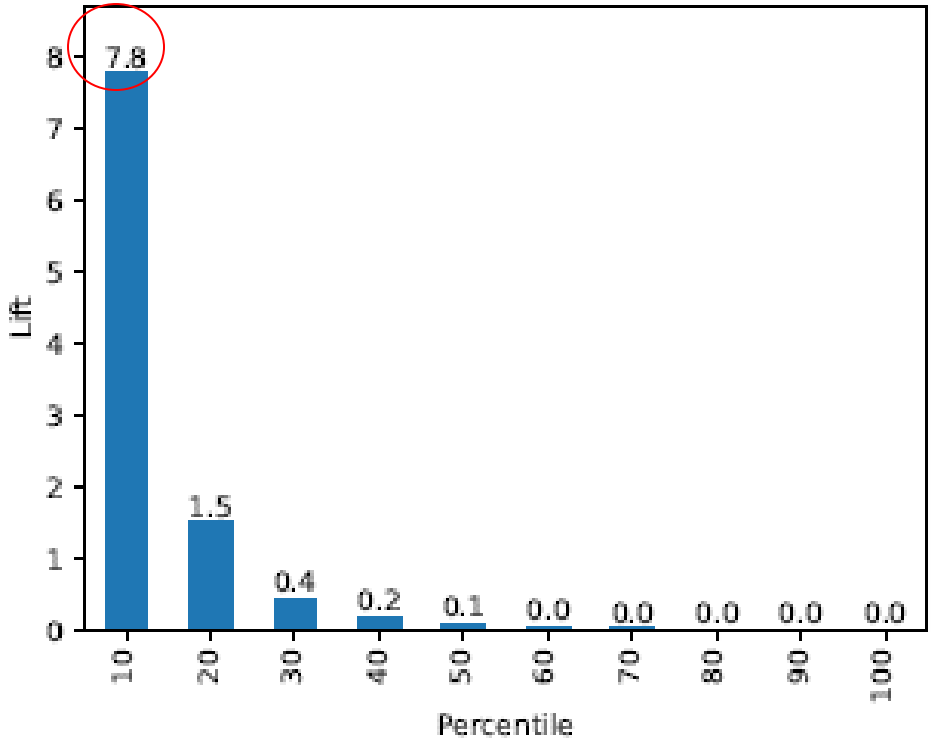
# of 1's yielded by model, moving thru records sorted by predicted prob. of being a 1

# of 1's yielded by selecting randomly

Python's Lift Chart

```
df = logit_result.sort_values(by=['p(1)'], ascending=False)
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))
gainsChart(df.actual, ax=axes[0])
liftChart(df['p(1)'], title=False, ax=axes[1])
plt.show()
```

Top decile (i.e. the 10% most probable to be 1's) are 7.8 times as likely to be 1's, compared to random selection



Multicollinearity

Problem: As in linear regression, if one predictor is a linear combination of other predictor(s), model estimation will fail

- Note that in such a case, we have at least one redundant predictor

Solution: Remove extreme redundancies (by dropping predictors via variable selection, or by data reduction methods such as PCA)

Variable Selection

This is the same issue as in linear regression

- The number of correlated predictors can grow when we create derived variables such as **interaction terms** (e.g. *Income x Family)*, to capture more complex relationships
- Problem: Overly complex models have the danger of overfitting
- Solution: Reduce variables via automated selection of variable subsets (as with linear regression)
- See “Dimension Reduction”

P-values for Predictors

- Test null hypothesis that coefficient = 0
- Useful for review to determine whether to include variable in model
- Important in profiling tasks (explaining the model structure), but less important in predictive classification

Logistic Regression for Multi-class Classification

- The logistic model for a binary outcome can be extended for more than two classes. Suppose that there are m classes. Using a logistic regression model, for each record we would have m probabilities of belonging to each of the m classes. Since the m probabilities must add up to 1, we need to estimate only $m - 1$ probabilities.

1. Ordinal Classes
2. Nominal Classes

Logistic Regression for Multi-class Classification

1. Ordinal Classes: They have a meaningful order. For example, in stock recommendations, the three classes buy, hold, and sell can be treated as ordered. As a simple rule, if classes can be numbered in a meaningful way, we consider them ordinal. When the number of classes is large (typically, more than 5), we can treat the outcome variable as continuous and perform multiple linear regression. When m = 2, the logistic model described is used. We therefore need an extension of the logistic regression for a small number of ordinal classes ($3 \leq m \leq 5$). There are several ways to extend the binary-class case. Here, we describe the proportional odds or cumulative logit method. For the three-class case, for example, we would have:

$$P(Y = 1) = P(Y \leq 1) = \frac{1}{1 + e^{-(a_0 + b_1 x)}},$$

$$P(Y = 2) = P(Y \leq 2) - P(Y \leq 1) = \frac{1}{1 + e^{-(b_0 + b_1 x)}} - \frac{1}{1 + e^{-(a_0 + b_1 x)}},$$

$$P(Y = 3) = 1 - P(Y \leq 2) = 1 - \frac{1}{1 + e^{-(b_0 + b_1 x)}},$$

where a_0, b_0, and b_1 are the estimates obtained from the training set.

