#h1 Chapter 12 - Transformations

#h2 Rotation, translation, scaling, and shearing

#h3 12.1 – Introduction

#pg At its core, machine learning attempts to find a mapping of its inputs on the desired outputs. In geometry, this is usually called a transformation. Finding this optimal mapping is the topic of the next module on learning and optimization. But before we start exploring how we transform our data to achieve the optimal mapping of input onto outputs, we first must understand the characteristics of the space of our input and output data. And the most important one is how we define distances.

#h3 12.2 - The Geometry of everyday space

#pg To most of us, geometry deals with the relative position of things in our familiar 3D world, the angles between them, their extent, size, width, height, depth, volume, and symmetry. But geometry is an abstraction away from that intuitive world of up, down, left, right, far, near. For starters, geometry is not necessarily 3-dimensional. Geometry doesn't care about the number of dimensions. The relationships between points in the spaces remain the same. Still, perhaps it takes a bit more effort to compute for high-dimensional spaces. To make matters even more complicated, most of us are only familiar with one out of many, if not an infinite number of possible geometries. Most of us learn about the geometry defined by the Greek mathematician Euclid because this geometry works particularly well for beings our size and dimensionality. Euclid based his geometry on a series of axioms or first principles. From those axioms, he derived all other relationships. One of those axioms is that parallel lines will never intersect, no matter how far out you extend them. This axiom makes Euclidean geometry special among all possible geometries. As for other geometries, this is not an axiom we can rely on. This parallelism axiom is a more profound statement about the nature of the space around us. Euclidian geometry is based on the idea that space consists of three dimensions that give rise to three strictly orthogonal axes. In other words, moving along one axis (say, the one we defined as up/down) does not change our position on the other axes. Again, the physical reality we observe and must contend with seems to favor this definition of our geometry. However, this orthogonality breaks down at much larger scales: space (as well as time) can start to bend due to forces that act upon them (like black holes). I can encourage you to explore this idea further outside of this course, but unfortunately, we will have to move on to the matter at hand: defining distances in our own geometries.

#im ../assets/figures/012/012-01.png 50 256 Figure 12.1 - The Euclidean Geometry of Everyday Space. In Euclidean geometry, the world has three spatial dimensions. What we define as up/down, left/right, or far/near is arbitrary. If these axes are orthogonal to each other, they conform to the rules of Euclidean geometry.

#h3 12.3 - The geometry of machine learning data

#pg We create our own geometry by building machine learning models using multidimensional data. As discussed, a multidimensional data set in machine learning consists of multiple input variables combined to compute some output of interest. For example, we can use ten demographic indicators to predict cancer risk. Each indicator is its own variable (also called a regressor) and has its own dimension in the geometry of our data space. When you use age, gender, race, ethnicity, income, education level, and BMI, you thus create a 7-dimensional data space. And in that data space, we can map each row of data (a single individual in our data to a point somewhere in that data space. From a geometric point of view, machine learning is about understanding the regularities of the relative positioning of all these points.

#im ../assets/figures/012/012-02.png 50 256 Figure 12.2 - The geometry of data spaces. In the geometry of data spaces, our machine learning algorithms look for regularities across data points that occupy the space. (A) In regression, the output changes along some composite axis of the input data. (B) In classification, data points are separated by their class. (C) In clustering, we associate points with some natural but unknown grouping in our data.

#br

#pg In geometry, there is a natural analog to (dis)similarity: the distance between data points. Do data points that are associated with high cancer risk cluster somewhere in the space, compared to the data points that are associated with low cancer risk? Is there some dividing plane that optimally separates these two clusters of points? Or are the points aligned along some direction in this space, where moving back and forth changes the cancer risk in a predictable manner? The key to answering those questions relies very much on how we measure the relative position of data points: how similar are two data points? And how dissimilar are both these data points to a third data point?

