Intermediate Data Science

Linear and Logistic Regression

Joanna Bieri DATA201

Important Information

- Email: joanna_bieri@redlands.edu
- Office Hours take place in Duke 209 – Office Hours Schedule
- Class Website
- Syllabus

Introduction to Linear and Logistic Regression

- **Linear Regression** - for predicting *continuous* outcomes (e.g., prices, scores). It is a true regression model where we are trying to fit a straight line to some data!
- **Logistic Regression** - for predicting *categorical* outcomes (e.g., yes/no, 0/1). This is a classification model! A way to take the idea of linear regression and turn it into classification.

Background: Linear Regression

The goal of linear regression is to model a relationship between one (or more) predictor variables x and a **continuous** target variable y. The simplest form (one predictor) is

$$\hat{y} = w_0 + w_1 x$$

Background: Linear Regression

For multiple predictors we would just have more variables:

$$\hat{y} = w_0 + w_1 x_1 + \cdots + w_p x_p$$

The goal is to find constants w_n that give us the best "fit" or prediction.

Background: Linear Regression

The common approach (ordinary least squares) is to choose the constants w_i to minimize the sum of squared errors or residuals:

$$\sum_{i=1}^{n}(y_i^{real} - y_i^{predicted})^2 = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

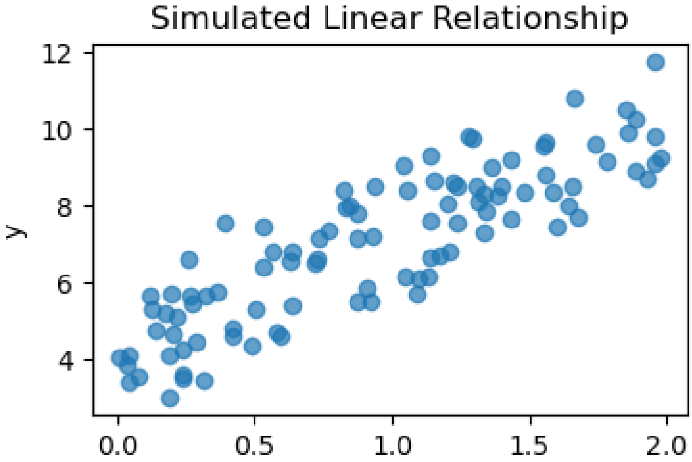
In this context the "machine learning" is figuring out what these constants should be!

Background: Linear Regression

In this basic formulation we are assuming that the data is in fact linear. My choosing a linear regression model this is your basic assumption. We will have ways to think, numerically, about whether or not this is a good assumption.

Linear Regression

Let's do linear regression on a fake dataset so we can see the overall process.



Linear Regression

We always want to split our data into a training and testing set so that we can text our model on data that it has not seen before!

```
X_train, X_test, y_train, y_test =
    train_test_split(X, y,
                     test_size=0.2,
                     random_state=42)
```

Linear Regression

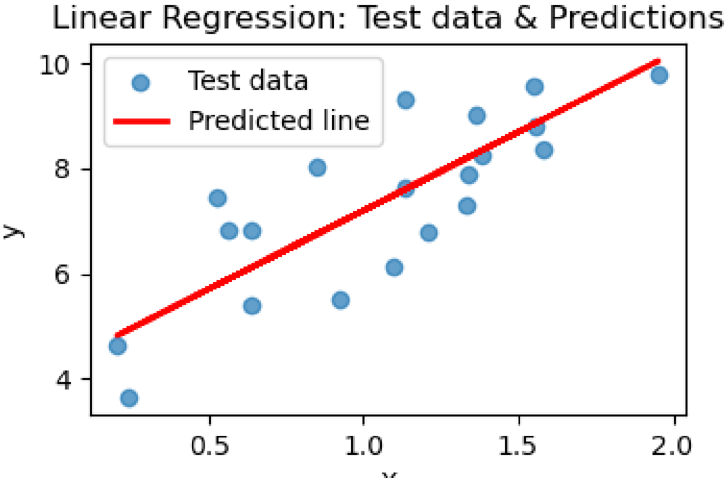
Then we define our model and train (fit) the model. In the case of linear regression with one variable we get two constants out of the model: the slope and the intercept.

```
linreg = LinearRegression()
linreg.fit(X_train, y_train)
```

```
Intercept (w0): 4.206340188711437
Coefficient (w1): 2.99025910100489
```

