Models for Classification

Introduction to Data Science

Kigali, Rwanda

July 8th, 2019

Outline

1. What is Classification
2. Logistic Regression
3. Interpreting and Evaluating Classifiers

What is Classification?

A regression problem is where we predict a **quantitative** outcome y based on some covariates x.

- **Example:** Given the number of room, the number of allowed guests and the neighborhood, predict price of an Airbnb listing.

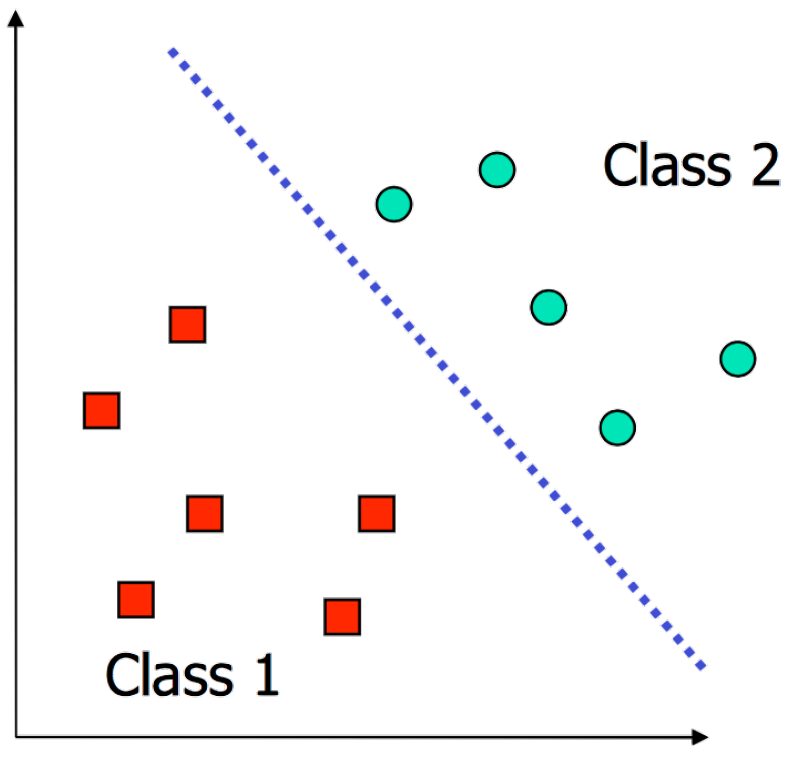
A ***classification*** problem is a **categorical** outcome y based on some covariates x.

- **Example:** Given the age, education and income level of a loan applicant, predict whether or not the loan application will be approved.

Logistic Regression

The Role of Decision Boundaries

Suppose our data has a small number of predictors, by visualizing the data we can intuitively check how easy it is to separate the classes.



Ideally, the classes are easily separated by a curve (or surface) in the input space, this curve (or surface) is called the ***decision boundary***.