#h3 12.4 - Defining distance

#pg In machine learning, we use distance to define similarities, differences, errors, margins, and probabilities. Therefore, how we define distance changes how we define these metrics. If our machine learning data lived in a perfect world, analogous to the Euclidean ideal, we wouldn't have to worry too much. However, in the same way gravity can bend space at cosmological scales, the statistical nature of our data can bend our data space. In Euclidean geometry, the axes are orthogonal. If this holds for our machine learning data, we expect a change along one variable (one dimension) does not affect any other dimensions. But we know this rule is often violated. Variables usually correlate; when they do, they aren't independent, and their axes aren't orthogonal. Another assumption of Euclidean space is that our space has equal units. One amount of distance along any axis is the same as that along any other axis. Equivalent units across dimensions are also not a given. Our variables often represent drastically different processes, values, quantities, and units. Comparing them isn't straightforward. To see where we usually must be careful in defining and quantifying distances between data points, let's start by implementing Euclidean distance, and see when it runs into problems.

#h3 12.5 - Euclidean distance

#pg Computing the Euclidean distance for any kind of dimensionality is straightforward. Consider a set of new data points compared to a large set of points separated in two dimensions. Equation 12.1 defines the distance between two points defined in 2D by their x and y coordinates.

#im ../assets/equations/012/png/012-01.png 50 48 Equation 12.1 - Euclidean distance D between points p and q, specified in 2D (x,y) coordinates.

#h3 12.6 - Violations of the Euclidean assumption

#pg Although highly intuitive, we will encounter many violations of the Euclidean assumption in real-world data sets. A violation means that if we compute the distance between two points, the resulting value would not be representative of that distance relative to other pairs of points. A common violation is that of unequal units across dimensions. For example, Figure 12.2 shows the hypothetical relationship between age and height. We express 'age' in years and 'height' in feet. This difference in units results in a distance of 1 between points A and B, but the distance between points B and C is 10. This difference causes an algorithm to place more weight on the latter point, even though we do not know how to interpret the differences in scale. How does one unit of 'year' compare to one unit of 'feet'?

#im ../assets/figures/012/012-03.png 50 256 Figure 12.3 - Unequal units across dimensions. When we express variables in different units, it makes their contribution to the overall performance of any machine learning model hard to assess.

#h3 12.7 - Unequal Variance

#pg A more subtle violation of using Euclidean distance can arise from datasets not having equal variance. Figure 12.2 shows the relationship between the years a person is married and the years they have spent in school. This time, the axis represents the same units. However, the variance of married years is much larger than those spent in school. Again, this makes it hard to compare these two variables directly.

#im ../assets/figures/012/012-04.png 50 256 Figure 12.4 - Unequal variance across dimensions. Even when two variables are specified in the same units, they can still differ substantially in their variance.

#h3 12.8 - Covariance

#pg Another problem arises from computing the distance between points whose dimensions are not independent. Figure 12.3 shows a strong relationship between hypothetical data correlating days of activity (exercise) with BMI. In this case, measuring a distance between points A, B and C becomes biased if we compute Euclidean distance using the formula in Figure 1. Think of the observed covariance as a transformation that has both rotated and compressed the space specified by the axes BMI and activity, creating a new coordinate system with a new set of axes. A distance metric is more meaningful when computed along these new axes.

#h3 12.9 - Alternative Distance Metrics

#pg Given that the Euclidean distance isn't the ideal metric, what are our options? Several different distance metrics have been proposed to mitigate some of the skew we might observe in our Euclidean distance metrics. Some of these metrics are specific to particular data types, like the Lowenstein distance to compute the difference between two strings and the Jaccard distance for sets of categorical variables. Others correct for a specific aspect that might mislead us.

#im ../assets/figures/012/012-05.png 50 256 Figure 12.5 - Strong covariance between our variables creates a distorted distance metric. In this example, in Euclidean terms, B and C are equidistant from A. However, most of us will argue that points B and A are more closely related and belong to the same cluster of points, where C is an outlier from this cluster.

#br

#pg For example, the Mahalanobis distance aims to define distance in terms of standard deviations. Looking back at Figure 12.4, notice that the two points B and C are equidistant from A; therefore, our Euclidean distance would assign them the same value. However, the variance of our data is elongated substantially along the direction of A to B. This elongation indicates that although B and C are the same absolute distance away, B is much closer to A than C in terms of the number of standard deviations. Where B still falls within the +/- 1 standard deviation region, accounting for 68 percent of the observed data points, C is almost three standard deviations away, making the likelihood of observing it much smaller than B.

