1 Linear Regression via Linear Algebra
1.1 Single Variable Regression
1.2 Multiple Regression with Linear Algebra
2 Useful Elements of Linear Algebra
3 Regression Framed via Vector Operations
4 Pre-Processing Regression Data
5 Homework
6 Appendix: The Eigen-Problem

Code ▾

Regression via Linear Algebra

… and some other gems from Linear Algebra

Dr. Eric Sullivan – DS401

1 Linear Regression via Linear Algebra

Let's stop messing around. Estimating linear regression parameters via Calculus and via Gradient Descent is just ridiculous. Beyond single-predictor problems, the calculus is gross, and gradient descient is somewhat slow for such an easy problem. Let's be civilized about this and just do linear regression the right way.

1.1 Single Variable Regression

We have a set of ordered pairs

$$S = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$$

where we want to build a model of the form

$$\hat{y} = \beta_0 + \beta_1 x$$

that predicts the y values from the x values with reasonable accuracy.

To find β_0 and β_1 with linear algebra we start by defining the vector $\boldsymbol{y}$ as

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix},$$

the matrix X as

$$X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix},$$

and the vector β as

$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}$$

Tasks:

1. Show that the product $X\beta$ indeed gives the intended estimate of y. Just do the matrix multiplication.
2. We want to solve the linear system of equations

   $$X\beta = \boldsymbol{y}$$

   for the vector β. Solve this equation for β ... but beware of how linear algebra works! (https://youtu.be/4F4qzPbcFiA). Take really careful note of the sizes of all of the matrices and vectors before you just try to solve this problem all willy nilly.
3. The way that you eventually had to solve $X\beta = \boldsymbol{y}$ via the normal equations actually has a great geometric interpretation. What is it?

You have just arrived at the *normal equations*. All statistics software actually implements the normal equations to solve linear regression problems instead of doing calculus or gradient descent. It is far more efficient!

1.2 Multiple Regression with Linear Algebra

The story doesn't really change for multiple regression. However, our set is a bit more complex in that we have points in much higher dimensions. If there are p predictors and 1 response then the points in our set are in $\mathbb{R}^{p+1}$. If we let x_{ij} be the value from the i^{th} observation and the j^{th} predictor then the set of points we are using for our regression is

$$S = \{(x_{11}, x_{12}, \dots, x_{1p}, y_1), (x_{21}, x_{22}, \dots, x_{2p}), \dots, (x_{n1}, x_{n2}, \dots, x_{np})\}.$$

We can define the vector $\boldsymbol{y}$ the same as before. The X matrix and the β vector, repectively, are

$$X = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}$$

and

$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{pmatrix}.$$

We are seeking to approximate the best entries of the β vector by solving the equation

$$X\beta = \boldsymbol{y}$$

just as before. And just as you found before, the solution is the normal equations

$$\beta = \left(X^T X\right)^{-1} X^T \boldsymbol{y}.$$

Again, this is how all statistics software actually finds the regression coefficients in a multiple regression setting.

1.2.1 Predicting the Concrete Compressive Strength Model with Linear Algebra

Let's put the normal equations to use on the Concrete Compressive Strength problem.

1. Read the concrete compressive strength data into `R` and use `lm` to find the multiple regression coefficients. We'll use these for reference.

Code

2. Now take the dataset and build the X matrix and the $\boldsymbol{y}$ vector. You'll need a few commands in `R` to get you going.
    a. The `matrix()` function is used to build a matrix. If you do `matrix(nrow=5, ncol=7)` it will build a 5×7 matrix filled with `NA` s.
    b. The `rep()` function is used to repeat the same value over and over (like to build a column of 1s for example). If you do `rep(1,50)` it will build a vector with 50 1s.
    c. The `t()` function takes the transpose of a matrix.
    d. The `%*%` command does matrix multiplication.
    e. Finally, `solve(A,b)` solves the linear equation $Ax = b$ In this case I'll give you a bit more of a nudge. We are solving $X\beta = \boldsymbol{y}$ and we start by multiplying both sides by X^T to get

    $$X^T X\beta = X^T \boldsymbol{y}.$$

    If you took one more step you would need the inverse, but we don't EVER do inverse computations on a computer since it is an unstable computation. If you look closely though, this equation has the typical form $Ax = b$ where $A = X^T X$, $x = \beta$, and $b = X^T \boldsymbol{y}$. Solving for β is now pretty easy with the `solve()` command.