Linear Decision Boundaries

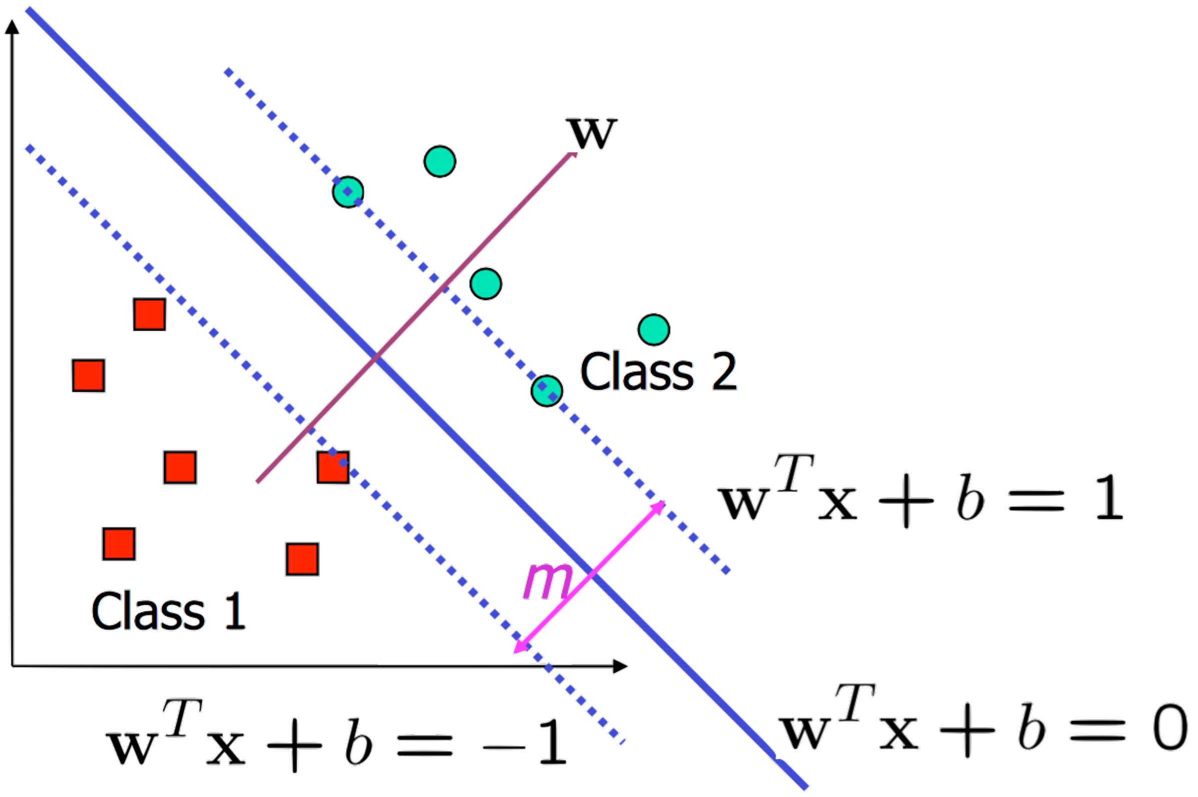
When the decision boundary is linear, it is defined by the equation

$$\mathbf{w}^\top \mathbf{x} = w_0 x_0 + w_1 x_1 + \ldots + w_D x_D = 0$$

$$\mathbf{w}^\top \mathbf{x} = w_0 x_0 + w_1 x_1 + \ldots + w_D x_D = 0$$

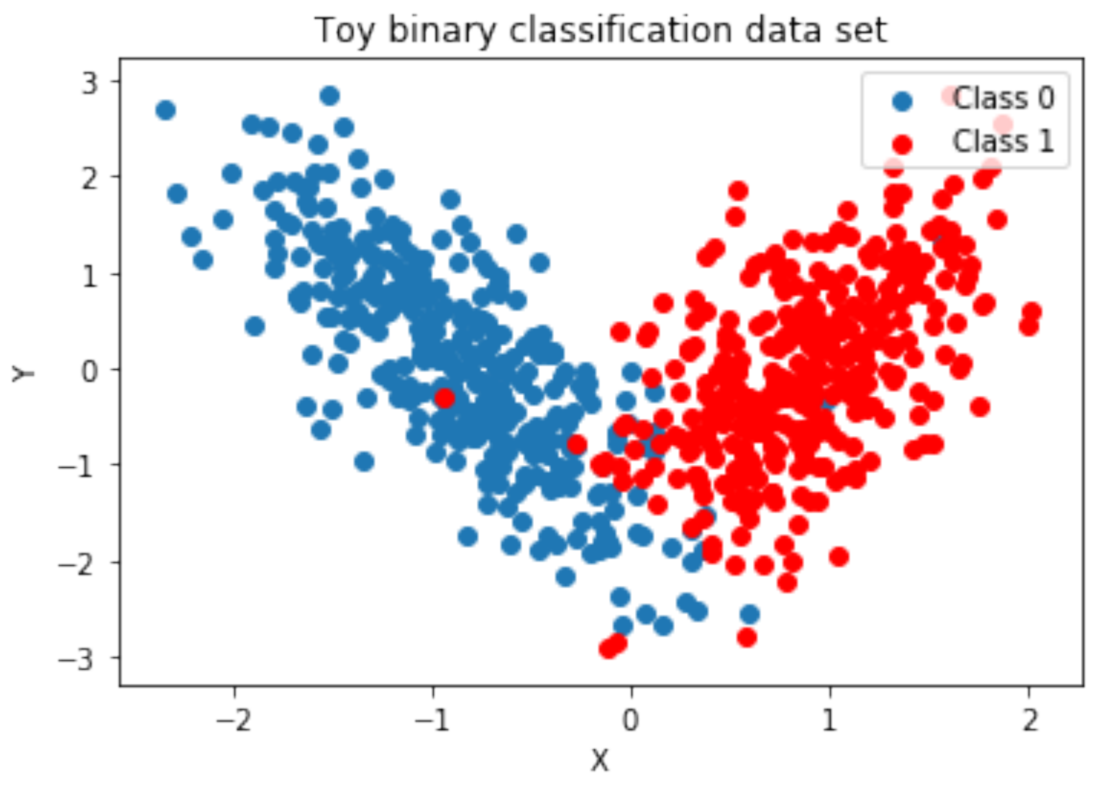
where $x_0 = 1$ $x_0 = 1$. Often we write $b = w_0 x_0$ $b = w_0 x_0$ and we call b b the ***bias*** term.

