**Introduction Video**

Hi everyone! My name is Helen Schmidt, and I’m a graduate student in the Social and Affective Neuroscience Lab at Temple University. The focus of this workshop will be on machine learning – what it is, how to use it in R, and why it’s a useful tool for data analysis.

To get us started, let’s first talk about what machine learning is. Broadly speaking, machine learning is the process of teaching machines to learn information. These machines can be computers or entire computing systems, or they can be simpler, something like an algorithm or a model. Machine learning to date has been widely used in computer science, data science, and neuroscience, and researchers with a psychology background are beginning to adopt this approach for more data-driven analyses.

Machine learning involves training a machine to learn about information and make predictions or classifications based on that learning. It’s a subset of artificial intelligence and has grown in popularity alongside the recent AI boom we’ve seen in the last year or so, as of the time I’m recording this video. While artificial intelligence describes the general development of computers that can mimic cognitive function, machine learning is more specific, using algorithms and large amounts of data to make informed decisions for specific tasks.

Machine learning has become a popular tool for researchers in psychology and neuroscience thanks to huge increases in computing power and access to datasets with thousands or millions of observations. Psychologists have seen great success recently in applying machine learning methods to gather insights about human behavior and cognition. Machine learning models can not only reduce time, effort, and human error costs, but they can also be applied flexibly, across projects and datasets, including behavioral, survey, and neuroimaging data.

The goal of this workshop tutorial is to introduce a variety of machine learning methods that can be applied to psychological data. We will examine the differences between unsupervised and supervised machine learning models, and talk about the contexts to which they are best applied. This tutorial will also touch on model evaluation and comparison and unpack how to determine certain model parameters given your data.

An important note here is that this tutorial will be by no means an exhaustive list of machine learning approaches. Rather, they are meant to serve as introductory examples and methods that can be employed specifically in the R programming language. If you’re interested in more complex machine learning approaches, a majority of these are supported in Python using libraries like PyTorch, TensorFlow, and Keras. These are more commonly what you might see in a machine learning tutorial online, but I hope this R-focused workshop serves as a complement to these other methods.

Like I said, this tutorial is intended to be an introduction to machine learning with R, but you should have some existing familiarity with scripting and R specifically before starting this tutorial.

To wrap up this introduction video, I’m going to briefly outline the format of this tutorial. We will start off by exploring the example dataset and getting familiar with its variables. Then we will introduce supervised machine learning methods and highlight two examples – k-nearest neighbors and classification trees. After that, we will introduce unsupervised machine learning methods and highlight two other examples – k-means clustering and principal components analysis. For each method that we introduce, there will be an interactive question for you to try out yourself.

**Dataset Exploration Video**

The dataset that you will be using throughout this tutorial comes from the palmer penguins R package. This dataset (and the images you will see sprinkled throughout the tutorial) were created by Allison Horst and features data from penguins. This data was collected by researchers at the Palmer Long Term Ecological Research Station in Antarctica. There are three different penguin species collected from three islands in the Palmer archipelago.

What kind of data is included in this dataset? If we look at the variables, we have species, which denotes the species name for each penguin. There’s island, which denotes the name of the island each penguin lived on. Bill\_length\_mm denotes the length of the penguin’s bill in millimeters. Bill\_depth\_mm denotes the depth of the penguin’s bill in millimeters.

I’ve included an image to help clarify the difference between bill depth and bill length. The bill depth refers to the depth in millimeters of the penguin’s bill if looking at its side profile. The bill length refers to the length in millimeters of the bill from front of the bill to where it meets the head.

In the variables we also have flipper\_length\_mm which refers to the length of the penguin’s flippers in millimeters. We have body\_mass\_g which refers to the mass of the penguin in grams. Finally, we have the penguin's sex denoted as either male or female and we have the year the data was collected.

Before we start with any machine learning, it’s important to get a sense of the data you’re working with. In the Load Data code chunk, we can load the data from the palmer penguins R package. Then we can examine the data using functions from the skimr R package to show more information about our categorical and numeric variables.

