Module 7 : Machine Learning

Session 7E : Synthetic Data

Dr Daniel Chalk

"I am the one and phoney"

HSMA 4

NIHR | Applied Research Collaboration South West Peninsula

@peninsula_ARC #HSMA4

HSMA 4 Everyone

Acknowledgments

With grateful thanks to my colleague Mike Allen for putting together the excellent Titanic workbook ~~from which I nicked most of this stuff~~ upon which much of this content is based.

Synthetic Data

Synthetic Data refers to data that is not real (“fake”). In Machine Learning, we can use AI-based methods to create Synthetic Data.

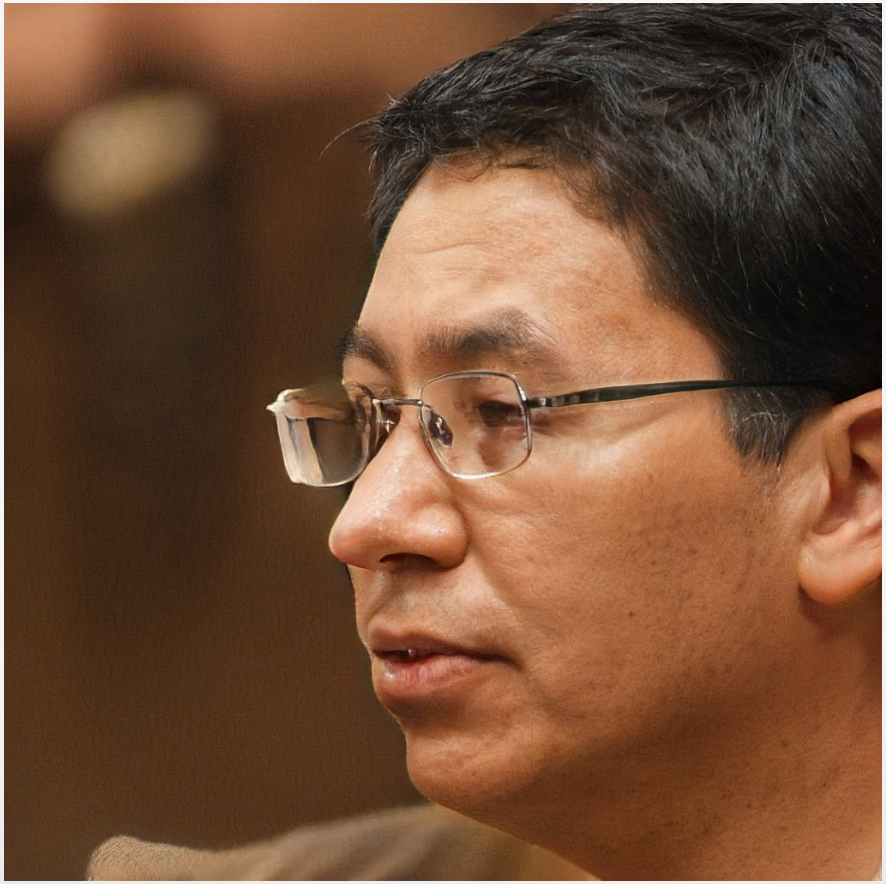
Why?

- Because we may want to share or publish data for use in a model, but the original data cannot be shared or published

- Because we may not have enough data for our Machine Learning model (e.g. for a particular outcome) and we need to create more to train and improve the performance of our model

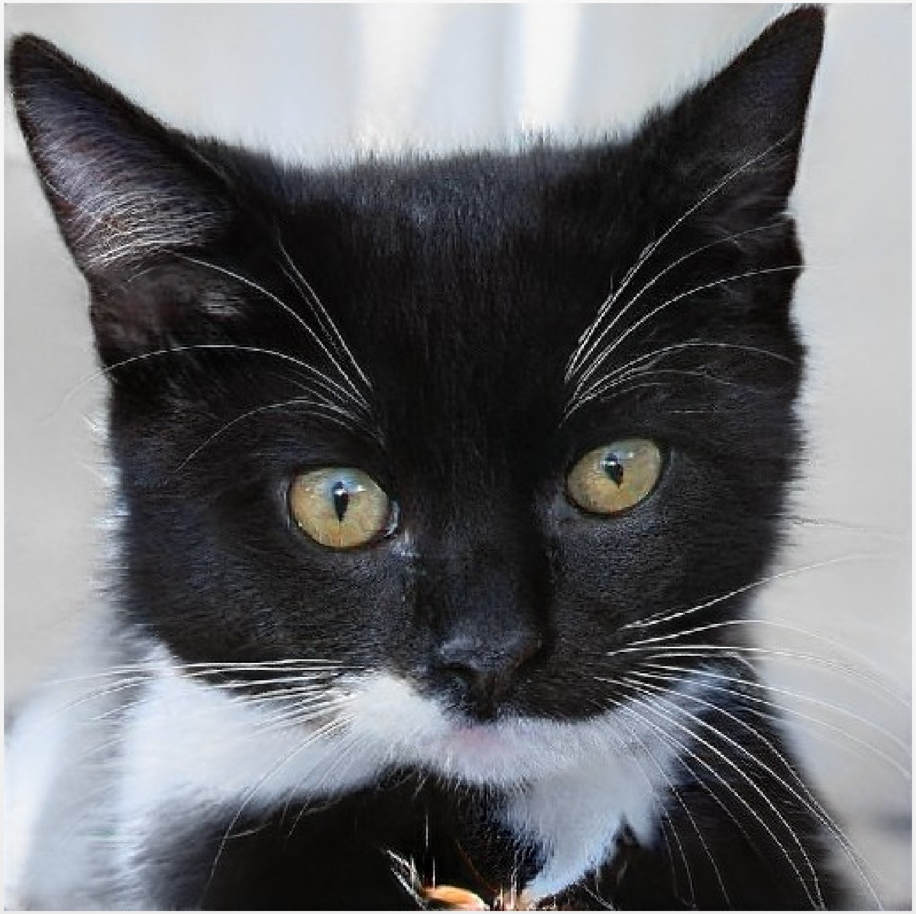
This Person Does Not Exist

https://this-person-does-not-exist.com/en



This Cat Does Not Exist

https://thiscatdoesnotexist.com/



This Cat Does Not Exist

https://thisshoedoesnotexist.com/

Synthetic Data Approaches

There are a number of different approaches that we can take to generate Synthetic Data. Today, we’re going to focus on the first of these, but we’ll also give an overview of the others.

