**Locally Linear Embedding (LLE)**

As you can see, Locally Linear Embedding (LLE) sits under the **Unsupervised** branch of Machine Learning within the group of **dimensionality reduction** algorithms.

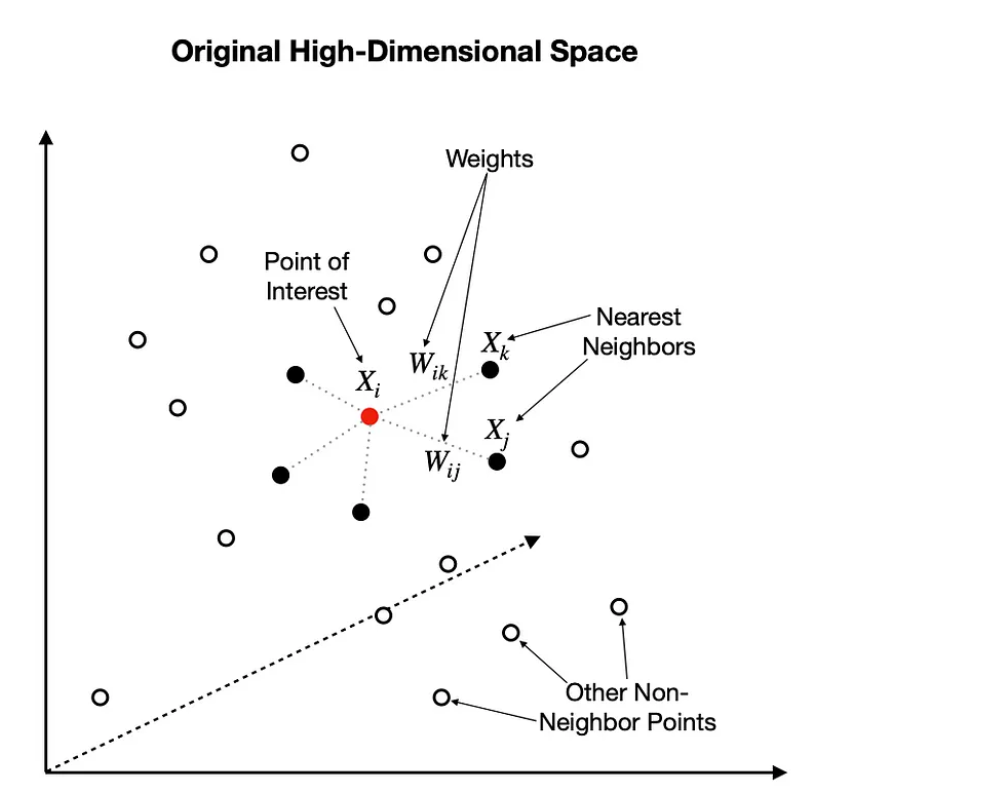
# How does Locally Linear Embedding work?