The next important step for the sake of simplicity, is to remove any penguins with missing data. While you can run permutation tests to estimate missing data, here, we will just remove the missing data. The final step in this code chunk is to get a correlation table of the numeric variables. This is helpful in determining what variables might be useful to examine in the context of a machine learning model. For example, flipper length and body mass are highly correlated – this makes sense since the bigger the flippers the heavier the penguin is likely to be.

Ok now that we’ve loaded the data and examined it a little bit, let’s create some visualizations. The first plot you’ll make is the frequency of body masses across penguin species. You can see that Gentoo penguins tend to be heavier than Adelie and Chinstrap penguins. Next, you’ll make a plot that compares bill length to bill depth across penguin species. While these two variables were slightly negatively correlated when collapsing across species, we can see here that they’re actually slightly positively correlated within each species.

Let’s also plot body mass compared to flipper length. Similar to the body mass distribution frequency plot we just made, you can see a clear separation of the data for Gentoo penguins in particular.

Finally, and this is just a fun plot to make, let’s combine all these variables and see how they compare across species. You’ll do this by creating a radar plot. This gives us a sense of how each of these species compare to each other on these four numerical variables.

Ok, so now that we’ve examined and visualized the data we’re working with, let’s outline how we can use machine learning to analyze this data. In the context of psychology, a common area of investigation is determining differences between individuals. We often care about what makes an intervention or cognitive strategy more effective for some individuals versus others. Machine learning can be particularly useful when we want to group data or compare groups, because we can reveal underlying structures or similarities that may not be apparent to human raters.

In the palmer penguins dataset, we can use the four data points for each penguin, including bill length, bill depth, flipper length, and body mass, as well as their species grouping, Gentoo, Adelie, and Chinstrap, to develop a model that can be applied to new data. This would be a supervised machine learning approach. We can also see how well a model can classify penguin data into species groups when we don’t tell the model the species groupings, using an unsupervised machine learning approach.

**Supervised Machine Learning Video**

Supervised machine learning is a technique for data that contains labeled examples, such that each data point contains a number of features and an associated label. The goal of supervised learning is to map these features to labels based on example data and apply that mapping to classify new data. In the case of the Palmer penguins dataset, our features include bill length, bill depth, flipper length, and body mass, and our labels are the penguin species.

The first supervised approach we will discuss is a technique called k-nearest neighbors. This technique classifies data into specified classes. In this example, we will use the k-nearest neighbors algorithm to try to classify our penguin species based on just two of these data features, bill length and bill depth.

The first thing we need to do is split our data into a training set and a testing set. In machine learning, it is pretty standard to use 80% of your data for training and 20% for testing. It is also standard to normalize your predictors, in this case bill depth and bill length, to ensure the scales of the two variables are the same.

Ok, let’s start in the KNN Setup code chunk. Step number one is to set a seed to ensure that we get the same results every time. Because we will randomly sample our training and testing sets from the data, we want to make sure that we select the same rows for training and testing every time. We can set a seed using the set.seed function.

Next, you’ll want to generate a random set of numbers that accounts for 80% of the total number of rows in your dataset. This will serve as the training set. I’ve created a function to normalize the predictors, but you can also use the native scale function in R. Using this normalization function, I will create a new data frame called df.norm and normalize the two predictors I’m interested in. I’ll check that this normalization worked using the summary function. If it has, my variables should both range from 0 to 1, which they do.

Now let’s move to the KNN Model code chunk. Using the random set of numbers, let’s split the data into a training and testing set. You also want to extract the species names to be used as the classification argument in the k-nearest neighbors model.

Now we have everything we need to run the k-nearest neighbors model! Using the knn function, we can set the training set to df.train, the testing set to df.test, the classification argument to df.category, and set k to be equal to 3, which means that a testing point used by the algorithm will classify based on the 3 closest neighbors to that point. We can then show the accuracy of the classifier by calculating the proportion of penguins that correctly had their species classified.

