Curse of Dimensionality Theoretical Questions

Q1: What is the Curse of Dimensionality?

Answer

The **curse of dimensionality** refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces that do not occur in low-dimensional settings. The common theme of these problems is that *when the dimensionality increases, the volume of the space increases so fast that the available data become sparse*.

Now let's say you have a square 100 yards on each side and you dropped a penny somewhere on it. It would be pretty hard, like searching across two football fields stuck together. It could take days.

Now a cube 100 yards across. That's like searching a 30-story building the size of a football stadium. Ugh.

The difficulty of searching through the space gets a lot harder as you have more dimensions. You might not realize this intuitively when it's just stated in mathematical formulas, since they all have the same "width". That's the **curse of dimensionality**.

Source: en.wikipedia.org

Q2: What is the *Curse of Dimensionality* and how can Unsupervised Learning help with it?

Answer

- As the amount of data required to train a model increases, it becomes harder and harder for machine learning algorithms to handle. **As more features are added to the machine learning process, the more difficult the training becomes.**
- In very *high-dimensional* space, supervised algorithms learn to separate points and build function approximations to make good predictions.

> When the number of *features* increases, this **search becomes expensive**, both from a time and compute perspective. It might become impossible to find a good solution fast enough. This is the *curse of dimensionality*.

- Using ***dimensionality reduction*** of unsupervised learning, the most *salient* features can be discovered in the original feature set. Then the dimension of this feature set can be reduced to a more manageable number while losing very little information in the process. This will help supervised learning find the optimum function to approximate the dataset.

Source: www.amazon.com

Q3: Why is data more *sparse* in a high-dimensional space?

Answer

> If we increase dimensions or features, we are giving our data a higher chance to **differentiate** itself from other data points.

The previous example illustrates that more dimensions give our data a higher chance of **being different**, of **being unique**, and that is why it becomes more **sparse**.

This also tells us that the distance between two random points increases with more dimensions: every data point is becoming increasingly separate and different from the rest.

Source: towardsdatascience.com

Q4: How does the *Curse of Dimensionality* affect Machine Learning models?

Answer

- As we increase the **number of dimensions**, our data becomes more **sparse**; every new dimension increases the volume of the feature space, giving our data a **higher differentiation** chance and therefore, the possibility of it becoming more spread out in a higher dimensional space than in a lower one. This means that if we need more **training samples** of a kind for our model to be able to learn about them and be able to predict them well, generalizing, in the future.

- As we increase the number of dimensions, especially for parametric models, we **increase the time** it takes to **train** them.

- Introducing features that don't add much value to our models and therefore increasing the number of dimensions, makes our model learn from these ***noisy*** or **irrelevant features** and can lead to a reduction in its performance.

- More features and thus dimensions lead to models that are more complex and harder to interpret than those with a low number of features.

Source: towardsdatascience.com

Q5: How does *High Dimensionality* affect *Distance-Based Mining Applications*?

Answer

- Many *distance-based* data mining applications **lose their effectiveness** as the dimensionality of the data increases.
- For example, a distance-based clustering algorithm may group unrelated data points because the distance function may poorly reflect the *intrinsic semantic distances* between data points with increasing dimensionality.
- As a result, distance-based models of clustering, classification, and outlier detection are often qualitatively ineffective. This phenomenon is referred to as the *curse of dimensionality*.

Source: www.amazon.com

Q6: How does the *Curse of Dimensionality* affect *Privacy Preservation*?

Answer

Anonymization is the process of modifying data before it is given for data analytics so that de-identification is not possible (so privacy is preserved) and will lead to K indistinguishable records if an attempt is made to de identify by mapping the anonymized data with external data sources.

1. **Computational challenges:** Optimal *k-anonymization* is **NP-hard**. This implies that with increasing dimensionality, it becomes more difficult to perform privacy preservation.
2. **Qualitative challenges:** The qualitative challenges to privacy preservation are even more fundamental. Recently, it has been shown that it may be difficult to perform effective privacy preservation without losing the utility of the anonymized data records. This is an even more fundamental challenge because it makes the privacy-preservation process less practical.