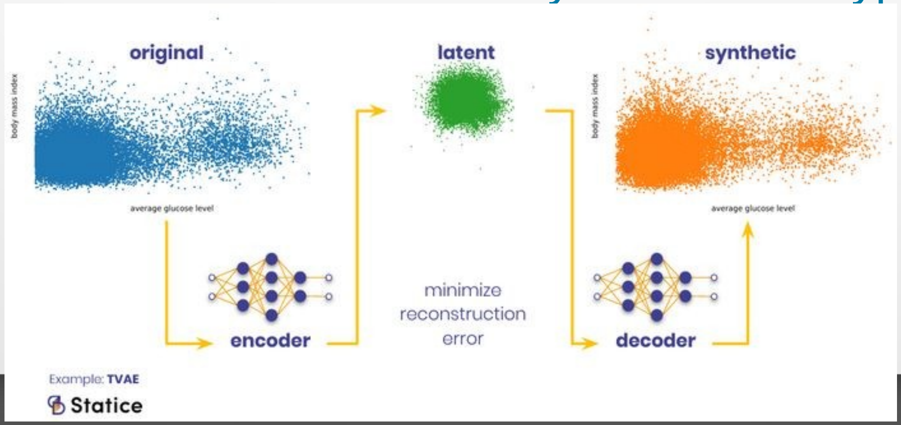
- Oversampling techniques, such as SMOTE
- Variational Autoencoders (VAEs)
- Generative Adversarial Networks (GANs)

Variational Autoencoders

Variational Autoencoders (VAEs) are a type of Deep Neural Network. They are very complex, but basically work using a two-step structure in which they *decode* the *distribution* of the original data, and then *encode* it again using this distribution whilst trying to minimise *reconstruction error* and maximise the *variance of the distribution in the encoding layer* (competing objectives). In other words, it is trying to create data that is similar enough but also different enough. You can tweak the parameters to get closer to one objective or the other. The decoding and encoding steps use neural networks.

In simple terms, for synthetic data, they try to work out the distribution of the real data, and then reassemble new points (the synthetic data) that follow the same kind of distribution. By minimising the reconstruction error, the idea is you get points that are close to the original data, without being identical.

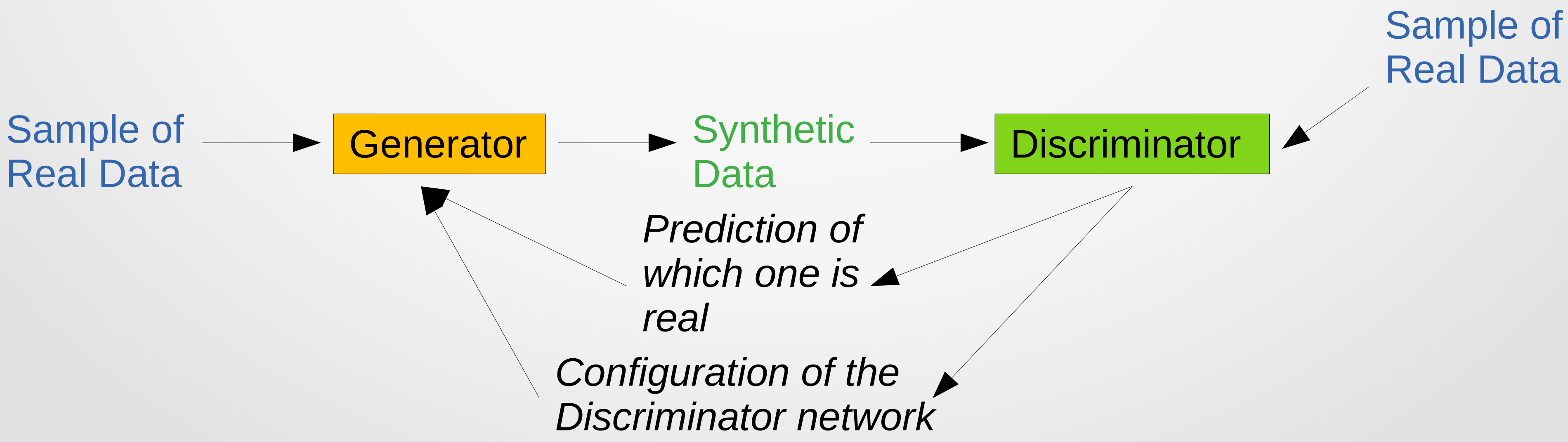
These links give some quite nice explanations (the image below is from the second link) :
https://visualstudiomagazine.com/articles/2021/05/06/variational-autoencoder.aspx
https://www.kdnuggets.com/2021/02/overview-synthetic-data-types-generation-methods.html



Generative Adversarial Networks

A Generative Adversarial Network (GAN) is an approach in which there are two networks working *competitively* – a *Generator* and a *Discriminator.* The Generator network tries to create synthetic data from a random sample of real data. The Discriminator is presented with samples from the real data and the synthetic data from the generator, and has to try to work out which is real data, and which is synthetic.

The two networks are linked during training, so that the Generator knows how the Discriminator is making its predictions, and gradually gets better at making more realistic synthetic data. But at the same time, the Discriminator gradually gets better at making predictions with increasingly difficult to differentiate data samples. They *reinforce* each other and cause each to get increasingly better – both trying to "outsmart" the other.



SMOTE

Synthetic Minority Oversampling Technique (SMOTE) is an *oversampling* (or “data augmentation”) approach. It is one of the most commonly used approaches for creating synthetic data for the purposes of augmenting the data in an under-represented class.

