LAB 11 - DATA SCIENCE AND PRINCIPLE COMPONENT ANALYSIS

1. Big data

We produce lots of data. Lots and lots of data. From the digital photos we take, to the emails we write and text messages we send, to the temperature data being measured by both scientific weather monitoring stations and the smart thermostat in your home, it's been estimated that mankind produces 2.5 million terabytes of data *each day*.[1] To put this into perspective, it's estimated that of all the data that has been produced by mankind since the dawn of civilization up until now (including all of the cave drawings, scrolls, books, letters, poems, emails, songs, pictures, and tweets produced), over 90% of it has been created in the past 2 years. This data contains valuable information which can be used to do many things, including informing government policy, designing safer self–driving cars, making better healthcare decisions, and helping companies target their marketing efforts. However, because of the sheer enormity of the data we create, it requires specialized tools to uncover the patterns and information which may be buried deep below the surface.

2. Classification problems

One of the most common types of problems that needs to be solved when dealing with data is how to classify data points within a set into specific types. As a concrete example, suppose that a researcher studying cancer is trying to find a way to determine in advance whether cells from a tissue sample will develop into cancer or not. The researcher gathers 300 samples of cells from people with a predisposition to this type of cancer, and measures the level of two different types of proteins in each sample, Protein A and Protein B. She then grows the cells in a culture, and waits to find out which develop into cancer and which do not, before plotting the data in Figure 1. Each sample that eventually became cancerous is plotted as a red point, while each sample that was not cancerous is plotted as a blue point.

From this data the researcher is able to observe that cells that eventually become cancerous tend to have higher levels of Protein A and lower levels of Protein B than cells which do not develop into cancer. This data then allows the researcher to predict whether a new tissue sample arriving at her lab will eventually become cancerous or not, without needing to wait to grow it in culture.

For example, suppose that three new tissue samples come into the lab. The researcher measures the levels of Protein A and Protein B and plots the results along with the data she previously gathered. See Figure 2, where the three new samples are labeled with a square, circle, and triangle respectively.

- Of the three tissue samples (square, circle, and triangle), can you predict which ones will turn cancerous, and which ones won't?
- Are you as confident about your prediction for the circle sample as you are for the square and triangle samples?

[1] *IBM Marketing Cloud Research Survey*, 10 Key Marketing Trends for 2017 and Ideas for Exceeding Customer Expectations.

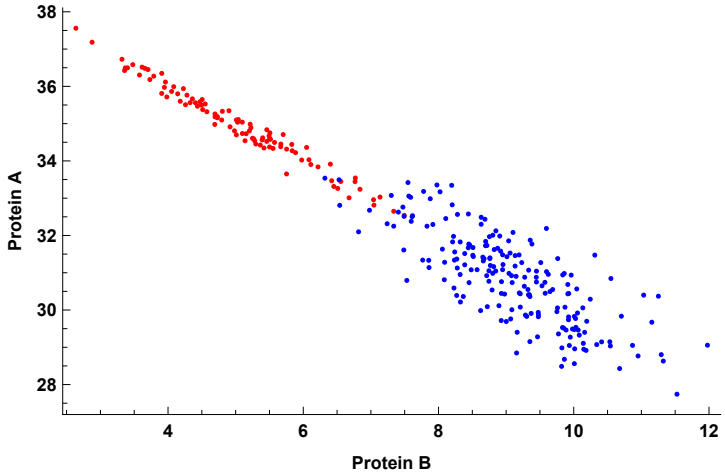

FIGURE 1. Measurements of protein levels in 300 tissue samples. The red points correspond to cells which eventually became cancerous, while the blue points correspond to cells which were not cancerous.

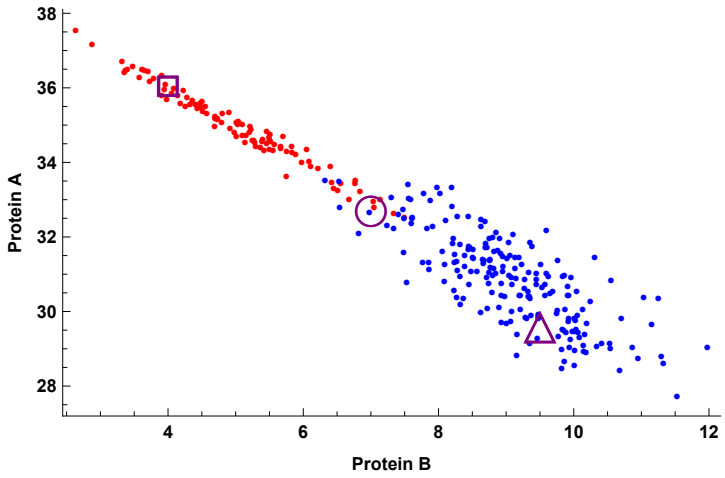

FIGURE 2. The protein levels of three new samples, labeled with a square, circle, and triangle.

Looking at the plot the researcher can be fairly confident that the sample corresponding to the triangle *will not* turn cancerous, since it is located in a region of the graph that is surrounded by blue points, all of which represent samples with similar protein counts that *didn't* turn cancerous. Similarly, the researcher predicts that the sample corresponding to the square *will* turn cancerous, since it is surrounded by red points, all of which represent samples with similar protein levels that *did* turn cancerous. It is not so straightforward, however, to make a prediction for the sample corresponding to the circle, since its protein levels are close to both red and blue points. For this sample the researcher would need to either collect more information, or grow it in a culture before she is able to confidently determine whether it will turn cancerous or not.