Sadly, our classifier wasn’t particularly accurate at only 39.1% accuracy. However, this is above chance at 33% so the algorithm is able to classify with some degree of accuracy. Let’s try running the KNN model again with a greater number of neighbors like 10.

Interestingly, that reduced our accuracy to 38.35%. This is still above chance, but shows that increasing the number of neighbors isn’t necessarily the way to get a better classification accuracy.

Ok, now it’s time for you to try an example. In example number 1, use the KNN algorithm to classify penguin species based on flipper length and body mass. Try a range of neighbors when employing the model to see if classification accuracy changes.

In example number 2, use the KNN algorithm to classify penguin species using all four data features. Again, try a range of neighbors when employing the model to see if classification accuracy changes.

Feel free to pause the video here while you try these examples yourself. Also as a note to the COG organizers, feel free to cut the video here if you’re trying to find places to split the video into smaller chunks.

Ok, now that you’ve had a chance to try k-nearest neighbors classification, let’s move on to another supervised learning technique. This technique is called classification or decision trees. This is a technique used to draw conclusions about a set of data observations. Classification trees specifically are models where the target variable, in our case the species variable, can take on a set of discrete values, like Chinstrap, Adelie, and Gentoo. We can train a decision tree model using a selection of our data and then apply it to a set of test data, like we did with k-nearest neighbors. Finally, we can examine how accurately the decision tree was able to classify our penguins into their correct species category.

Let’s begin with the Classification Trees code chunk in the tutorial. Because we’re taking a training and testing set from our data, we need to set a seed to ensure the same results every time. This time, we’ll create a function to split our data into training and testing sets. We will use 80% of the data to train the model and 20% to test the model.

Now we can fit our decision tree model using the rpart function. We want to specify species in the model formula because that is how we’re trying to classify the individual penguins. We also want to specify the method as “class” because the species outcome can only take on three possible discrete values. We can now run the model using our training data set!

Once we run the model, the rpart.plot function creates a really helpful decision tree plot. When we previously investigated the data, we saw that adelie penguins had the shortest flippers. The model identified flipper length as a way to distinguish between adelie penguins and other penguins species. If we move to looking at the next level of the tree, once the model classifies using the flipper length data, we see that it further distinguishes between adelie and chinstrap penguins based on bill length and between gentoo and chinstrap penguins using the island that the penguin was found on.

Because we used the entire dataset, the model had the option of using information from island, sex, and year as part of its classification process, so we might expect that the accuracy of this model with more data information might be more accurate than our KNN model. Let’s find out using our test dataset.

We can use the predict function on the test dataset along with our model and get a table of the penguins that the model could correctly classify. And this model indeed did pretty well! It correctly predicted 26 Adelie penguins but incorrectly classified two Chinstrap penguins as Adelie penguins. If we measure the overall performance of the model, we see that it classified penguins at a staggering 95.52% accuracy, well above our KNN model! But again, this makes sense since we’re using more data to train the model.

Now it’s your turn to try a classification tree for yourself. Since island name and flipper length were important factors for this model to classify the penguins, what happens to the decision tree if you remove island name and flipper length from the data frame, and only classify based on the remaining data? Does it change the model accuracy? What does that tell you about the removed data?

Give that example a try and then join me after the break to discuss unsupervised machine learning models.

**Unsupervised Machine Learning Video**

Unsupervised machine learning is a machine learning technique for data that does not contain associated labels or categories. There are two main goals to unsupervised learning – clustering and dimensionality reduction. In this section of the tutorial, we will introduce two techniques. The first is k-means clustering, which will help cluster the data into categories, and the second is called principal components analysis, which will help reduce dimensionality within the data.

The first thing we need to do is remove the species labels from our dataset. In unsupervised machine learning, there are no labels provided alongside the data. Since we’re working with a dataset that already has labels, we can remove these labels and see how well unsupervised machine learning models perform. In our new dataset, we will only select bill length, bill depth, flipper length, and body mass variables and remove labels like penguin species, penguin sex, and island location.