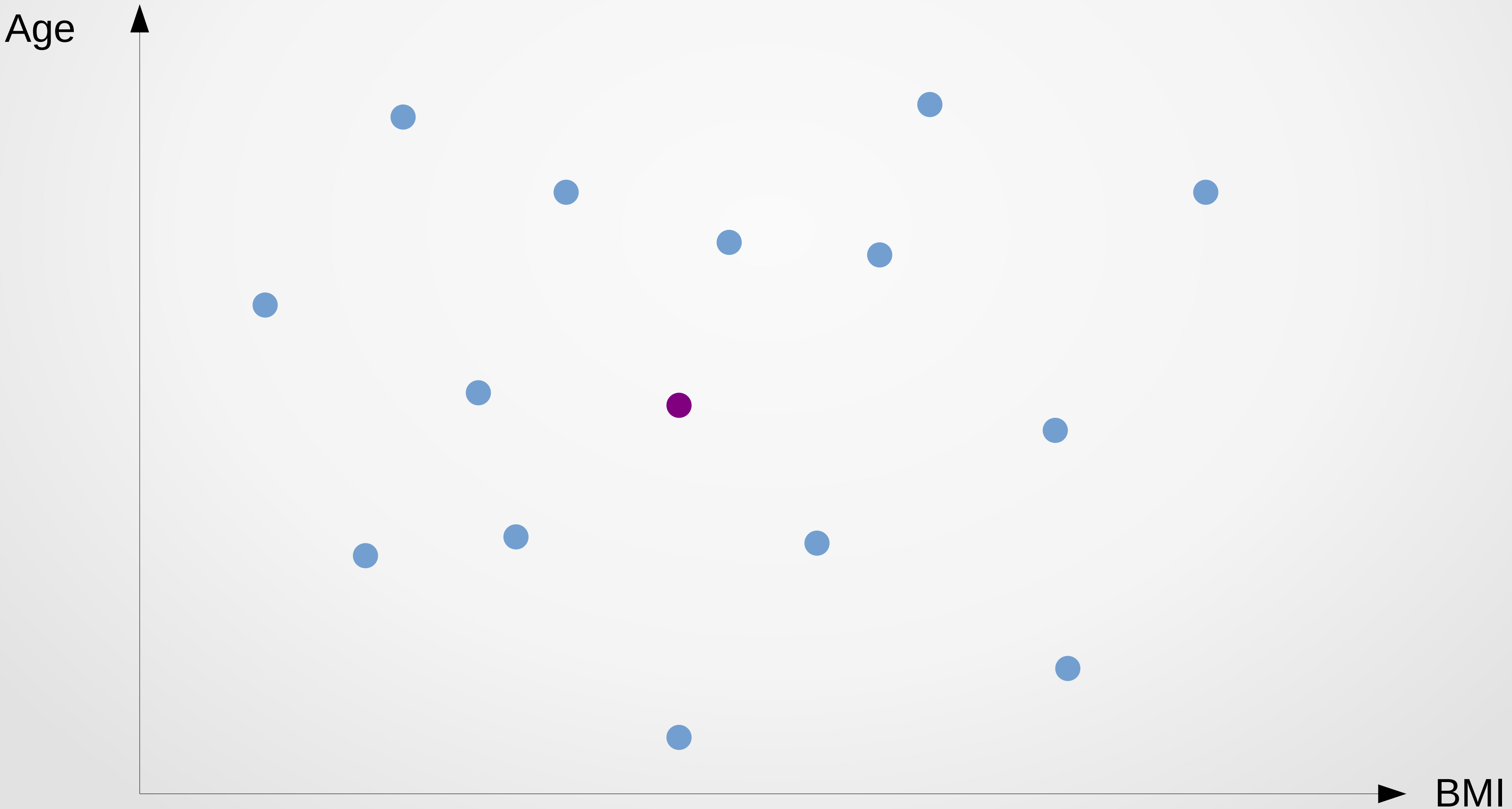
SMOTE works by creating new data points that are close to existing data points. This allows us to expand the number of data points we can use (or replace our real data points with synthetic data points).

SMOTE

Let’s see how it works. Here we’ll use a 2-dimensional example (two features) to make this easier to visualise, but obviously in reality this would typically be happening in multi-dimensional space.