The above scenario is an example of a *classification problem*, which are very common in data science and show up under many different guises. Instead of protein measurements and cancer

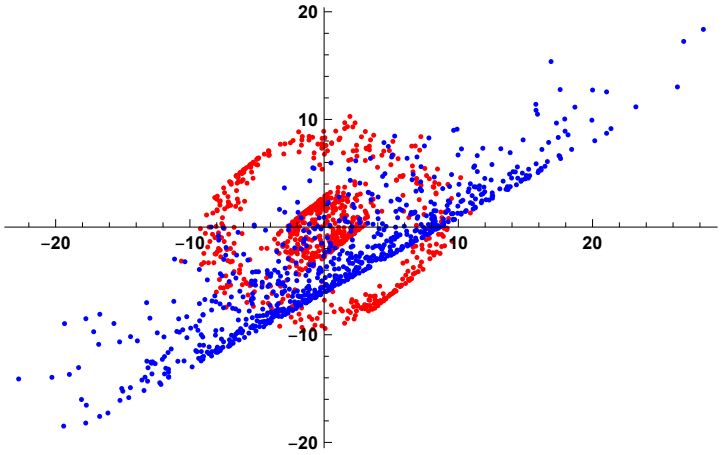

FIGURE 3. A classification problem where the different data point types are not clearly separated.

detection, it can just as easily be an online retailer tracking mouse clicks and trying to predict whether or not a webpage visitor will purchase a product. Or a basketball analyst tracking various NCAA player statistics, trying to predict the winner of a given college basketball game. Or a Twitter data scientist tracking information about various tweets, trying to determine which ones come from actual humans and which ones are generated by bots. The specifics of each situation may be very different, but the goal is the same in each: learn patterns from previous observations to make predictions about new data points.

While in the cancer detection example we introduced above the distribution of the data was very simple—and thus we could make predictions about new samples by visually inspecting the plot—in general this will not always be possible. Indeed, often times the different classes of data points aren't so clearly separated as in the above scenario (see Figure 3, for example).

Another issue in trying to answer the classification problem is that often times the data we want to study can't be plotted on a graph at all, because it's *dimension* is too large. In the above example, our cancer researcher was interested in the levels of two different protiens in the tissue samples, and hence we could plot the results in $\mathbb{R}^2$ (hence we think of our data as being 2–dimensional). What if there were actually four different proteins the researcher was interested in? How would she plot this data, and make predictions using this 4–dimensional data set? Or suppose that in addition to the proteins of interest each patient filled out a 100 question survey providing details about their lifestyle and family medical histories. How would our researcher include information about protein counts along with the answers to all 100 questions in a single plot?

Fortunately, there are many sophisticated algorithms which data scientists can use to help make accurate predictions, even when the data can't be plotted and inspected visually like above. These algorithms are part of the field of *machine learning*, and are used frequently in many fields of science, business, and industry. Despite how successful these algorithms can be at learning patterns and making predictions, however, when the dimension of the data becomes too large even they become less effective at making accurate predictions. This difficulty to make accurate predictions when the dimension of the data grows too large is called *the curse of dimensionality*, and it plagues even the most sophisticated machine learning algorithms.

One way to avoid the curse of dimensionality—and thereby make accurate predictions even when our data is high dimensional—is by finding ways to reduce the dimension of the data

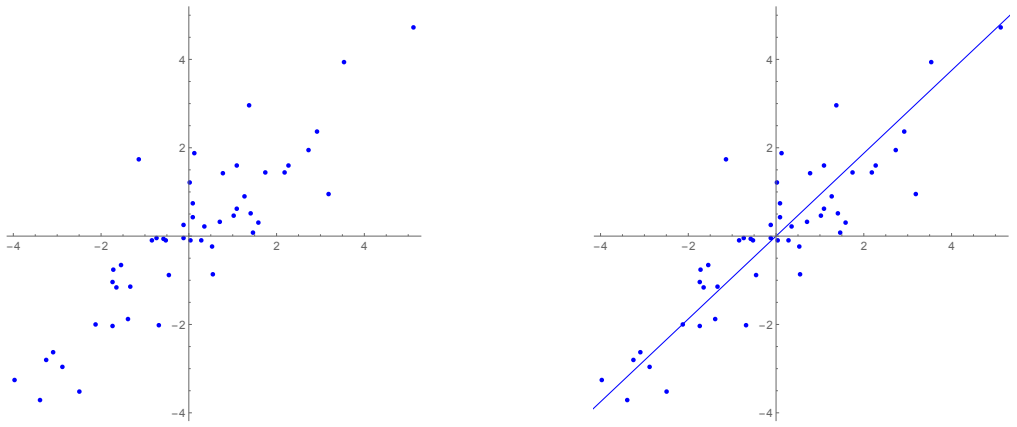

FIGURE 4. A 2-dimensional dataset that lies close to a line.

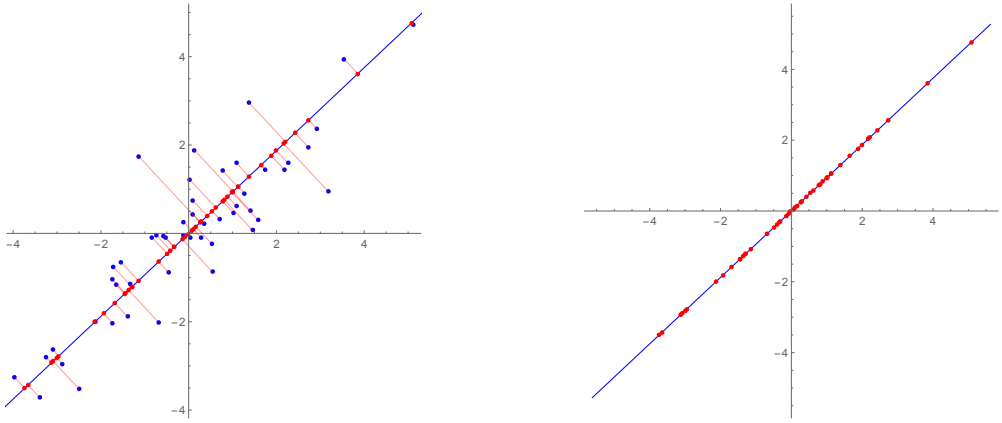

FIGURE 5. Projecting our 2-dimensional data set to a 1-dimensional line.

before applying any machine learning algorithms. This approach is called *dimensionality reduction*, and in this lab we will focus on a specific example of dimensionality reduction called *principle component analysis*.

