Lab 8 – Distances and PCA

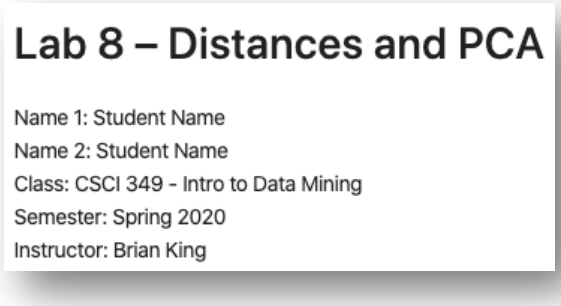
Submission: `lab08.ipynb`, pushed to your Git repo on `master`.
Points: 10
Due: Friday, February 14, 2pm

Objectives

- Practice more pandas
- Use chi-squared test to evaluate two categorical variables for independence
- Learn about implementing distance matrices
- Learn how to employ PCA
- More visualizations

Partnerships

Remember – you are now being encouraged to work with a partner. It's not required yet, but strongly encouraged to do so. And, if you do, BOTH partners MUST submit the same notebook file, AND both partners must be listed in the top header cell. For example:



Introduction

As you have learned, before you can model your data, you spend a lot of time preprocessing it. Sometimes, this means that you need to work out a set of variables that are numeric, unbiased and evenly weighted with respect to the rest of the dataset. Most of this lab will have you preprocess a very small dataset to prepare for computing a distance matrix. (It's very similar to what you saw in class.) This will give you an opportunity to understand how to perform proximity measurements on data with different types of variables on your own. The second part of the lab will have you perform a PCA on an interesting dataset of car crash data by state.

Preparing for your lab

Create a **`lab08.ipynb`** file. Create your header cell, then your first cell to set up your imports:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
```

```
from scipy.stats import zscore
from sklearn.decomposition import PCA
```

Work through this lab, and enter the answers to questions that are scattered throughout this lab. It is quite likely you'll need to add additional imports as you work through the lab.

Exercises

1) [P] Set up a pandas data frame with the following 8 observations and 3 variables:

	test1	test2	test3
A0	A	excellent	25
A1	C	fair	32
A2	C	good	60
A3	B	fair	53
A4	A	poor	23
A5	B	excellent	37
A6	C	good	45
A7	B	good	49

Be sure to set the types of each variable as follows:

- `test1`: Nominal, levels = { "A", "B", C" }
- `test2`: Ordinal, levels = { "poor", "fair", "good", "excellent" }
- `test3`: Numeric

Use `pd.Categorical` to properly set up categorical variables (i.e. test1 and test2) and pay attention to the `ordered` parameter. Name the data frame `df`. Set the `index` correctly to match the row names in the table above.

Display the df data frame, and then show the output of `info()` method to show the type of each variable in `df`.

2) [P] Show the output of `df.describe(include='all')`. What does the `include='all'` parameter do?

3) [P] Show the output of `df.test1.cat.categories`, and `df.test2.cat.categories`. What is this showing? Does it work for `df.test3.cat.categories`? (If not, then comment this line out.)

4) [P] Show the output of `df.test1.cat.codes`, and `df.test2.cat.codes`. What is this showing?

5) [P] Report the counts of each level of the categorical variables.

6) [P] Report a cross tabulation (i.e. contingency table) between `test1` and `test2`. Include the margins (i.e. the sum of the rows and the columns) in your reported table (HINT: Look up pandas `crosstab()` function)

7) [P] From the previous table, store the contingency table without the margins in a variable called `observed`

8) [P] Run a chi-squared test to determine whether test1 and test2 are dependent. Use the contingency table from the previous step. Clearly report the chi2 statistic, the p value, and the degrees of freedom, and then use the p-value to clearly state whether test1 and test2 are independent (assume p=0.05 threshold to test for

independence)

Now, you're going to explore some similarities between observations. Yes, this is a SMALL dataset! (Don't bother to explore large data until you can understand how they work on small, toy experimental data!)

Before you can compute a pairwise distance matrix to report the distance between all pairs of observations, always remember that **you can only compute distances between strictly numeric data**. Therefore, you must convert your data to numeric types for ALL variables you want to include in your distance metric.

9) [P] Create a new data frame called `df_num`, that represents a numeric version of the above. Do NOT do any rescaling of your variables yet!

   NOTE: If you do this from a dataframe that has the categorical variables set up properly, then this step is simple to do. The two choices I generally follow are either: 1) use the `cat` member of your categorical data, which stores a `CategoricalAccessor` object (look it up), or use one of the encoders in the `sklearn.preprocessing` module. The first option is easier, and yet another reason why it's so important to take the time to preprocess your data as correctly and error-free as possible.

   Your resulting data frame at the end of this step should be as follows:

	test1	test2	test3
A0	0	3	25
A1	2	1	32
A2	2	2	60
A3	1	1	53
A4	0	0	23
A5	1	3	37
A6	2	2	45
A7	1	2	49

10) [P] As you learned in lecture, you absolutely must rescale your data to fall on a similar scale. There are different approaches to doing so. A standardized zscore is among the most common, but not necessarily always the best approach, especially when you are dealing with numeric representations of true categorical data. Rescaling your data to all fall between 0 and 1 is also a common approach, particularly when you have categorical data.

    Let's first try to rescale our data to all fall between the values of 0 and 1. Use the `MinMaxScaler` in `sklearn.preprocessing` to rescale all variables to fall between 0 and 1. Store the transformed data as a pandas data frame called `df_num_zeroone`,. Your result should look as follows:

	test1	test2	test3
A0	0.0	1.000000	0.054054
A1	1.0	0.333333	0.243243
A2	1.0	0.666667	1.000000
A3	0.5	0.333333	0.810811
A4	0.0	0.000000	0.000000
A5	0.5	1.000000	0.378378
A6	1.0	0.666667	0.594595
A7	0.5	0.666667	0.702703

11) [P] Notice the value of `test3`. Quite often, when we have solid knowledge of what we expect our range to be, then we can rescale our data using that knowledge. In the case of `test3`, you learned that the data must fall between 0 and 100. Therefore, reassign `test3` so that the min and max before rescaling are assuming to be between 0 and 100, respectively. (i.e. simply divide the original variable by 100)

12) [P] Compute a single distance matrix called `distmat_zeroone`. Use a standard Euclidean distance measure. Your reported result should be an 8x8 matrix with appropriately labeled rows and columns. (HINT – study the output of the distance matrix functions! They do not output a square matrix. Look at the `squareform` function. The `pdist` and `squareform` functions are in `scipy.spatial.distance`).

13) [P] Output the top three closest pairs of observations. You MUST write Python code to report these results! Do not simply print out your distance matrix and tell me your answers! Consider that this may have been thousands of observations! Always generate reported answers in code!

    For each pair, output the pair of observations from the original dataframe, and the distance between them.

    For example, your first closest pair output might have output that looks as follows:

```
Closest # 0 :  ['A2', 'A6'] dist =  0.15
   test1 test2  test3
A2     C  good     60
A6     C  good     45
```

14) [P] Now, output the three most distant (least similar) pairs of observations. Again, for each pair, output the two observations, and the distance between them

15) [P] Create a new data frame, `df_num_binarized`, that stores the a *binarized* version for test1 and test2.

    For example, `test1` has three distinct values, "`A`", "`B`", and "`C`". Therefore, you should end up with three new variables that replace the one categorical variable. Best practice is to name your variables accordingly, with the variable name prefix, and the value as a suffix, usually with an _ in between. Thus, your result for the first variable should look like:

	test1_A	test1_B	test1_C
A0	1	0	0
A1	0	0	1
A2	0	0	1
A3	0	1	0
A4	1	0	0
A5	0	1	0
A6	0	0	1
A7	0	1	0

Do this for BOTH categorical variables `test1`, and `test2`. However, `test3` is already numeric, and we scaled it between zero-one already. Thus just copy the result of `test3` from df_num_zeroone.

HINT - HOW? One approach is to use `OneHotEncoder` from `sklearn.preprocessing`. For this purpose, you can set the parameter `sparse=False` so that you can easily view the data, and set your `dtype=int`.

Your result should look something like the following:

	test1_A	test1_B	test1_C	test2_poor	test2_fair	test2_good	test2_excellent	test3
A0	1	0	0	1	0	0	0	0.25
A1	0	0	1	0	1	0	0	0.32
A2	0	0	1	0	0	1	0	0.60
A3	0	1	0	0	1	0	0	0.53
A4	1	0	0	0	0	0	1	0.23
A5	0	1	0	1	0	0	0	0.37
A6	0	0	1	0	0	1	0	0.45
A7	0	1	0	0	0	1	0	0.49

Another approach is to use the pandas function `get_dummies()`.

16) [P] Now, compute distmat_binarized by computing the distance matrix for the df_binarized.

17) [P] Report the three closest pairs, and the three most distant pairs from distmat_binarized

18) [M] Take a moment and compare and contrast your results. Which method do you think have the better results? Why? Which variable do you think was the distinguishing player in affecting the different outcomes between both of the above approaches to transforming your data to numeric results? Why? Summarize what would have been the best transformation to make for all three variables that would have given the most accurate results.

OK – Take a breather. That was cool. Consider how valuable that is going to be when the time comes to assess similarity between observations for, say, clustering hundreds of thousands of observations. These considerations are precisely the same types of considerations you need to make whether you are searching for similar observations on

---

For this next exercise, you are going to use the dataset built into the seaborn library, called `'car_crashes'`. This time, the aim will be determine states that are outliers, but also to apply a PCA to a bit of visualization to help you confirm states that are outliers.

19) [P] Load in your next dataset using the following:

```
df_car_crashes = sns.load_dataset('car_crashes')
```

The dataset is directly downloaded from:
https://github.com/mwaskom/seaborn-data/blob/master/car_crashes.csv

Figure out what this dataset is all about. Sometimes, it's easy to figure out. Just Google! And, sometimes we find interesting publicly available data but we need to make the best guess regarding our variables. For now, don't worry too much about specifics. This is a very simplified, highly aggregated dataset from much more extensive statistics drawn from the Insurance Institute for Highway Safety - https://www.iihs.org/iihs/topics/t/general-statistics/fatalityfacts/state-by-state-overview (NOTE - this might make a great repository for some project?)

20) [P] Preprocess your data. Minimally, you should move the state code to become the index for the dataframe, and then drop that column from your dataframe. Show the first five rows.

21) [P] Create a new dataframe called `df_car_crashes_zscore` that represents the zscore transformation for `df_car_crashes`. Again, show the first five rows.

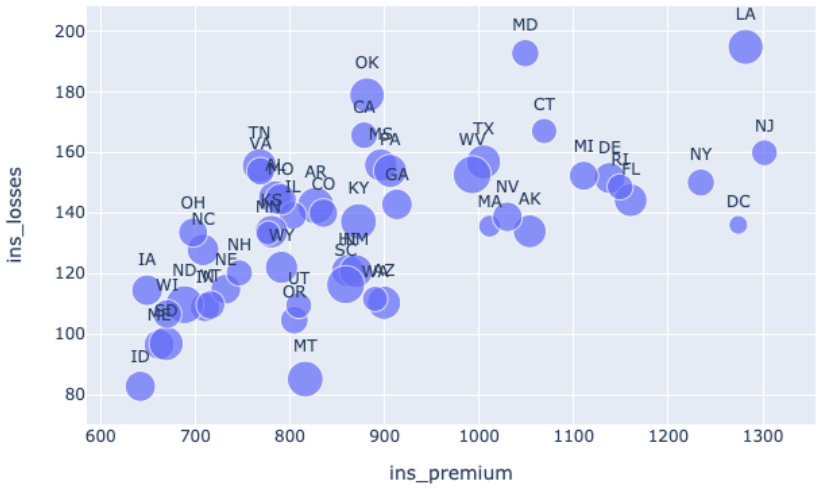
22) [P] Create a distance matrix called `distmat_cars` based on the `df_car_crashes_zscore`. Display the entire distance matrix.

23) [P] An interesting way to suggest outliers is to take a distance matrix, aggregate the mean over each row or column, then sort the output in order. Why would this work? Because an observation that is an outlier should have a relatively high mean distance to all other observations! Do this, and output the entire ordered list in descending order. (HINT: DC should be your largest outlier.)

24) [M] From this analysis, which 4 states seem to be strongest outliers?

25) [P] OK. Let's explore the data visually. First, using the original, unscaled data frame `df_car_crashes`, create a scatter plot of insurance premiums vs. insurance losses, with total number of accidents as the size of the point. Create a label near to every point representing the two letter state code.

Here's one example in Plotly:



Here's an identical plot in Seaborn:

You should be able to see that LA certainly stands out as an outlier, at least with respect to insurance premiums vs. losses.

26) [P] Next, generate **<u>two</u>** interesting plots that show some relationships between variables in the data. Try to use as many variables as you can without creating chaos! Don't just throw in multiple variables for the sake of showing them, only include them if it makes sense to do so. **Your aim is to derive meaning from your data. Good visualizations tell a story.** Strive to use at least one additional variable as size, color, or shape in your data, so you can show more than just 2 variables on a single plot. Add titles, legends and label your axes as appropriate.

After each plot, create a markdown cell and briefly draw conclusions from your plot.

Now, we're going to look at the data visually, but use a PCA transformation to help give a better sense of trends and differences among our data.

27) [P] Run a full PCA on the z_score transformed data. Set `n_components` to be the same number of columns as the data. Be sure to fit the data to your PCA model, and then output the components, explained variance,

and the explained variance ratio.

28) [M] Use your intuition – what do the weights of the first couple of components suggest explains most of the variance in the data?

29) [P] Create a plot of the cumulative sum of the explained variance. How many components will get you to 90% of the explained variance?

30) [P] Transform the z_score transformed data using your PCA model (i.e. using the `transform` function of the pca object.)

    (NOTE: I often just store the transformed data temporarily as some arbitrary variable, X, to make it easier to manipulate the data for plotting.)

31) [P] Generate a 2D plot using the first two principal components as your x and y coordinates. Be sure to label each point, and label your axes as component 1 and component 2, respectively.

32) [M] Compare the states you reported as potential outliers above to those that appear to be outliers from your plot. Do the same results seem to hold?

33) [P] Read how to generate a 3D scatterplot in seaborn or plotly, and use it to generate a scatterplot of the first 3 components.

34) [M] Do the same outliers still stand out?

That's it!

Deliverables

Commit and push `lab08.ipynb`. Be sure you have every cell run, and output generated. All plots should have plt.show() as the last command to ensure the plot is generated and viewable on Gitlab. Verify that your file is pushed properly on Gitlab. Be sure each question is properly annotated.