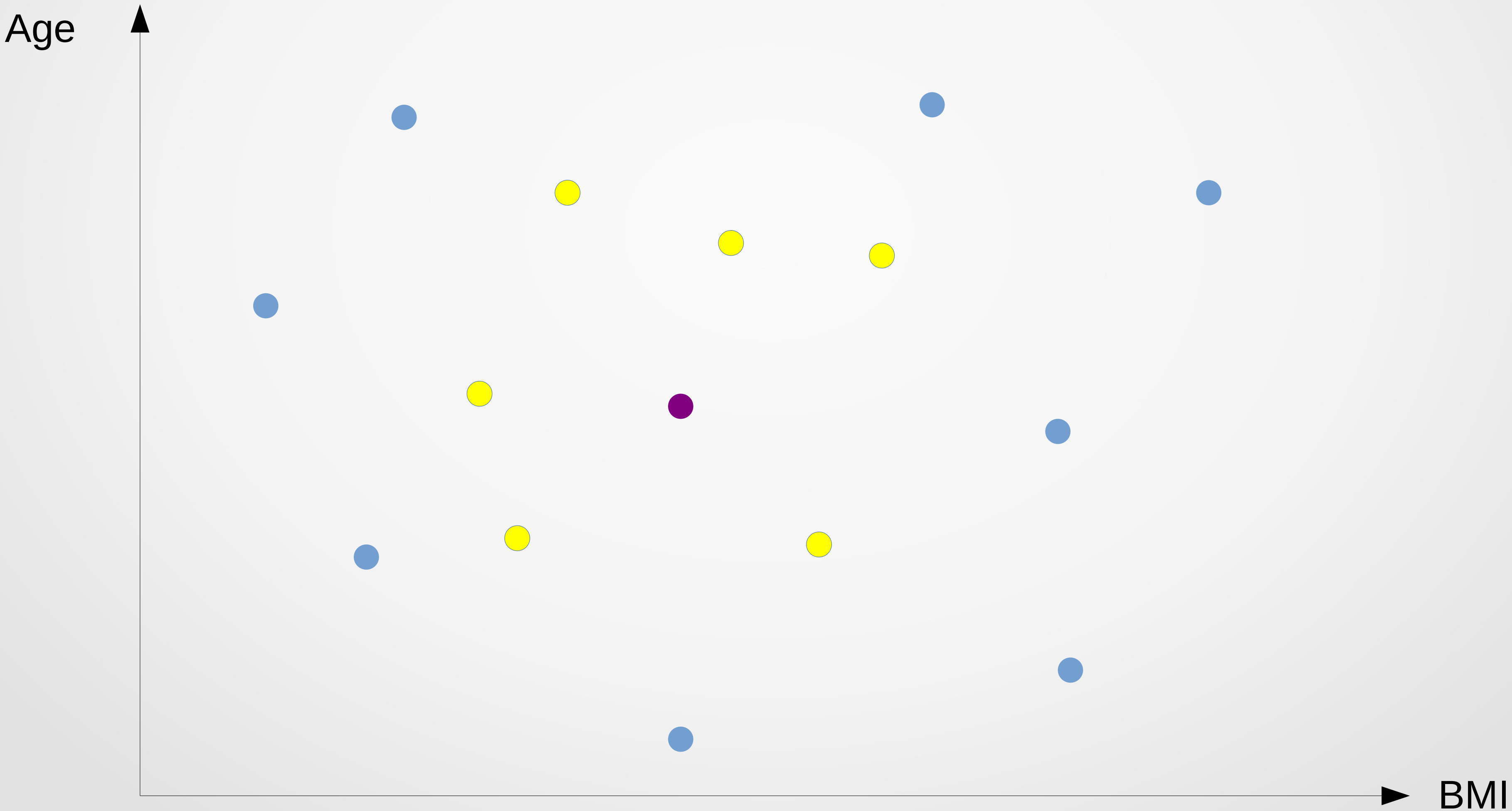
SMOTE

1. Pick a random data point.



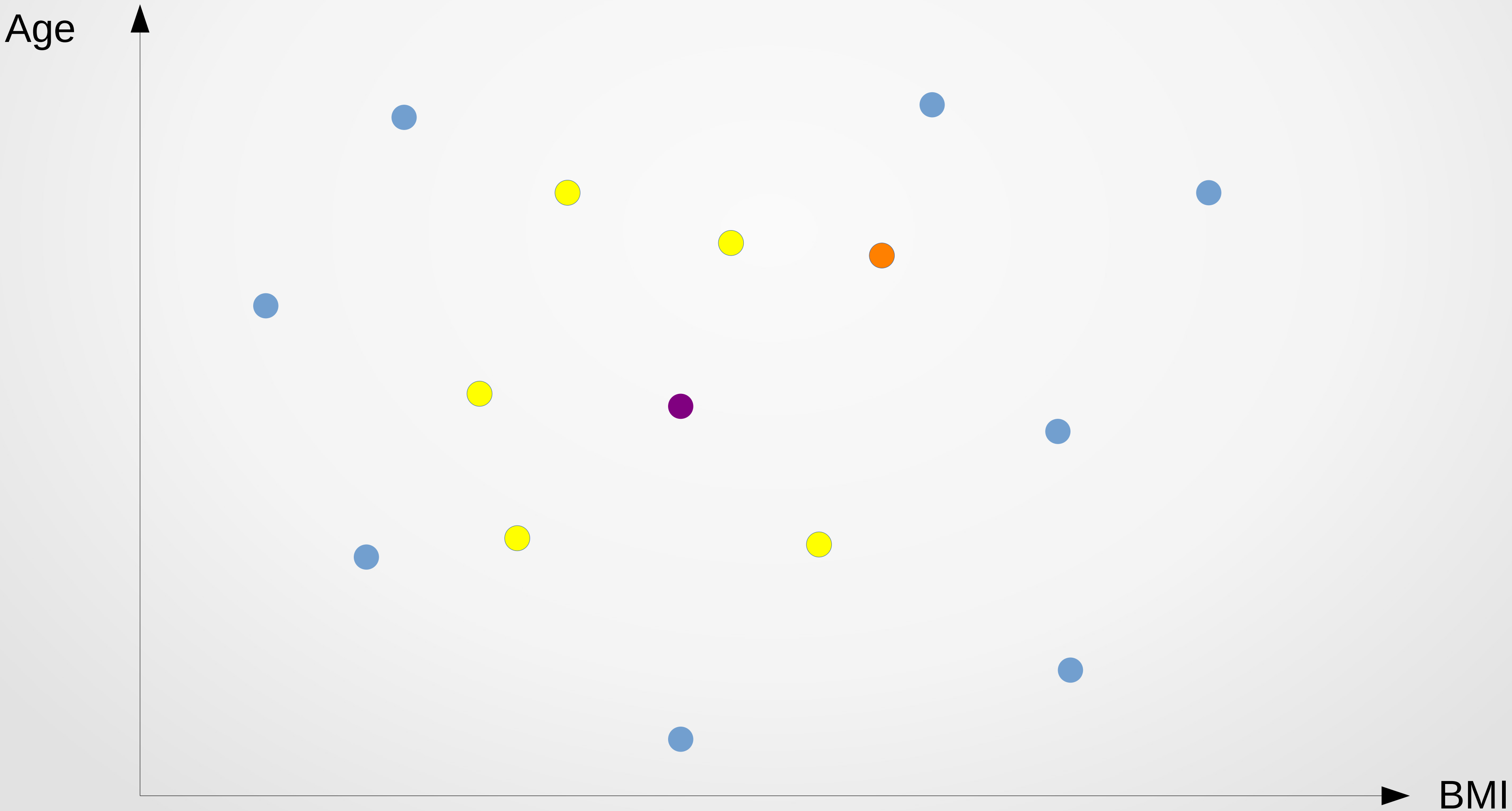
SMOTE

2. Find *k* nearest neighbours (default = 6)



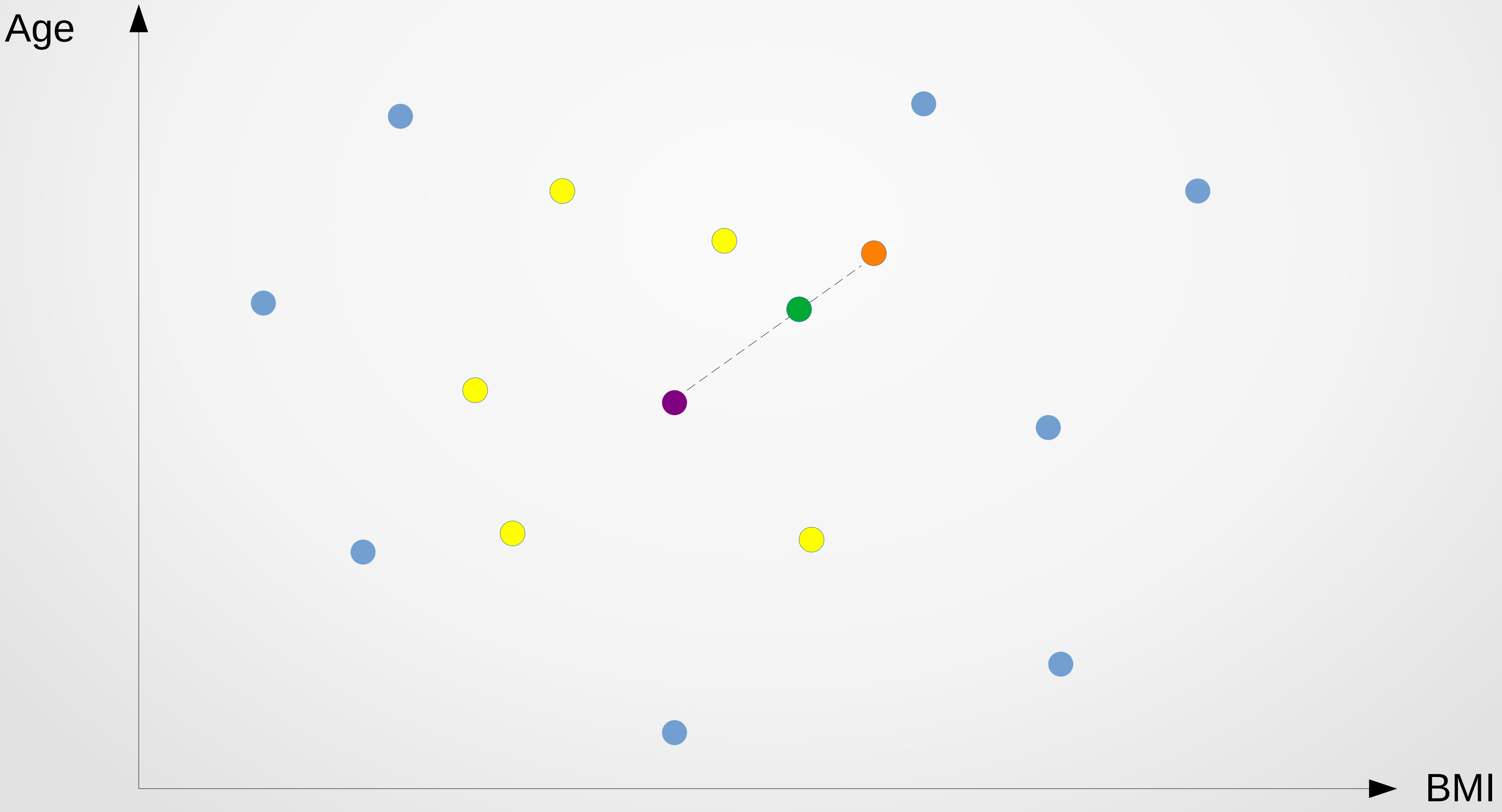
SMOTE

3. Pick one of the *k* nearest neighbours at random



