MATHS 7107 Data Taming Tutorial

Principal Component Analysis (PCA)

Today I recommend you complete this tutorial in a script, not an R Markdown document as you may see some errors with some plots.

Getting the data

Welcome to a living, working example of the beast that is PCA. We are going to delve into how to do PCA with the use of TidyModels, and see how to use it for data exploration and dimension reduction.

For this example, we are going to use the `ad_data` from the `modeldata` package (make sure `modeldata` is installed). Craig-Schapiro et al. (2011) describe a clinical study of 333 patients, including some with mild cognitive impairment (early stages Alzheimer's disease) as well as healthy individuals. For each patient, cerebral spinal fluid (CSF) samples were taken with the goal to determine if patients with the impairment could be differentiated from healthy controls. A whopping 131 variables were collected for each patient, including demographic data, genotype data, protein measurements of various biomarkers and clinical dementia scores.

For the purposes of this dataset, the dementia scores have been converted into a hard classification of 'control' or 'impaired'. If we were to approach this from a supervised learning framework, this variable (called `Class` in the dataset) would be our outcome in our classification model. Although this might be an interesting approach to take, today is all about the unsupervised framework, so let's explore what interesting tidbits we can get out of the 130 predictor variables in the data.

First up, let's load the data and give it a skim (I might regret this).

```r
# This wont work in R Markdown, but does in an R script
ad_data %>%
  skim()
```

I definitely regret skimming that data. That is a lot of variables. You see, we have two categorical variables (the outcome `Class` and the `Genotype`), and then the rest are numeric variables. All in all, this is just too much to get our heads around right now. If you want to look at all the possible combinations of predictors you would have to examine 8385 plots all up. To be completely honest, nobody's got time for that. So, let's see what PCA can do for us.

Using a recipe

TidyModels has a very neat package that makes preprocessing really simple, and that package is the `recipes` package. A recipe is a way for us to tell R what we want it to do to our data in a precise manner, and is completely analogous to a cooking recipe. As a working example, we are going to preprocess the data before we perform PCA. In our recipe, there are going to be three steps:

- We will need to convert `Genotype` to a dummy variable, since PCA works purely for numerical variables and not categorical variables.
- We will need to `normalize` our predictor variables. PCA works better with data that is all on the same scale and normalised.
- We need to actually perform the PCA.

So, let's go ahead and make a recipe for our data.

Start a recipe

The first thing you need to do in a recipe is declare that you are starting one. For instance, we start with:

```
ad_recipe <- recipe( Class ~ ., data = ad_data )
ad_recipe
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor        130
```

I have used the `recipe` function, and have given it two important pieces of information. The first is the formula I want to fit in our model. So, I have given it our `Class ~ .` formula from before. This defines what we want our `outcome` variables and our `predictor` variables to be. So, `Class` is the outcome and everything else are predictors. I have also given it the dataset we are interested in, so it knows where to look for the outcomes and predictors. As you can see, the recipe has correctly identified that there is 1 outcome, and 130 predictor variables.

Add steps

The next thing we need to do is add 'steps' to our recipe. This is where you tell R what you want to do to the data. What you want to do is convert `Genotype` to a dummy variable, normalise our predictors and perform the PCA. You do this with:

```
ad_recipe <- recipe( Class ~ ., data = ad_data ) %>%
  step_dummy( Genotype ) %>% # Convert Genotype to a dummy variable
  step_normalize( all_predictors() ) %>%  # Normalise our predictors
  step_pca( all_predictors() ) # Do the PCA.
ad_recipe
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor        130
##
## Operations:
##
## Dummy variables from Genotype
## Centering and scaling for all_predictors()
## PCA extraction with all_predictors()
```

Where you see `all_predictors`, this tells R to do the preprocessing step to all the predictor variables.

Alright, so our unprepped recipe tells us we have one outcome and 130 predictors. We are going to dummy up genotype, normalise everything and then grab some PCA. So, let's prep it to see the outcome.

Prep it

The final thing to do to complete a recipe is to cap it off by prepping it:

```
ad_prepped <- ad_recipe %>%
  prep()
ad_prepped
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor        130
##
## Training data contained 333 data points and no missing data.
##
## Operations:
##
## Dummy variables from Genotype [trained]
## Centering and scaling for ACE_CD143_Angiotensin_Converti, ACTH_Adrenocort... [trained]
## PCA extraction with ACE_CD143_Angiotensin_Converti, ACTH_Adrenocorti... [trained]
```

So, it has run nice and smoothly. Notice how our predictors have increased to 134. This is because we converted Genotype to a dummy variable, which will always add more predictors to the model if there are more than two classes in the categorical variable.