#im ../assets/equations/012/png/012-02.png 50 32 Equation 12.2 - Manhattan or City Block distance. This distance metric simply sums the absolute distance between p(x,y) and q(x,y) on each dimension. Like all other distance metrics discussed here, this distance matrix can be generalized to any number of dimensions.

#h3 12.10 - Manhattan and Cosine Distance

#pg Two more distance metrics are worth describing (and are plotted in Figure 12.5). First, the Manhattan or city block distance sums the absolute distance along each axis. You can see where this metric got its name grid-like layout, like streets in a city. The distance traveled is along the roads, as we can't go through the buildings. But it is not literal urban planning why we use this distance metric often. The Manhattan distance is preferred when we have sparse categorical or binary variables or data. Cosine distance is more a measure of similarity in direction rather than a difference in space. In geometry, a point is best seen as a vector, extending from the origin (0,0) to the location of the point (x,y). Formulated in this way, we can define a point (read: vector) as having a direction and a length. Two vectors with the same direction but different lengths will be parallel, with the longer just extending outward from the origin (0,0) further. Two vectors with different directions but the same length diverge, stopping at the same distance from the origin. Sometimes, the difference in direction is what is important to us, not the difference in length, or a combination of length and direction. Imagine we are classifying documents. One document has ten repeats of the word 'cancer' and two mentions of the word leukemia. The other document has five repetitions of the word 'cancer' and one mention of 'leukemia'. A third document mentions 'cancer' 5 times, but instead of 'leukemia', it mentions 'Capricorn'. You can see that even though documents A and B have different numbers of mentions, they still overlap in what they mention. In other words, they point toward the same topic with different strengths. Article 1 might be said to be 'more' about cancer and leukemia, but it is not necessarily different in content than article 2. In contrast, the third article has something in common with articles 1 and 2 but also has its own unique keywords. Therefore, it points towards a different direction in 'topic space'. This difference in direction, rather than length, is captured by the cosine similarity metric. It is common to define this metric as similarity, which is the complement of distance.

#im ../assets/equations/012/png/012-03.png 50 128 Equation 12.3 - Cosine similarity. Cosine similarity captures the angular difference between two vectors p and q, while being agnostic to any possible difference in the length of the two vectors A and B. Therefore, the cosine similarity between two vectors A = [2,2] and B = [4,4] is maximal, as they point in the same direction, even though they differ in length.

#h3 12.11 - The curse of dimensionality

#pg An additional potential issue that most distance metrics share is their behavior with high dimensional data sets. With many dimensions contributing to the distance, noise or non-defined distances along most dimensions overwhelms whatever signal might exist in informative dimensions. This effect is especially problematic for sparse matrices, where most values are 0. Compare two rows of a sparse matrix; you will find 0 distance along most dimensions (because both rows are mostly 0). The few times the rows differ, the resulting distance is non-zero, diluted by all the distances that are 0.

#im ../assets/figures/012/012-06.png 50 256 Figure 12.6 - Three different distance metrics visualized. (A) Euclidean distance between Q and P. (B) Manhattan distance between Q and P. (C) Cosine similarities between R and S and U and S.

#h3 12.12 - Normalization

#pg The above violations are not necessarily a problem for all algorithms when they are mild. However, some form of normalization typically increases accuracy for bigger data sets. There are two approaches we can take to mitigate these issues. First, we can transform the data itself to remove those violations. This transformation typically includes subtracting out the mean of each regressor and subsequentially dividing it by its standard deviation, a statistical method known as Z-scoring. Z-scoring is a common step in preprocessing data before we use it to train our models and typically involves standardizing the data across regressors.

#im ../assets/equations/012/png/012-04.png 50 64 Equation 12.4 – Z-score transform. By subtracting the mean of x from all x and dividing it by the standard deviation, we center our data on a mean of 0, with a standard deviation of 1.

#br

#br

#im ../assets/figures/012/012-07.png 50 256 Figure 12.7 - Effect of normalization. (A) Before normalization, the two variables had different means, as well as different amounts of variance. This unequal variance can lead to problems fitting machine learning models and makes interpreting the resulting learned weight/parameters of the machine learning model difficult. Just because a weight on a variable is larger than other weights doesn't necessarily mean it provides more evidence for the prediction. (B) Mitigating this using Equation 12.4, we now have our two variables in the same ballpark: roughly equal means and variances.