Code

```
##                       BetaFromLinAlg    BetaFromR
## (Intercept)             -23.33121358 -23.33121358
## Cement                    0.11980433   0.11980433
## Blast_Furnace_Slag        0.10386581   0.10386581
## Fly_Ash                   0.08793432   0.08793432
## Water                    -0.14991842  -0.14991842
## Superplasticizer          0.29222460   0.29222460
## Coarse_Aggregate          0.01808621   0.01808621
## Fine_Aggregate            0.02019035   0.02019035
## Age                       0.11422207   0.11422207
```

2 Useful Elements of Linear Algebra

2.1 Vectors, Length, and Distance in Vector Spaces

In this section I'll only list a handful of the primary definitions and thoughts from linear algebra that you should be keeping in mind as you learn Machine Learning.

- **Vectors**, in the strict linear algebra sense, are abstraction objects that are elements of a vector space. While that abstraction is useful at times, vectors in Machine Learning will literally be columns of numbers since we are most often working with data that comes in this form. For example, a vector $\boldsymbol{u}$ could be defined as

$$\boldsymbol{u} = \begin{pmatrix} 1 \\ 3 \\ \pi \\ e^3 \\ 42 \end{pmatrix}.$$

  In this vector the entries are $u_1 = 1$, $u_2 = 3$, $u_3 = \pi$, $u_4 = e^3$, and $u_5 = 42$. Notice a notational convention: the symbol for a vector will always be bold and the symbols for the entries will not be.

Mathematically, a vector is really just a mathematical object where the notions of addition and scalar multiplication apply and make sense. Other examples of vectors are functions, matrices, and polynomials. This idea will show up a few times in machine learning, but for now you can think of all vectors as lists of numbers.

- The **dot product** is one of many natural ways to multiply vectors. Recall that the dot product is defined as follows. If $\boldsymbol{u}$ and $\boldsymbol{v}$ are defined as

  $$\boldsymbol{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} \quad \text{and} \quad \boldsymbol{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}.$$

  then the dot product $\boldsymbol{u} \cdot \boldsymbol{v}$ is defined as

  $$\boldsymbol{u} \cdot \boldsymbol{v} = u_1 v_1 + u_2 v_2 + \cdots u_n v_n = \sum_{i=1}^{n} u_i v_i.$$

  Other notions of vector multiplication like the cross product, Kronecker product, or the tensor product will not be widely used in Machine Learning.
- The **length**, or **norm**, of a vector is typically motivated by the Pythagorean Theorem whereas $\|\boldsymbol{u}\|$ is defined as

  $$\|\boldsymbol{u}\| = \sqrt{u_1^2 + u_2^2 + \cdots u_n^2}.$$

  It is very useful to note that this norm of a vector can also be defined as

  $$\|\boldsymbol{u}\| = \sqrt{\boldsymbol{u}^T \cdot \boldsymbol{u}}$$

  since the dot product of $\boldsymbol{u}$ with itself is $\boldsymbol{u} \cdot \boldsymbol{u} = u_1^2 + u_2^2 + \cdot u_n^2$.
  It would be rather short sighted, though, to say that the norm of a vector that we defined above is the *only* one. There are actually infinitely many ways to measure length!
  - The 1-Norm (aka the Taxicab norm):

    $$\|\boldsymbol{u}\|_1 = |u_1| + |u_2| + \cdots + |u_n|$$

  - The 2-Norm (aka the Pythagorean norm):

    $$\|\boldsymbol{u}\|_2 = \sqrt{u_1^2 + u_2^2 + \cdots + u_n^2}.$$

  - The 3-Norm:

    $$\|\boldsymbol{u}\|_3 = \sqrt[3]{u_1^3 + u_2^3 + \cdots + u_n^3}.$$

- ...
- The ℓ_p Norm:

$$\|\boldsymbol{u}\|_p = \sqrt[p]{u_1^p + u_2^p + \cdots + u_n^p}.$$

- The Infinity-Norm (aka the Max norm):

$$\|\boldsymbol{u}\|_\infty = \max_i |u_i|$$

In machine learning we'll find that the ℓ_1, ℓ_2, and ℓ_∞ norms are the most often used.