Linear Regression

```
MSE on test set: 0.918
R² on test set: 0.652
```



Understanding MSE and R^2 on the Test Set

Mean Squared Error (MSE)

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Measures the **average squared difference** between predicted values $(\hat{y}_i)$ and actual values (y_i).

Understanding MSE and R^2 on the Test Set

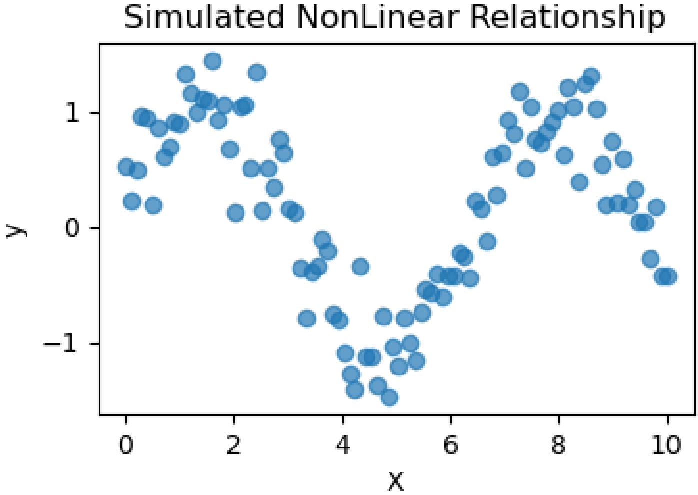
Coefficient of Determination (R^2)

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

- Measures how much of the **variance in the target** is explained by the model.
- Interpretation of R^2 values:
  - $1 =$ perfect fit
  - $0 =$ model predicts no better than the mean
  - $<0 =$ model performs worse than predicting the mean

What if our data was nonlinear?

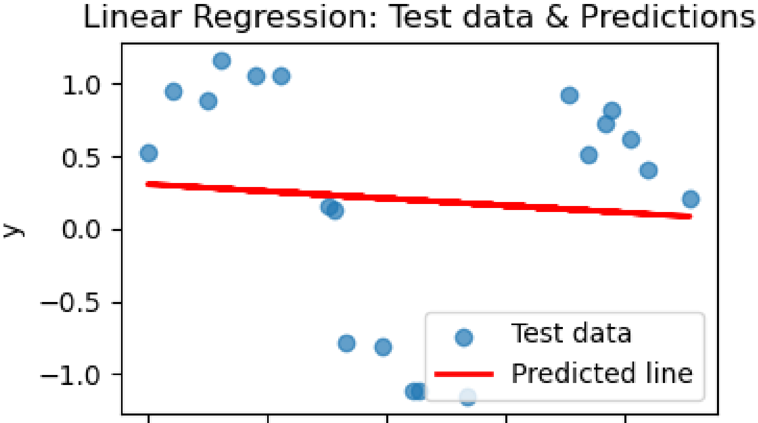
Let's try to apply linear regression to data that is clearly nonlinear!



What if our data was nonlinear?

Try a straight line prediction!

```
MSE on test set: 0.612
R² on test set: 0.005
```



What if our data was nonlinear?

How did we do? Is this a good predictor?

Polynomial vs Linear Regression

Not all data is linear!

Polynomial Regression

Our goal is to extend linear regression by including **powers of predictors** and/or interactions:

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + \ldots$$

This allows us to model **curved relationships** between predictors and the target.