The vector $\mathbf{w}$ $\mathbf{w}$ allow us to gauge the 'distance' of a point from the decision boundary


$\mathbf{w}$
Class 2
$\mathbf{w}^T\mathbf{x} + b = 1$
m
Class 1
$\mathbf{w}^T\mathbf{x} + b = -1$
$\mathbf{w}^T\mathbf{x} + b = 0$

Hard Decisions or Soft

Once we have a decision boundary should we always classify points on one side of the boundary as 1 and points on the other side as 0?



Logistic Regression: Modeling Probabilities of Labels

To model the probability of labeling a point a certain class, we have to convert distance, $\mathbf{w}^\top\mathbf{x}$$\mathbf{w}^\top\mathbf{x}$ (which is unbounded) into a number between 0 and 1, using the ***sigmoid function***:

$$\text{Prob}(y = 1|\mathbf{x}) = \text{sigmoid}(\underbrace{\mathbf{w}^\top\mathbf{x}}_{\text{distance}})$$

$$\text{Prob}(y = 1\,|\,\mathbf{x}) = \text{sigmoid}(\underbrace{\mathbf{w}^\top\mathbf{x}}_{\text{distance}})$$

where $\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$ $\text{sigmoid}(z) = \frac{1}{1+e^{-z}}$.

To fit our model, we **choose** to learn the parameters of $\mathbf{w}$$\mathbf{w}$ that maximizes the likelihood of our training data. This is equivalent to minimizing ***binary cross-entropy***:

$$\min_{\mathbf{w}} \text{CrossEnt}(\mathbf{w}) = \min_{\mathbf{w}} \sum_{n=1}^{N} y_n \log(\mathbf{w}^\top\mathbf{x}_n) + (1 - y_n) \log\left(1 - \mathbf{w}^\top\mathbf{x}_n\right)$$

$$\min_{\mathbf{w}} \text{CrossEnt}(\mathbf{w}) = \min_{\mathbf{w}} \sum_{n=1}^{N} y_n \log(\mathbf{w}^\top\mathbf{x}_n) + (1 - y_n) \log\left(1 - \mathbf{w}^\top\mathbf{x}_n\right)$$

Unfortunately, we cannot solve for the minimum of the binary cross-entropy loss analytically!

Tomorrow we will see how to minimize this loss function algorithmically.

