Understanding Logistic Regression

Introduction to Logistic Regression

Logistic regression is a statistical method for predicting binary outcomes from data.

- Used when the dependent variable is categorical.
- Estimates the probability of occurrence of an event by fitting data to a logistic function.

Applications of Logistic Regression

Logistic regression is widely used in various fields:

1. Medicine: Predicting the likelihood of a patient having a certain disease based on symptoms and test results.
2. Finance: Credit scoring to predict the probability of a customer defaulting on a loan.
3. Marketing: Predicting customer churn based on usage patterns and customer feedback.
4. Political Science: Predicting election outcomes based on demographic data and past voting patterns.

Basic Concept

- The logistic model (or logit model) is used to model the probability of a certain class or event existing.
- Mathematical Form: $P(Y = 1) = \frac{1}{1+e^{-(\beta_0+\beta_1 X)}}$

Python Code Example: Fitting a Logistic Model

```
1  import pandas as pd
2  from sklearn.linear_model import
       LogisticRegression
3
4  # Sample data
5  data = pd.DataFrame({
6      'X': [value1, value2, ...],
7      'Y': [0, 1, ...]})
8
9  # Initialize and fit the logistic regression
       model
10 model = LogisticRegression()
11 model.fit(data[['X']], data['Y'])
```

Interpreting the Coefficients

The coefficients in logistic regression represent the change in the log odds of the outcome for a one unit change in the predictor variable, this is not the same as a normal linear regression model. However we can generalize:

- Positive coefficients mean a variable has a positive relationship with the target being equal to 1. If $\beta_1 > 0$ then an increase in X_1, all else held equal, increases the probability that $Y = 1$.
- Negative coefficients mean a variable has a negative relationship with the target being 1. If $\beta_1 < 0$ then an increase in X_1, all else held equal, decreases the probability that $Y = 1$.

Interpreting the Coefficients

Example: In logistic regression, coefficients indicate the direction and strength of the relationship between each predictor and the log odds of the outcome.

Model Coefficients:

Variable	Coefficient
Credit Score	0.03
Income Level	-0.02

Interpretation:

- **Credit Score:** A positive coefficient (0.03). An increase in the credit score, all else held equal, increases the probability of loan approval.
- **Income Level:** A negative coefficient (-0.02). An increase in income level, all else held equal, decreases the probability of loan approval. This might indicate that higher-income individuals are not the primary target for this specific loan product.

Using Statsmodels for P-values

In statsmodels, the summary of the fitted model includes p-values for each coefficient, which indicate the statistical significance.

```
import statsmodels.api as sm

# Assuming 'data' is a DataFrame with the predictor and outcome variables
X = sm.add_constant(data['Predictor']) # Add constant for intercept
model = sm.Logit(data['Outcome'], X).fit()

# Display summary to get p-values
print(model.summary())
```

Understanding Prediction in Logistic Regression

- In logistic regression, prediction means estimating the probability of an event occurring.
- The model outputs probabilities between 0 and 1.
- A threshold (commonly 0.5) is used to classify predictions into binary outcomes.

Understanding Prediction in Logistic Regression

Example: Consider a logistic regression model predicting whether a student passes (1) or fails (0) an exam based on their study hours. If a student studies for 3 hours, the model might predict a probability of passing as 0.7. Using a threshold of 0.5, we would classify this student as likely to pass (since $0.7 > 0.5$).

Measuring Model Accuracy

Model accuracy can be assessed using:

- Accurate Predictions - what percent of prediction were correct
- Confusion Matrix
- AUC Score

Introduction to the Confusion Matrix

A confusion matrix is a summary of the performance of a classification model.

- It shows the counts of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN).
- Essential for understanding the model's performance beyond just accuracy.

Labeling the Confusion Matrix

Standard Confusion Matrix

	Predicted: No	**Predicted: Yes**
Actual: No	True Negative (TN)	False Positive (FP)
Actual: Yes	False Negative (FN)	True Positive (TP)

Confusion Matrix with Error Types

	Predicted: No	**Predicted: Yes**
Actual: No	Correct Rejection	Type I Error
Actual: Yes	Type II Error	Correct Detection

Confusion Matrix Example: High Accuracy

Example Matrix: High Accuracy

	Predicted: No	**Predicted: Yes**
Actual: No	TN: 90	FP: 10
Actual: Yes	FN: 5	TP: 95

Interpretation:

- High number of true positives and negatives.
- Low number of false positives and negatives.
- Model is highly accurate in predicting both classes.

