transforming-data

April 21, 2018

0.1 Transforming Data

Your goal during the data gathering phase is to record as much working data about your observations as possible since you never know which feature is going to end up being the golden one that allows your machine learning algorithm to succeed. Due to this, there are usually a few redundant or even poor features in your dataset. Think back to those long word problems in grade school that were essentially a simple math question, but came filled with red herrings to throw you off; feeding an unfiltered soup of features to your machine learning algorithms is pretty similar to trying to get it to solve those word problems.

To be effective, many machine learning algorithms need the data passed to them be discerning, discriminating and independent. In this module, you're going to discover methods to get your data behaving like that using transformers. This will help improve your own knowledge of your data, as well as improve your machine learning algorithms' performance.

A transformer is any algorithm you apply to your dataset that changes either the feature count or feature values, but does not alter the number of observations. You can use transformers to mung your data as a pre-processing step to clean it up before it's fed to other algorithms. Another popular transformer use is that of dimensionality reduction, where the number of features in your dataset is intelligently reduced to a subset of the original.

Once you've used a few basic transformers, you will also learn about some data cleansing techniques that attempt to rectify problematic observations.

0.1.1 Principal Component Analysis (PCA)

Unsupervised learning aims to discover some type of hidden structure within your data. Without a label or correct answer to test against, there is no metric for evaluating unsupervised learning algorithms. Principal Component Analysis (PCA), a transformation that attempts to convert your possibly correlated features into a set of linearly uncorrelated ones, is the first unsupervised learning algorithm you'll study.

What is principal component analysis **PCA falls into the group of dimensionality reduction algorithms**. *In many real-world datasets and the problems they represent, you aren't aware of what specifically needs to be measured to succinctly address the issue driving your data collection. So instead, you simply record any feature you can derive, usually resulting in a higher dimensionality than what is truly needed.* This is undesirable, but it's the only reliable way you know to insure you capture the relationship modeled in your data.

If you have reason to believe *your question has a simple answer*, or that *the features you've collected are actually many indirect observations of some inherent source you either cannot or do not know how to measure*, then **dimensionality reduction applies to your needs**.

telephone-front-view

telephone-above-view

PCA's approach to *dimensionality reduction* is to **derive a set of degrees of freedom that can then be used to reproduce most of the variability of your data**. Picture one of those cartoon style telephone poles; once you have a figure in mind, compare it to this one:

Your envisioned image probably looked similar. You could have pictured it from any other viewing angle, for instance, as if you were floating directly above it looking down:

However you probably didn't, since that view doesn't contain enough variance, or information to easily be discernible as a telephone pole. The frontal view, however, does. Looking at a telephone pole or any other object from various viewing angles gives you more information about that object. If the view angles are really close to one another, the information you get from the views ends up being mostly the same, with a lot of duplicate information. However if you're able to move to a completely different angle, you can get a lot more information about the object you're examining. And if you're wise in choose your view angles, with just a few calculated glimpses of an object, you can build a rather comprehensive understanding of it. PCA calculates those best view angles:

How Does PCA Work? **PCA is one of the most popular techniques for dimensionality reduction**, and *we recommend you always start with it when you have a complex dataset*. **It models a linear subspace of your data by capturing its greatest variability**. Stated differently, **it accesses your dataset's covariance structure directly using matrix calculations and eigenvectors to compute the best unique features that describe your samples**.

An iterative approach to this would **first find the center of your data**, *based off its numeric features*. Next, it would **search for the direction that has the most variance or widest spread of values**. *That direction is the principal component vector, so it is then added to a list*. By **searching for more directions of maximal variance that are orthogonal to all previously computed vectors**, *more principal component can then be added to the list*. **This set of vectors form a new feature space that you can represent your samples with**.

On Dimensions, Features, and Views Each sample in your dataset represents an observable phenomenon, such as an object in the real world. Each feature in your dataset tells you details about your samples. **Recall from earlier chapters that features and views are synonymous terms; this isn't accidental!** Just like looking at an object from different views gives you more information about the object, so too does examining a sample from different features. Similar or correlated features will produce an "overlapped" view of your samples, the same way similar views of an object also overlap.

PCA ensures that each newly computed view (feature) is orthogonal or linearly independent to all previously computed ones, minimizing these overlaps. PCA also orders the features by importance, assuming that the more variance expressed in a feature, the more important it is. In our telephone pole example, the frontal view had more variance than the bird's-eye view and so it was preferred by PCA.

With the ***newly computed features ordered by importance, dropping the least important features on the list intelligently reduces the number of dimensions needed to represent your dataset****, with minimal loss of information*. This has many practical uses, including boiling off high dimensionality observations to just a few key dimensions for visualization purposes, being used as a noise removal mechanism, and as a pre-processing step before sending your data through to other more processor-intensive algorithms. We'll look at more real life use cases in the next unit.

When should I use PCA? PCA, and in fact all dimesionality reduction methods, have three main uses: + To handle the clear goal of reducing the dimensionality and thus complexity of your dataset. + To pre-process your data preparation for other supervised learning tasks, suchs as regression and classification. + To make visualizing your data easier, since we can only preceive three dimensions simultaneously.