Logistic Regression in `sklearn`

```
# import the logistic regression model from sklearn
from sklearn.linear_model import LogisticRegression
# create a logistic regression model with linear boundary
logistic = linear_model.LogisticRegression(C=1.)
# fit our logistic regression model, i.e. find the parameters w
# that maximizes the likelihood of x_train, y_train
logistic.fit(x_train, y_train)
```

Hard Predictions with Logistic Regression

Remember that logistic regression predicts the **probability** that `y=1` :

$$\text{Prob}(y=1|\mathbf{x}) = \text{sigmoid}(\underbrace{\mathbf{w}^\top \mathbf{x}}_{\text{distance}})$$

$$\text{Prob}(y=1|\mathbf{x}) = \text{sigmoid}(\underbrace{\mathbf{w}^\top \mathbf{x}}_{\text{distance}})$$

But what if we have to make a hard decision `y=1` or `y=0` ?

To do this, we **choose** a threshhold, say 0.5, and decide:

$$y = 1 \text{ if } \text{Prob}(y=1|\mathbf{x}) > 0.5$$
$$y = 0 \text{ if } \text{Prob}(y=1|\mathbf{x}) < 0.5$$

$$y = 1 \text{ if } \text{Prob}(y=1|\mathbf{x}) > 0.5$$
$$y = 0 \text{ if } \text{Prob}(y=1|\mathbf{x}) < 0.5$$

In `sklearn` :

```
logistic.predict(x_train)
```

Interpreting and Evaluating Classifiers

Interpreting Logistic Regression

Suppose that you fit a logistic regression model to predict whether a loan application should be approved. Suppose that you have three attributes:

1. `x_1` representing gender: 0 for male, 1 for female
2. `x_2` for the income
3. `x_3` for the loan amount

Suppose that the parameters you found are:

$$p(y=1|x_1, x_2, x_3) = \text{sigmoid}(-1 + 3x_1 + 1.5x_2 + 1.75x_3).$$

$$p(y=1|x_1, x_2, x_3) = \text{sigmoid}(-1 + 3x_1 + 1.5x_2 + 1.75x_3).$$

What are the parameters telling us about the most influential attribute for predicting loan approval? What does this say about our data?

Evaluating Classifiers Using Accuracy

Suppose your train/test accuracies are as follows:

In [4]: Slide Type -

```
# evaluate model on training and testing data
scores_df = pd.DataFrame(data={'Linear regression': regression.score(x_train, y_train),
                                                    regression.score(x_test, y_test)},
                         index=['train score', 'test score'])
scores_df.head()
```