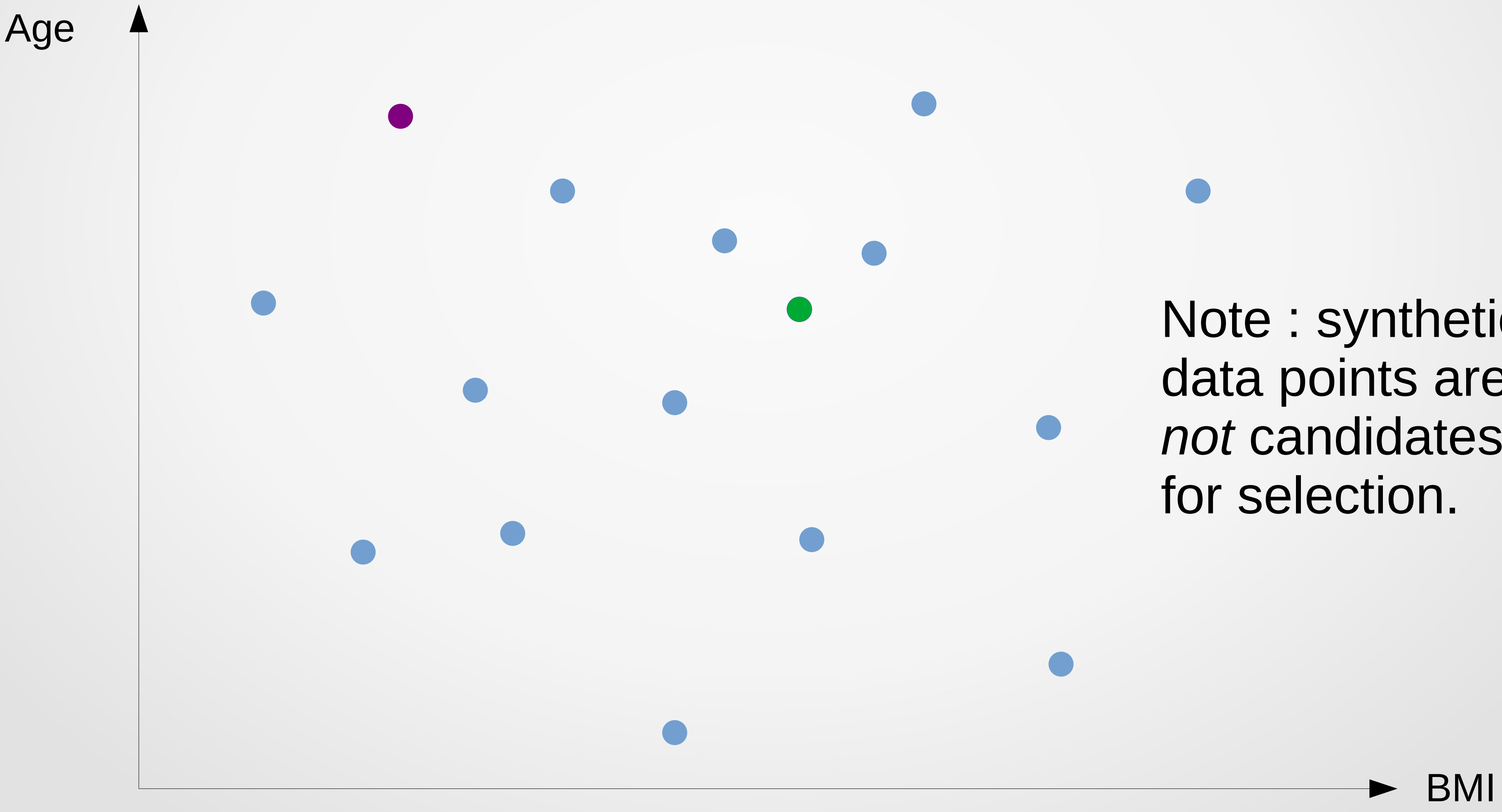
SMOTE

4. Generate a new data point at a random distance between the two.



SMOTE

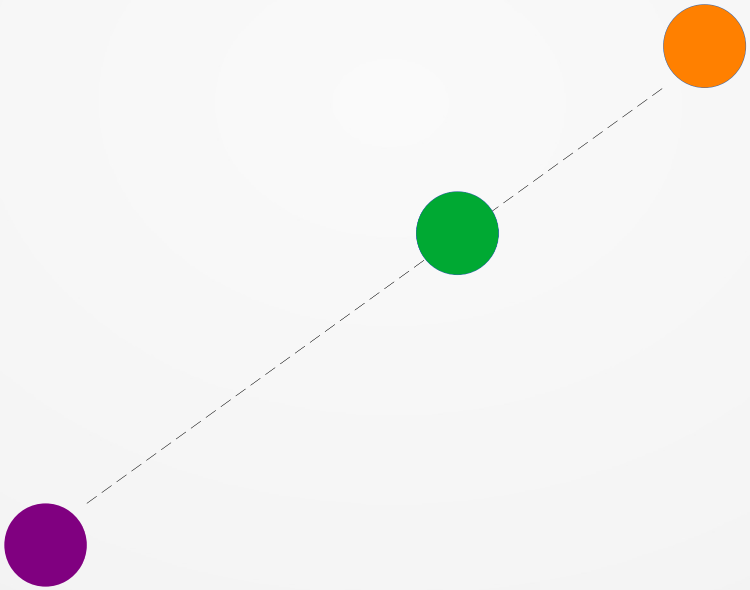
5. Repeat 1 – 4 until required number of new data points created.