3. Reducing the dimension of data

In this section we will introduce a straightforward—yet extremely useful—way to reduce the dimension of a dataset, by using orthogonal projections to project the data points to a subspace of lower dimension. To see how this is done, consider the data points plotted on the left-hand side of Figure 4. Although the data is 2-dimensional (meaning each point consists of an x- and a y-coordinate), it appears as though the data points are all clustered around a straight line L. For the time being we will ignore the question of how we found the line L, though from your work in Lab 6 you may have an idea of how to do this using the method of least-squares. In this lab we will use a different approach which yields the same results, but which generalizes better to higher dimensions.

To reduce the dimension of the data, we simply use orthogonal projection to project each data point to the line L, and take the projected data points as our new dataset (see Figure 5). Recall that if $\mathbf{u}$ is a vector which spans the line L, then the orthogonal projection $\text{proj}_{\mathbf{u}}\,\mathbf{x}$ of

a vector $\mathbf{x}$ onto the line L is given by the formula

$$\text{proj}_L \mathbf{x} = \left(\frac{\mathbf{u}\cdot\mathbf{x}}{\mathbf{u}\cdot\mathbf{u}}\right)\mathbf{u}. \tag{1}$$

We can simplify this formula a bit, if we assume that $\mathbf{u}$ is a unit vector, i.e., $\|\mathbf{u}\| = \sqrt{\mathbf{u}\cdot\mathbf{u}} = 1$. In this case $\mathbf{u}\cdot\mathbf{u} = 1$, and equation (1) becomes

$$\text{proj}_L \mathbf{x} = (\mathbf{u}\cdot\mathbf{x})\,\mathbf{u}. \tag{2}$$

To see what this looks like in practice, suppose that the line L is spanned by the unit vector

$$\mathbf{u} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \approx \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix}$$

and let $\mathbf{x}$ be the vector

$$\mathbf{x} = \begin{bmatrix} 0.81 \\ 0.67 \end{bmatrix}.$$

Using equation (2) then gives

$$\text{proj}_L \mathbf{x} = (\mathbf{u}\cdot\mathbf{x})\,\mathbf{u} = \left(\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \cdot \begin{bmatrix} 0.81 \\ 0.67 \end{bmatrix}\right)\begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = 1.04652 \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} 0.739993 \\ 0.739993 \end{bmatrix}.$$

But wait! We were promised that by projecting our data point $\mathbf{x}$ to the line L we would obtain a 1-dimensional vector, but all we've found above is another 2-dimensional vector. Why is this any better than what we had before?

To see why this is an improvement, notice that this allows us to think of our (projected) data point in a slightly different way. Instead of keeping track of the entire vector $\text{proj}_L \mathbf{x} = (\mathbf{u}\cdot\mathbf{x})\,\mathbf{u}$, we just record the coefficient $\mathbf{u}\cdot\mathbf{x} = 1.04652$, which we can think of as being a 1-dimensional vector! We call this scalar the *coordinate* of the vector $\text{proj}_L \mathbf{x}$ with respect to the basis vector $\mathbf{u}$.

If we do this for an entire dataset worth of points $\mathbf{x}_1, \cdots, \mathbf{x}_p$, then we get a list of scalar values $(\mathbf{u}\cdot\mathbf{x}_1), \cdots, (\mathbf{u}\cdot\mathbf{x}_p)$ which represents our dataset. For example, if we were given the initial dataset

$$\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\} = \left\{\begin{bmatrix} 0.81 \\ 0.67 \end{bmatrix}, \begin{bmatrix} 0.21 \\ 0.31 \end{bmatrix}, \begin{bmatrix} 1.11 \\ 1.19 \end{bmatrix}, \begin{bmatrix} -1.21 \\ -0.94 \end{bmatrix}\right\},$$

we would replace it by the list of values

$$\{(\mathbf{u}\cdot\mathbf{x}_1), (\mathbf{u}\cdot\mathbf{x}_2), (\mathbf{u}\cdot\mathbf{x}_3), (\mathbf{u}\cdot\mathbf{x}_4)\} = \{1.04652, 0.367696, 1.62635, -1.52028\}.$$

In this way, we can think of our data set as being (approximately) 1-dimensional, since each point can be represented by a 1-dimensional vector (i.e., a scalar). If we ever need to recover the original data points, all we have to do is multiply each of the scalar coordinates by the basis vector $\mathbf{u}$ to obtain an approximation to our dataset.

Problem 1. Define a function `projection_coordinate(u,x)` which takes as input NumPy vectors `u` and `x` of the same size, and returns the coordinate of the projection $\text{proj}_L \mathbf{x}$ with respect to the basis vector `u`. You may assume that `u` is a unit vector.

Make sure that your output is the scalar *coordinate* of the projection vector $\text{proj}_L \mathbf{x}$ with respect to the basis $\mathbf{u}$, and not the actual projection vector. In particular, the output of your function should be a scalar. You may find the function `np.dot(a,b)`, which computes the dot product of NumPy arrays `a` and `b` helpful here.

For example, if `u=np.array([1/np.sqrt(6),1/np.sqrt(6),2/np.sqrt(6)])` and `x=np.array([2,1,-3])`, then `projection_coordinate(u,x)` should return the scalar `-1.2247448713915892`.