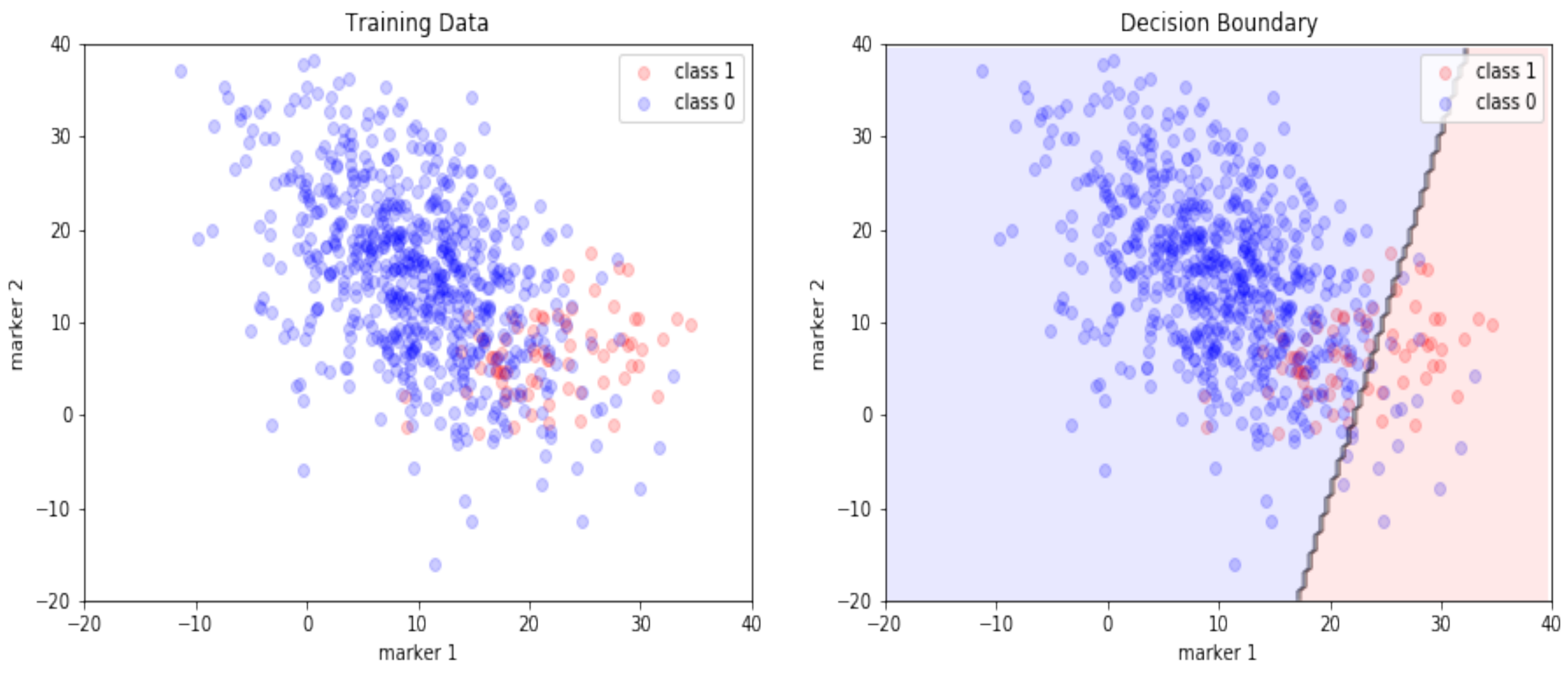
Out[4]:

	Linear regression
train score	0.84105
test score	0.63544

Would you say that your classifier is fitting the data well?

Evaluating Classifiers by Visualizing the Decision Boundary

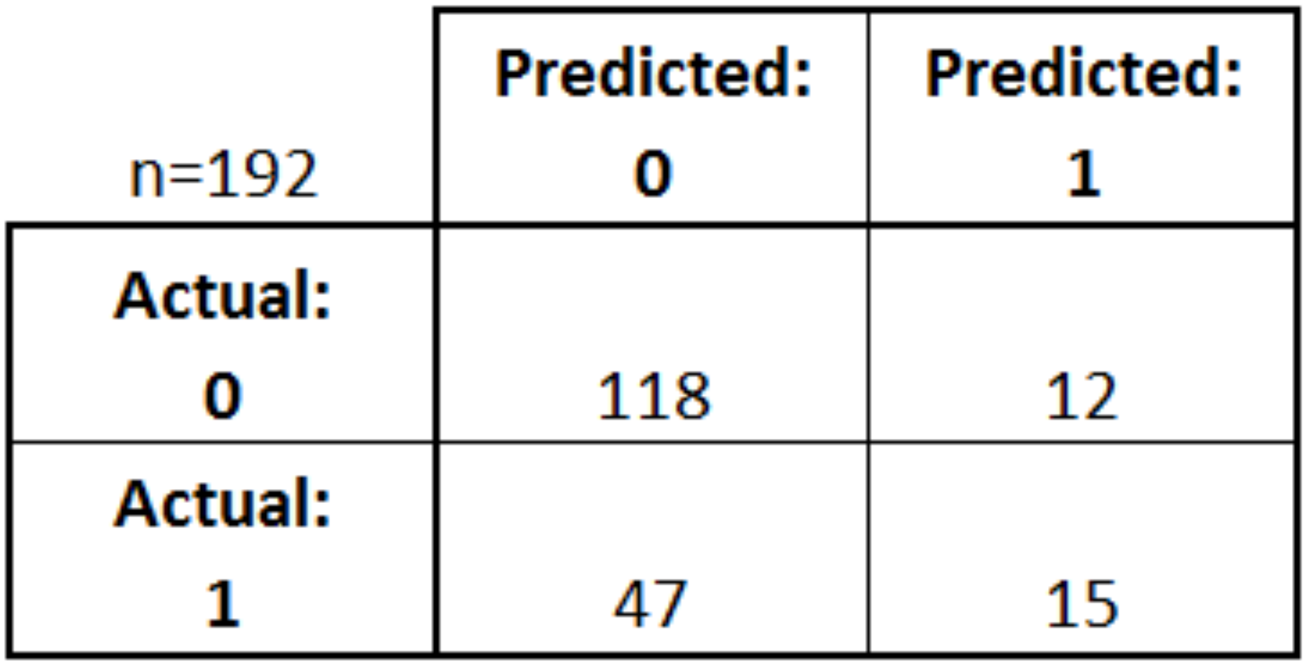
Suppose you visualize the decision boundary:



