Date: 15-2-2022

1. Use of eigenvalues and eigenvectors in data science

For any squared matrix, an eigenvector is a non zero vector where the product of multiplying by this vector returns a matrix that is a multiple of the eigenvector. Eigenvector is a vector which when multiplied with a transformation matrix results in another vector multiplied with a scalar multiple having the same direction as Eigenvector. This scalar multiple is known as Eigenvalue. Eigenvectors are unit vectors, which means that their length or magnitude is equal to 1. They are often referred to as right vectors, which simply means a column vector. Whereas, eigenvalues are coefficients applied to eigenvectors that give the vectors their length or magnitude. Eigenvectors and Eigenvalues are key concepts used in feature extraction techniques such as Principal Component analysis which is an algorithm used to reducing dimensionality while training a machine learning model.
Eigendecomposition is used to decompose a matrix into eigenvectors and eigenvalues which are eventually applied in methods used in machine learning, such as in the Principal Component Analysis method or PCA. Decomposing a matrix in terms of its eigenvalues and its eigenvectors gives valuable insights into the properties of the matrix. The whole matrix can be summed up to just a multiplication of scalar and a vector. In fact PCA uses this technique to effectively represent multiple columns to less number of vectors by finding linear combinations among them and evaluating eigenvalues and eigenvectors for them.

2. Non relational databases

A non-relational database stores data in a non-tabular form, and tends to be more flexible than the traditional, SQL-based, relational database structures. It does not follow the relational model provided by traditional relational database management systems.

Non-relational databases are often used when large quantities of complex and diverse data need to be organized. For example, a large store might have a database in which each customer has their own document containing all of their information, from name and address to order history and credit card information. Despite their differing formats, each of these pieces of information can be stored in the same document.

Non-relational databases often perform faster because a query doesn’t have to view several tables in order to deliver an answer, as relational datasets often do. Non-relational databases are therefore ideal for storing data that may be changed frequently or for applications that handle many different kinds of data. They can support rapidly developing applications requiring a dynamic database able to change quickly and to accommodate large amounts of complex, unstructured data.

There are several advantages to using non-relational databases, including:

- Massive dataset organization:
  In the age of Big Data, non-relational databases can not only store massive quantities of information, but they can also query these datasets with ease. Scale and speed are crucial advantages of non-relational databases.
- Flexible database expansion:
  Data is not static. As more information is collected, a non-relational database can absorb these new data points, enriching the existing database with new levels of granular value even if they don't fit the data types of previously existing information.
- Multiple data structures:
  The data now collected from users takes on a myriad of forms, from numbers and strings, to photo and video content, to message histories. A database needs the ability to store these various information formats, understand relationships between them, and perform detailed queries. No matter what format your information is in, non-relational databases can collate different information types together in the same document.
- Built for the cloud:
  A non-relational database can be massive. And as they can, in some cases, grow exponentially, they need a hosting environment that can grow and expand with them. The cloud's inherent scalability makes it an ideal home for non-relational databases.

Examples:

- MongoDB
- Apache Cassandra
- Redis
- Couchbase
- Apache HBase

3. Max number for each numerical data type

int8	-128 to 127	uint8	Unsigned integer (0 to 255)		
int16	-32768 to 32767	uint16	Unsigned integer (0 to 65535)	float16	-65504.0 to 65504.0
int32	-2147483648 to 2147483647	uint32	Unsigned integer (0 to 4294967295)	float32	-3.4028234663852886e+38 to 3.4028234663852886e+38
int64	-9223372036854775808 to 9223372036854775807	uint64	Unsigned integer (0 to 18446744073709551615)	float64	-1.7976931348623157e+308 to 1.7976931348623157e+308

np.iinfo(‘dtype’).max **np.finfo('dtype').max**
np.iinfo(‘dtype’).min **np.finfo('dtype').min**