Source: www.amazon.com

Q7: Does *kNN* suffer from the *Curse of Dimensionality* and if it why?

Answer

Yes, very much so, **K-Nearest Neighbors** operates on the distance between the data points. The distance of the data points is inversely proportional to the exponential increase in the number of data points that leads to the curse of dimensionality.

Source: www.quora.com

Q8: What are some trade-offs when using *Embeddings* in Machine Learning?

Answer

An **embedding** is a relatively low-dimensional space into which you can translate high-dimensional vectors. Embeddings make it easier to do machine learning on large inputs like sparse vectors representing words.

The **lossiness of the representation** is controlled by the **size of the embedding layer**. In practice, the exact dimensionality of the **embedding space** is something that we choose as a practitioner:

- By choosing a very small output dimension of an embedding layer, too much information is forced into a small vector space and context can be lost.
- On the other hand, when the embedding dimension is too large, the embedding loses the learned contextual importance of the features.
- The optimal embedding dimension is often found through **experimentation**, similar to choosing the number of neurons in a deep neural network layer.

However, if we're in a hurry, there are two rules of thumb that we could take:

1. Use the **fourth root of the total number of unique categorical elements** as the embedding dimension.
2. The embedding dimension should be approximate `1.6` **times the square root of the number of unique elements in the category**, and **no less than** `600`.

For example, suppose we wanted to use an embedding layer to encode a feature that has `625` unique values:

- Using the first rule of thumb, we would choose an embedding dimension for a plurality of `5`.
- Using the second rule of thumb, we'd choose `40`.
- If we are doing *hyperparameter tuning*, it might be worth searching within this range.

Source: www.oreilly.com

Q9: Explain *Curse of Dimensionality* to a child

Answer

The analogy I like to use for the curse of dimensionality is a bit more on the geometric side, but I hope it's still sufficiently useful for your kid.

- To catch a caterpillar moving along the branch
- To hunt a dog and maybe catch it if it were running around on the plain (two dimensions)
- To hunt birds, which now have an extra dimension they can move in

---

Source: stats.stackexchange.com

Q10: How does the *Curse of Dimensionality* affect *k-Means Clustering*?

Answer

- **k-Means** is a search strategy to *minimize* the squared Euclidean distance.
- As the number of dimensions increase, the objects become approximately *equidistant (at equal distances)* from each other given that the data is distributed uniformly throughout the space.
- What matters is not necessarily the number of variables, but the *effective dimensionality* of the data. It is *quite common that most of the variation exists in a couple of dimensions*. If this is the case then there will be fewer problems with *k-means* in the sense that the effective dimensionality is much smaller.
- Therefore, under the assumption that there are meaningful latent groupings in the data, they do not necessarily exist in all of the dimensions or in constructed dimensions that maximize variation. The clusters might be in the *lower-variation dimensions*.

Source: stats.stackexchange.com

Q11: How does a Deep Neural Network escape/resist the *Curse of Dimensionality*?

Answer

The **curse of dimensionality** normally comes about because in data there are relevant and too many **irrelevant** (noise) **features**. The neurons in deep learning (DL) architectures, use lots of data in order to model a problem and thereby a DL system reduces the influence of irrelevant features while **increasing the influence of relevant features** during learning.

$$v = [v_1, v_2, \cdots, v_n]$$

For n very large number such as an image n = *width* × *height*, we know that the actual information exists in a much lower dimensional space than n. That is why **dimensionality reduction** works well because it eliminates the curse of dimensionality by projecting the data into a much lower relevant representational space. The process of learning in machine learning (ML) algorithms finds that smaller dimensional representation space in the large raw vector v.