Note : synthetic data points are *not* candidates for selection.

SMOTE

Quick question : why is it important that the distance between the two data points (at which the new synthetic data point will sit) is random?



Integer, Binary and Categorical Data

The data points that SMOTE generates are *floating point* numbers (eg 25.7, 6.17359).

But very often our features might contain data that is :

Integer (whole numbers)
Binary / Boolean (1 or 0, True / False)
Categorical (eg Module 1 / Module 2 / Module 3)

There are different ways in which we can deal with these kinds of data so we can generate synthetic data points for these kinds of features.

Integer, Binary and Categorical Data

Here’s how we’ll deal with each of these types of data.

Integer (whole numbers)
We’ll take the synthetic data floating point number, and round it to the nearest integer.

Binary / Boolean (1 or 0, True / False)
We’ll code our binary / boolean values as 0s and 1s before we generate the synthetic data. Then, we’ll round our synthetic data points to the nearest integer (as above).

Categorical (eg Module 1 / Module 2 / Module 3)
See next slide...

Categorical Data

Categorical (eg Module 1 / Module 2 / Module 3)
First, before we generate the synthetic data, we’ll *One-Hot Encode* the categorical data :

Patient	Ward 1	Ward 2	Ward 3	Ward 4
1	0	1	0	0
2	1	0	0	0
3	0	0	1	0
4	0	0	0	1
5	1	0	0	0

Categorical Data

Then we can generate the synthetic data. This will generate floating point numbers for each classification for a new data point (representing a new “patient” in the above), as they are separate *features*. So we’ll just find the category with the highest value, and set that value to 1, and set the values of the other categories to 0. For example :

Raw created data points (floating points)

Patient	Ward 1	Ward 2	Ward 3	Ward 4
New 1	0.34789	0.78906	0.98761	0.45808

This is the highest value, so we’ll set this to be the selected category (1) for this new synthetic data patient. We’ll set the others to 0.

Finalised Synthetic Data

Patient	Ward 1	Ward 2	Ward 3	Ward 4
New 1	0	0	1	0

imbalanced-learn

imbalanced-learn is a Python package which contains resampling techniques that are useful when you have imbalance between classes in your machine learning data. This includes an implementation of SMOTE, and it’s the implementation we will use in this lecture.

To install imbalanced-learn, we simply use :

pip install imbalanced-learn

Total Data Points

When we use the imbalanced-learn SMOTE implementation, we need to provide it with the final number of data points in each class, comprised of all the data points in the original data + however many synthetic data points we want.

Let’s consider an example. Let’s imagine we have two classes – 0 and 1. For class 0, we have 100 data points, and for class 1 we have just 50 data points.

Now let’s imagine that we want to rebalance things so we have a total of 200 data points for each class, so we’ll need an additional 100 synthetic data points for class 0 and an additional 150 synthetic data points for class 1.

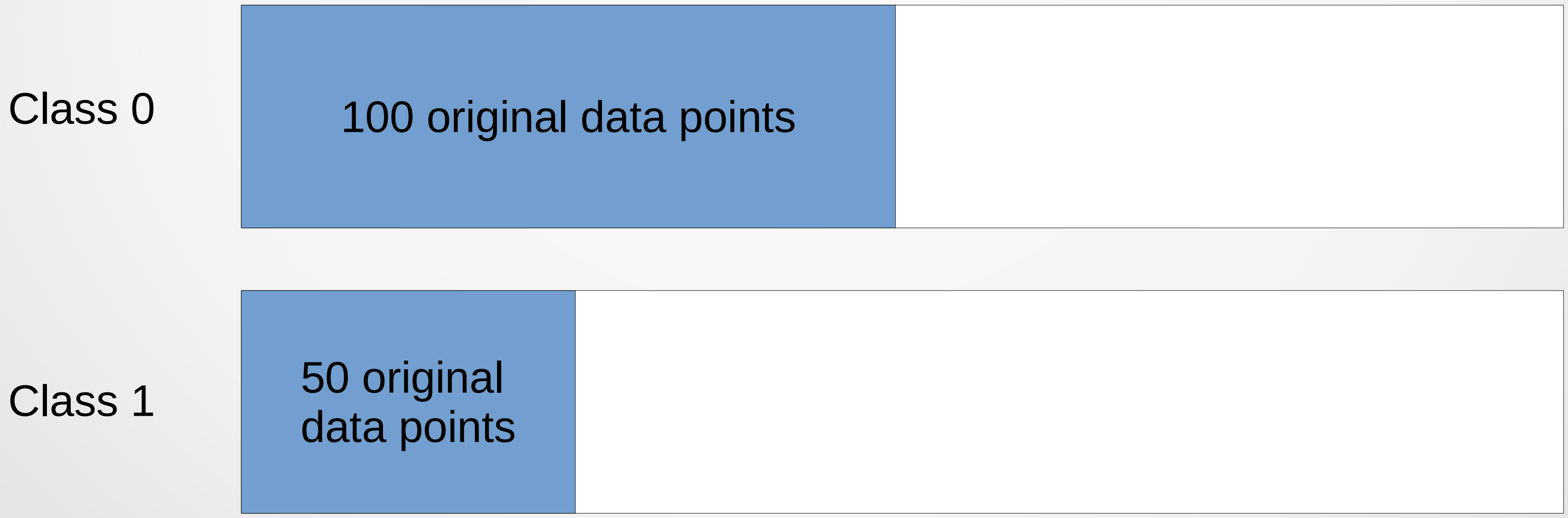
Total Data Points

Here’s how the SMOTE implementation works. We first tell it how many *total* data points we will need for each class. This is the number of data points we have in the original data for a class + however many new (synthetic) data points we want to create for that class.

Class 0	100 original (real) data points	+	50 new (synthetic) data points	TOTAL = 200
Class 1	50 original (real) data points	+	150 new (synthetic) data points	TOTAL = 200

Total Data Points

SMOTE now knows it has 200 “slots” to fill for each class. First it fills the slots in each class with the original data for each class.