#h3 12.13 - Categorical Variables

#pg One final situation in which the Euclidean distance will fail is if we try to compute a distance that is not defined. As we saw earlier, categorical variables have ordinal or nominal scales. If the space between two points is not continuous, it has no actual definition of length or distance. Something is either a statue of Abraham Lincoln or last year's tax return. There is no in-between. We can circumvent this issue using a method called one-hot encoding, by which a categorical variable we transformed into a binary representation, something we will see in action later. Still, you can find a small example in Figure 12.8.

#im ../assets/figures/012/012-08.png 50 128 Figure 12.8 – One-Hot Encoding is a process of encoding categorical variables as a binary representation, where we need n new regressors to encode a categorical variable with n unique states. Each new regressor encodes for the presence (1) or absence (0) of one of the n unique states (red, blue, green, and purple).

#h3 12.14 - Transformations

#pg From a geometric perspective, we think of algebraic operations as transformations of points. We can move them, scale them, rotate them, etc. Approximating a regression function (finding the best fitting function through our data) then means we aim to find a transformation of our data into a particular function that minimizes the difference between the points and the function. These transformations are applied to our data using transformation matrices. These specify how we will transform the data using a combination of rotation, translation, scaling, and shearing. If we rotate and translate our points, we are performing a rigid transformation since these transformations do not change the relative position of our points. If we combine it with scaling and or shearing, we perform a so-called affine transformation. Points that were neighbors before the transformation will remain neighbors after the transformation.

#im ../assets/figures/012/012-09.png 50 256 Figure 12.9 - Why we need a bias term. Without a bias term, our machine learning model has no mechanism or parameter to introduce an offset of one variable relative to another.

#h3 12.15 - Bias terms and intercepts

#pg An essential detail we need to discuss is bias terms. We need to introduce a bias term to combine our transformation into a single matrix to apply to our data. The name is slightly confusing, and it is known differently in different fields of mathematics and statistics. But the key idea is always the same. The weights that we estimate have a multiplicative effect on our data. Not additive (or subtractive). That means we cannot have weights that simply translate our data (move it uniformly up, down, sideways, etc.). Specifically, when we have the basic linear equation, we can find values for a that multiply with x to get us close to y. But we cannot use it to translate. The way out is to introduce a bias term. Think of b as just another column vector. The only difference is that it is constant, with all values set to 1. We are adding such a column allows us to apply a purely additive or subtractive offset to our data x to match y. Where a scales x, b only scales a column that is one everywhere. We will need this extra column in our data to apply the most basic type of transformation, a translation, see Figure 12.5.

#h3 12.16 - Translation

#pg Translation can move our points along each axis of our data independently. In the case of a set of 2D points, we can add or subtract from our x and y coordinates independently. Add 2 to all our x coordinates, subtract four from all our y coordinates, and we independently translate our entire data set along both axes. That is straightforward enough, but we might want to rotate and scale the same data. For our case, it is much better to think of translation as a matrix operation, where we apply a transformation matrix (specifying our translation) to our original data. Let us start with some very fundamental algebra. Our data is a matrix of nx3 elements. Each row i of n is a point, with an x coordinate (first column), a y coordinate (second column), and a constant value of 1 in the third column. As stated above, this column of ones serves as our bias or intercept. The matrix we will use to translate the original matrix (let us call it P for points) in both the x and y dimensions is a 3x3 (rows by columns) matrix, where each element or combination of elements is responsible for encoding the amount translation, rotation, scaling, and shearing.

#im ../assets/figures/012/012-10.gif 70 132 Figure 12.10 - Multiplying M with identify matrix I results in the same matrix M asdasd asdasdasdasdasdasdasd.