For each record, we now have the estimated probabilities that it belongs to each of the classes.

Example: If an event has P(Y=buy) = 0.2, P(Y=hold) = 0.3, and P(Y=sell) = 0.5, it is classified in the third category.

Logistic Regression for Multi-class Classification

1. Ordinal Classes:

More than one threshold (θ) is needed in case of multi-class classification. The general cumulative form which incorporates thresholds, and Python uses, is shown below:

$$\log\left(\frac{P(Y \leq j)}{P(Y > j)}\right) = \theta_j - (\beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k)$$

$$P(Y \leq j) = \frac{1}{1 + e^{-(\theta_j - (\beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_k X_k))}}$$

The general individual probabilities are shown below.

For the first class (j=1): $P(Y = 1) = P(Y \leq 1)$

For subsequent classes (j=2, 3, J): $P(Y = j) = P(Y \leq j) - P(Y \leq j - 1)$

Where j represents outcome variable classes – 1 (j = 1 : J – 1, J = Total # of outcome classes)

For example, if J = 3, j = 2.

The probability for last class is calculated as:

$$P(Y \leq J) = 1 - P(Y \leq J - 1)$$

Logistic Regression for Multi-class Classification

2. **Nominal Classes:** When the classes cannot be ordered and are simply different from one another, we are in the case of nominal classes. An example is the choice between several brands of cereal. A simple way to verify that the classes are nominal is when it makes sense to tag them as A, B, C, …, and the assignment of letters to classes does not matter. For simplicity, let us assume that there are m = 3 brands of cereal that consumers can choose from (assuming that each consumer chooses one). Then we estimate the probabilities P(Y = A), P(Y = B), and P(Y = C). As before, if we know two of the probabilities, the third probability is determined. We therefore use one of the classes as the reference class. Let us use C as the reference brand.

   The goal, once again, is to model the class membership as a function of predictors. So in the cereals example, we might want to predict which cereal will be chosen if we know the cereal's price x.

$$P(Y = A) = \frac{e^{a_0 + a_1 x}}{1 + e^{a_0 + a_1 x} + e^{b_0 + b_1 x}},$$

$$P(Y = B) = \frac{e^{b_0 + b_1 x}}{1 + e^{a_0 + a_1 x} + e^{b_0 + b_1 x}},$$

$$P(Y = C) = 1 - P(Y = A) - P(Y = B),$$

   where a_0, a_1, b_0, and b_1 are the coefficient estimates obtained from the training set.

   A record is assigned to the class that has the highest probability.

   Example: If an event has P(Y=brandA) = 0.2, P(Y=brandB) = 0.3, and P(Y=BrandC) = 0.5, it is classified in the third category.

Comparing Ordinal and Nominal Models

1. The estimated logistic models will differ, depending on whether the outcome variable is coded as ordinal or nominal. Let`s take, for example, data on accidents from the US Bureau of Transportation Statistics. This data can be used to predict whether an accident will result in injuries or fatalities based on predictors such as alcohol involvement, time of day, and road condition. The severity of the accident has three classes: 0 = No Injury, 1 = Nonfatal Injuries, 2 = Fatalities.

2. We first consider severity as an ordinal variable, and fit a simple model with two nominal predictors, alcohol and weather. Each of these predictors is coded as No (not involved in the accident) or Yes (involved in the accident). Next, we change the modeling type for severity to nominal, and fit a model with the same predictors. The reference severity level, in both models, is severity = 2.

Comparing Ordinal and Nominal Models

Python code for ordinal and nominal multinomial regression applied to the accidents data, displaying the resulting estimated models and predicted probabilities for a few sample records.

```
data = pd.read_csv('accidentsFull.csv')
outcome = 'MAX_SEV_IR'
predictors = ['ALCHL_I', 'WEATHER_R']

y = data[outcome]
X = data[predictors]
train_X, valid_X, train_y, valid_y = train_test_split(X, y,  test_size=0.4, random_state=1)

print('Nominal logistic regression')
logit = LogisticRegression(penalty="l2", solver='lbfgs', C=1e24, multi_class='multinomial')
# multi_class='multinomial': Specifies that the logistic regression model should handle multiple classes using the multinomial logistic regression approach. This is relevant when you # have more than two classes in your target variable.
logit.fit(X, y)
print('  intercept', logit.intercept_)
print('  coefficients', logit.coef_)
print()
probs = logit.predict_proba(X)
results = pd.DataFrame({'actual': y, 'predicted': logit.predict(X), 'P(0)': [p[0] for p in probs], 'P(1)': [p[1] for p in probs], 'P(2)': [p[2] for p in probs]})
print(results.head())
print()

print('Ordinal logistic regression')
logit = LogisticIT(alpha=0) # Classifier that implements the ordinal logistic model (All-Threshold variant) (alpha: Regularization parameter. Zero is no regularization)
logit.fit(X, y)
print('  theta', logit.theta_)
print('  coefficients', logit.coef_)
print()
probs = logit.predict_proba(X)
results = pd.DataFrame({'actual': y, 'predicted': logit.predict(X), 'P(0)': [p[0] for p in probs], 'P(1)': [p[1] for p in probs], 'P(2)': [p[2] for p in probs]})
print(results.head())
```