Would you say that your classifier is fitting the data well?

Evaluating Classifiers by Computing the Confusion Matrix

A confusion matrix breaks down the performance of a classifier into categories.



n=192	**Predicted: 0**	**Predicted: 1**
Actual: 0	118	12
Actual: 1	47	15

Evaluating Classifiers by Computing the Confusion Matrix

Suppose the confusion matrix of your classifier is:

```
In [8]: y_pred = logistic.predict(x_test)
        print('Confusion matrix for the classification on test:\n', confusion_matrix(y_test, y_pred))
```

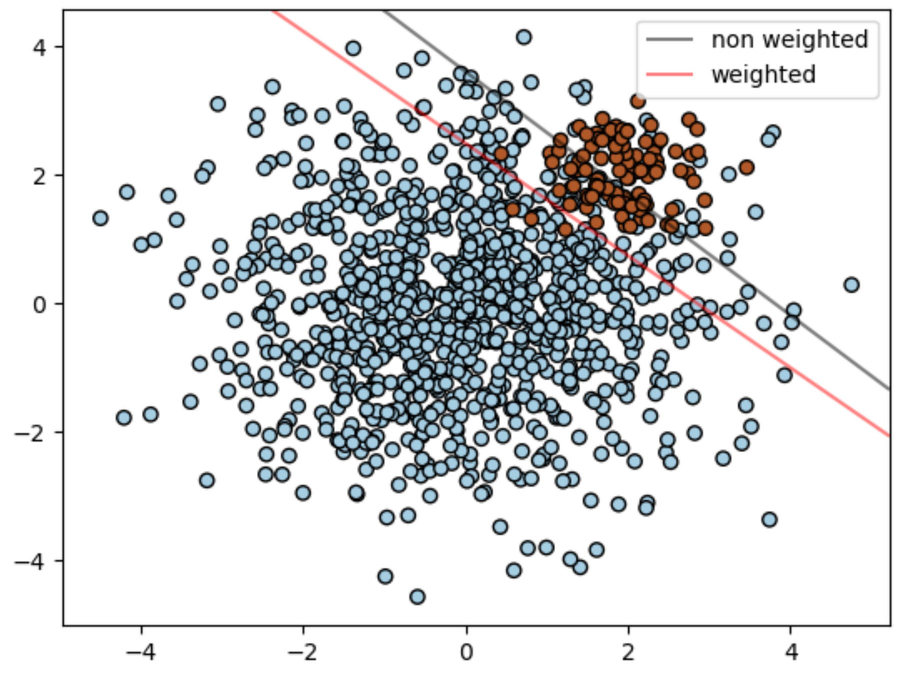
```
Confusion matrix for the classification on test:
 [[332   6]
 [ 20   5]]
```

Would you say that your classifier fit the data well?

Addressing Class Imbalance: Reweighting

When classes in your data set are extremely unbalanced, the models you train can be unincentivized to predict correctly on the rare class -- specializing on the overrepresented classes will result in low average loss.