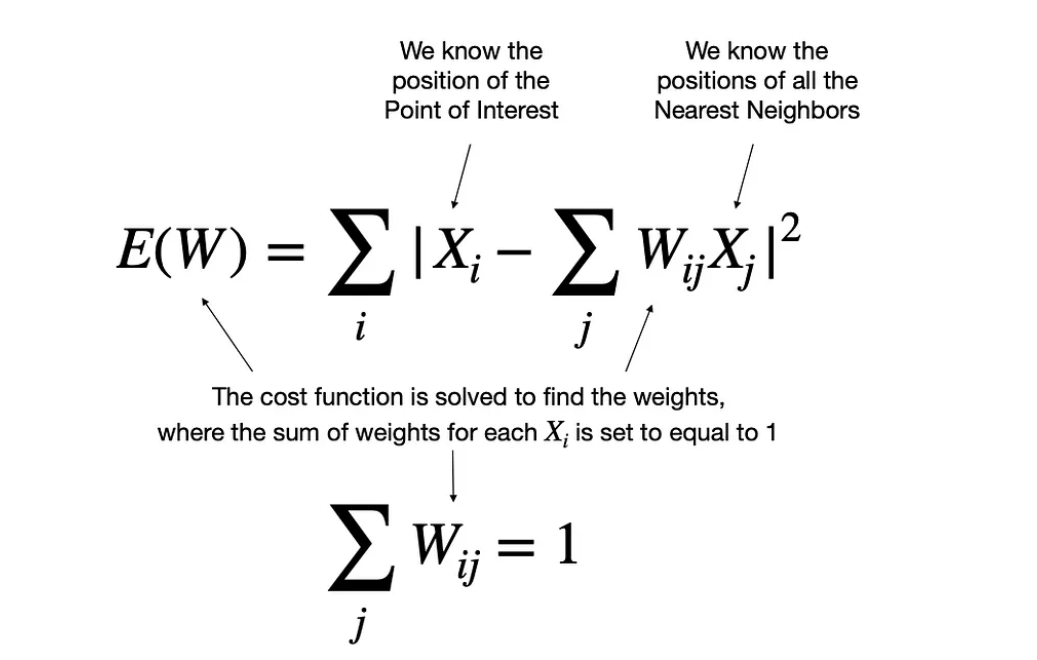
## The high-level steps

Similar to Isomap, LLE combines several steps to produce the lower-dimensional embedding. These are:

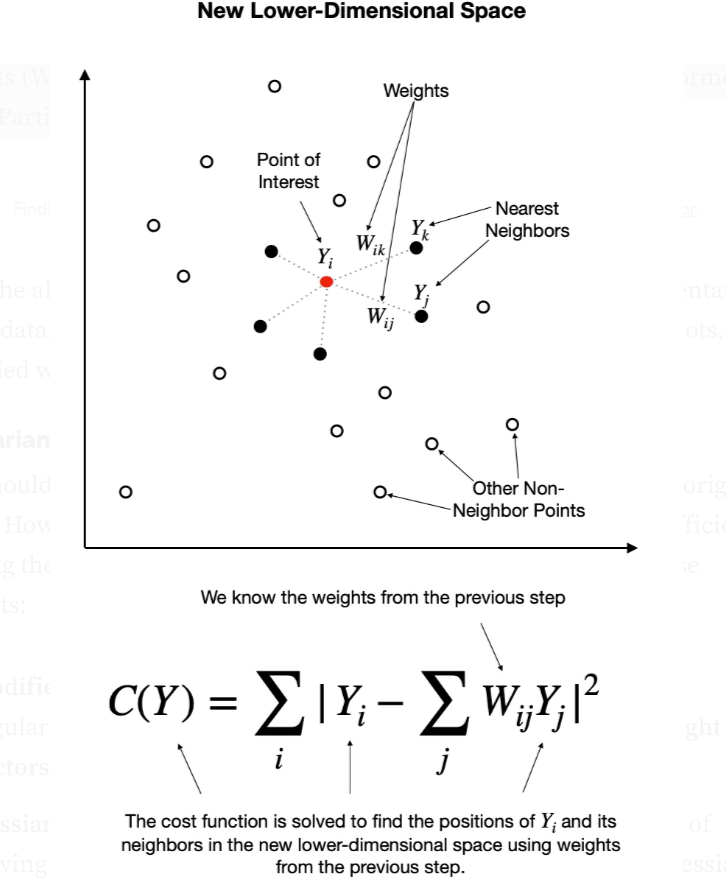
1. Use a KNN approach to **find the k nearest neighbors** of every data point. Here, “k” is an arbitrary number of neighbors that you can specify within model hyperparameters.

2. **Construct a weight matrix** where every point has its weights determined by minimizing the error of the cost function shown below. Note that every point is a linear combination of its neighbors, which means that **weights for non-neighbors are 0**.





 3- **Find the positions** of all the points in the **new lower-dimensional embedding** by minimizing the cost function shown below. Note, here we use weights (W) from step two and solve for Y. The actual solving is performed using Partial Eigenvalue Decomposition.



With the above steps completed, we get a lower-dimensional representation of the data, which we can typically visualize using standard scatterplots, provided we reduce the dimensionality to 3D or less.

## LLE variants

You should be aware of a few LLE variants, which improve upon the original setup. However, note that these improvements come at the cost of efficiency, making the algorithm slower. Here is how [scikit-learn](https://scikit-learn.org/stable/modules/manifold.html#modified-locally-linear-embedding) describes these variants:

* **Modified LLE (MLLE)**— One well-known issue with LLE is the regularization problem. A way to address it is to use **multiple weight vectors** in each neighborhood. This is the essence of MLLE.
* **Hessian LLE (HLLE)**— Hessian Eigenmapping is another method of solving the regularization problem of LLE. It revolves around a **hessian-based quadratic form** at each neighborhood used to recover the locally linear structure.

While I will not go into details, I recommend you experiment with them to see which variant yields the best results for your data. Personally, I find MLLE to perform well in most scenarios (see an example of this in the next section).

## Difference between LLE and Isomap

The two algorithms are similar in the way they approach dimensionality reduction, but they do have their differences.