According to Nielson Tetrad Demographics, the group of people who watch the most movies are people between the ages of 24 through 35. Let's say you had a list of 100 movies and surveyed 5000 people from within this demographic, asking them to rate all the movies they've seen on a scale of 1-10. By having considerably more data samples (5000 people) than features (100 ordinal movie ratings), you're more likely to avoid the curse of dimensionality.

Having collected all that data, even though you asked 100 questions, what do you think truly is being measured by the survey? Overall, it is the collective movie preference per person. You could attempt to solve for this manually in a supervised way, by break down movies into well-known genres: + Action + Adventure + Comedy + Crime & Gangster + Drama + Historical + Horror + Musicals + Science Fiction + War + Western + etc.

Being unsupervised, PCA doesn't have access to these genre labels. In fact, it doesn't have or care for any labels whatsoever. This is important because it's entirely possible there wasn't a single western movie in your list of 1000 films, so it would be inappropriate and strange for PCA to derive a 'Western' principal component feature. By using PCA, rather than you creating categories manually, it discovers the natural categories that exist in your data. It can find as many of them as you tell it to, so long as that number is less than the original number of features you provided, and as long as you have enough samples to support it. The groups it finds are the principal components, and they are the best possible, linearly independent combination of features that you can use to describe your data.

Being unsupervised, PCA doesn't have access to these genre labels. In fact, it doesn't have or care for any labels whatsoever. This is important because it's entirely possible there wasn't a single western movie in your list of 1000 films, so it would be inappropriate and strange for PCA to derive a 'Western' principal component feature. By using PCA, rather than you creating categories

manually, it discovers the natural categories that exist in your data. It can find as many of them as you tell it to, so long as that number is less than the original number of features you provided, and as long as you have enough samples to support it. The groups it finds are the principal components, and they are the best possible, linearly independent combination of features that you can use to describe your data.

One warning is that again, being unsupervised, PCA can't tell you exactly know what the newly created components or features mean. If you're interested in how to interpret your principal components, we've included two sources in the dive deeper section to help out with that and highly recommend you explore them.

Once you've reduced your dataset's dimensionality using PCA to best describe its variance and linear structure, you can then transform your movie questionnaire dataset from its original [1000, 100] feature-space into the much more comfortable, principal component space, such as [1000, 10]. You can visualize your samples in this new space using an Andrew's plot, or scatter plot. And finally, you can base the rest of your analysis on your transformed features, rather than the original 100 feature dataset.

PCA is a very fast algorithm and helps you vaporizes redundant features, so when you have a high dimensionality dataset, start by running PCA on it and then visualizing it. This will better help you understand your data before continuing.

Projecting a Shadow By transforming your samples into the feature space created by discarding under-prioritized features, a lower dimensional representation of your data, also known as **shadow** or **projection** is formed. In the shadow, some information has been lost—it has fewer features after all. You can actually visualize how much information has been lost by taking each sample and moving it to the nearest spot on the projection feature space. In the following 2D dataset, the orange line represents the principal component direction, and the gray line represents the second principal component. The one that's going to get dropped:

By dropping the gray component above, the goal is to project the 2D points onto 1D space. Move the original 2D samples to their closest spot on the line:

Once you've projected all samples to their closest spot on the major principal component, a shadow, or lower dimensional representation has been formed:

The summed distances traveled by all moved samples is equal to the total information lost by the projection. An an ideal situation, this lost information should be dominated by highly redundant features and random noise.

SciKit-Learn and PCA To get started, import PCA from `sklearn.decomposition` and then create a new instance of the model setting the `n_components parameter` to the number of dimensions you wish to keep. This value has to be less than or equal to the number of features in your original dataset, since each computed component is a linear combination of your original features:

```
In [4]: import pandas as pd
        from sklearn.decomposition import PCA

        df = pd.read_csv("Datasets/students.data")

        pca = PCA(n_components=2)
        pca.fit(df)
        print (pca)
        T = pca.transform(df)

        print (df.shape) # (430, 6) - 430 Student survey responses, 6 questions..
        print (T.shape) # (430, 2) - 430 Student survey responses, 2 principal components
```

```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
(649, 29)
(649, 2)
```

Once you've fit the model against your dataframe, you can use it to transform your dataset's observations (or any other observation that share its feature space) into the newly computed, principal component feature space with the `.transform()` method. This transformation is bidirectional, so you can recover your original feature values using `.inverse_transform()` so long as you don't drop any components. If even one component was removed, then after performing the inverse transformation back to the regular feature space, there will be some signs of information loss proportional to which component was dropped.

There are a few other interesting model attribute that SciKit-Learn exposes to you after you've trained your PCA model with the `.fit()` method:

- `components_` These are your principal component vectors and are linear combinations of your original features. As such, they exist within the feature space of your original dataset.
- `explained_variance_` This is the calculated amount of variance which exists in the newly computed principal components.
- `explained_variance_ratio_` Normalized version of `explained_variance_` for when your interest is with probabilities.