- The **distance between two vectors** is the length (or norm) of the difference between the vectors. For example, if $\boldsymbol{u}$ and $\boldsymbol{v}$ are vectors then the ℓ_2 distance is $\|\boldsymbol{u} - \boldsymbol{v}\|_2$. More clearly,

$$\|\boldsymbol{u} - \boldsymbol{v}\|_2 = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2 + \cdots + (u_n - v_n)^2},$$

and you should recognize this as just a generalization of the distance formula from geometry.

3 Regression Framed via Vector Operations

We can make use of the ideas of vectors when we write down our formulas for RSS and MSE (as well as others). For example, recall that the residual sum of squared errors is defined in multiple regression as

$$RSS = \sum_{i=1}^{n} \left(y_i - \left[\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots \beta_p x_{ip}\right]\right)^2.$$

We can greatly simplify the way that we write the RSS by using vector notation. Let the vector $\boldsymbol{y}$ be defined as

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix},$$

the matrix X be defined as

$$X = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix},$$

and the vector β be defined as

$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}.$$

Then a moment's thought reveals that

$$RSS = \|\mathbf{y} - X\beta\|_2^2.$$

We can rewrite one more time to see that

$$RSS = (\mathbf{y} - X\beta)^T \cdot (\mathbf{y} - X\beta)$$

A fun exercise is now to figure out what it means to take the derivative of the RSS function with respect to the vector β. Moreover, how does one take the derivative of a function with respect to a vector? Making sense of this will allow us to talk about the gradient of the RSS function with only one computation.

$$\frac{\partial RSS}{\partial \beta} = ???$$

You should do this exercise to verify that you get exactly the same gradient as you did in the gradient descent unit previously, but now we have more compact notation (for better or for worse).

All of this is to say that we can reframe the linear regression problem as:

Regression: *Minimize the ℓ_2 distance between the response vector $\mathbf{y}$ and the vector $X\beta$ by varying the entries of the vector β.*

Furthermore, we know that this minimum is achieved by projecting $\mathbf{y}$ onto the column space of X. This should make intuitive geometric sense: if we want to find the minimum distance between two objects, draw a line from one object perpendicular to the other.

4 Pre-Processing Regression Data

We have not dealt with this in the past, but every time we do regression we should really pre-process the data with a z transformation so that every predictor has mean 0 and standard deviation 1. If you do this you will lose some interpretability of the resulting model but you can always convert back by undoing the z transformations in the end. For example, if

$$y = \alpha_0 + \alpha_1 z_1 + \alpha_2 z_2 + \cdots \alpha_p z_p$$

then we need only to recall that

$$z_j = \frac{x_j - \bar{x}_j}{\sigma_{x_j}}$$

to recover

$$\begin{aligned} y &= \alpha_0 + \alpha_1 \frac{x_1 - \bar{x}_1}{\sigma_{x_1}} + \alpha_2 \frac{x_2 - \bar{x}_2}{\sigma_{x_2}} + \cdots + \alpha_p \frac{x_p - \bar{x}_p}{\sigma_{x_p}} \\ &= \alpha_0 + \frac{\alpha_1}{\sigma_{x_1}} x_1 - \frac{\bar{x}_1}{\sigma_{x_1}} + \frac{\alpha_2}{\sigma_{x_2}} x_2 - \frac{\bar{x}_2}{\sigma_{x_2}} + \cdots + \frac{\alpha_p}{\sigma_{x_p}} x_p - \frac{\bar{x}_p}{\sigma_{x_p}} \\ &= \left(\alpha_0 - \sum_{i=1}^{p} \frac{\bar{x}_i}{\sigma_{x_i}} \right) + \frac{\alpha_1}{\sigma_{x_1}} x_1 + \frac{\alpha_2}{\sigma_{x_2}} x_2 + \cdots \frac{\alpha_p}{\sigma_{x_p}} x_p \\ &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots \beta_p x_p \end{aligned}$$

where the β_j values are the regression parameters from regression without pre-processing.