Something that is blatantly obvious here is that there are no PCA components to play with yet. This is easily remedied. First up, let's explore the PCA loadings.

Loadings

To get to the loadings for PCA, we need to use the `tidy` function from `yardstick`. So, what happens when we `tidy` a (prepped) recipe?

```
tidy( ad_prepped )
```

```
## # A tibble: 3 x 6
##   number operation type      trained skip  id
##    <int> <chr>     <chr>     <lgl>   <lgl> <chr>
## 1      1 step      dummy     TRUE    FALSE dummy_cn1PT
## 2      2 step      normalize TRUE    FALSE normalize_Kd7Nv
## 3      3 step      pca       TRUE    FALSE pca_yTeeX
```

What you see here is that it is nicely outlining the preprocessing steps we have done. The `trained` column indicates whether the step in the recipe has been completed (you can see everything has been completed). We also have the `number` column. This is important, since we can add an extra argument to `tidy` that will allow us to gather more information about each step, according to its number. For instance, if we want to get more information about the normalisation, we can type in `tidy( ad_prepped, 2 )` (give it a try). What we are after is more information on PCA, so we want to use:

```
tidy( ad_prepped, 3 )
```

```
## # A tibble: 17,956 x 4
##    terms                              value component id
##    <chr>                              <dbl> <chr>     <chr>
##  1 ACE_CD143_Angiotensin_Converti  -0.103  PC1       pca_yTeeX
##  2 ACTH_Adrenocorticotropic_Hormon -0.0347 PC1       pca_yTeeX
##  3 AXL                             -0.113  PC1       pca_yTeeX
##  4 Adiponectin                     -0.0777 PC1       pca_yTeeX
##  5 Alpha_1_Antichymotrypsin        -0.125  PC1       pca_yTeeX
##  6 Alpha_1_Antitrypsin             -0.0823 PC1       pca_yTeeX
##  7 Alpha_1_Microglobulin           -0.122  PC1       pca_yTeeX
##  8 Alpha_2_Macroglobulin           -0.114  PC1       pca_yTeeX
##  9 Angiopoietin_2_ANG_2            -0.114  PC1       pca_yTeeX
## 10 Angiotensinogen                 -0.0302 PC1       pca_yTeeX
## # ... with 17,946 more rows
```

Note: Only the first 10 rows are shown in this output.

```
tidy( ad_prepped, 3 ) %>%
  dim()
```

```
## [1] 17956     4
```

Question: How many rows and columns are there?

The first column corresponds to our different predictor variables, the second column corresponds to our loading values, the third is the principal component we are looking at, and the final is just an ID column.

Question: Why are there 17,956 rows?

What you see here is that the loading value corresponding to the predictor AXL for the first principal component is about -0.112686, the loading value for Adiponectin for the first principal component is -0.077717, and so on. But what does this tell us?

If you look at the loadings, they can tell you which variable is contributing strongly to which principal component. Since the principal components describe our most variable directions in our data, the loadings will tell us which variables are causing the most variation in our data in a given direction.

We can explore this with a lovely graph. To get this graph is a little complicated, so I am going to show it and then discuss what I have done.

```
# This may cause an error in R Markdown so use an R script
tidy( ad_prepped, 3 ) %>%
  filter( component %in% c("PC1", "PC2", "PC3", "PC4") ) %>%
  group_by( component ) %>%
  top_n(10, abs(value) ) %>%
  ungroup() %>%
  ggplot( aes( x = abs(value), y = terms, fill = value > 0 ) ) +
  geom_col(show.legend = F) +
  facet_wrap( ~ component, scales = "free") +
  ylab(NULL) # We do not need the y axis label.
```

Okay, what have we got here?

- We took our tidied PCA information.
- We filtered to get the first four principal components.
- We grouped by component to get the top 10 influential loadings by absolute value.
- We plotted the absolute value on the x-axis and the term on the y-axis to get a nice column graph. We have also coloured the bars by whether the value is positive or negative.
- We have put each component on its own plot.

That was a lot to take in. So, what does this tell us?

For instance, it tells us that things like `VCAM_1` and `TNF_RII` are the highest contributing predictors in the first principal component. You can look at these and get a real understanding for which predictors are contributing to the variance in our data, which is very useful in understanding your data.