PCA Gotchas! To use PCA effectively, you should be aware of its weaknesses. The first is that **PCA is sensitive to the scaling of your features**. PCA maximizes variability based off of variance (the average squared differences of your samples from the mean), and then projects your original data on these directions of maximal variance. If your has a feature with a large variance and others with small variances, PCA will load on the larger variance feature. Being a linear transformation, it will rotate and reorient your feature space so it diverts as much of the variance of the larger-variance feature (and some of the other features' variances), into the first few principal components.

When it does this, a feature might go from having little impact on your transformation to totally dominating the first principal component, or vice versa. **Standardizing your variables ahead of time is the way to free your PCA results of such scaling issues**. *The cases where you should not use standardization are when you know your feature variables, and thus their importance, need to respect the specific scaling you've set up*. In such a case, PCA would be used on your raw data.

PCA is extremely fast, but for very large datasets it might take a while to train. All of the matrix computations required for inversion, eigenvalue decomposition, etc. boggle it down. Since most real-world datasets are typically very large and will have a level of noise in them, razor-sharp, machine-precision matrix operations aren't always necessary. If you're willing to sacrifice a bit of accuracy for computational efficiency, *SciKit-Learn offers a sister algorithm called* ***RandomizedPCA*** *that applies some approximation techniques to speed up large-scale matrix computation*. You can use this for your larger datasets.

The last issue to keep in mind is that PCA is a linear transformation only! In graphical terms, it can rotate and translate the feature space of your samples, but will not skew them. *PCA will only, therefore, be able to capture the underlying linear shapes and variance within your data and cannot discern any complex, nonlinear intricacies. For such cases, you will have to make use different dimensionality reduction algorithms, such as* ***Isomap***.

Knowledge checks

Review Question 1 1/1 point (graded)
Please complete the sentence so that it makes the most sense:
Principal component analysis…

- Requires you have labeled features to use as a metric for determining which features are most important
- Is a dimensionality reduction technique that builds a simpler, non-linear projection or 'shadow' of your dataset
- Asserts you have more features than samples so you can avoid the curse of dimensionality and the matrix math works out
- Ensures each newly computed feature is orthogonal to all previously computed ones, minimizing overlaps

Answer: **D** ##### Review Question 2 1 point possible (graded)
Which of these statements is wrong?

- PCA can be used to discover the underlying features being assessed by a dataset
- The results of PCA depend on the scaling of your data, so having a feature with units of 'light-years' and another feature with units of 'GHz' may be disastrous
- When applied to non-linear data, PCA generally isn't as effective as when applied to linear data
- Since PCA is sensitive to feature scaling, if you have a feature that is a linear transformation of the other, e.g. feature2 = 10 * feature1, then both features will be ignored

Answer: **D**

0.1.2 PCA Lab

Assignment 1

Lab Assignement 1 In this assignment, you're going to experiment with a real life armadillo sculpture scanned using a Cyberware 3030 MS 3D scanner at Stanford University. The sculpture is available as part of their 3D Scanning Repository, and is a very dense 3D mesh consisting of 172974 vertices! The mesh is available for you, located at /Module4/Datasets/**stanford_armadillo.ply**. It is *not* a Python file, so don't attempt to load it with a text editor!

Open up the Module4/**assignment1.py** starter code and read through it carefully. You will notice the use of a new library, Plyfile. This library loads up the 3D binary mesh for you. The mesh is further converted into a Pandas dataframe for your ease of manipulation. Complete the following tasks:

1. Before changing any of the code, go ahead and execute assignment1.py. You should see the 3D armadillo. Your goal is to reduce its dimensionality from three to two using PCA to cast a shadow of the data onto its two most important principal components. Then render the resulting 2D scatter plot.
2. Fill out the proper code in the `do_PCA()` and `do_RandomizedPCA()` methods. Be sure to **return** the result of your transformation! You may even want to read the SciKit-Learn documentation on `.transform()`, just for future reference so you know what data type comes out of it.
3. Re-run the application! Then, answer the questions below:

Lab Question 1 1 point possible (graded)
The first time you see the armadillo in 3D, what direction was its face pointing towards?

- Left, Towards the negative X-Axis
- Up, Towards the positive Z-Axis
- Right, Towards the positive X-Axis
- Down, Towards the negative Z-Axis

Answer: **A** ##### Lab Question 2 2 point possible (graded)
Were you able to discern any **visual** differences between the transformed PCA results and the transformed RandomizedPCA results?

- Yes, the RandomizedPCA version was no longer even recognizable as an armadillo
- Yes, the RandomizedPCA version was a lot less true to the original than the regular PCA version
- Yes, but it wasn't a lot… just minor differences
- No, they pretty much looked the same to me

Which executed faster, RandomizedPCA or PCA?

- PCA
- RandomizedPCA

Answer: **RandomizedPCA**

Lab Assignement 2 In Lab Assignment 1, you applied PCA to a dataset generated by 3D-scanning an actual sculpture. Real life 3D objects are a good segue to PCA, since it's **fun** to see its effects on a dataset we can see and touch. Another benefit is that all three spatial dimensions, **x**, **y**, and **z**, each measure the same unit-length relative to one another, so no extra consideration need be made to account for PCA's weakness of requiring feature scaling.