Let’s start off with a technique called k-means clustering. This unsupervised technique assigns each data observation to one of K clusters based on the nearest cluster centroid. In this technique, we need to specify the number of clusters we might expect from the data. We can do this manually or iteratively to find the optimal number of clusters based on the data.

I’ve included a series of visualizations in the tutorial that I find really helpful in visualizing the process of assigning data observations to these cluster centroids.

In this example, we will use a K value of 3 as a hypothesized number of clusters if we want to classify species. We can also use a K value of 2 if we want to classify a number of clusters based on penguin sex. For simplicity, we will only use flipper length and bill length observations. K-means clustering can handle more than 2 dimensions of data, but in order to compare the classifications to the visualizations made with the full dataset, we will just use two dimensions.

Let’s start in the K-means clustering code chunk. From the new unlabeled dataset, I just want to extract the flipper length and bill length variables. We can create another plot to visualize how bill length compares to flipper length by species.

Running k-means clustering is as simple as using the kmeans function in the R stats package. We have to set a seed and then we can run the kmeans function on our unlabeled dataset that contains just flipper length and bill length. We can set the centers or K value to 3 since we know there are three different species of penguin in this dataset. I will also set the nstart to 25, which tells the model to choose 25 random rows in the dataset as initial centers.

Once we run the function, we can examine the structure of the resulting clusters and visualize them using the fviz\_cluster function. If we eyeball the plot made with the actual species labels and the plot made from the kmeans clustering, they are generally pretty similar!

What if we try to cluster based on penguin sex, therefore using a K value of 2? We can create a similar whole-dataset visualization based on penguin sex and repeat the k means clustering. As we can see, there is more overlap between the two groups of penguins based on sex, so the kmeans clustering plot doesn't look as similar to the real data.

But what about if we have data where we don’t KNOW the true number of groups, as we so often do with multidimensional behavioral or neuroimaging data? Well, we can determine the optimal number of clusters based on our data using a few different methods. This process can be somewhat subjective, but we can use three different methods together to attempt to reach a consensus about the optimal K.

The first method is called the elbow method and aims to find a K that has the smallest within-cluster sum of squares, or WSS. The total WSS measures how compact the clusters are. The elbow method examines the total WSS as a function of the number of clusters, and aims to find an optimal number of clusters such that adding another cluster wouldn’t improve the total WSS. We can use the fviz\_nbclust function in the Optimizing Clusters - WSS code chunk to run the elbow method on our unlabeled, subset dataset.

Based on this plot, it looks like the optimal number is 4, since the total WSS doesn’t get much lower for 5 clusters compared to 4. Comparatively, there are notable jumps down in total WSS from 1 to 2, 2 to 3, and even 3 to 4.

We can also use a method called the average silhouette method which measures the quality of a clustering. A high average silhouette width indicates good clustering. With this method, the optimal number for K is the number that maximizes the average silhouette over a range of possible values for K. We can use the fviz\_nbclust function again in the Optimizing Clusters - ASM code chunk, and specify the silhouette method.

Based on this plot, it looks like the optimal number is 2 because 2 clusters corresponds to the maximum average silhouette width.

The final method we’ll try is a statistical method called the gap statistic method. This method compares the data against a null hypothesis. The gap statistic compares the total within intra-cluster variation for different values of K with their expected values under a null reference distribution of the data. The optimal number of clusters is a value that maximizes the gap statistics and means that the clustering structure is far from a random, uniform distribution of observations. We can use a function called clusGap in the Optimizing Clusters - Gap Statistic code chunk.

This method also suggests that 4 is the optimal number of clusters. This is consistent with the 4 clusters recommended by the Elbow Method, but inconsistent with the number recommended by the silhouette method. There is not a clear consensus, but we can be relatively confident that using a K value between 2 and 4 is likely optimal. And we can visualize how our data is classified based on these varying K values.