Nonlinear Regression

If we wanted we could also use non polynomial functions for either the variables or the target!

$$\hat{y} = w_0 + w_1 \ln(x)$$

$$\ln(\hat{y}) = w_0 + w_1 x$$

Nonlinear Regression

This process still fits a linear model in terms of coefficients, but the model can capture nonlinear relationships in the data

So if we look at our data above, should we have used a linear function or maybe something else?

Polynomial Regression

Let's try doing polynomial regression on the data above to see if we can get a better fit. In this case we are going to use the Polynomial Features function to create new polynomial variables.

```
degree = 2
poly = PolynomialFeatures(degree=degree, include_bias=False)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
```

Polynomial Regression

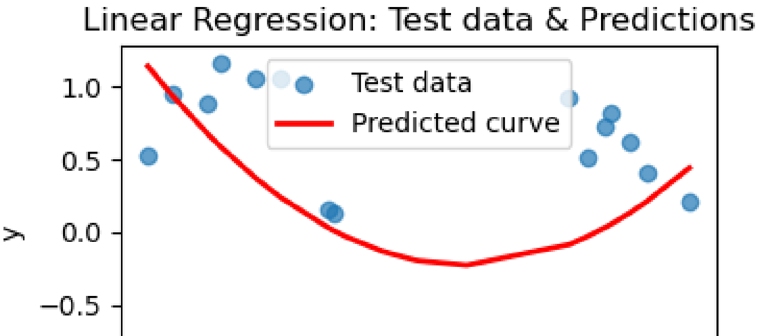
```
Intercept (w0): 1.1429933136914656
Coefficient (w1): -0.5125048890339421
Coefficient (w2): 0.04789627715549682
```

Polynomial Regression

So our model is given by

$$\hat{y} = w_0 + w_1 x + w_2 x^2$$

```
MSE on test set: 0.412
R² on test set: 0.329
```



Polynomial Regression

Did we do better that before? What happens if we try a cubic?

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$

or higher?

Polynomial Regression

We can actually choose a polynomial of too high a degree. This encourages your model to just pick a really jagged line that goes exactly through all the points.

This is **overfitting** and **poor model selection**.

You do not want your model to memorize the data! You want to choose a curve that matches the trend of the data without memorizing it.

Background: Logistic Regression

While linear regression predicts a continuous outcome, logistic regression is used for **classification** (typically binary) modeling the probability that an outcome belongs to class 1 (versus 0).

Background: Logistic Regression

Mathematically one models the log-odds (the "logit") as a linear combination:

$$\log \frac{p}{1-p} = w_0 + w_1 x_1 + \cdots + w_p x_p$$

This is making the assumption that the data can be separated by a straight line!

Background: Logistic Regression

Then the probability is

$$p = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + \cdots + w_p x_p)}}$$

That is, the "sigmoid" ("logistic") function maps the linear part back into the interval $(0, 1)$. The closer the probability is to 0, the more likely the observation belongs to class 0.

Background: Logistic Regression

You have to decide on what the cutoff is (e.g., $p > 0.5 =$ class 1 and $p <= 0.5 =$ class 0)

It is STILL a linear model in the log-odds space (it is a linear combination of predictors) but when we apply the sigmoid function it becomes a non-linear mapping in output.

Logistic Regression

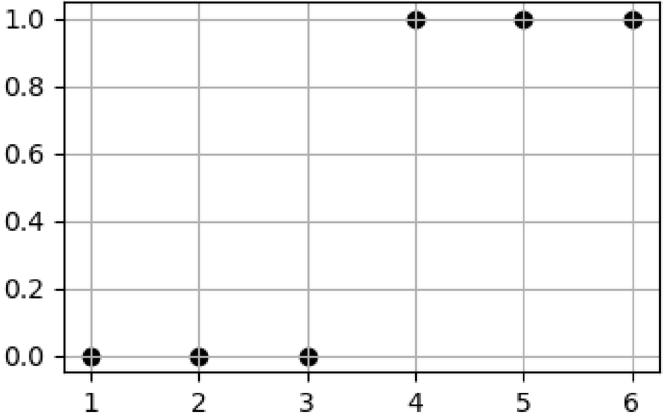
Let's visualize this in one dimension

Imagine we have the following data:

(x)	(y)
1	0
2	0
3	0
4	1
5	1
6	1

And we want to use x to predict y

Logistic Regression

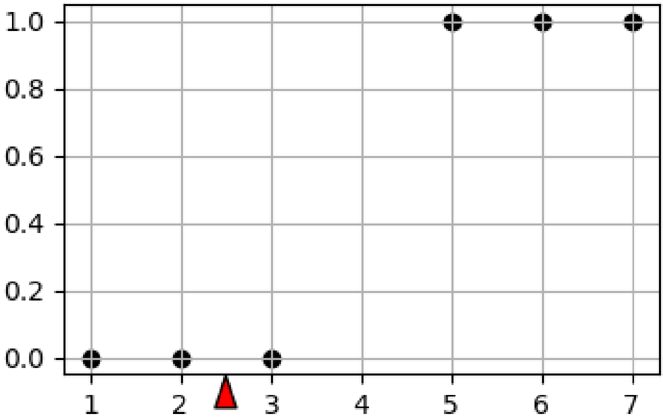


Logistic Regression

Well if we were doing this bu hand we would say, we'll if I draw a line between 3 and 4, then I can use that as a cutoff for how to classify future data.

eq. Here is a new data point x=2.5, is it more likely to belong to class 0 or class 1?

Logistic Regression



Logistic Regression

So we say "Given a data point x, what is the probability that it is in class 1?"

$$p(y = 1|x)$$

Well the "odds" that x is a member of class 1 are given by:

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

this is just the ratio of the probability of yes vs probability of no.

- p=0.5, odds = 1 so there is an equal chance of class 1/class 2.
- p=0.8, odds = 4 so it is 4 times more likely to be class 1.
- p=0.2, odds = 0.25 so it is 4 times more likely to be class 0.

Logistic Regression

We assume that the odds are a linear relationship with one variable x:

$$\log \frac{p}{1-p} = w_0 + w_1 x$$

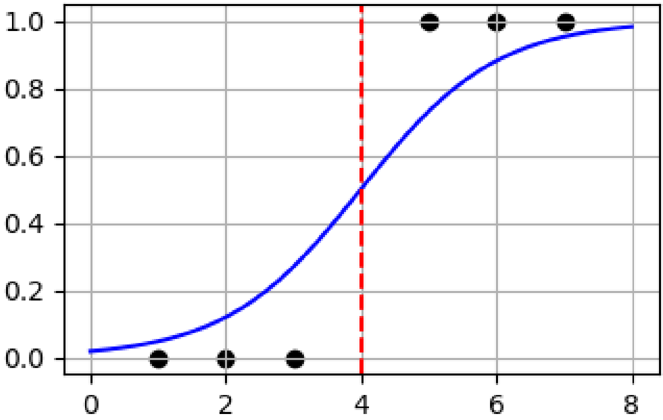
Logistic Regression

In this simple case we find $w_0 = -4$ and $w_1 = 1$

Then we can convert this back into a probability

$$p = \frac{1}{1 + e^{-(-4+x)}}$$

Logistic Regression

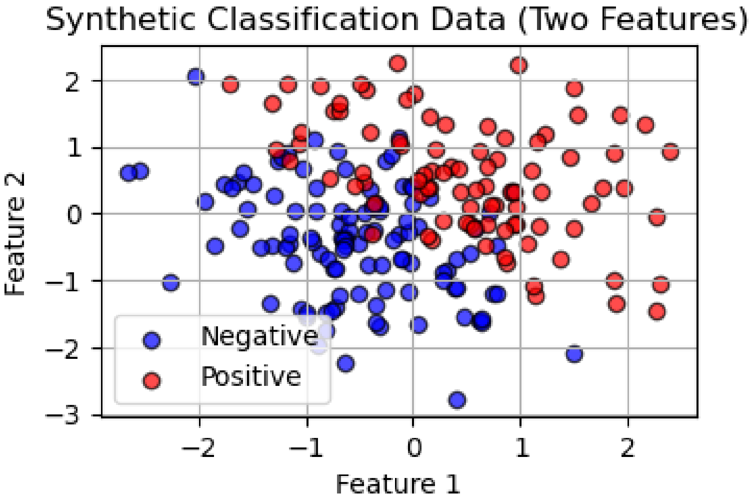


Logistic Regression

Now we can interpret this as given a point if the probability is less then 0.5 the it belongs to class 0 and otherwise to class 1. This matches our intuition.

Logistic Regression

Here is an example with more than one variable:



Logistic Regression

We still want to to a test train split!

Now train the model using Logistic Regression!

```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

```
Intercept (w0): -0.1949041499390587
Coefficients (w1,w2): [2.28689566 2.12914333]
```

Logistic Regression

Here we see our coefficients. This means that:

$$\log \frac{p}{1-p} = w_0 + w_1 x_1 + w_2 x_2$$

In this simple case we find $w_0 = -4$ and $w_1 = 1$

Logistic Regression

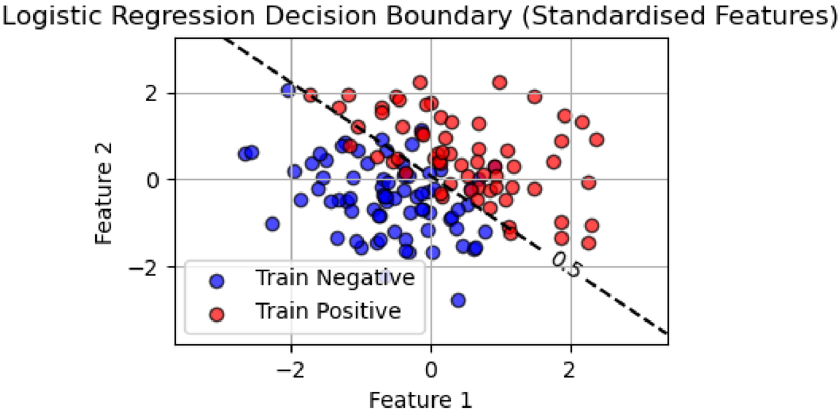
Then we can convert this back into a probability

$$p = \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}}$$

So we might say if $p > 0.5$ then we are positive and if $p < 0.5$ we are negative.

Logistic Regression

Here is a plot of this linear decision boundary.



Logistic Regression

There are many ways to think about accuracy when talking about classification problems. Here we will use the `accuracy score` `classification report` and the `confusion matrix`

Logistic Regression

```
# Predictions
y_pred = logreg.predict(X_test)

acc = accuracy_score(y_test, y_pred)
print(f"Accuracy on test set: {acc:.3f}")

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

Logistic Regression

```
Accuracy on test set: 0.850
Confusion Matrix:
[[28  6]
 [ 3 23]]

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.82      0.86        34
           1       0.79      0.88      0.84        26

    accuracy                           0.85        60
   macro avg       0.85      0.85      0.85        60
weighted avg       0.86      0.85      0.85        60
```

Understanding Classification Metrics

The output above has a lot of information!

- **TN** $=$ True Negatives - predicted class 0 belongs to class 0
- **TP** $=$ True Positives - predicted class 1 belongs to class 1
- **FN** $=$ False Negatives - predicted class 0 belongs to class 1
- **FP** $=$ False Positives - predicted class 1 belongs to class 0

Understanding Classification Metrics

1. Accuracy

First we see that the accuracy $= 0.85$. This seems pretty good, but sometimes accuracy can be deceiving!