Now, I haven't mentioned how to get the principal components. To do this, we need to `juice` our prepped recipe:

```
ad_juiced <- juice( ad_prepped )
ad_juiced %>%
  head()
```

```
## # A tibble: 6 x 6
##   Class        PC1   PC2    PC3    PC4    PC5
##   <fct>      <dbl> <dbl>  <dbl>  <dbl>  <dbl>
## 1 Control  -11.5   -1.72 -3.03  -1.02   0.701
## 2 Control   -3.02  -4.97 -1.50   0.965 -1.44
## 3 Control    0.750 -1.24  1.62  -1.92   1.20
## 4 Control   -0.531 -6.48  0.382 -0.620  2.92
## 5 Control  -11.6    4.56 -3.52   2.77  -1.04
## 6 Impaired   4.40   8.55  3.40   3.16   0.908
```

Look at that! It has returned our class (outcome) and the first five principal components. Each value you see in the principal component is the score for that subject. So, basically, we have re-jigged the predictors for each person.

Now, why five components?

Because R chose for us (we did not specify how many principal components we wanted). We will return to this in a second, but let's first look at our data in these components.

Produce a scatterplot of our first two principal components against each other, coloured by what class they are in.

The plot shows us a nice random scattering. A down side to principal components is that the interpretation of this plot is much harder than your basic scatterplot. For instance, you would interpret the points on the left of the plot (with negative `PC1`) as those with an above average (since everything was normalised) value of `VCAM_1` and `TNF_RII`, and those on the right as having below average value of `VCAM_1` and `TNF_RII` (this is because the loadings of `VCAM_1` and `TNF_RII` are negative).

Despite this obvious disadvantage, it still has the advantage of making finding relationships in our data easier and reducing our predictor space to something much more manageable.

Proportion of variation

How do you pick the number of dimensions you wish to reduce to?

This is generally done with something called the proportion of variation explained by the principal components. The idea is that we pick how much explained variance is enough for our situation. For instance, we might say: "If 90% of our variation is explained, this is good enough". We then pick the smallest number of components that explain that much variance.

So, now the question remains: "How do we do this in R?" Sadly, this is pretty ugly, but only to get our variances for our principal components. Buckle up, here we go:

```
sdev <- ad_prepped$steps[[3]]$res$sdev
```

This has given us the standard deviations of each principal component. What this is doing is the following:

- We look at our prepped recipe.
- We look at the `steps` component of the recipe.
- We access the third list item (because the PCA was our third step).
- Inside this step, we look at the residuals.

- Inside the residuals, we look at the standard deviation (`sdev`).

The good thing about this is that it is just about copy and paste. The only things that will possibly change are the name of the prepped recipe (ours was `ad_prepped`), and the step number we want to look at (ours was 3 since PCA was the third step).

Now we have some standard deviations, so let's get the proportion of variation explained by each component.

```r
ve <- sdev^2 / sum(sdev^2)
head(ve)
```

```
## [1] 0.26406736 0.10931413 0.04161419 0.02922091 0.02816129 0.02700291
```