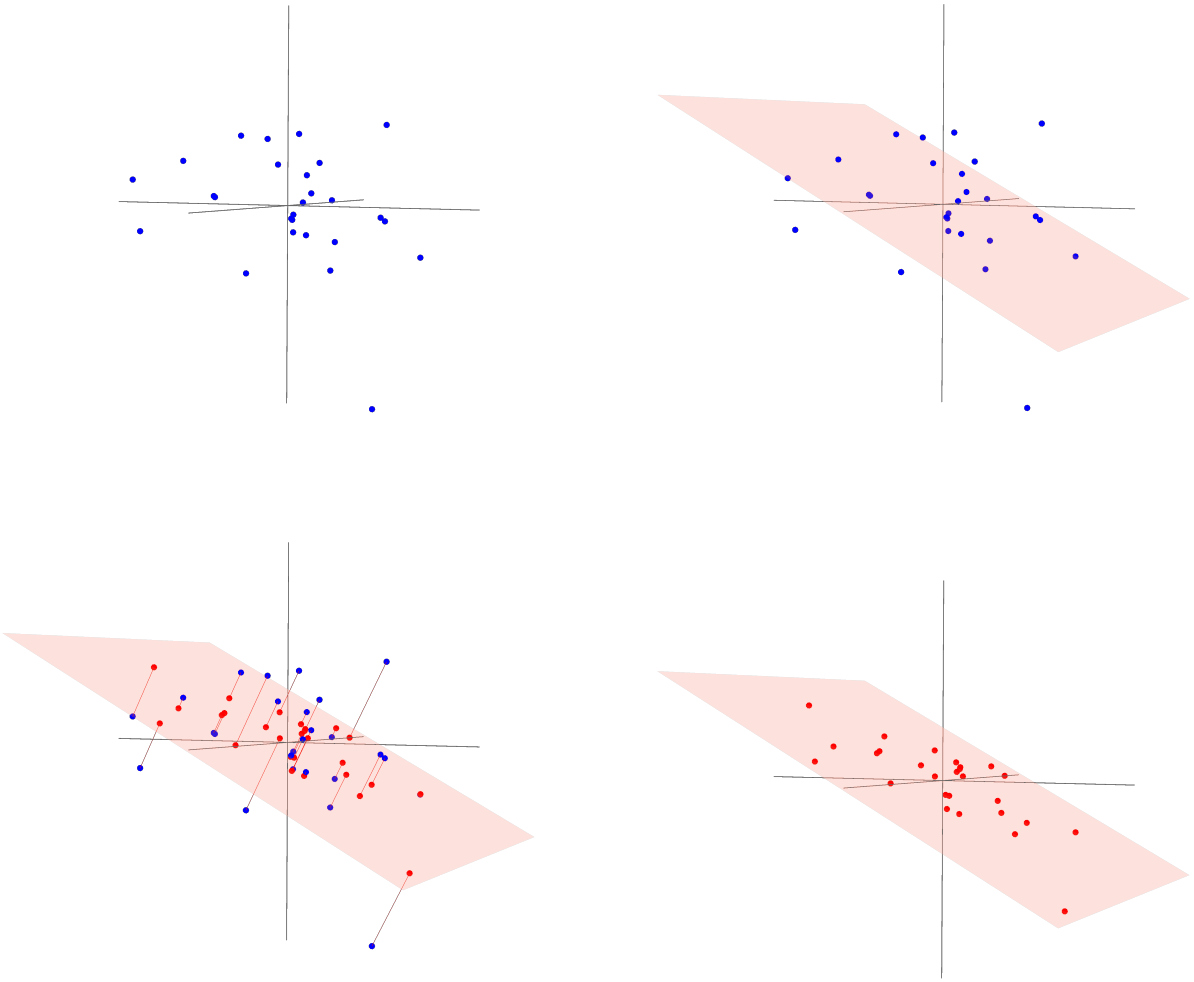

FIGURE 6. Projecting 2-dimensional data to a 3-dimensional plane.

It is worth noting that we can project to subspaces of any dimension, not just to 1-dimensional lines. Because our ultimate goal will be to project high-dimensional data down to $\mathbb{R}^2$ so that we can plot the resulting data and inspect it visually, we will describe how to project

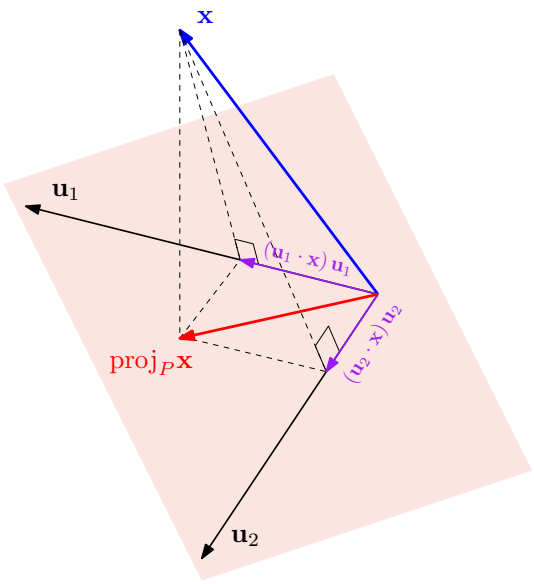

FIGURE 7. Projecting a vector $\mathbf{x}$ to a plane with basis $\{\mathbf{u}_1, \mathbf{u}_2\}$. The coordinates $(\mathbf{u}_1 \cdot \mathbf{x})$ and $(\mathbf{u}_2 \cdot \mathbf{x})$ determine how much each basis vector contributes to the projection vector $\text{proj}_P \mathbf{x}$.

to a 2-dimensional plane. The techniques we describe, however, can easily be generalized for use in any dimension.