The question is *why*. Why would you want to do this? Why would you need to do this?

The answer lies deeply in linear algebra. Instead of proving the underlying theorem we'll look at what happens in 1 and 2 predictor models and extend from there.

4.1 Why Pre-Process Regression Data

Step 1: Load the `Concrete_Data.csv` dataset. Build the regression matrix X with 1s in the first column and the `Cement` variable in the second column.

Code

Step 2: Build the matrix $A = X^T X$ from the normal equations.

Code

Step 3: Find the eigenvalues and eigenvectors of the matrix A using the `eigen` command in `R`. What do you notice about the eigenvectors? What do you notice about the eigenvalues?

Code

Step 4: Now let's pre-process the data by doing a z transformation on the cement predictor.

Code

Step 5: Repeat steps 2 and 3 on the transformed data. What do you notice about the eigenvalues and eigenvectors now?

Code

- Repeat this numerical experiment with different predictor columns. How does the scale of the values in the predictor play a role?
- Now repeat this numerical experiment with two predictors (so X is 1030×3 this time). What do you notice about the eigenvalues and eigenvectors before and after the z transformation?

4.2 The Linear Algebra of Pre-Processing

Theorem: The matrix $A = X^T X$ will always be a symmetric matrix (where the line of symmetry is the main diagonal).

Proof: You should at least prove this for a single-predictor regression matrix.

Theorem: If A is a symmetric matrix then

a. all of the eigenvalues are real, and
b. all of the eigenvectors are perpendicular to each other.

Proof: This particular proof is beyond the scope of this course.

The last theorem says that we will always have a full suite of perpendicular eigenvectors in the matrix $X^T X$ … the best basis we could possibly ask for. Remember that the eigenvalues tell you how much the space is scaled in the direction of each eigenvector. What we saw in the previous section suggests that if one (or more) of the columns in X is on a vastly different scale relative to the others then the resulting eigenvalues are going to be on vastly different orders of magnitude. This means that the space is being stretched in one direction and not another.

Now let's recall another theorem from linear algebra.

Theorem: If a matrix A has an eigenvalue of 0 then A^{-1} does not exist.

Corollary: If a matrix A has an eigenvalue *very close* to 0 then A^{-1} *almost doesn't exist*.

Let's think of this by analogy. If a positive number x is very close to 0, then computing $1/x$ gives you a number, but that number is really really large. If you were to wiggle x by a little bit the value of $1/x$ would wiggle by A LOT!! The analogy to matrices is the same: if a matrix is *close* to non-invertible then any small wiggle in the values inside the matrix will cause massive changes in the inverse. This makes the inverse useless from a practical sense.

Conclusion: If one (or more) of the predictors in a regression matrix X are on a much larger scale than the other(s) then the eigenvalues of $X^T X$ will be on very different orders of magnitude. In the extreme case, the smallest eigenvalue is computationally close to 0 relative to the largest eigenvalue. This means that the matrix $X^T X$ will be *very close* to not having an inverse. Hence, **finding regression coefficients when the predictors are on different scales results in an unstable matrix inversion problem**!

> **Key Takeaway:** If your predictors are on vastly different scales, then you should definitely pre-process your regression predictors with a z transformation. This way you avoid computational issues with the matrix inversion behind the scenes.

5 Homework

5.1 Problem 1

Let X be a regression matrix as described in these notes (1s in the first column, predictors in the subsequent columns). We showed that the system of equations $X\beta = \boldsymbol{y}$ can be approximated by solving $X^T X\beta = X^T\boldsymbol{y}$.

1. Explain what multiplying by X^T both sides of $X\beta = \boldsymbol{y}$ is doing geometrically.
2. Prove that the entry in row 1 column 1 of $X^T X$ will always be the same as the number of columns of X.
3. Prove that the matrix $X^T X$ will always be a symmetric matrix. Recall that a symmetric matrix has a line of symmetry along the main diagonal. Therefore, in a symmetric matrix the entry A_{ij} will have the same value as the entry A_{ji}.