But now the **fun** is over. Gaining some practical experience with real-world datasets, which rarely allot you the luxury of having features all on the same scale, will help you see how critical feature scaling is to PCA. In this lab, you're going to experiment with a subset of UCI's Chronic Kidney Disease data set, a collection of samples taken from patients in India over a two month period, some of whom were in the early stages of the disease. The starter code over at /Module4/**assignment2.py**.

1. Start by looking through the attribute information on the dataset website. Whenever you're given a dataset, the first thing you should do is find out as much about it as possible, both by reading up on any metadata, as well as by prodding through the actual data. Particularly, pay attention to what the docs say about these three variables: **bgr**, **rc**, and **wc**.
2. Load up the `kidney_disease.csv` dataset from the /Module4/Datasets/ directory, and drop **all rows** that have *any* nans. You're probably already a pro at doing that by now. In addition to getting rid of nans, did you know that the `.dropna()` method (upon completion) also automatically re-checks your features and assigns them an appropriate inferred data types?
3. Use an appropriate indexer command to select only the following columns: **bgr**, **rc**, and **wc**. Or alternatively, you can drop every other column, but it's probably easier to just use an indexer to select the one's you wish to keep.
4. Do a check of your dataframe's dtypes. Anything that didn't make it to the right type, you may want to investigate. Look through the data and identify *why* the conversion failed. These types of problems often arise when you aren't in control of how your data is organized. Luckily the issue isn't too bad so once you've identified it, you can fix it through simple numeric coercion.
5. Print the variance of your dataset, as well as a `.describe()` printout.
6. Reduce your dataset to two principal components by run it through PCA, then check out the resulting visualization.

Lab Question 1 1 point possible (graded)
The first time you see the armadillo in 3D, what direction was its face pointing towards?

12. Serum Creatinine (numerical) sc in mgs/dl
13. Serum (numerical) sod in mEg/L
14. Potassium (numerical) pot in mEg/L
15. Hemoglobin (numerical) hemo in gms

Having reviewed the dataset metadata on its website, what are the units of the **wc**, White Blood Cell Count feature? An example of where units are defined is shown above. NOTE: In case the UCI site is down, here is a mirror.

- mm/Hg
- mgs/dl
- mEq/L
- cells/cumm
- gms

Answer: ** **

Lab Question 2 2 point possible (graded)
Why did the `.dropna()` method fail to convert all of the columns to an appropriate numeric format?

- It actually did successfully convert them

- There were a few erroneous leading tab / whitespace characters

- There were a few erroneous trailing tab / whitespace characters

- The dataset comma offset was incorrect in a few rows causing nans to move into the next column
- Yes, the RandomizedPCA version was no longer even recognizable as an armadillo
- Yes, the RandomizedPCA version was a lot less true to the original than the regular PCA version
- Yes, but it wasn't a lot… just minor differences
- No, they pretty much looked the same to me

Sort the features below from the largest to the smallest variance amount.

- bgr, rc, wc
- bgr, wc, rc
- rc, wc, bgr
- rc, bgr, wc
- wc, bgr, rc
- wc, rc, bgr

Answer: ** **

You're almost there! The last thing you have to do, and the purpose of this lab really, is to see how feature scaling alters your PCA results.

1. Make a backup of your **assignment2.py** file for safe keeping.
2. Change the line that reads: `python scaleFeatures = False` So that is now reads: `python scaleFeatures = False`.
3. Also take a look inside of **assignment2_helper.py**. There are some *important* notes in there about what SKLearn's *star transform()* methods do, and why they do it. You will need to know this information for future labs!
4. Re-run your assignment and then answer the questions below:

Answer: ** **

Lab Questions (Continued) 2 points possible (graded)
Did scaling your features affect their variances at all? + Yes + No
Answer: ** **
After scaling your features, are the green patients without chronic kidney disease more cleanly separable from the red patients with chronic kidney disease?

- They are less separable
- There isn't much change
- They are more separable

Answer: ** **

0.1.3 Isomap

Similar to PCA, **Isomap is also an unsupervised learning technique that reduces the dimensionality of your dataset. No labels or classifications are needed to guide it except your raw data**. You can literally apply Isomap to any dataset during your exploratory analysis. In fact *you should transform your high dimensionality datasets with Isomap when they do not exhibit the behavior you want, having been passed through PCA*.

PCA is faster than Isomap and works well in most situations, but its limitation is that it assumes a linear relationship exist between your features. What happens when your data has a non-linear structure? Take for instance a set of images produced by photographing the rotation one of those fancy, $1500 office chairs about its Y-axis. From a stationary camera's perspective, the rotation is a non-linear transformation, not well suited at all for PCA:

Authman's Commando Chair: Non-linear, and linear transformation examples...

In reality though, just by spinning the chair, you're really only altering a single degree of freedom. Dimensionality reduction aims to derive a set of degrees of freedom that can then be used to reproduce a lower dimensional embedding of your data, so a non-linear reduction algorithm should be able to recognize that within this image space, the sequence of pictures lie along a single-dimensional, continuous curve.