Confusion Matrix Example: Low Accuracy

Example Matrix: Low Accuracy

	Predicted: No	**Predicted: Yes**
Actual: No	TN: 50	FP: 50
Actual: Yes	FN: 45	TP: 55

Interpretation:

- Similar numbers of false positives and negatives compared to true positives and negatives.
- Indicates the model is not very accurate in differentiating between classes.

Confusion Matrix Example: Imbalanced Data

Example Matrix: Imbalanced Data

	Predicted: No	**Predicted: Yes**
Actual: No	TN: 95	FP: 5
Actual: Yes	FN: 50	TP: 50

Interpretation:

- High number of true negatives, but also a high number of false negatives.
- Model is biased towards predicting the majority class (No).
- Important in situations where failing to detect the positive class is costly (e.g., medical diagnosis).

Understanding the Importance of Specific Cells

- **Minimizing Type I Errors (FP):** Important when the cost of a false positive is high (e.g., spam filters).
- **Minimizing Type II Errors (FN):** Crucial when missing a positive is costly (e.g., medical tests).
- Balance between sensitivity (True Positive Rate) and specificity (True Negative Rate) depends on the application.

Python Code: Generating a Confusion Matrix

```
1 from sklearn.metrics import confusion_matrix
2
3 # Predicted values
4 predicted = model.predict(data[['X']])
5
6 # Generating the confusion matrix
7 cm = confusion_matrix(data['Y'], predicted)
```

Understanding AUC (Area Under the Curve)

AUC represents the area under the ROC (Receiver Operating Characteristic) curve. It provides an aggregated measure of performance across all possible classification thresholds.

Intuition:

- AUC measures the model's ability to discriminate between positive and negative classes.
- A higher AUC indicates a model is better at correctly classifying true positives and true negatives.
- Unlike accuracy, AUC takes into account the prediction probabilities, not just the final classification.

Comparing Models Using AUC: A Scenario

Scenario: Imagine a diagnostic tool for a rare disease. We have two models:

Model 1:

- Predicts 'Disease' with high confidence (e.g., 0.9 probability) in true cases. Meaning when it correctly predicts a true positive, it is very confident in that prediction.
- Accurate Predictions: 80%; AUC score: 0.85

Model 2:

- Predicts 'Disease' with moderate confidence (e.g., 0.6 probability) in true cases. Meaning when it correctly predicts a true positive, it is only moderately confident.
- Accurate Predictions: 80%; AUC score: 0.75

Comparing Models Using AUC: A Scenario

Comparison:

- Both models might have similar accuracy in predicting the disease.
- Model 1 has a higher AUC, indicating it is better at distinguishing between patients with and without the disease.
- In a medical context, higher confidence in correct predictions can be crucial.

Python Code: Calculating AUC

```
1 from sklearn.metrics import roc_auc_score
2
3 #Coompute the AUC
4 roc_auc_score((data['Y'], model.predict_proba(
      data[['X']])[:,1])
```

Introduction to Classification Trees

Classification trees are a type of decision tree used for categorizing instances into discrete classes. They are like regression tress but a little different.

Key Points:

- Splits the data according to the gini index instead of MSE.
- Essentially the gini index is a measure of how pure a node is. If a node contains only data where the target is 1, then the Gini index is 0. Likewise, if a node contains only data where the target is 1, then the Gini index is also 0. However, if it is perfectly split, the Gini index is 0.5.
- We want to split the nodes into as pure as possible so the splits are made to minimize the Gini index.
- Each leaf node represents a class label (decision taken after computing all attributes). For example if a leaf node is 80% data where the target is 0, we would label the predictions for that leaf node as 0, or we could say an 80% chance of being 0.

Python Code: Building and Evaluating a Classification Tree

```
1  from sklearn.tree import DecisionTreeClassifier
2  from sklearn.model_selection import
       train_test_split
3  from sklearn.metrics import roc_auc_score
4  import pandas as pd
5
6  # Split data
7  X_train, X_test, y_train, y_test =
       train_test_split(X, y, test_size=0.3)
8
9  # Build the tree model
10 tree_model = DecisionTreeClassifier()
11 tree_model.fit(X_train, y_train)
```

Comparing Models Using AUC

Comparing AUC Scores:

- AUC scores of both models reflect their ability to correctly classify positive and negative cases.
- A higher AUC score indicates better model performance.

Example Comparison:

- Classification Tree AUC: 0.75
- Logistic Regression AUC: 0.80
- Conclusion: Logistic Regression performs better in this scenario, with a higher AUC score.