5.2 Problem 2

Previously we stated the RSS for a regression problem is the same as the quantity $\|\boldsymbol{y} - X\beta\|_2^2$. Recall further that $\|\boldsymbol{u}\|_2^2 = \boldsymbol{u}^T \cdot \boldsymbol{u}$ for any vector $\boldsymbol{u}$. Therefore

$$RSS = (\boldsymbol{y} - X\beta)^T(\boldsymbol{y} - X\beta).$$

Show the algebra to verify that if we differentiate RSS with respect to the vector β we get the correct values for β_0 and β_1 in the single-predictor regression case. (look back to notes 1 or in chapter 3 of the text for the algebraic forms of β_0 and β_1)

5.3 Problem 3

Regression simply does not work if the number of observations (rows) are less than the number of predictors (columns)! Let's consider a simple/ridiculous case where there are 2 observations and three predictors. This would mean that the regression matrix X would be 2×4, the matrix X^T would be 4×2, and the matrix $X^T X$ would be 4×4.

1. What can you say about the matrix $X^T X$ in this simple case?
2. What are the implications on solving $X\beta = \boldsymbol{y}$?
3. Extend your statement to a theorem about the matrix $X^T X$ in the case that there are fewer observations that predictors. State your theorem clearly and provide a sketch of a proof.

5.4 Problem 4:

From the `MASS` package load the `Boston` dataset. (Remember that the `MASS` package has a `select` command just like the tidyverse, so if you are going to use the `tidyverse` select command you need to load `tidyverse` last.)

1. Start by building a model for the median household value (`medv`) as explained by everything else in the dataset. Save this model so you can compare the coefficients in future steps of this problem.
2. Explicitly build the normal equations and use them to find the coefficients for the linear model. Compare to what you found in part 1. Be sure to include your code in your writeup.
3. Find the eigenvalues of the matrix $X^T X$ and compute the ratio between the largest and the smallest eigenvalues. This ratio is called the *condition number* of the matrix $X^T X$. The condition number in this case should be huge! What does this number mean about the matrix $X^T X$? (Hint: think about what it means geometrically for the ratio of largest to smallest eigenvalues to be very very large)
4. Now pre-process the predictors by doing a z transformation on each. Recall that

$$z_j = \frac{x_j - \overline{x_j}}{\sigma_{x_j}}.$$

   a. The coefficients in the model will be different that what you found. Explicitly show how you convert this new model to the original model algebriacally.
   b. Repeat problems 2 and 3 on the pre-processed data. What does the conditions number tell you now?
5. The z transformation is not the only type of pre-processing that you could do. We can also pre-process our data so that every predictor has a minimum of 0 and a maximum of 1. This will force every predictor to live on the unit interval. The logic here is that we give every predictor the same *range* instead of the same mean and standard deviation. We'll call this the unit transformation

$$u_j = \frac{x_j}{\text{max(x\_j)} - \min(x_j)}.$$

   a. Perform this pre-processing on the data and build the normal equations for this pre-processed data. Again, what does the condition number tell you?
   b. Show that you can reverse this transformation to arrive back at the model for the non-processed data.
6. At first glance it may appear that the model coefficients that you found in parts 1, 2, 4, and 5 are all the same, but they are not! Devise a way to measure which collection of coefficients most closely matches the coefficients from `R`'s `lm` command.
7. Summarize what we've done in this problem. Why is pre-processing important, and for this particular problem which type of pre-processing is the best?

6 Appendix: The Eigen-Problem

6.1 The Geometry of Eigenvectors

Fact: Matrix multiplication by a square matrix takes a vector, rotates it, and scales it. For example, if