#br

#pg The way it achieves this is one of those extremely useful intuitions you can develop when thinking about a point in the space of arbitrary dimensionality (we will use 2 for now). Here is a little trick for those unfamiliar with algebra and matrix multiplication. The first thing to know is that to multiply two matrices A of size n x m (n rows by n columns) and B of size k x l, the number of columns of A (m) must be the same as the number of rows k in B. To apply a transformation, we multiply the nx3 data matrix T with our 3x3 transformation matrix P. The multiplication result equals the number of rows n in A and columns l in B. So, what if we multiply our points matrix P with our transformation matrix T? Because the number of columns in P (3) is the same as the number of rows in T (also 3). And what is the size of the resulting matrix? P has n number of points, and T has three columns. Therefore, the output P is a matrix the same size as the input matrix P. That is what we need: a new set of points in the same format as the original set of points with its values transformed by matrix T. Before we use this approach to translate our points, we should consider a transformation matrix that does not affect our points: the identity matrix. An identity matrix is a square matrix (n x n) of zeros, except for the diagonal, where the elements are all one. As the animation below of matrix multiplication using an identity matrix will show, this matrix leaves our points untouched. It serves as the basis of the transformation matrices we will consider next, i.e., translation, rotation, shear, and scaling. To specify a translation using a transformation matrix, we start with an identity matrix and change two elements to implement a translation. Specifically, we change the value in the last row and the first column to the value we want to shift our data along the first dimension (x). Similarly, we can move our data along the second dimension (y) by changing the value in the last row and the second column. When we compute the dot product of our data with this new transformation matrix, we obtain a new matrix, shifted along its dimension as specified.

#im ../assets/figures/012/012-11A.gif 70 192

#im ../assets/figures/012/012-11B.gif 70 192 Figure 12.11 - Multiplying matrix M with a transformation matrix T moves the points along the x and y-axis.

#h3 12.17 - Rotation

#pg The second type of transformation is rotation. Per definition, rotations are around the origin of the space, which is not necessarily the origin of your data. To translate your data around its center rather than the origin, you will need to translate the data to center it on the origin, rotate it, and translate it back to its original location. A helpful way to think about rotation is to see it as a particular type of translation. Whereas a translation can move a point further away or closer to the origin, a rotation can translate the point to a new position for which the distance to the origin stays the same. Imagine a piece of wire attached to your point and the origin. To rotate a set of 2D points, we specify a transformation matrix. Starting with an identity matrix, we use the top left block of 2x2 cells to determine the rotation as the cosines and sines of our specified angle (in radians).

#im ../assets/figures/012/012-12A.gif 70 256

#im ../assets/figures/012/012-12B.gif 70 256 Figure 12.11 - Multiplying matrix M with a rotation matrix R rotates the original points around the origin (0,0).

#h3 12.18 - Scaling

#pg A translation will center data to a new point (usually the origin), and a rotation will rotate the points around a point, but neither affects the absolute distance between points. We call these rigid body transformations. We can push and pull all the data points at once, but we cannot squash them closer to either or pull them apart. For that we need a third type of transformation: scaling. Whereas we can see a translation as a scalar subtraction or addition operation, scaling is a multiplication. Notice that after we translated and rotated our data, the data elongates along one axis. Such a distortion could be due to the dimensions simply reflecting different quantities or the fact that even though both axes represent the same physical quantity (say: length), one variable has a larger spread than the other. For example, in absolute terms, body length across different humans will show a broad range of values and, say, the average body length of a sparrow.

#im ../assets/figures/012/012-13A.gif 70 256

#im ../assets/figures/012/012-13B.gif 70 256 Figure 12.12 - Multiplying matrix M with a rotation matrix S results in a scaling applied to the original data points.

#h3 12.19 - Shear

#pg Shear is perhaps a bit difficult to describe in plain English but easy to recognize when you see it. Like scaling, it does change the absolute distances between points, whereas translation and rotation do not. By scaling, we can imagine pushing and pulling the points apart in the direction of the axes. With shearing, this pushing and pulling appear to happen at an angle relative to the x and y-axis.

#im ../assets/figures/012/012-14A.gif 70 256

#im ../assets/figures/012/012-14B.gif 70 256 Figure 12.13 - The multiplying matrix M with a shearing matrix H results in a shear applied to the original data points.