Class 0

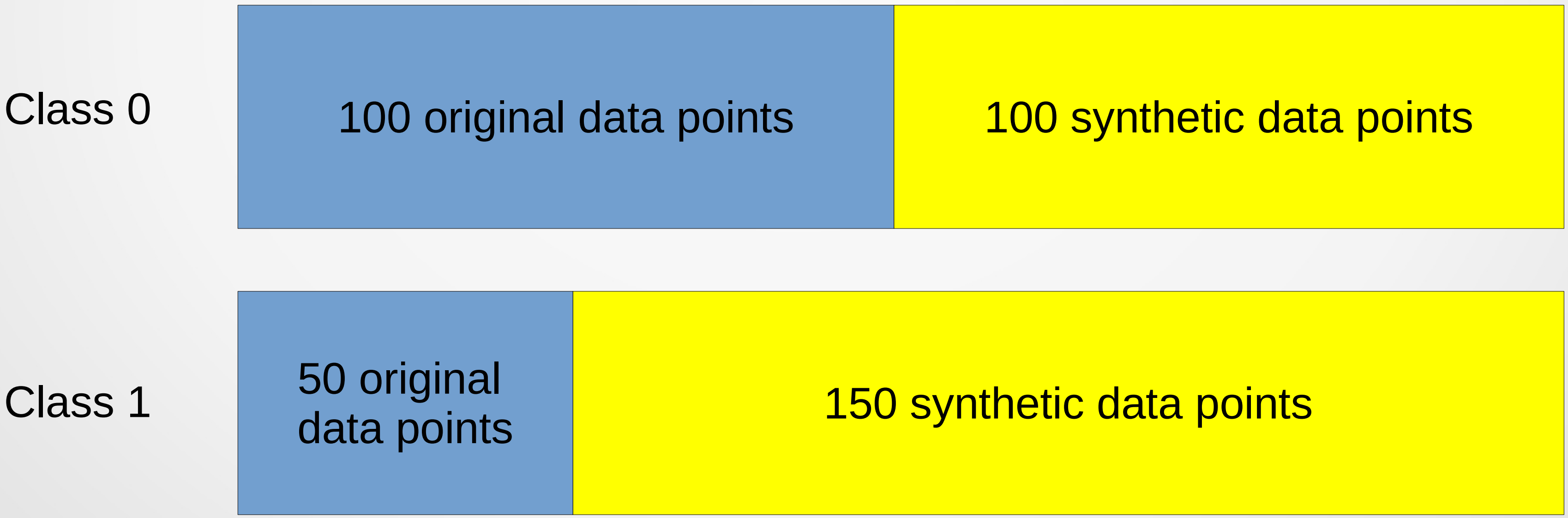
100 original data points

Class 1

50 original data points

Total Data Points

Then it applies the SMOTE oversampling technique we described earlier to generate synthetic data points to fill the remaining slots in each class



Class 0

100 original data points

100 synthetic data points

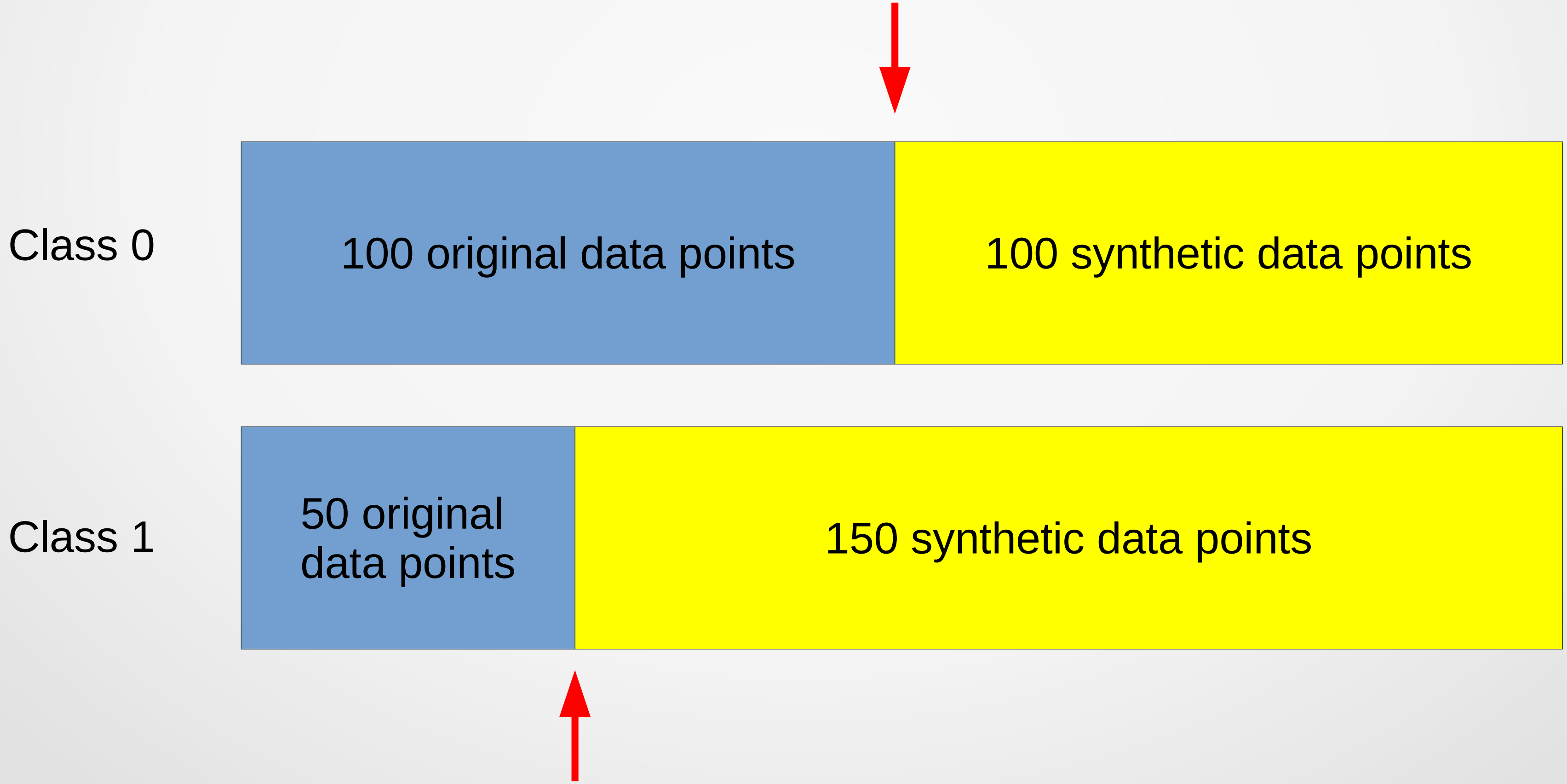
Class 1

50 original data points

150 synthetic data points

Total Data Points

When we want to grab out the synthetic data, we can just grab it from after however many original data points we've got



Removing Duplicate or Close Data

When we generate synthetic data, we remove data points that are identical or very close to the real world data.