Comparing Ordinal and Nominal Models

Nominal logistic regression

intercept [-0.17870851 0.81673248 -0.63802397]
coefficients [[0.52491466 0.40469045]
[0.15763357 0.16071816]
[-0.68254823 -0.56540861]]

	actual	predicted	P(0)	P(1)	P(2)
0	1	1	0.490569	0.498933	0.010498
1	0	0	0.554023	0.441483	0.004494
2	0	0	0.554023	0.441483	0.004494
3	0	1	0.490569	0.498933	0.010498
4	0	1	0.393700	0.578117	0.028183

Ordinal logistic regression

theta [-1.06916285 2.77444326]
coefficients [-0.40112008 -0.25174207]

	actual	predicted	P(0)	P(1)	P(2)
0	1	1	0.496205	0.482514	0.021281
1	0	0	0.558866	0.424510	0.016624
2	0	0	0.558866	0.424510	0.016624
3	0	1	0.496205	0.482514	0.021281
4	0	1	0.397402	0.571145	0.031453

The ordinal logistic model has estimates for two intercepts (the first for severity = 0 and and the second for severity = 1), but one estimate for each predictor. In contrast, the nominal logistic model has estimates for three intercepts, and also has three separate sets of coefficients for the predictors for each level of the outcome variable.

Finally, the resulting predicted probabilities also differ across the two models. Hence, it is useful to determine which model is more suitable for the outcome at hand, and to evaluate predictive performance on a validation set.

Comparing Ordinal and Nominal Models

Nominal logistic regression

intercept [-0.17870851 0.81673248 -0.63802397]

coefficients [[0.52491466 0.40469045]
[0.15763357 0.16071816]
[-0.68254823 -0.56540861]]

Ordinal logistic regression

theta [-1.06916285 2.77444326]

coefficients [-0.40112008 -0.25174207]

	actual	predicted	P(0)	P(1)	P(2)
0	1	1	0.490569	0.498933	0.010498
1	0	0	0.554023	0.441483	0.004494
2	0	0	0.554023	0.441483	0.004494
3	0	1	0.490569	0.498933	0.010498
4	0	1	0.393700	0.578117	0.028183

	actual	predicted	P(0)	P(1)	P(2)
0	1	1	0.496205	0.482514	0.021281
1	0	0	0.558866	0.424510	0.016624
2	0	0	0.558866	0.424510	0.016624
3	0	1	0.496205	0.482514	0.021281
4	0	1	0.397402	0.571145	0.031453

Considering the first row of training data set where ALCHL_I = 2 and WEATHER_R = 1:

Manual Calculations for Nominal Regression:

$\log\_odds_0 = -0.17870851 + (0.52491466 * X_1) + (0.40469045 * X_2) = 1.2758$ $\log\_odds_1 = 1.2927$ $\log\_odds_2 = -2.568$

$e^{\log\_odds_0} = e^{1.2758} = 3.582$ $e^{\log\_odds_1} = e^{1.2927} = 3.628$ $e^{\log\_odds_2} = e^{-2.568} = 0.076$

$sum = e^{\log\_odds_0} + e^{\log\_odds_1} + e^{\log\_odds_2} = 7.287$

$P(0) = \frac{e^{\log\_odds_0}}{sum} = 0.49$ $P(1) = \frac{e^{\log\_odds_1}}{sum} = 0.49$ $P(2) = \frac{e^{\log\_odds_2}}{sum} = 0.01$

Manual Calculations for Ordinal Regression:

Individual/cumulative probability of Class 0: $P(Y = 0) = P(Y \leq 0) = \frac{1}{1+e^{-(\theta_0-(\beta_1 X_1+\beta_2 X_2))}} = \frac{1}{1+e^{-(-1.069-(-0.401)*2+(-0.251)*1)}} = 0.469$

Individual probability of Class 1: $P(Y = 1) = \frac{1}{1+e^{-(\theta_1-(\beta_1 X_1+\beta_2 X_2))}} - P(Y \leq 0) = \frac{1}{1+e^{-(2.774-(-0.401)*2+(-0.251)*1)}} - 0.469 = 0.978 - 0.469 = 0.482$

Individual probability of Class 2: $P(Y = 2) = P(Y \leq 2) - P(Y \leq 1) = 1 - 0.978 = 0.021$