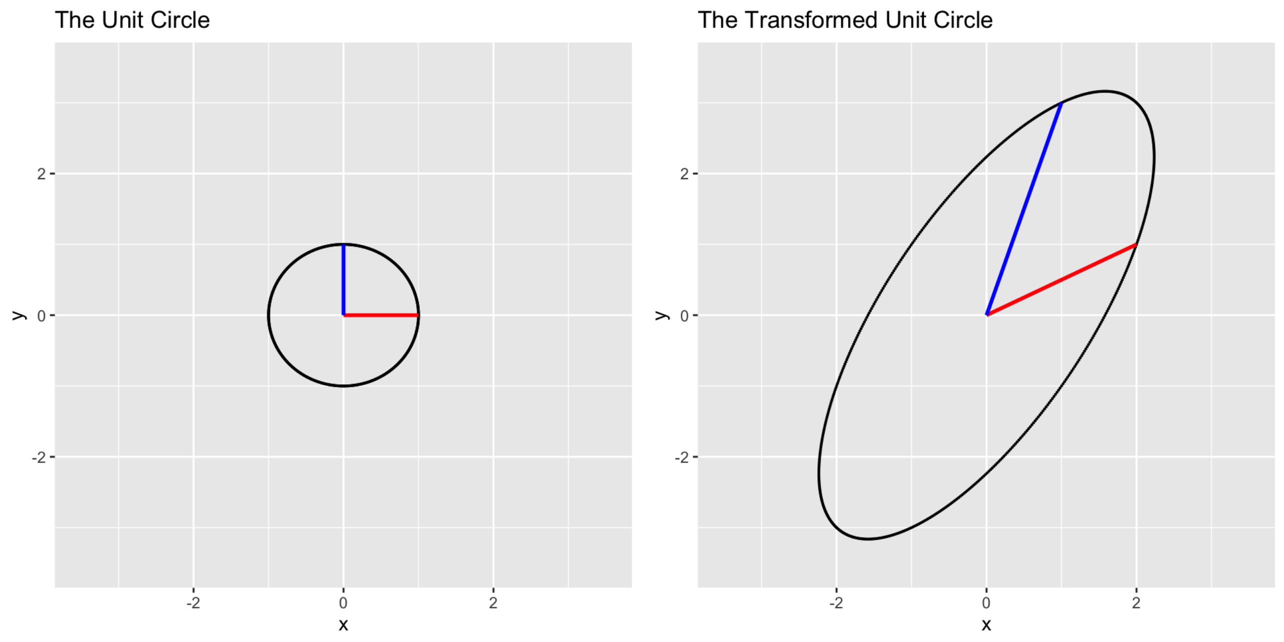
$$A = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}$$

and $\boldsymbol{u}$ is any vector in $\mathbb{R}^2$ then the vector resulting from the multiplication $A\boldsymbol{u}$ will be a scaling and a rotation of $\boldsymbol{u}$. Let's try a few.

$$\begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

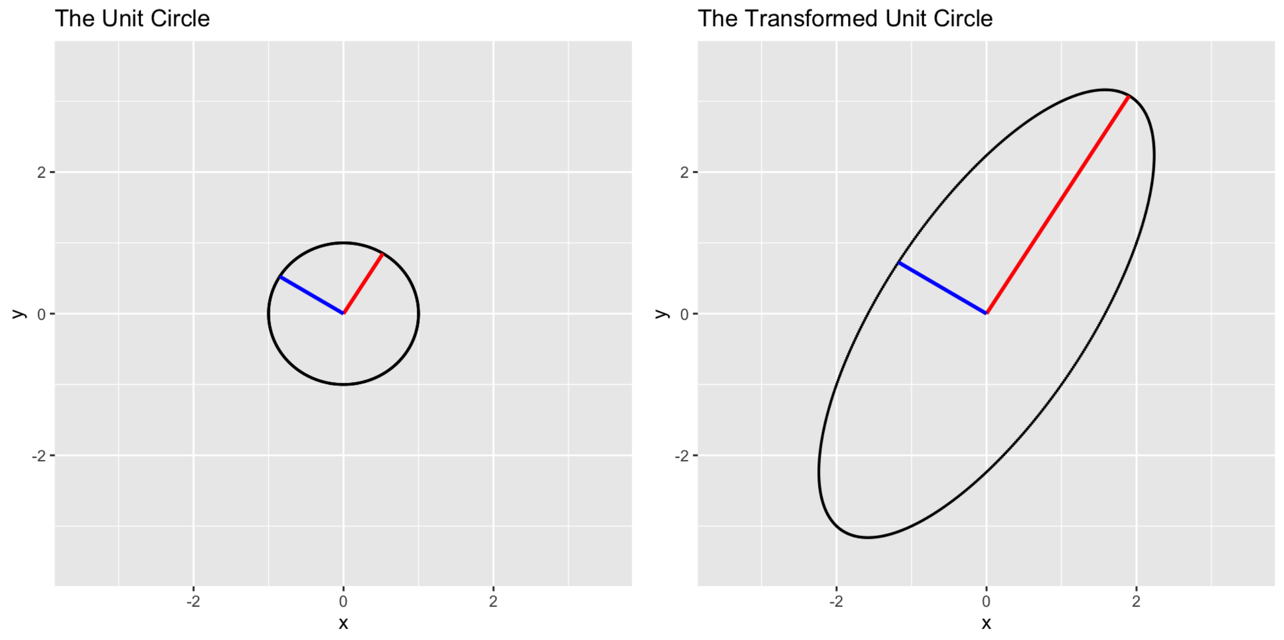
$$\begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