On the other hand, if the camera were translated directly up and down rather than rotating the chair itself, then that would be an example of a linear transformation from the camera's perspective, and PCA and other linear reduction techniques would be better suited for reducing the dimensionality of the resulting dataset:

Authman's Commando Chair: linear transformation examples (the camera were translated directly up and down rather than rotating)

In order to address non-linear situations, **Isomap uses an entirely different approach to the dimensionality reduction problem**. One that is highly efficient, albeit more processor intensive than PCA. Nonetheless, for non-linear relationships it is a must. **Its goal: to uncover the intrinsic, geometric-nature of your dataset, as opposed to simply capturing your datasets most variant directions**.

How Does Isomap Work? Isomap operates by first computing each record's nearest neighbors. This is done by comparing each sample to every other sample in the dataset. Only a sample's K-nearest samples qualify for being included in its nearest-neighborhood samples list. A neighborhood graph is then constructed by linking each sample to its K-nearest neighbors. The result is similar to map of roads that is traversed in order to move from point to point.

A comparison by Josh Tenenbaum of the direct Euclidean vs manifold neighborhood path distances of the classic 'swiss-roll', as appears in Science 290 (5500), December 2000.

You've already encountered one form of multi-dimensional scaling. Do you recall what it was? It was actually none other than PCA itself! Multi dimensional scaling is a process of taking samples in N-dimensional space, and representing them on some other M-dimensional space, while attempting to preserve the inter-sample distances as much as possible. If you're going from a lower to higher dimensionality, then the distances can be preserved perfectly. But when you reduce dimensions, as you've seen some information is lost.

Fun fact, the actual distance formula used to calculate the K-nearest samples doesn't even have to be 100% precise. As long as it works reasonably well for 'close by' samples. The truth of the matter is that with Isomap, samples far away from one another aren't included in each other's neighborhood map, and therefore do not affect the final simplified manifold produced by multi dimensional scaling. Due to this, some very nice speed optimizations can be enacted with the distance calculations.

When Should I Use Isomap? Isomap has so many real-world use cases. The most straightforward of which being whenever you believe a simpler surface can describe and preserve the inherent structure of your higher dimensionality dataset, and want to retrieve that simpler embedding.

Isomap is better than linear methods when dealing with almost all types of real image and motion tracking. Recalling the expensive office chair example from earlier, either rotating the chair, having the camera zoom into the chair, or panning around the chair, etc. are all examples of very natural, non-linear motions we undertake often. If you're dataset comes from images captured in a natural way, or your data is characterized by similar types of motions, consider using isomap. Simply translating the camera up and down, or striding it left and right, are linear motions that aren't as common unless you're doing this kind of movement.

Isomap can also be better than linear methods at estimating a more accurate distance metric between different observations of your samples. Consider the following image sequence of an air dancer being deformed in the wind:

Even trained against the full set of images (plus any intermediary images), a linear model would calculate the distance between the first and last pictures using a straight-line, high dimensionality, Euclidean metric. This of course doesn't take into account the manifold's geometry. Isomap would do a better job by traversing the nearest neighborhood map:

Another example of this would be trying to get from one spot on a sphere to another; the Euclidean distance would be the shortest path connecting the two locations, namely, a straight line. The manifold path on the other hand, would do a much better job sticking to the surface of the sphere, based on how far apart the neighborhood sampling is. Note: In the above graph, the orange isomap plot as been interpolated; a more accurate representation would be polyline segments connecting the four points in order.

So long as the underlying relationship is non-linear, another area isomap tends excel at is the grouping and identifying of similar variations in similar data samples. Due to this, it is extremely useful as a preprocessor step before conducting supervised learning tasks, such as classification or regression. Even the official example of isomap in SciKit-Learn's documentation is of it being applied to group similar variations of handwritten digits, and outperforming PCA while doing so!

73 Pepper Variations From Dr. Baumler's Garden, Courtesy Denise Thornton

Finally, isomap's benefits also include all the other reasons you would use PCA or any other dimensionality reduction technique, including visualization and data compression.

SciKit-Learn and Isomap Isomap is a method of SciKit-Learn's manifold module:

```
In [2]: from sklearn import manifold

In [3]: iso = manifold.Isomap(n_neighbors=4, n_components=2)

In [6]: iso.fit(df)

Out[6]: Isomap(eigen_solver='auto', max_iter=None, n_components=2, n_jobs=1,
            n_neighbors=4, neighbors_algorithm='auto', path_method='auto', tol=0)
```

As with PCA, **n_components** is the number of features you want your dataset projected onto, and n_neighbors defines the neighborhood size used to create the node neighborhood map. Be sure to experiment with **n_neighbors** particularly, as your neighborhood size will directly effect how the manifold mapping is generated:

The *larger* your **n_neighbors** value is, the *longer* it will take to calculate the node neighborhood map. In the above animation, the neighborhood sizes in order are: 2, 3, 4, 5, 6, 7, 8, 16, 32, 64. Notice that at 64 connectivity, the manifold almost looks similar to PCA. You will have to experiment with the neighborhood connectivity in order to get the best results.