The case of projecting 3-dimensional data (represented as points in $\mathbb{R}^3$) to a 2-dimensional plane is illustrated in Figure 6. Let P be a 2-dimensional subspace of $\mathbb{R}^3$, and let $\{\mathbf{u}_1, \mathbf{u}_2\}$ be an *orthonormal* basis for the plane P. In other words, the vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ are both unit vectors, which are orthogonal to each other:

$$\mathbf{u}_1 \cdot \mathbf{u}_1 = 1, \quad \mathbf{u}_2 \cdot \mathbf{u}_2 = 1, \quad \text{and} \quad \mathbf{u}_1 \cdot \mathbf{u}_2 = 0.$$

Then the orthogonal projection of a vector $\mathbf{x}$ to P is given by

$$\text{proj}_P \mathbf{x} = (\mathbf{u}_1 \cdot \mathbf{x})\, \mathbf{u}_1 + (\mathbf{u}_2 \cdot \mathbf{x})\, \mathbf{u}_2.$$

(Note that the denominators that sometimes appear in the projection formula are both equal to 1, since $\mathbf{u}_1$ and $\mathbf{u}_2$ are both unit vectors.). As above, instead of keeping track of the vector $\text{proj}_P \mathbf{x}$ in $\mathbb{R}^3$, we will keep track of it's coordinates $(\mathbf{u}_1 \cdot \mathbf{x})$ and $(\mathbf{u}_2 \cdot \mathbf{x})$ with respect to the basis $\{\mathbf{u}_1, \mathbf{u}_2\}$ of P. Notice here that we need two coordinates to represent the projected points in the plane P, since P is 2-dimensional. See Figure 7 for an illustration of such a projection.

To see how to easily compute these coordinates in practice, let U be the matrix with rows $\mathbf{u}_1^T$ and $\mathbf{u}_2^T$

$$U = \begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \end{bmatrix}.$$

Then if we take the matrix U and multiply by one of our data points $\mathbf{x}$ we obtain the vector

$$U\mathbf{x} = \begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{u}_1^T \mathbf{x} \\ \mathbf{u}_2^T \mathbf{x} \end{bmatrix} = \begin{bmatrix} (\mathbf{u}_1 \cdot \mathbf{x}) \\ (\mathbf{u}_2 \cdot \mathbf{x}) \end{bmatrix}.$$

Notice that the components of this vector are the coordinates of $\text{proj}_P \mathbf{x}$ with respect to the basis $\{\mathbf{u}_1, \mathbf{u}_2\}$.

As a concrete example, suppose that the plane P is spanned by the vectors

$$\mathbf{u}_1 = \begin{bmatrix} \frac{1}{\sqrt{6}} \\ \frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{6}} \end{bmatrix} \approx \begin{bmatrix} 0.408 \\ 0.816 \\ 0.408 \end{bmatrix} \quad \text{and} \quad \mathbf{u}_2 = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} \approx \begin{bmatrix} 0.707 \\ 0 \\ 0.707 \end{bmatrix}$$

and that our data point is

$$\mathbf{x} = \begin{bmatrix} 1.15 \\ 1.45 \\ -0.40 \end{bmatrix}.$$

Then our matrix U is given by

$$U = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}$$

and we can compute the coordinates of $\text{proj}_P \mathbf{x}$ with respect to the basis $\{\mathbf{u}_1, \mathbf{u}_2\}$ as follows:

$$U\mathbf{x} = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1.15 \\ 1.45 \\ -0.40 \end{bmatrix} = \begin{bmatrix} 1.4901 \\ -1.0960 \end{bmatrix}.$$

One nice feature of the above procedure is that it can be applied to an entire dataset worth of points all in one step. For example, suppose that we were given the dataset

$$\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\} = \left\{ \begin{bmatrix} 1.15 \\ 1.45 \\ -0.40 \end{bmatrix}, \begin{bmatrix} -0.45 \\ 0.45 \\ -0.20 \end{bmatrix}, \begin{bmatrix} 0.75 \\ -2.15 \\ 0.40 \end{bmatrix}, \begin{bmatrix} -1.45 \\ 0.25 \\ 0.20 \end{bmatrix} \right\}.$$

Then we can compute the projection coordinates for all four of these points at once, by first combining them into a single matrix X and then multiplying by U.

$$UX = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ -\frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1.15 & -0.45 & 0.75 & -1.45 \\ 1.45 & 0.45 & -2.15 & 0.25 \\ -0.40 & -0.20 & 0.40 & 0.20 \end{bmatrix} \tag{3}$$

$$= \begin{bmatrix} 1.4901 & 0.1021 & -1.2860 & -0.3062 \\ -1.0960 & 0.1768 & -0.2475 & 1.1667 \end{bmatrix}.$$

Having done this, our 3-dimensional dataset $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$ can now be represented as 2-dimensional points by the set

$$\left\{ \begin{bmatrix} 1.4901 \\ -1.0960 \end{bmatrix}, \begin{bmatrix} 0.1021 \\ 0.1768 \end{bmatrix}, \begin{bmatrix} -1.2860 \\ -0.2475 \end{bmatrix}, \begin{bmatrix} -0.3062 \\ 1.1667 \end{bmatrix} \right\}.$$

- How would the steps we took to project the data change if we were instead projecting from a 4-dimensional dataset to a 3-dimensional subspace? Or a 4-dimensional dataset to a 2-dimensional plane?

Problem 2. Define a function `projection_2D(u1,u2,X)` which takes as input two k-dimensional NumPy vectors `u1` and `u2`, and a $k \times p$ NumPy matrix `X`. Your function should return a matrix whose columns are the coordinate vectors obtained by projecting the columns of `X` to the plane spanned by `u1` and `u2`. In other words, your function should take the vectors `u1` and `u2`, build the matrix U as in equation (3) above, and then return the product U`X`. You may assume that the vectors `u1` and `u2` are unit vectors, and that they are orthogonal.