We can ***reweight*** the data so that contributions from error associated with the rare class is increased while the error associated with the overrepresented classes is decreased. Intuitively, the model is penalized more for being 'wrong' on the rare class.



Reweighting in `sklearn`

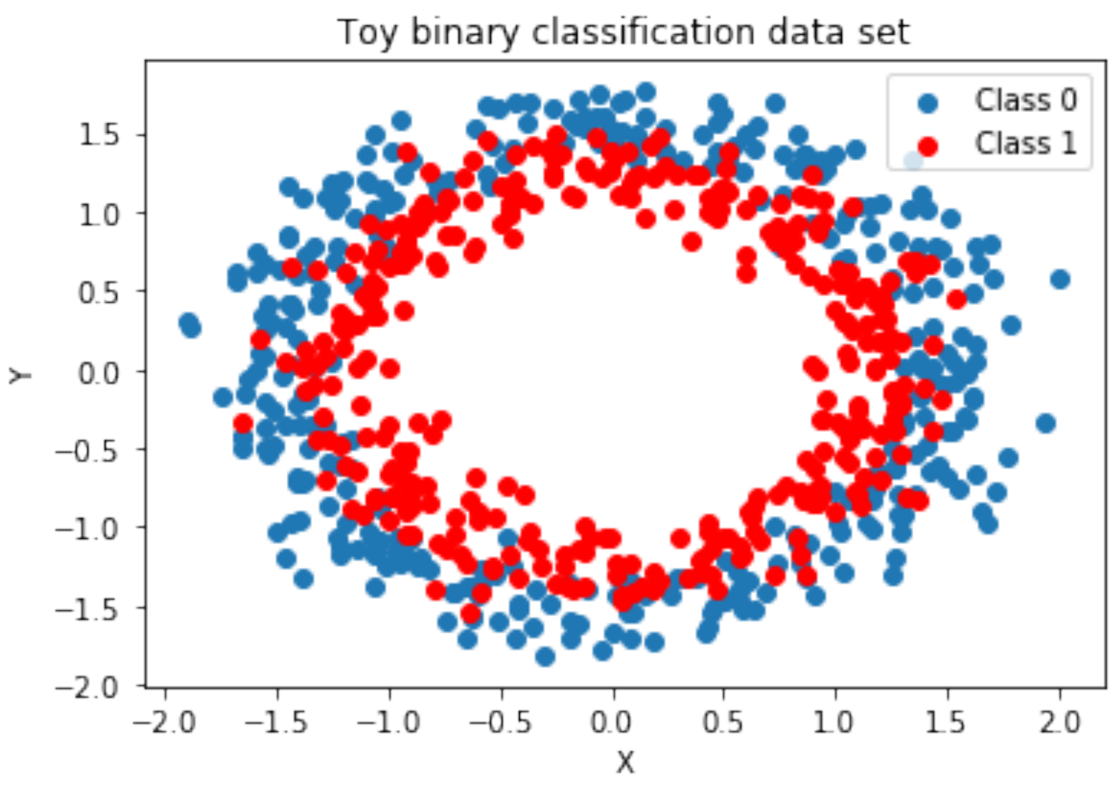
Use the the `class_weight` parameter in `sklearn`'s' `LogisticRegression` model to put more emphasis on the smaller class. For example,

```
# define a dictionary specifying the emphasis given to each class
weights = {0: 10, 1: 1}

# make an instance of the linear regression model, alpha is the strength of
the penalty, class_weight is the relative weight of the two classes
logistic = LogisticRegression(C=1., class_weight=weights)
```

Logistic Regression with Polynomial Boundaries

How would you parametrize a ellipitical decision boundary?