#h3 12.20 - Combining rotations, scaling, shearing, and translations

#pg One beneficial consequence of linear algebra is that the matrices we specified for translation, rotation, and scaling can be combined beforehand. We obtain a single transformation matrix that applies all three to a dataset by simply multiplying all three matrices in a row. And this applies to all transformation matrices. By combining two (or more) transformation matrices already specifying their translation, rotation, and scaling, we end up with a matrix that combines that all. Furthermore, we can inverse the matrix, which is useful should we ever want to transform our points back into the opposite way. Finally, we can quickly decompose these matrices to isolate the rotation, translation, scaling, or shearing embedded in the matrices.

#h3 12.21 - Putting it all together - Mappings

#pg Now that we understand how to manipulate the position of points, we can start to understand what a machine learning algorithm aims to do. In essence, it attempts to find the best mapping of our input data onto our desired output data by translating, rotating, scaling, shearing, and projecting it. Consider the parameters our models learn as the values we need to plug into our translation, rotation, and scaling matrices to make this happen. And the complexity of this mapping dictates the complexity of the machine learning algorithm. We typically do not think these mappings are a series of translations, rotations, and scaling (and shears and reflections). Rather, it is perhaps more common for engineers to see them as weights we apply to our input data. Figure 12.1 provides a visual representation of the geometric interpretation of machine learning. We want our machine learning algorithm to produce a value close to -1 when the class is 'red' and a value of +1 when the class is 'blue'. Furthermore, we want confidence or probability in this prediction. By scaling, translation, and rotating our initial point cloud, we can align its x-axis with this required output. Again, in the geometric interpretation, machine learning involves finding the most optimal mapping of an input space that is typically very high-dimensional onto a single value or a limited number of values representing some decision and confidence level. However, it is common for additional nonlinear transformations to occur by applying nonlinear functions to our data in addition to weights specifying the affine transform of inputs onto outputs. Nevertheless, non-linearities can still be interpreted as the transformation or mapping of inputs onto outputs, albeit in a more complex, non-rigid way.

#im ../assets/figures/012/012-15A.gif 70 256

#im ../assets/figures/012/012-15B.gif 70 256 Figure 12.14 - Multiplying all the above matrices together into a single transformation matrix will combine the rotation, translation, scale, and shear into a single composite transformation matrix

#h3 12.22 - Conclusions

#pg This chapter highlights my intuition, perspective, and approach to machine learning: visual imagery of the structure of our data, as represented as a high dimensional state space. But in this case, the dimensions do not correspond to Euclidian space of up, down, left, right, near, and far. Instead, dimensions represent single variables we use as inputs. What we aim to unearth in this complex space is a structure. Our data points do not fall randomly in this space. If they did, no dimension would ever correlate with other dimensions. Still, more importantly, there would be no organization between data points we can exploit to infer some prediction or outcome of interest. In the vast state space representing all the exciting bits about our pets, surely all the data points representing cats have structure. Their tails are long, they have claws, meow, purr, hunt for mice, and take long naps, preferably on our laps. Each characteristic adds a dimension representing it to our data state space. However, dogs and cats do not differ in most of these dimensions (they are both pets, mammals, do not hold PhDs, have poor color vision, walk on four feet, and like to be petted (mostly). But still, in some dimensions, they differ in the structure, form, and shape their data points take on. We can use this difference in the structure to tell a machine whether 'Fluffy' is a dog or a cat.

#h3 12.23 - Demo Notebooks

#pg Notebook 012.01 has example code that walks through the transformation detailed above in Python.

#h3 12.24 - References

#bs

#be

#bp Strang, G. (2021). Introduction to Linear Algebra (5th ed.). Wellesley-Cambridge Press.

#bp Shilov, G. E. (2012). Linear Algebra. Dover Publications.

#bp Wallisch, P., Lusignan, M., Benayoun, M., Baker, T. I., Dickey, A. S., & Hatzopoulos, N. (2014). MATLAB for Neuroscientists: An Introduction to Scientific Computing in MATLAB. Academic Press.

#bp Thompson, D. W. (1992). Canto: On growth and form (J. T. Bonner, Ed.). Cambridge University Press.

#bp Ellenberg, J. (2022). Shape: The hidden geometry of information, biology, strategy, democracy, and everything else. Penguin.

#be