Your function should be able to handle vectors of any size (provided the number of entries in `u1` and `u2` match the number of rows of `X`).

For example, if the basis vectors are given by

```
u1=np.array([1/3,2/3,2/3])
u2=np.array([0,-1/np.sqrt(2),1/np.sqrt(2)])
```

and the data vector is given by

```
X=np.array([[1,2,3],[4,5,6],[7,8,9]]),
```

then the function call `projection_2D(u1,u2,X)` should return the array

```
array([[ 7.66666667, 9.33333333, 11.    ],
      [ 2.12132034, 2.12132034, 2.12132034]]).
```

4. Finding projections with maximal variance

Now that we know how to project our data points to a subspace of lower dimension we return to the question of how to choose the subspace which we project onto. Are there some choices of subspace that are better than others?

To answer this question consider the projection show in in Figure 8. Here we have projected the same dataset as in Figures 4 and 5, but using a different line.

- Do you think the projection in Figure 8 will lead to better or worse results than the projection in Figure 5? Why do you think this is the case?

As you may have guessed, our new choice of projection is a worse choice than the choice we originally made, for a number of reasons. First, when we project our data points to the line we are making an approximation, replacing each data point in our set with the closest point to it on the line. The further the line is from the points, the worse our approximation will be. By visually inspecting the two lines in Figure 5 and Figure 8 we can see that the first choice of line is much closer to the datapoints than the second choice of line (in the terminology of Lab 6 the line in Figure 5 is the *line of best fit*, and it minimizes the *mean squared error* introduced in Lab 7).

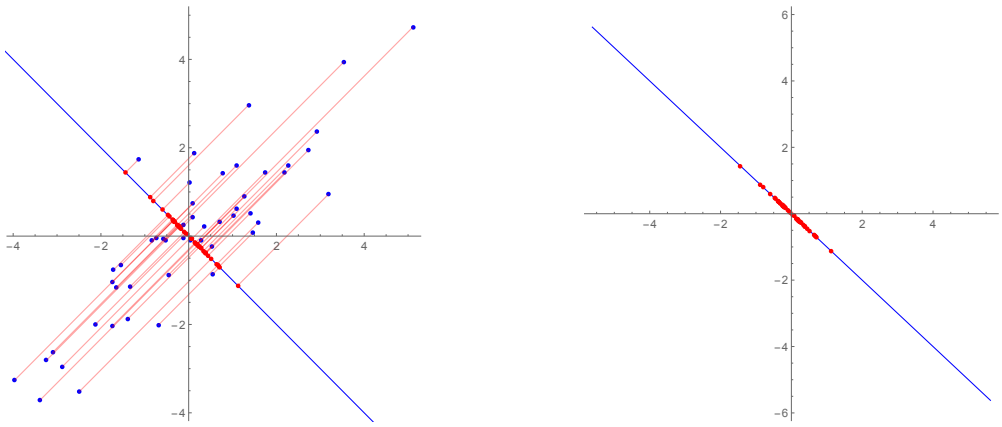

FIGURE 8. An alternate projection of the data in Figures 4 and 5.

Another reason that our initial choice of projection is preferable is related to the distribution of the points on the line after projection. Comparing Figures 5 and 8, we see that the projected points are much more tightly clustered on the line in Figure 8, while the points on the line in Figure 5 are much more spread out.

To see why a projection that keeps the data spread out is preferable, consider the comparison plots in Figure 9. Here we have highlighted a pair of points in each plot, denoted by large blue circles. Because these two points are far apart in the original dataset, we would expect them to represent very different objects. For example, in our cancer detection scenario, these two points would represent tissue samples with very different protein counts. We would like to find a projection which keeps these two points as far apart as possible, so that whatever algorithm we apply after projection can distinguish them. In Figure 9 we can see that our first choice of projection sends these points to points which are far apart on the line, while the second choice of projection sends them to a pair of points on the line which are very close, and therefore much more difficult to distinguish.

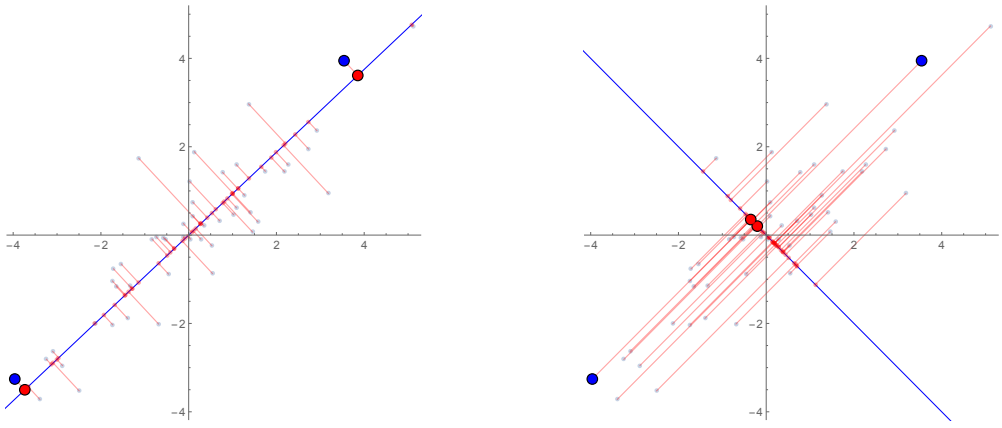

FIGURE 9. Comparing the projections of a pair of far-away points, projected onto the lines in Figures 5 and 8.

When deciding which line, or plane, or other subspace to project along we therefore look to find the subspace which keeps the data as spread out as possible. In other words, we look for a subspace which maximizes the *variance* of the projected data. (You can think of the variance

as measuring how spread out the dataset is.) Although the line of best fit described in Lab 6 will give us such a line, we would like to know how to find planes and higher dimensional subspaces which also maximize the variance of the data.