For simplicity let's consider a single node,

$$y = \psi(\sum_{i=1}^{n} v_i w_i + b)$$

The node makes a decision by weighing each feature *vi* thus after training the weights for corresponding relevant features will be high. Further, consider v is a concatenation of the *relevant feature vector* and the *irrelevant feature vector* such as

$$v = [v_{relevant}, v_{irrelevant}]$$

The weight vector can also be seen as a concatenated vector:

$$w = [w_{relevant}, w_{irrelevant}]$$

So we can further write

$$\begin{aligned} y =& \psi(v^T w + b) \\ y =& \psi(v_{relevant}^T w_{relevant} + v_{irrelevant}^T w_{irrelevant} + b) \end{aligned}$$

After learning,

$$w_{irrelevant} \approx 0$$

So that reduces the effective dimensionality of the problem to the dimensionality of the **relevant features**. This is a form of dimensionality reduction. This process occurs at every layer of the deep neural nets (DNN) because each of the neurons in the DNN will only be sensitive to a particularly relevant feature.

Source: www.quora.com

Q12: Does *linear SVMs* suffer from the *Curse of Dimensionality*?

Answer

The performance bounds on which the maximal margin classifier is based in **linear SVM** are independent of the input vector, but instead, depending on the **margin of separation** which is defined with the **hyperparameter *C***.

- For *large values* of ***C***, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly.
- A *very small* value of ***C***, will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points.

Instead of the complexity of the model being defined in terms of the number of model parameters, the parameter ***C*** is used to define a nested set of classes of models of increasing complexity.

This is the reason that the **SVM** with an **infinite-dimensional feature space** (e.g. that induced by the *radial basis function kernel*) can still give good generalisation performance. However, this requires careful tuning of the regularisation parameter. In other words, the **linear SVM** is robust to the **curse of dimensionality** when the ***C*** parameter is tuned very carefully.

Source: stats.stackexchange.com

Q13: Why does the hyperparameter optimisation method *GridSearch* suffer from the *Curse of Dimensionality*?

Answer

Consider the standard classification framework, we have a sample which we divide into **training sample** and **validation sample**. We are solving an optimization problem `P` (which would usually be something like minimize training error plus a regularization term), which is a function of the ***model parameters***, say `w`, the ***training sample*** and some ***hyperparameters***, say `α` and `β`.

$$w^*(\alpha, \beta) = \arg min_w P(w, \alpha, \beta, S_{train})$$

Now we use this `w*` to predict on the **validation sample** to get **validation error**. We can view this scenario in terms of a ***validation error function***: the function takes as inputs the hyperparameters `α` and `β`, and returns the validation error corresponding to `w*(α,β)`.

So the goal of hyperparameter optimization is to find the set of values of `α` and `β`, that **minimize** this validation error function. Therefore, we resort to ***grid search***: pick a bunch of values of `α:(α1,α2,…)`, pick a bunch of values of `β:(β1,β2,…)` and for each pair of values, evaluate the ***validation error function***. Then pick the pair that gives the minimum value of the validation error function.

Now, the dimension of the space we're looking at or the dimension of the grid we're working with is `2` here (one dimension is `α` and the other one is `β`). So if we have `n` values to try for `α` and values to try for `β`, the total number of pairs would be `n×n`.

Generalizing from here, if we have *d* parameters, the number of *pairs* (or ordered sets) would be `n^d`. That `d` in the exponent is the **curse of dimensionality**, as the number of hyperparameters (`d`) increase a little, the procedure quickly becomes much more expensive.

Just to have some numbers, it is typical to have `n = 5` or around that. So `d = 2` hyperparameters require solving `5^2 = 25` optimization problems, which is doable. But if we have `d = 5` hyperparameters, we now need to solve `5^5 = 3125` optimization problems, which is basically **too expensive** for most practical purposes.

Source: www.quora.com

Q14: Does *Random Forest* suffer from the *Curse of Dimensionality*?

Answer

Random Forests use a collection of **decision trees** to make their predictions, but instead of using all the features of the problem, individual trees only use a **subset of the features**. Since they use *bootstrapped* data and a random set of features, they ensure diversity and robust performance they are **immune to the curse of dimensionality** as they do not consider all the features at one time for individual trees. The main disadvantage is that they are complex and computationally expensive.

Source: stats.stackexchange.com