Why do we do this?

Removing Duplicate or Close Data

Remember, there are typically two reasons for creating Synthetic Data :

- Because we may want to share or publish data for use in a model, but the original data cannot be shared or published

- Because we may not have enough data for our Machine Learning model (e.g. for a particular outcome) and we need to create more to train and improve the performance of our model

For the first reason, we need to ensure that somebody can't tell what the original data was. We need it to be a *good* fake, but not *too good* (or identical).

Removing Duplicate or Close Data

So, once we’ve generated our synthetic data, we typically :

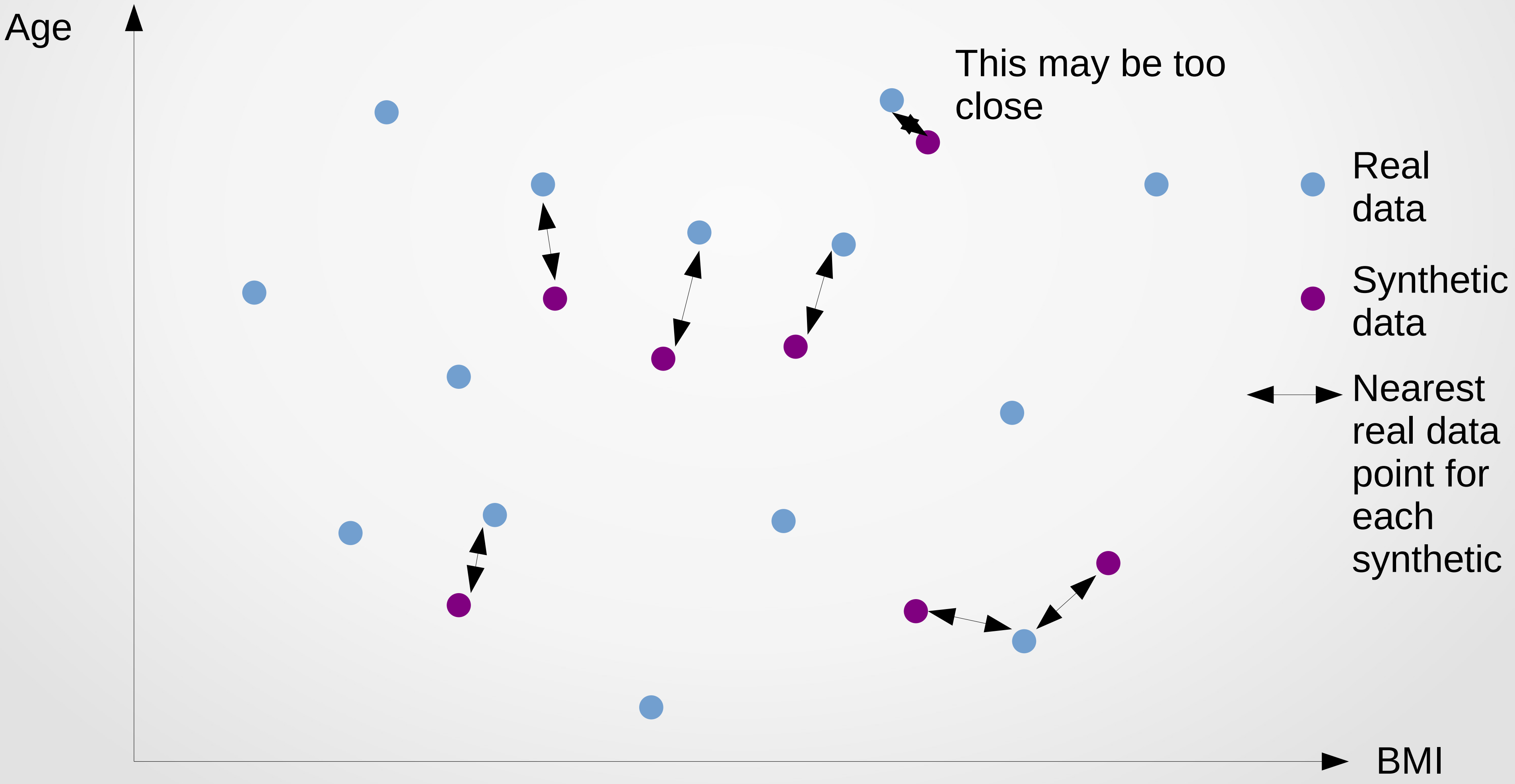
- look to remove synthetic data points that are identical to real world data
- remove the x% of synthetic data points that are closest to the real world data

The specific % we choose to remove will vary depending on our data (and the synthetic data that’s been generated). 10% is a good initial value to go with. You could also do a more in-depth analysis that looks at how close your synthetic values are.

Because we need to get rid of duplicates and close values, when we tell SMOTE we want synthetic data points, it’s useful to generate *far more* than we might need (e.g. double) to allow for the fact that we’ll have to remove some of them.

Using the Cartesian Distance

We identify the closeness of points using Cartesian distances – the same way in which nearest neighbours are selected in SMOTE.



Exercise 1

After a 10 minute break, you'll assemble in your Peer Support Groups. You'll have 90 minutes to do the following :

a) First, work through the 30_synthetic_data_SMOTE.ipynb notebook as a group. Read through it carefully and make sure you understand what the code is doing. Much of the initial stuff we covered earlier in the course so there isn't so much explanation here – refer back to your notes from earlier in Module 7 if you've forgotten any of this.

b) Once you're all happy that you understand how things are working, you must then, as a group, generate some synthetic data for a new dataset, and use it to train a Linear Regression model, comparing its accuracy with the accuracy from using the original data. I've supplied you with data on predicting strokes from a Kaggle dataset (https://bit.ly/3IktstF), but I've preprocessed the data for you (so use the data I've given you, rather than the raw data). The pre-processed data is stored in the *data* folder – the file is *processed_stroke.csv*. Take a look at the data first to make sure you understand what it contains and to get a feel for it.

You should either create a new notebook, or you may, if you prefer, use the template Exercise_1_template.ipynb, which splits things into sections for you.

After we come back, I'll ask a couple groups to share their solutions, and we'll also have a bit of discussion of ideas for data to which you could apply these methods, so have a chat through some ideas as you go through the exercise.