Surprisingly, it turns out that the subspace which maximizes variance can be described in terms of the eigenvectors of a certain matrix, called the *covariance matrix* of the data. To find the covariance matrix of our data, suppose that we have a data set with n points in it. We combine our data points as the columns of a single matrix X, similar to how we presented our dataset as a matrix in equation (3). We must also assume that data is centered around the origin, meaning that the average of each coordinate in our dataset is zero (this is easy to achieve in practice by shifting the data points appropriately, and we won't say any more about it). Then the covariance matrix W of our dataset is given by

$$W = \frac{1}{n-1} XX^T. \tag{4}$$

For example, suppose we are again working with the data set

$$\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\} = \left\{ \begin{bmatrix} 1.15 \\ 1.45 \\ -0.40 \end{bmatrix}, \begin{bmatrix} -0.45 \\ 0.45 \\ -0.20 \end{bmatrix}, \begin{bmatrix} 0.75 \\ -2.15 \\ 0.40 \end{bmatrix}, \begin{bmatrix} -1.45 \\ 0.25 \\ 0.20 \end{bmatrix} \right\}.$$

Notice that the average of all of the first coordinates is zero, the average of the second coordinates is zero, and similarly with the third coordinates. Then the covariance matrix of this data set is given by

$$\begin{aligned} W =& \frac{1}{n-1} XX^T \\ =& \frac{1}{4-1} \begin{bmatrix} 1.15 & -0.45 & 0.75 & -1.45 \\ 1.45 & 0.45 & -2.15 & 0.25 \\ -0.4 & -0.2 & 0.4 & 0.2 \end{bmatrix} \begin{bmatrix} 1.15 & 1.45 & -0.4 \\ -0.45 & 0.45 & -0.2 \\ 0.75 & -2.15 & 0.4 \\ -1.45 & 0.25 & 0.2 \end{bmatrix} \\ =& \begin{bmatrix} 1.397 & -0.170 & -0.120 \\ -0.170 & 2.33 & -0.493 \\ -0.120 & -0.493 & 0.133 \end{bmatrix}. \end{aligned}$$

- Notice that the covariance matrix $W = \frac{1}{n-1} XX^T$ above is symmetric. Can you see why this will always be the case?
- Since W will always be symmetric, what does this tell you about its eigenvectors?

Let $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$ be the eigenvectors of the covariance matrix, with corresponding eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$. Assume that the eigenvectors are ordered so that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. We can use these eigenvectors, along with the following theorem, to determine which subspace to project onto.

Theorem 1. *The subspace spanned by the eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k$ is a k-dimensional subspace on which the data can be projected with maximum total variance. After projecting the data it will have total variance $\lambda_1 + \lambda_2 + \cdots + \lambda_k$.*

In other words, if we want to project our data onto a line in such a way that the projection is as spread out as possible (and the error is as small as possible), then we should project onto

the line spanned by $\mathbf{u}_1$. If we want to project onto a plane so that the data remains as spread out as possible, then we should project onto the plane spanned by $\mathbf{u}_1$ and $\mathbf{u}_2$, and so on.

Returning to our above example, notice that the eigenvalues of W are

$$\lambda_1 = 2.4546, \qquad \lambda_2 = 1.3947, \qquad \text{and} \qquad \lambda_3 = 0.0106.$$

This tells us that our original dataset has total variance is $\lambda_1+\lambda_2+\lambda_3 = 3.86$. Notice, however, that the third eigenvalue is much smaller than the other two (and hence the data is much more spread out in the directions of the eigenvectors $\mathbf{u}_1$ and $\mathbf{u}_2$ than it is in the direction of $\mathbf{u}_3$). If we project onto the plane spanned by the eigenvectors $\mathbf{u}_1$ and $\mathbf{u}_2$ then we will obtain a dataset with total variance $\lambda_1 + \lambda_2 = 3.8494$. Since this is very close to our initial variance of 3.86 it suggests that we can project to the plane spanned by $\mathbf{u}_1$ and $\mathbf{u}_2$ without losing much of the information in our original dataset.

Computing $\mathbf{u}_1$ and $\mathbf{u}_2$ gives orthogonal unit vectors

$$\mathbf{u}_1 = \begin{bmatrix} 0.1334 \\ -0.9708 \\ 0.1994 \end{bmatrix} \qquad \text{and} \qquad \mathbf{u}_2 = \begin{bmatrix} 0.9849 \\ 0.1074 \\ -0.1357 \end{bmatrix}.$$

Using the eigenvectors $\mathbf{u}_1$ and $\mathbf{u}_2$ we can construct our projection matrix U as above, and use it to project our data points down to the plane as before.

$$\begin{aligned} UX &= \begin{bmatrix} 0.1334 & -0.9708 & 0.1994 \\ 0.9849 & 0.1074 & -0.1357 \end{bmatrix} \begin{bmatrix} 1.15 & -0.45 & 0.75 & -1.45 \\ 1.45 & 0.45 & -2.15 & 0.25 \\ -0.4 & -0.2 & 0.4 & 0.2 \end{bmatrix} \\ &= \begin{bmatrix} -1.3341 & -0.5368 & 2.2670 & -0.3962 \\ 1.3427 & -0.3677 & 0.4534 & -1.4284 \end{bmatrix}. \end{aligned}$$

Thus the 2-dimensional representation of our dataset which maximizes variance and minimizes error is given by

$$\left\{ \begin{bmatrix} -1.3341 \\ 1.3427 \end{bmatrix}, \begin{bmatrix} -0.5368 \\ -0.3677 \end{bmatrix}, \begin{bmatrix} 2.2670 \\ 0.4534 \end{bmatrix}, \begin{bmatrix} -0.3962 \\ -1.4284 \end{bmatrix} \right\}.$$

We can now plot this data to visualize it more easily, or apply machine learning algorithms to answer questions about this information more effectively.