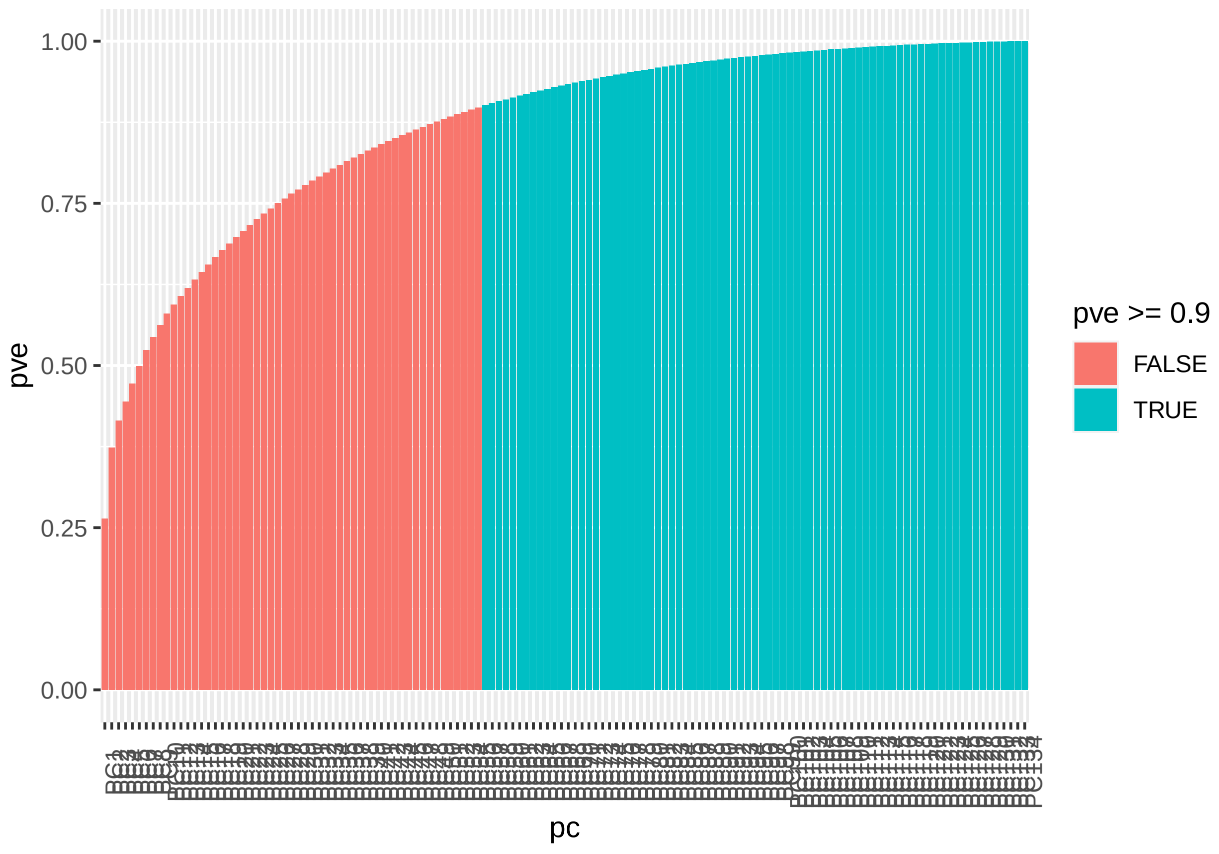
We can then get the proportion of variance explained (PVE) of each principal component by taking the cumulative sum of our vector `ve`. Let's create a tibble that has our principal components in one column and the PVE in another:

```r
PC.pve <- tibble(
  pc = fct_inorder( str_c("PC", 1:134) ), # This will create PC1, PC2, ..., PC134
  pve = cumsum( ve ) # Takes the cumulative sum to get our PVE
  )
PC.pve
```

```
## # A tibble: 134 x 2
##    pc      pve
##    <fct> <dbl>
##  1 PC1   0.264
##  2 PC2   0.373
##  3 PC3   0.415
##  4 PC4   0.444
##  5 PC5   0.472
##  6 PC6   0.499
##  7 PC7   0.524
##  8 PC8   0.544
##  9 PC9   0.563
## 10 PC10  0.580
## # ... with 124 more rows
```

Let's viusualise this:

```r
PC.pve %>%
  ggplot( aes( x = pc,
               y = pve,
               fill = pve >= 0.9 ) ) + # Let's find the PC that explains
                                        # 90% of our variance
  geom_col() +
  theme( axis.text.x = element_text( angle = 90 ) ) #rotate the x-axis labels
```



This is a pretty plot. We can see that we need less than half the number of predictors to explain 90% of the variation in our data. This type of plot is a bit more interpretable when there are fewer predictors and we can read that x-axis a bit better, so let's grab the best number of principal components with:

```
PC.pve %>%
  filter( pve >= 0.9)  # Let's look at those explaining 90% or more
```

```
## # A tibble: 79 x 2
##    pc      pve
##    <fct> <dbl>
##  1 PC56  0.901
##  2 PC57  0.904
##  3 PC58  0.907
##  4 PC59  0.910
##  5 PC60  0.913
##  6 PC61  0.916
##  7 PC62  0.919
##  8 PC63  0.921
##  9 PC64  0.924
## 10 PC65  0.926
## # ... with 69 more rows
```

What this tells us is that we need 56 components to explain at least 90% of the variance in our predictors. This is significantly less than the original 130! In other words, we can reduce our dimension from 130 to 56.