That said, in addition to considering how many neighbors each sample should have, you also need to reflect on how many samples realistically even need to be collected in order for you to properly capture your lower dimensional manifold! A rule of thumb is the curvier your dataset is, and by curvier we mean the sharpness of the edges, the more dense your samples must be in order to capture its latent relationships:

With your Isomap instance created, you can calculate your new feature set the same way you did with PCA using `fit()`. Fitting the data just calculates the basis of the lower dimensional encoding. To actually project your data into that space, transforming it by calling:

```
In [12]: manifold = iso.transform(df)
         print('df shape: ', df.shape)
         print('manifold shape: ', manifold.shape)

df shape:  (649, 29)
manifold shape:  (649, 2)
```

Unlike PCA, Isomap transformations are unidirectional so you will not be able to `.inverse_transform()`your projected data back into your original feature space, even if it has the same number of dimensions as your original dataset.

Running Isomap is a lot slower than PCA since a lot more is happening under the hood, particularly **for large n_neighbors values**, but it provides a simple way to analyze and manipulate high dimensional samples in terms of its intrinsic nonlinear degrees of freedom. This is because **isomap attempts to keep the global structure of your data as it reduces its dimensionality**. You can use it in any case where nonlinear geometry degrades the effectiveness of PCA.

Isomap is also a bit more sensitive to noise than PCA. Noisy data can actually act as a conduit to short-circuit the nearest neighborhood map, cause isomap to prefer the 'noisy' but shorter path between samples that lie on the real geodesic surface of your data that would otherwise be well separated.

When using unsupervised dimensionality reduction techniques, be sure to use the feature scaling on all of your features because the nearest-neighbor search that Isomap bases your manifold on will do poorly if you don't, and PCA will prefer features with larger variances. As explained within the labs' source code, SciKit-Learn's **StandardScaler** is a good-fit for taking care of scaling your data before performing dimensionality reduction.

Knowledge Checks

Review Question 1 1/1 point (graded)
Which of the following explanations of isomap is true?

- Isomap can be used as a powerful noise removal tool, since a smooth manifold is created by "short circuiting" the nearest neighbor map when calculating distances
- Isomap is usually faster than PCA because it's quicker to compute a nearest neighbor map than to do matrix decomposition
- **A one sentence summary of isomap's implementation is that at its core, it is essentially a node distance map that has been fed into a special type of PCA correct**
- Isomap will not function without a completely accurate distance metric, since it needs to know the precise distance to every single sample, including distant ones

Review Question 2 1/1 point (graded)
Isomap is most beneficial. . .

- When your data lacks an inherent manifold
- **When a non-linear, geometric structure is expressed in your data correct**
- When you are uncertain how many samples are needed to capture the underlying nature of your data
- When your high dimensionality data has a hidden, linear relationship expressed within it

Assignment 4

Lab Assignment 4 After having a brief conversation with Joshua Tenenbaum, the primary creator of the isometric feature mapping algorithm, it only seems right that we make your first lab assignment be replicating his canonical, dimensionality reduction research experiment for visual perception! In fact, you will also be using his original dataset (cached website]) from December 2000. It consists of 698 samples of 4096-dimensional vectors. These vectors are the coded brightness values of 64x64-pixel heads that have been rendered facing various directions and lighted from many angles. Replicate Dr. Tenenbaum's experiment by:

1. Applying both PCA and Isomap to the 698 raw images to derive 2D principal components and a 2D embedding of the data's intrinsic geometric structure.
2. Project both onto a 2D scatter plot, with a few superimposed face images on the associated samples.

3. Extra: If you're feeling fancy, increase n_components to three, and plot your scatter plot on a 3D chart.

NOTE: If you encounter issues with loading .mat files using SciPy, you might want to see this Stack Overflow post and check the version of SciPy you're using.

Lab Questions 4 points possible (graded) (option in bold text is the answer)
Between linear PCA and the non-linear Isomap, which algorithm is better able to capture the true nature of the faces dataset when reduced to two component?

- PCA
- **IsoMap**

Each coordinate axis of your 3D manifold should correlate highly with one degree of freedom from the original, underlying data. In the isomap plot of the first two components (0 and 1), which '**degree of freedom**' do you think was encoded onto first component (the X-axis) encoded? In other words, what varies as you move horizontally in your manifold rendering?

- **Left and Right Head Position**
- Down and Up Head Position
- Lighting Position
- Something else

Alter your code to graph the second and third components (index=1 and 2) instead of the 0th and 1st, for both PCA and Isomap. Look *closely* at the Isomap plot. Can you tell what 'degree of freedom' the X axis represents?

- Left and Right Head Position
- **Down and Up Head Position**
- Lighting Position
- Something else

In his experiment, Dr. Tenenbaum set his K-parameter (n_neighbors is SciKit-Learn) to 8. Try reducing that figure down to 3 and re-running your code. Does the X-Axis still represent the same degree of freedom?

- **Yes**
- No

0.1.4 Assignment 5

Lab Assigment 5 Now that you've had your first taste of isomap, let's take your knowledge of it to the next level.