It reports the proportion of total predictions that are correct:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Understanding Classification Metrics

2. Confusion Matrix

The confusion matrix tells us a count of which predictions we got right and wrong,

TN	FP
FN	TP

Understanding Classification Metrics

3. Precision

- Measures the **accuracy of positive predictions**.
- Formula:

$$\text{Precision} = \frac{TP}{TP + FP}$$

- High precision means few false positives.

Understanding Classification Metrics

3. Recall (Sensitivity or True Positive Rate)

- Measures how well the model **finds all actual positives**.
- Formula:

$$\text{Recall} = \frac{TP}{TP + FN}$$

where:

- High recall means few false negatives.

Understanding Classification Metrics

4. F1-Score

- The **harmonic mean** of precision and recall.
- Formula:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Balances precision and recall. Useful when classes are imbalanced.

Understanding Classification Metrics

5. Support

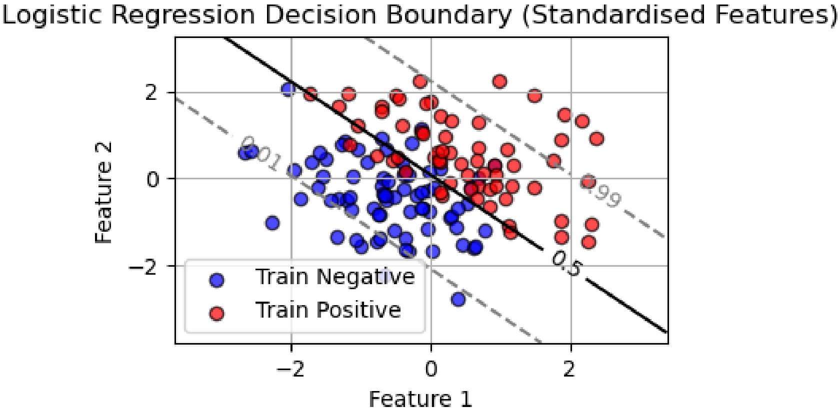
- The **number of actual occurrences** of each class in the dataset.
- Indicates how many samples of each class were present when computing the metrics.

Now all of this depends on where we set our decision boundary! It was somewhat arbitrary to choose $p = 0.5$ as the cutoff.

Choosing The Decision Boundary

The decision boundary is the “line” that shows when we would say the observations falls under class 0 vs when it would fall under class 1. By default we use a probability of $p = 0.5$.

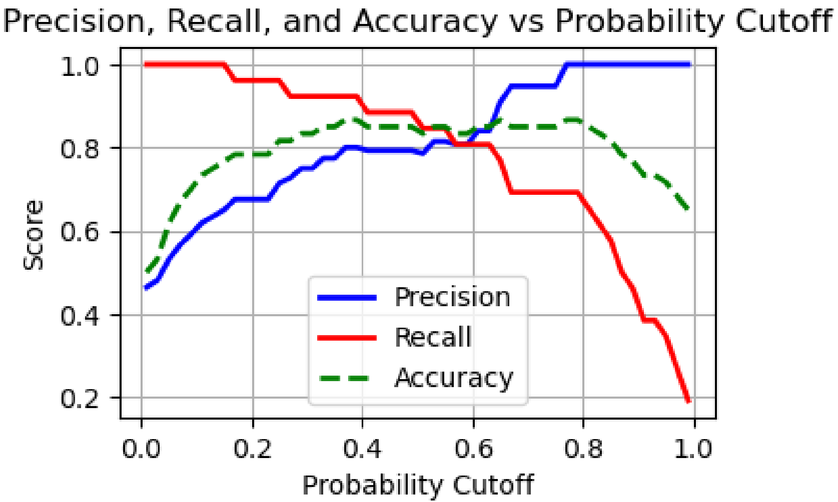
Choosing The Decision Boundary



Choosing The Decision Boundary

But the choice of this cutoff is arbitrary! Sometimes a BIG part of the classification problem is figuring out what the cutoff should be!