Finally, let's make sure we get at least 56 components in our juiced data. To do this, we need to go back to our recipe and define the `num_comp` command in `step_pca`:

```
ad_56_comps <- recipe( Class ~ ., data = ad_data ) %>%
  step_dummy( Genotype ) %>%
  step_normalize( all_predictors() ) %>%
  step_pca( all_predictors(), num_comp = 56 ) %>%
  prep()
juice( ad_56_comps ) %>%
  head()
```

```
## # A tibble: 6 x 57
##   Class       PC01  PC02  PC03   PC04   PC05   PC06   PC07   PC08  PC09    PC10
##   <fct>      <dbl> <dbl> <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl> <dbl>   <dbl>
## 1 Control  -11.5   -1.72 -3.03 -1.02   0.701 -3.21   0.229 -0.245 1.09  -0.346
## 2 Control   -3.02  -4.97 -1.50  0.965 -1.44   1.27   0.332  1.47  1.18   2.62
## 3 Control    0.750 -1.24  1.62 -1.92   1.20   4.05  -1.54   2.10  1.50  -0.0869
## 4 Control   -0.531 -6.48  0.382 -0.620  2.92  -1.29   0.889 -1.31  1.29   1.23
## 5 Control  -11.6    4.56 -3.52  2.77  -1.04  -5.03  -2.68   0.668 1.67   1.97
## 6 Impaired   4.40   8.55  3.40  3.16   0.908  0.679 -0.462  3.48  0.753  0.882
## # ... with 46 more variables: PC11 <dbl>, PC12 <dbl>, PC13 <dbl>, PC14 <dbl>,
## #   PC15 <dbl>, PC16 <dbl>, PC17 <dbl>, PC18 <dbl>, PC19 <dbl>, PC20 <dbl>,
## #   PC21 <dbl>, PC22 <dbl>, PC23 <dbl>, PC24 <dbl>, PC25 <dbl>, PC26 <dbl>,
## #   PC27 <dbl>, PC28 <dbl>, PC29 <dbl>, PC30 <dbl>, PC31 <dbl>, PC32 <dbl>,
## #   PC33 <dbl>, PC34 <dbl>, PC35 <dbl>, PC36 <dbl>, PC37 <dbl>, PC38 <dbl>,
## #   PC39 <dbl>, PC40 <dbl>, PC41 <dbl>, PC42 <dbl>, PC43 <dbl>, PC44 <dbl>,
## #   PC45 <dbl>, PC46 <dbl>, PC47 <dbl>, PC48 <dbl>, PC49 <dbl>, PC50 <dbl>, ...
```

This has reduced our data from the original 130 dimensions to a nice and manageable 56 dimensions. This is one of the true powers of PCA.

Recap

1. When you have lots of predictors, you can use PCA to reduce the dimension and break your data up into the most variable directions.
2. You use `recipes` to perform PCA with the command `step_pca`.
3. It is important to normalise the predictors before you shove them into PCA.
4. You can access PCA loadings by using the `tidy` function.
5. You can access the PCA scores by juicing the prepped recipe.
6. You choose your optimal dimension using the proportion of variance explained.
7. You calculate the proportion of variance explained by extracting the standard deviations with code that looks like `PREPPED_RECIPE$steps[[ STEP NUMBER OF PCA ]]$res$sdev`.
8. When you know the number of components you want, you specify `num_comp` in the `step_pca` command.

Your turn

For this exercise, you are going to look at the `USArrests` dataset. This data contains statistics—in arrests per 100000 residents—for assault, murder and rape in each of the 50 US states in 1973. It also gives the percentage of the population living in urban areas for each state.

You are going to load the data, but you are going to do something a little snazzy as well. The `USArrests` dataset has row names that will tell you which state the statistics correspond to, but when you convert it to a tibble, you lose this information. To prevent this loss of information, you are going to add an extra argument:

```
data( "USArrests" )
USArrests <- USArrests %>%
  as_tibble( rownames = "state" ) #This will convert the row names to a new variable called state
```

What you need to do:

1. Skim the data. For each of the numeric variables, give the mean and standard deviation. Do you think the data should be normalised?
2. Create a recipe to perform PCA on this data. Don't forget to normalise the data before performing PCA and prep the recipe. (Hint: So you don't have to worry about the categorical variable `state`, use the formula `state ~ .`.)
3. Use `tidy` to inspect the loadings of the PCA components. Obtain a column plot for each component with `value` on the x-axis and `terms` on the y-axis. (Hint: What number step was PCA in your recipe? You are going to need this.)
4. What is the most influential variable in each component, i.e. which variable has the largest (in absolute value) loading value in each principal component?
5. Juice the recipe to get the PCA scores for each state.
6. Obtain a scatterplot of the principal components PC1 and PC2 labelled by state. (Hint: Use `geom_label` instead of `geom_point`.)
7. Consider the points for Florida and Mississippi.
    a. Do you think Florida has an above- or below-average amount of arrest for assault per 100000?
    b. Do you think Mississippi has an above- or below-average percentage of population living in urban areas?
8. Obtain a scatterplot of the principal components PC3 and PC4 labelled by state.
9. Consider the points for Georgia and Alaska.
    a. Do you think Georgia has an above- or below-average amount of arrest for assualt per 100000?
    b. Do you think Alaska has an above- or below-average amount of arrest for rape per 100000?
10. Calculate the PVE for each principal component. What is the proportion of variation explained by the first two principal components?