Whatever your high-dimensional samples are, be they images, sound files, or thoughtfully collected attributes, they can all be considered single points in a high dimensional feature-space. Each one of your observations is just a single point. Even with a high dimensionality, it's possible that most or all your samples actually lie on a lower dimension surface. Isomap aims to capture that embedding, which is essentially the motion in the underlying, non-linear degrees of freedom.

By testing isomap on a carefully constructed dataset, you will be able to visually confirm its effectiveness, and gain a deeper understanding of how and why each parameter acts the way it does. The **ALOI**, Amsterdam Library of Object Images, hosts a huge collection of 1000 small objects that were photographed in such a controlled environment, by systematically varying the viewing angle, illumination angle, and illumination color for each object separately. To really drive home how well isomap does what it claims, this lab will make use of two image sets taken from the ALOI's collection.

Manifold extraction, and isomap specifically are really good with vision recognition problems, speech problems, and many other real-world tasks, such as identifying similar objects, or objects that have undergone some change. In the case of the 3D rotating object such as the office chair example from earlier, if every pixel is a feature, at the end of the day, the manifold surface is parametrizable by *just* the angle of the chair—a single feature!

1. Start by having a look through the Module4/Datasets/ALOI/ directory. There are two directories filled with 192 x 144 pixel images. Identify their ordering and try to figure out what's changing between the images. They might not be perfectly ordered, but that doesn't matter to isomap.

2. Create a regular Python list object. Then, write a for-loop that iterates over the images in the Module4/Datasets/ALOI/32/ folder, appending each of them to your list. Each .PNG image should first be loaded into a temporary NDArray, just as shown in the Feature Representation reading.

   Optional: Resample your images down by a factor of two if you have a slower computer. You can also convert the image from `0-255` to `0.0-1.0` if you'd like, but that will have no effect on the algorithm's results.

3. Convert the list to a dataframe and run isomap on it to compute the lower dimensional embedding. Be sure to set n_components to 3 so you can visualize your manifold. You can also set the neighborhood size to six.

4. Plot the first two manifold components using a 2D scatter plot, then plot the first three components using a 3D scatter plot. Run your assignment and then answer the questions below.

Lab Questions 2 points possible (graded)

Please describe the results of your isomap embedding–either the 3D or 2D one, it doesn't matter:

- It is completely sporadic. It's hard to detect the pattern at this point, since we discarded too many dimensions
- **The embedding appears to follow an easily traversable, 3D spline correct**
- It looks like a geometric pattern, but we probably need to increase the neighborhood resolution before a discernible shape emerges incorrect
- Isomap rendered the result as a straight line, as expected, since only a single degree of freedom is altered in the images

Explanation

Encode each image as a single sample. Since SciPy loads images into an NDArray, rather than converting the individual NDArrays to Pandas DataFrames and concatenating DataFrames, it makes more sense to just append all the NDArrays to a list and mass convert the thing in one go.

When rendering the manifold, you should see a smooth curve. In 2D, it should look like a near perfect circle (check your window axes scale, if your X-Axis is stretched). In 3D, it should look like the circle as well, but with the Y-axis values oscillating up and down as if bound to a sine curve:

Try reducing the 'n_neighbors' parameter one value at a time. Keep re-running your assignment until the results look visible different. What is the smallest neighborhood size you can have, while maintaining similar manifold embedding results?

- 1
- **2**
- 3
- 4
- 5
- 6

Explanation The correct number is two. The dataset was generated carefully enough that there really shouldn't be any cause for noise in the path, unless you introduced it.

Lab Questions (Continued) Almost done! Two more steps to complete this lab:

1. Once you're done answering the first three questions, right before you converted your list to a dataframe, add in additional code which also appends to your list the images in the Module4/Datasets/ALOI/32_i directory.
2. Create a colors Python list. Store a 'b' in it for each element you load from the /32/ directory, and an 'r' for each element you load from the '32_i' directory. Then pass this variable to your 2D and 3D scatter plots, as an optional parameter c=colors. Re-run your assignment and answer the final question below.

Reset your 'n_neighbors' if you changed it from 6. After adding in the additional images from the 32_i dataset, do examine your 2D and 3D scatter plots again. Have the new samples altered the shape of your original (blue) manifold?

- No, not in the slightest
- **Only very slightly… correct**
- It looks like a completely different shape

What is the arrangement of the newly added, red samples?

- **Isomap rendered the result as a straight line, intersecting the original manifold**
- They are completely sporadic compared to the original manifold, so no real pattern exist incorrect
- They are aligned in a smaller circle, intersecting the original manifold

0.1.5 Data Cleaning

In *data wrangling*, irrelevant, incomplete, and missing data is either defaulted to a specific value or removed entirely. NaNs are stripped out, typographical errors are patched, and perhaps even some data normalization occurs. The goal of *data cleansing* is to *take wrangling a step further by rectifying inaccurate and inconsistent data to standardize it*. Inconsistent data can lead to false intelligence being produced by your machine learning algorithms, or no intelligence at all.

Simple data cleansing tasks might be automated and applied out of the box. More occupation specific tasks require you fully understand the working environment that generated your raw data. Knowledge of the range of values you expect to see for a particular feature will help you find any anomalies that need attention.