We can say that the decision boundary is given by a ***quadratic function*** of the input:

$$w_1x_1^2 + w_2x_2^2 + w_3 = 0$$

$$w_1x_1^2 + w_2x_2^2 + w_3 = 0$$

We say that we can fit such a decision boundary using logistic regression with degree 2 polynomial features

Logistic Regression with Quadratic Decision Boundary

Recall that polynomial regression is simply a linear regression fit on polynomial features of the inputs `x` :

1. transform $x x$ into [1, $x x$, $x^2 x^2$, $x^3 x^3$, $x^4 x^4$, ... etc]
2. fit linear regression on the polynomial features [1, $x x$, $x^2 x^2$, $x^3 x^3$, $x^4 x^4$, ... etc]

Similary, if we want to fit a logistic regression with quadratic features, we:

1. transform $x x$ into [1, $x x$, $x^2 x^2$]
2. fit logistic regression on the quadratic features [1, $x x$, $x^2 x^2$]

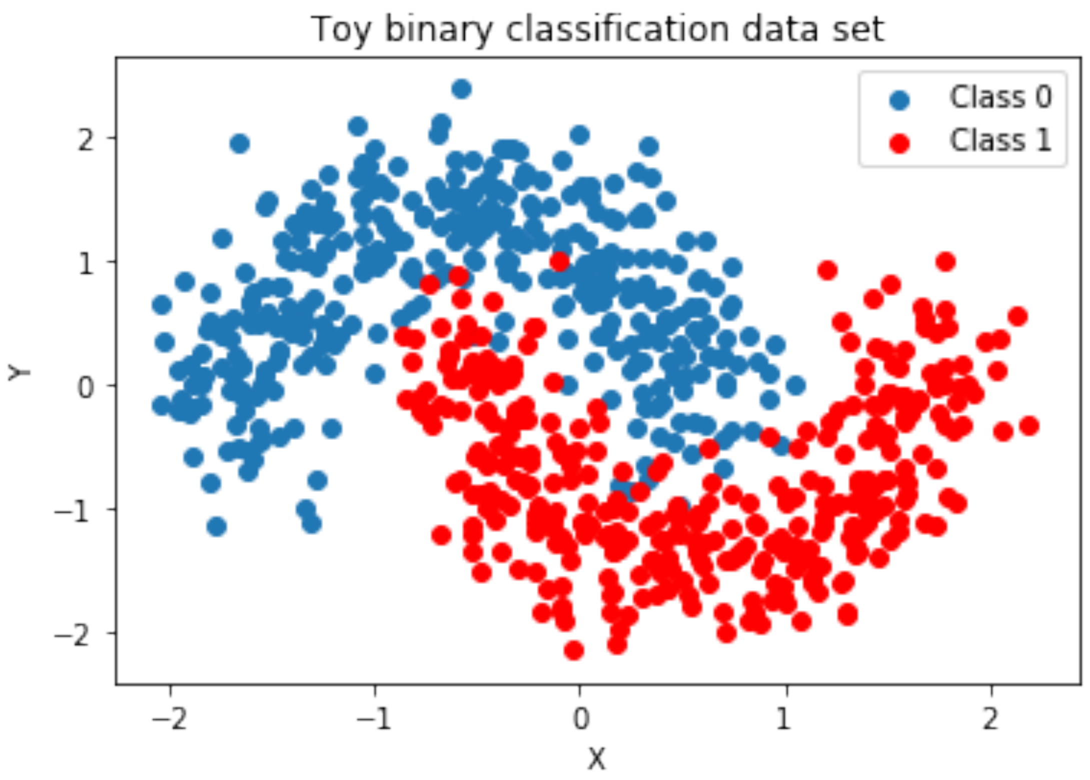
Logistic Regression with Polynomial Boundary Implementation in `sklearn`

```python
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures

#transform your training and testing data into polynomial features
polynomial_features = PolynomialFeatures(2)
polynomial_features.fit(x)
x_poly = polynomial_features.transform(x)

#fit a logistic regression on top of your polynomial features
logistic_poly = LogisticRegression(solver='lbfgs', max_iter=1000)
logistic_poly.fit(x_poly, y)
```

How would you parametrize an arbitrary complex decision boundary?



It's not easy to think of a function $g(x)$ can capture this decision boundary.

GOAL: Find models that can capture *arbitrarily complex* decision boundaries.