5. Cancer prediction revisited

We now return to our cancer researcher, who has turned her attention from early cancer detection through protein counts to the problem of identifying other factors that lead to higher incidences of cancer. In her effort to identify additional risk factors, she has 1000 people answer a 100 question survey with information about their demographics, lifestyles, and family histories. She then tracks these individuals for a period of 15 years to determine which of them develop cancer of a particular form. Their answers to these survey questions are converted to numerical values, and recorded.

- Using the code provided in the lab notebook, create three data matrices. The first is called `X_neg`, which contains the survey data of the individuals in the study who tested negative for cancer. The second is called `X_pos`, and contains the survey data of all of the individuals who tested positive for cancer. The third dataset is called `X_total` and contains the survey data of all positive and negative individuals combined.

- The data in the matrices `X_neg`, `X_pos`, and `X_total` is formatted like the matrix X above, with each row representing a different survey question, and each column representing a different individual.

Suppose now that two new individuals, Alice and Bob, fill out the same survey. Using the survey data we have gathered so far, is it possible to estimate the likelihood that Alice or Bob will develop this form of cancer in the future?

- Vectors were created containing Alice and Bob's survey answers, called `Alice` and `Bob` respectively, when you imported the data. Print these vectors in your notebook, and compare them to the survey answers of several of the patients in `X_neg` and `X_pos` (each of which is represented as a column of the matrix). Can you determine whether their answers are closer to those of patients who developed cancer or to those who didn't?
- Recall that you can view column `j` of an array `A` by slicing with the command `A[:,j]`, and you can view row `i` using the command `A[i,:]`. When comparing values recall that Python uses `e` for powers of 10, so for example `-8.63438966e+02` and `-863.438966` are the same thing.

Because our dataset is 100–dimensional it is difficult to make predictions for Alice and Bob. We will therefore try to reduce the dimension so we can analyze it more easily.

Problem 3. Compute the covariance matrix of our total dataset using equation (4). Save it's value as a NumPy array `W`.

Hint: You should use the full dataset `X_total` *here.*

Recall that the total variance of our dataset and its various projections can be found by computing the eigenvalues of `W`. To do this we can use the command

```
1 L, P = np.linalg.eig(W)
```

This will array create an array `L` with all of the eigenvalues of `W`, and a matrix `P` whose *columns* are the corresponding unit eigenvectors (with the same ordering as `L`).

Problem 4. Compute the eigenvalues and eigenvectors of `W`. Save the three largest eigenvalues as the variables `L1`, `L2`, `L3`, where `L1`$\geq$`L2`$\geq$`L3`. Save the corresponding eigenvectors as `u1`, `u2`, and `u3` respectively.

If we let $\lambda_1, \ldots, \lambda_{100}$ denote the eigenvalues of `W`, and $\mathbf{u}_1, \ldots, \mathbf{u}_{100}$ the corresponding unit eigenvectors, then recall that the variance (or spread) of the data in the direction of $\mathbf{u}_j$ is measured by λ_j. If we project our data onto the subspace spanned by $\mathbf{u}_1, \ldots, \mathbf{u}_k$ the total

variance of our projected data will thus be $\lambda_1 + \cdots + \lambda_k$. This is true even if we don't project at all (which is the same thing as projecting onto the space spanned by $\mathbf{u}_1, \ldots, \mathbf{u}_{100}$).

- Looking at the relative size of the eigenvalues of `W`, what can you conclude about our original dataset? Do we need to keep all 100–dimensions, or can we project to a lower-dimensional subspace without reducing the variance too much? How many dimensions do we actually need to keep?

Problem 5. Compute the total variance of our original dataset, and save its value as the variable `total_variance`.

Find the variance of the dataset we obtain by projecting onto the subspace spanned by the eigenvectors `u1` and `u2`, and save its value as the variable `reduced_variance`.

To illustrate how close these two values are, compute the ratio of `reduced_variance` to `total_variance`, and save the value as `relative_variance`.

Hint: You might find the function `np.sum(L)` *which adds up the values of an array* `L` *to be useful here.*

- Keeping in mind the values you found in Problem 5, does projecting our 100–dimensional data onto a 2–dimensional plane seem like a good idea? Will we obtain an accurate approximation of our original dataset, or will it be too distorted to make reliable predictions?

We would like to compare the data contained in `X_neg` and `X_pos` with the survey answers given by Alice and Bob. We will therefore project all of the data to the 2–dimensional subspace spanned by `u1` and `u2`, so they can be plotted.

Problem 6. Project the data in the arrays `X_neg` and `X_pos` to the 2–dimensional subspace spanned by `u1` and `u2`. Save the 2–dimensional datasets as `X_neg_2D` and `X_pos_2D` respectively.

Similarly, project the vectors `Alice` and `Bob` and save the resulting 2–dimensional vectors as `Alice_2D` and `Bob_2D` respectively.

- Using the function provided in the lab notebook, plot the various 2–dimensional data points in `X_neg_2D` and `X_pos_2D`, along with `Alice_2D` and `Bob_2D`. You can use the function to plot them separately or together in any combination.

Problem 7. Comparing the plots of `X_neg_2D` and `X_pos_2D` with `Alice_2D` and `Bob_2D` make a prediction about the likelihood of Alice or Bob developing this form of cancer.

If you think that Alice is likely to develop this form of cancer enter a value of `1` for the variable `Alice_prediction`. If you think it is likely she will not develop this form of cancer, save the value of `-1` for the variable `Alice_prediction`. Similarly, assign a value of `1` or `-1` for the variable `Bob_prediction`.