In the KMC code chunk, we first repeat the kmeans clustering using a K value of 2, 3, and 4. We can then plot these together to compare. Now, because we know that this data can be grouped into three known species, the recommended cluster of 4 doesn’t actually make sense. But when we don’t know the true category structure for the data, using these techniques to find an optimal number of clusters can be a critical step to *identifying* that underlying structure.

Now it’s time for you to try kmeans clustering for yourself! Using only the bill depth and body mass variables, repeat the k-means clustering and the steps to find an optimal K. Does the number of optimal clusters change?

Feel free to pause the video here while you try these examples yourself. Also as a note to the COG organizers, feel free to cut the video here if you’re trying to find places to split the video into smaller chunks.

Alright, let’s introduce the final technique in this tutorial, an unsupervised approach called principal components analysis, or PCA. This is a technique that reduces dimensionality in a dataset. In the Palmer penguins data, we have four data dimensions – bill length, bill depth, body mass, and flipper length. As we just discussed with k-means clustering, it’s difficult to visualize all four of these data dimensions together. Using PCA, we can transform the data into lower-dimensional space and collapse highly correlated variables together. We can then extract important information and visualize the data more easily.

The first step within the PCA code chunk is to normalize all four variables in the unlabeled dataset. We can do this using the normalization function we created earlier in the tutorial. Then, we can compute and visualize a correlation matrix for all these variables to see which are highly correlated with each other. Like we saw during our data exploration, body mass and flipper length are highly correlated, so these may be good dimension reduction candidates.

Next, we’ll conduct our PCA using the princomp function in the stats package. We conduct the PCA using the correlation matrix in order to help the model identify these highly correlated variables. Using the summary function, we can see the standard deviation, the proportion of variance in the data explained, and cumulative proportion of variance explained for each newly created component. The number of components corresponds to the number of variables in the data. The first component accounts for 95.8% of the variance and the second component accounts for 3.85%, so together they account for nearly 100% of the total variance in the data.

But what do these components even mean? We can investigate that using the loadings of each component. Component 1 has positive values for bill length, flipper length, and body mass but a negative value for bill depth. This implies that penguins in component one are heavier and have longer flippers and bill lengths, but shorter bill depths. Component 2 has positive values for bill length and bill depth but negative values for flipper length and body mass. This implies that penguins in component 2 are lighter and smaller, but have bigger bills.

We can visualize the relative importance of each component using something called a scree plot and can be used to determine the number of components to retain. Clearly we want to retain component 1 and component 2 explains some variance, so we will retain components 1 and 2.

We can also examine a biplot to see similarities and differences between the samples and show the impact of each attribute on each of the components. We can gather a few pieces of information from this plot. Variables that are grouped together are positively correlated to each other. The higher the distance between the variable and the origin, the better represented that value is. Finally, variables that are negatively correlated are displayed to the opposite sides of the biplot’s origin.

Now it’s time for you to try out PCA. Let’s pretend that the penguin species variable is the only “label” variable. If you re-code penguin sex as a numeric variable where 0 is male and 1 is female, what is similar and / or different in the PCA analysis?

**Conclusion Video**

I hope this tutorial has served as a useful introduction to machine learning methods in R. We discussed how to apply both supervised and unsupervised machine learning models to a publicly available dataset. These techniques include k-nearest neighbors, decision trees, k-means clustering, and principal components analysis.

By using these relatively simple tools in a programming language that is already popular amongst psychology and neuroscience researchers, we are able to make these techniques more easily accessible and applicable to a variety of datasets, research projects, and topics.

This is by no means a comprehensive tutorial about machine learning, and if you’re interested in learning more, or how to apply more advanced machine learning models to your data, I highly recommend checking out some machine learning tutorials in the Python programming language. There are libraries available like PyTorch, TensorFlow, and Keras that are designed to support powerful machine learning models. I hope this tutorial has served as a good introduction and that you feel empowered to continue learning more about these techniques.