In any example that we cook up there will be a rotation and a scaling. If we apply this matrix to every vector on the unit circle then we will see an image of all of the resulting scalings and rotations. The red and blue vectors below are shown for reference for how our two examples were rotated and scaled.



The Eigenvalue-Eigenvector problem asks a simple question: *which vectors are only scaled by this matrix multiplication?* We want to find the vectors that were NOT rotated by the matrix multiplication – these are the eigenvectors. The amount by which they were scaled are the eigenvalues.

In the next figure we now show the eigenvectors on the same picture. Notice that each of these vectors is only scaled, not rotated, by the matrix A.



6.2 The Mathematics of Eigenvectors

First remember that it only makes sense to talk about eigenvectors and eigenvalues of a square matrix. There are extensions to non-square matrices that we'll get to in due course.

Mathematically we state the eigenvector problem as: *Find the pairs λ and $\boldsymbol{v}$ such that*

$$Av = \lambda v.$$

The vectors $\boldsymbol{v}$ are the **eigenvectors** and the scalars λ are the eigenvalues. Notice that the matrix equation $A\boldsymbol{v} = \lambda\boldsymbol{v}$ is just a restatement of the geometric property that the vectors end up parallel to themselves after being multiplied by the matrix A. … make your own inner peace with this.

The thing to keep in mind about the eigenvectors of a matrix are that they form, in some sense, the *best* basis for the column space of the matrix. Moreover, they are also the most convenient vectors for computation since if you create a vector $\boldsymbol{u}$ as a linear combination of the eigenvectors $\boldsymbol{v}_1$ and $\boldsymbol{v_2}$

$$\boldsymbol{u} = c_1\boldsymbol{v}_1 + c_1\boldsymbol{v}_1$$

then the product $A\boldsymbol{u}$ becomes easy since

$$A\boldsymbol{u} = A(c_1\boldsymbol{v}_1 + c_1\boldsymbol{v}_1) = c_1\lambda_1\boldsymbol{v}_1 + c_1\lambda_2\boldsymbol{v}_1.$$

This last fact is worth mentioning again: *If you have a basis of eigenvectors then you have drastically reduced any computation time associated with the matrix.* For this reason, most linear algebra algorithms will leverage the eigenvalue computation first in order to get speedup later.

Typically in Machine Learning we will not directly compute the eigen-structure of a matrix. If you wanted to actually do the computation then you only need to remember the steps:

$$\begin{aligned} & A\boldsymbol{u} = \lambda\boldsymbol{u} \\ \implies \quad & (A - \lambda I)\boldsymbol{u} = \boldsymbol{0} \\ \implies \quad & (A - \lambda I)^{-1} \text{ does not exist} \\ \implies \quad & \det(A - \lambda I) = 0. \end{aligned}$$

Solving the equation $\det(A - \lambda I) = 0$ for λ will give the eigenvalues. Then solving $(A - \lambda I)\boldsymbol{u} = \boldsymbol{0}$ for $\boldsymbol{u}$ using each λ will give the associated eigenvectors.

Don't worry about actually doing any of this math by hand in Machine Learning. Just use the `eigen()` command in `R` to do the work for you.