Similar to LLE, Isomap also uses KNN to find the nearest neighbors in the first step. However, the second step constructs neighborhood graphs instead of describing each point as a linear combination of its neighbors. Then it uses these graphs to compute the shortest path between every pair of points.

Finally, Isomap uses those pairwise distances between all points to construct a lower-dimensional embedding.

## Should I choose LLE over Isomap?

In general, LLE is a more efficient algorithm as it eliminates the need to estimate pairwise distances between widely separated data points. Furthermore, it assumes that the manifold is linear when viewed locally. Thus it recovers the non-linear structure from locally linear fits.

However, because LLE focuses on preserving only the local structures, it may introduce some unexpected distortions on the global scale.

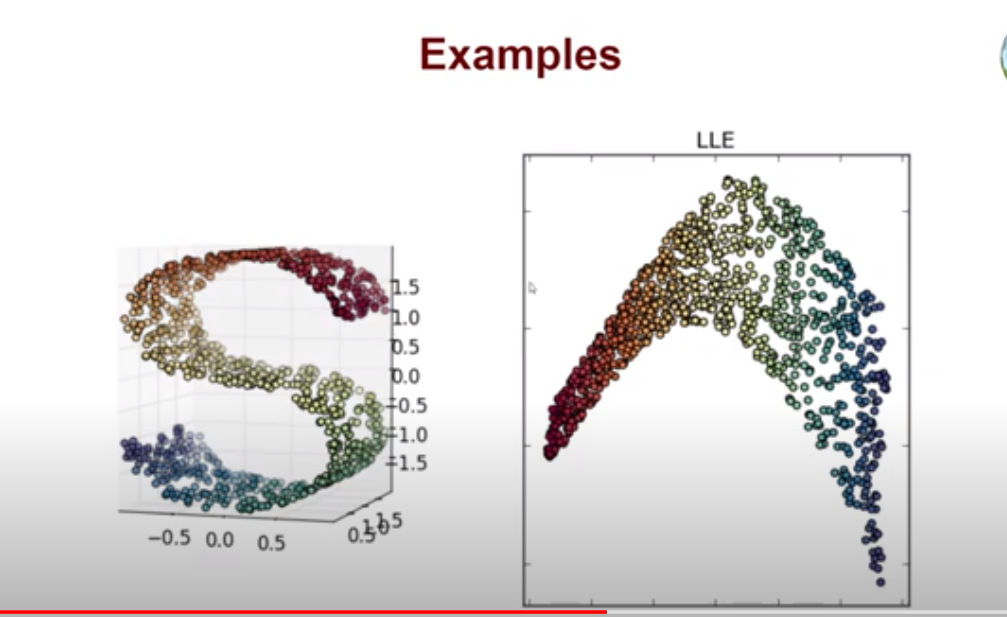
It tries to reduce these n-Dimensions while trying to preserve the geometric features of the original non-linear feature structure.

* **Hessian Locally-Linear Embedding (Hessian LLE)** – it is just like a basic LLE based on the solving sparse matrix technique and tends to give much higher quality results than the basic LLE. it has no internal models.
* **Modified Locally-Linear Embedding(MLLE)**– Modified LLE (MLLE) is another LLE variant in the local weight matrix addressed by multiple weights of each neighbourhood. This process leads to distortions in LLE maps. More formally we can say that if the generated weights from the basic LLE are orthogonally projected we can consider them as multiple weights.
* **Local Tangent Space Alignment(LTSA)** – Local Tangent Space Alignment can also be considered as the variant of LLE because it is pretty similar to LLE when we talk about the algorithm of LTSA. The only difference between LLE and LTSA is that LLE focuses on preserving the neighbourhood distance where LTSA characterizes the local geometry of each neighbourhood.

**In sklearn:**

"Standard LLE": LocallyLinearEmbedding(         n\_neighbors=n\_neighbors, n\_components=2, method="standard"     ),     "Modified LLE": LocallyLinearEmbedding(         n\_neighbors=n\_neighbors, n\_components=2, method="modified"     ),     "Hessian LLE": LocallyLinearEmbedding(         n\_neighbors=n\_neighbors, n\_components=2, method="hessian"     ),     "LTSA LLE": LocallyLinearEmbedding(         n\_neighbors=n\_neighbors, n\_components=2, method="ltsa"     ), }

* **The best method to model manifolds is to treat the curved surface as being composed of several neighborhoods. If each data point manages to preserve the distance not with all the other points but only the ones close to it, the geometric relationships can be maintained in the data.**

****