**PCA**

**Class Architicture:**

We have totally **7** function:

* **The constructor:**

We specify the number of the component (new features) that we want our data to have, and it specifies two variables, one to store the **component**, and another one to store the **mean**

* **Normalization:**

To standardize the data using the mean

* **Covariance:**

To calculate the covariance matrix

* **Calc\_eigen:**

To calculate the eigen values and vectors

* **Fit:**

It is the main function of our class

* **Transform:**

It takes the data matrix then transform it to the new **reduced data** using the same object after performing its **Fit function**

* **Reverse:**

Trying to de-transform the new (reduced) data to its original values

**Eigen:**

To calculate the Eigen Values and Vectors we made custom function using **sympy** library, the function has the following sub-functions:

* **calc\_determinant:**

its custom **recursive** function using **divide and conquer** method, to calculate the determinant of a given matrix by:

1. checking if the shape of a function is (2,2) if its true then the determinant is the:

(value [0][0] \* value [1][1]) – (value [0][1] \* value [1][0])

1. if the shape of the matrix is not a (2,2) then I will divide the matrix into **M** **sub-matrices**, which **M** is the number of **columns** or **rows** in the parent matrix “note the matrix is squared matrix so number of rows = number of columns”.

these new matrices will have the shape of (M-1, M-1) because we are removing the first row and **T’th column**, which **T** is the number of the **sub-matrix**.

We calculate the determinant to the sub-matrices using the same function, the final determinant to the original matrix will be:

the summation of all the sub-determinant each sub-determinant multiplied by (-1)^T and its **T** value “value [0][T] from the parent matrix” from the first row

see the following formula

A computer screen with text

Description automatically generated

* calc\_eigenvalues:

its custom function takes the result of:

A black symbols of a mathematical equation

Description automatically generated with medium confidence

Let’s make the above formula equals to matrix called Result, our calc\_eigen function takes the Result function then forming from it the equation of the determinant

then finding the values of Lambda that makes the determinant equals to zero by using Eq, and solve functions from sympy library

* calc\_eigenvectors:

in this function we are calculating the eigen vectors using the eigen values and Result matrix.

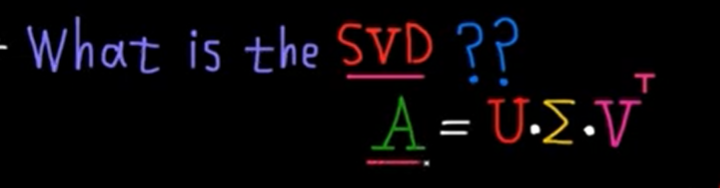
but we faced a problem:

**Problem Definition**

that the determinant of The Result matrix after substituting the Lambda with an Eigen value not always Equal to zero some-times the determinant is a very small value it almost equal to zero, that’s due to the Floating-point precision , and this make the solve functions from sympy library consider the matrix is not singular and give us trivial solution witch is all the vectors are equals to zeros, when in the same time its theoretically should equal to zero for the eigenvalues

**Attempt (1) using SVD Only as an attempt to solve Floating-point**

So we used **SVD** (Singular Value Decomposition) matrix analyzation its robust to numerical issues because it works with floating-point approximations and doesn't rely directly on symbolic solvers, because it’s can give us a singular matrix and its determinant is exactly equal to **zero**



We can choose U or V^T use V^T because there are both squared matrices, and their vectors are perpendicular on each other, we apply the SVD on our Result matrix then we, took a single vector from the V^T for its corresponding eigen value but we found another problem the dot product between the resulting eigen vectors its not equals to zero

A screen shot of a computer code

Description automatically generated

Its equal to very small number almost equal zero

Comparing to the built-in numpy array:A black screen with white text

Description automatically generated

We found it’s having the same problem, and we found out that the complexity of the determinant function is O(n!) so we changed the method of finding the eigen value to be zeros of a polynomial using root function from numpy, and this solution solved the problem

**FuzzyEntropy**

The model takes ‘B’ the similarity between the centroid and the vectors, and ‘y’ to set a threshold if the class contains less than y% of the size of the original data the model will mark the vectors and in that class as outliers.

we have made comments everywhere in the code, to make it easier to understand the functions, the class has the following functions:

* Ecledian\_distance: to create the distance matrix
* Similarities: to create the similarity matrix
* Calc\_alpha: to calculate the value of the constant alpha from the law to use it in the similarities function
* Entropies: to calculate the entropy for each vector
* Get\_next\_entropy: decide which vector will be the next class centroid
* Fit: is the main function for the Fuzzy Entropy class, in this function we:

1. Getting the distance matrix
2. Calculating the mean
3. Getting alpha using the mean and the similarity B
4. Getting the similarity matrix
5. Getting the entropies
6. Making the vector with minimum entropy as a class centroid
7. Calculating the similarity between the class centroid and all the untaken vectors and consider them after that as taken by a class
8. Selecting the vector with minimum entropy from the untaken vectors then go to 7
9. Stop if all data marked as taken
10. If a class contains less than y% of the whole data mark its vectors as outliers