Choosing The Decision Boundary



Choosing The Decision Boundary

Remember perfect recall means that we do not see any false negatives. In this case when we move our probability cutoff to 0.1 we see that we predict almost everything as positive, we don't miss a single positive case with our prediction. It would be important to have high recall when missing positives is a big problem (eg. cancer detection - you don't want false negatives) But our accuracy is very low!

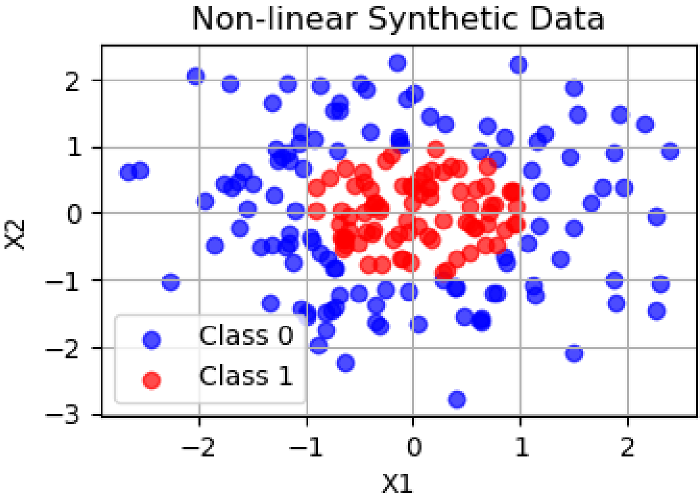
Choosing The Decision Boundary

Perfect precision means that we do not see any false positives. In this case when our probability cutoff is moved to 0.99 we see that we predict almost everything as negative, we don't miss a single negative case. It would be important to have high precision in cases when a positive outcome has a large impact (eg. detecting fraud - you don't want false positives). But again our accuracy is low.

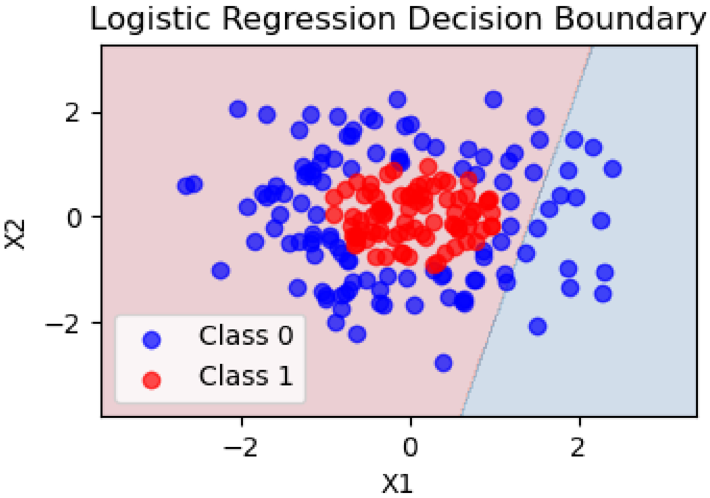
Choosing The Decision Boundary

The best choice is somewhere in the middle and is highly dependent on the problem you are trying to solve.

What if our decision boundary was not linear?



What if our decision boundary was not linear?



What if our decision boundary was not linear?

How did we do at predicting these classes? We got ALL of the class 1 data right! But our accuracy was actually pretty low and we can see from the picture that the decision boundary really did not match our data. We were particularly bad a predicting class 0!

What if our decision boundary was not linear?

We can use polynomial features here too!

```
poly = PolynomialFeatures(degree=2, include_bias=False)
X_train_poly = poly.fit_transform(X_train)
X_test_poly  = poly.transform(X_test)
```

What if our decision boundary was not linear?