7 Exploring Ethical Concerns in Machine Learning

Following on from the previous lecture, in this worksheet you will be using a technique that allows you to explain and interpret predictions made by a neural network. After that you'll have some time to return to your classifier from the first worksheet and evaluate it in the context of the ethical concepts we just covered.

The neural network we'll be looking at for the first part of this session requires *tensorflow*, which is an industry standard collection of tools and libraries used in high-end machine learning. Before we start R we need to tell the system that it should load the tensorflow code (so that we can use it). On the lab machines, we do this via Anaconda (tensorflow is written in a language called Python, and software engineers are very good and wasting a lot of time thinking of funny names). We also need to download an image to classify using a utility called *wget*.

```
module load Anaconda
wget("https://www.website.com/image.jpg")
```

The file you choose should be a jpeg file, and preferably small—you can filter search results in Google to only show you smaller images. Note the name of the file, which should be printed out by the *wget* utility, as you'll need this in the next step.

8 Setting Up the Image Classifier

Load the *keras*, *lime*, *magick*, and *ggplot2* packages in the same way you loaded *caret* in the earlier practical. Yes, we are actually using a package that purports to do magic. Welcome to computer science.

Training a complicated model from scratch can take hours, days, or even weeks/months on the computers we have in the lab. For today, we're going to use a pre-trained model called vgg_16, which is a *massive* convolutional neural network already configured for image detection.

```
model = application_vgg16(weights = "imagenet", include_top = TRUE)
```

Load in the image you downloaded earlier from the internet and view it:

```
img = image_read(file_name_from_earlier.jpg')
plot(as.raster(img))
```

Because typing out the whole file name is tedious, R will auto-complete it for you if you type a part of it and press <*Tab*>.

Images that you download are defined as a set of pixels. This is a problem given that the neural network thinks of the image as sets tensors, so we need to set up some pre-processing for the model to make sure that everything that goes in is in the right format:

```
image_prep = function(x) {
    arrays = lapply(x, function(path) {
        img = image_load(path, target_size = c(224,224))
        x = image_to_array(img)
        x = array_reshape(x, c(1, dim(x)))
        x = imagenet_preprocess_input(x)
    })
    do.call(abind::abind, c(arrays, list(along = 1)))
}
```

Lets see what the network makes of our downloaded image:

```
pred = predict(model, image_prep(file_name_from_earlier.jpg))
imagenet_decode_predictions(pred)
```

9 Explaining the Neural Network

Set up the LIME explainer using the model from earlier:

```
mod_lbls = readRDS(system.file('extdata', 'imagenet_labels.rds', package='lime'))
explainer = lime(file_name_from_earlier.jpg, as_classifier(model, mod_lbls), image_prep)
```

The explainer works by creating local models for different parts of the image, and then analysing the results of these models to see which parts of the image are responsible for its overall classification. The technical term for these mini images is *superpixels*. We can see the effect of partitioning our image into different number of super pixels by using Lime's built in tool:

```
plot_superpixels(file_name_from_earlier.jpg)
plot_superpixels(file_name_from_earlier.jpg, n_superpixels = 50)
```

The number of superpixels you want to use for the explanation is entered as n_features below. Note that this is a much more computationally complex task than the original classification, as Lime has to feed each superpixel into the neural network multiple times in order to build the local models. Expect this process to take a few minutes. **You might want to open another terminal window to continue with the rest of the worksheet and come back to this later.**

```
explanation = explain(file_name_from_earlier.jpg, explainer, n_labels = 2, n_features =
```

Finally, we want to visualise our explanation (otherwise, what would be the point). Lime gives us a number of nice options here:

- Pass no options to just see supporting superpixels highlighted in green
- Pass show_negative=TRUE to also plot contradictory superpixels in red
- Pass display='block' to only show the parts of the image that supported the classification

```
plot_image_explanation(explanation, show_negative=TRUE)
```

10 Revisiting the Classifier Challenge

In the previous session we explored four different definitions of fairness that we might apply to algorithms:

1. Profit Maximisation: develop the classifier with the highest overall accuracy
2. Demographic Parity: ensure that the probability of predicting a particular outcome is equal across groups
3. Equal Accuracy: ensure that we predict outcomes for different groups with the same accuracy
4. Equal Opportunity: ensure that a we make particular types of mistakes with the same frequency across groups

Revisit the classifier that you built in the first practical session. Explore how it treats different groups of people. To do this you'll need to use a combination of tables and confusion matrices like you did in the previous practical. You'll also need to make subsets of your data set in order to compare different groups of people. You can make subsets of your test data in R using commands like the following:

```
test[test$race=="African-American",]
```

You can use these subsets when making predictions like so:

```
predict_tree = predict(object=model_tree,test[test$race=="African-American",7:9])
```

Because it is easy to make typos here that R will helpfully neglect to tell you about, it's worth running *nrow* on your subset to make sure that it actually contains some data (i.e if you type race=="Caucadion", then R will return you a subset that correctly, but unhelpfully, contains no data).
You'll need to use a combination of subsets, tables, and confusion matrices in order to do properly comparisons.
Compare your classifier's predictions on different groups—how do the accuracy and false positive rates change?

What happens if you train a model for each group you want to classify (maybe try an example with a low number of categories, like gender or age_cat)?

Does your classifier meet any of the four fairness models above? What might the ethical implications be if your classifier was used in real life?

11 Comparisons with COMPAS

If you have time, also load the "crime_data_full.csv" data set. This contains the same records as we've been using, but also includes two outputs from COMPAS, the decile score (a risk rating out of 10) and a categorical variable that represents whether the decile score is considered high or low risk.

We saw some of the problems with COMAS ratings in the previous session. Explore the new data set a little using the same techniques you used on your own classifier. How does COMPAS compare to your own efforts in terms of accuracy and ethics?