A classical example of when cleansing is necessary is when data comes from multiple sources. If, on average, a specific source consistently reports figures offset from others, identifying the source of the error, be it a faulty sensor, or bad reporting, etc., and then making calculated adjustments is a way to improve your overall data accuracy. *But without carefully balancing keeping your data as close as possible to its raw from and making these error corrections, you might get accused of cooking your data*. After all, **it's always possible that there is no error at all**.

0.1.6 Case study

Climate change is a hotly debated topic. Climatologist claim the world is at a teetering point, and the damages will soon be irreversible if we don't make swift changes. Opponents claim there is no solid proof that demonstrates climate change is real, and that the data behind it is *cooked*, or falsified. Is there any truth to their claim?

The **NOAA**, National Oceanic and Atmospheric Administration, has long produced the only spatially complete, long-term (1895-2013) dataset for climate analyses within the US. In the Climate Divisional Dataset, each state is represented by 6-10 climate divisions, as shown on the map below. The **monthly** temperature readings per climate division are then calculated by averaging the **daily** observations from their ground stations. By using climate divisions and not individual weather stations, and by using monthly divisional averages for data-samples, the overall data collection process was eased considerably. After all, we are talking about government agency that formed over 200 years ago! Another added benefit is that less concern need be placed on the accuracy of specific station measurements, in case of local inconsistencies. Everything should just get *smoothed over* by averaging out.

In 2012, the NOAA must have come into better funding because they exerted an open effort to recalculate their historical climate dataset. They added in thousands of past observations that had only recently been digitized (remember when we used to use paper and pencil?), and adjusted their interpolation formula to remove some *known biases*, such as jumps introduced by replacing instruments, observation practices such as differences in readings based on when in the day temperatures are recorded, inaccuracies caused by station moves, urbanization around the station, etc. The NOAA further went on to retrospectively adjust their historic dataset as well, to be more in line with the current observations at each station, and to make use of the new interpolation method.

Skeptics were quick to criticize the NOAA's alterations to the historic dataset, claiming they've manipulated temperature records to create a warming trend by literally *cooling* the past and *warming* the present. It seemed that early decades of temperature records were consistently adjusted downward, and the current, century-long temperature trend was higher than in the original dataset. Even some of the historic 'hottest days ever' lost their titles.

Part of the data science umbrella of topics include the use of the scientific method. When experimenting, a control is usually introduced to reduce variations in the data due to anything except the features being measured. Knowing how to apply and what type of control to use typically requires human intervention, as we haven't found a way to get machines to learn to automatically account for that just yet. There are different types of controls, for instance negative control or randomization, which are used depending on the type of experiment being conducted and the data being gathered. Unfortunately, we only have one planet to test with and can't go backwards in time. Retrospective efforts like the NOAA's to polish up their data, will likely continue to be met with some level skepticism—at least until it gets a lot warmer.

Knowledge Checks

Review Question 1/1 point (graded)
Given the following options:

- If you encounter errors in your data, don't let anyone know and try to hide them by deleting the affected rows, or by cooking your data.
- While gathering data, identify issues that might cause inconsistencies, and capture additional features that'll help you rectify them.
- cRetrospectively adjust your data to account for discovered problems.

Order these options from most desirable to least desirable:

- A, B, C
- A, C, B
- B, A, C
- **B, C, A**
- C, A, B
- C, B, A

Further Reading

Dive Deeper Congratulations on making it this far! In the previous module, you directly explored your data using many visualizations. This time you learned how to take complex datasets and simplify them using two popular methods: keeping the most variant set of orthogonal components, and manifold multi-dimensional scaling of your sample's distance map.

After that you learned about a way errors can creep into your dataset and potential methods of handling that. Keep all these techniques fresh in your toolbox by being sure you record some notes about them in the transforming section of your course map. Remember, there is no hard order you must stick to while applying these methods. Try something out, visualize your data, then continue experimenting until you get your desired results.

Below, as usually, we've included some added details about the techniques you just studied in case you're interested in further broadening your knowledge, and taking it to the next level! For instance, regarding the isomap lab, you learned what each parameter does and how isomap works; but one thing you might be wondering from our examples is how interpolation between video frames is performed using isomap. You know it's impossible to .inverse_transform() an isomap manifold, so how in the world might you go about interpolating data in your original feature space / dataset? Check below!

PCA

- Interpreting PCA
- Another Method for Interpreting PCA
- Interactive PCA Demo
- PCA on Binary Data
- The Best Explanation of the Math Behind PCA You'll Ever Read
- RandomizedPCA
- Correlation or Covariance?

Isomap

- The Historic Stanford Isomap Paper, By Josh Tenenbaum
- Lower Dimensional Embedding
- Manifold Learning
- Interpolating Images Between Video Frames Using Non-Linear Dimensionality Reduction
- Cambridge Hand Gesture Data set

Climate Change

- NOAA U.S. Climate Divisional Dataset
- Interactive Discovery Tool For Comparing Raw Historical Data With the 'Enhanced' Data | Flash Based
- False Equivalence | Logic
- Control Types