1.Introduction

I implemented PCA algorithm on my own in Java and compared the result with the result of Weka tool. I would like to share implementation of my PCA algorithm first then comparison result of it with result of Weka.

2.Implementation

2.1 Code Explanation

In this part I would like to briefly explain code. I wrote the code by looking at the notes that I took during the lectures. This is my Main, I declared all my functions as “public static” so I directly call them in my Main function.

```
public static void main(String[] args) {

    getNumber_of_attributes_reduced=Integer.parseInt(args[1]);
    reader(args[0]);
    find_means();
                                                        //shur
    covariance_matrix();
    eigenvalue_eigenvector_finder();
    srter_and_picker(getNumber_of_attributes_reduced);
    createAmatrix_and_multiply();
    writer();

}
```

Here is the all the variables, and functions that I declared. Data holds all the data that was read in **reader()** function. Means hold all the mean values of attributes that were calculated in **find_means()** function.

```
public static List<List<Float>> data = new ArrayList<List<Float>>();
public static List<Float> means;
public static int number_of_attributes,number_of_samples;
public static double[][] matrix; // covariance matrix
public static EigenvalueDecomposition eigenvalues;
public static Matrix eigenvector;
public static List<Integer> top_indexes=new ArrayList<>();

public static Matrix final_result;
public static int getNumber_of_attributes_reduced;
```

```
public static void find_means()
{
    means=new ArrayList<Float>();

    number_of_attributes=data.get(0).size();
    number_of_samples=data.size();
    for(int j=0; j<data.get(0).size(); j++) {
        float sum=0;
        for (int i = 0; i < data.size(); i++) {

            sum = sum + data.get(i).get(j);
        }
        means.add(sum/data.size());
    }
}
```

Reader() and **writer()** functions are easy, so I skipped that part.
In **find_means()** function I sum all attribute values of specific attribute and divide it to number of instances and store result for all attributes in **means** variable.

In the **covariance_calculator()** function I calculate covariance of any two attributes. And in **covariance_matrix()** function I create covariance matrix. In last function I find eigenvalues and eigenvectors of covariance matrix.

I used **Jama** library to find eigenvalues and eigenvectors of covariance matrix and also to do matrix operations.

```
public static float covariance_calculator(int a1, int a2)
{
    float sum=0;
    for(int i=0; i<data.size(); i++)
    {
        //(a1-mean'a1)*(a2-mean'a2)
        sum=sum+(data.get(i).get(a1)*means.get(a1) + data.get(i).get(a2)*means.get(a2));
    }
    return sum/number_of_samples;
}

public static void covariance_matrix()
{
    matrix=new double[number_of_attributes][number_of_attributes];

    for(int i=0; i<number_of_attributes; i++)
    {
        for(int j=0; j<number_of_attributes; j++)
        {
            matrix[i][j]=covariance_calculator(i,j);
        }
    }
}

public static void eigenvalue_eigenvector_finder()
{
    Matrix m=new Matrix(matrix);
    eigenvalues = m.eig();
    eigenvector=eigenvalues.getV();

}
```

```java
public static void srter_and_picker(int reducenumber) //srter cause sout is important )
{
    for(int i=0; i<reducenumber; i++)
    {
        double min=Double.MIN_VALUE; //nedense altdakide dogru result basyo cok ilginc
        double min = -1;

        int index=0;
        for (int j = 0; j < eigenvalues.getRealEigenvalues().length; j++)
        {
            if(min<eigenvalues.getRealEigenvalues()[j] && !top_indexes.contains(j))
            {
                index=j;
            }
        }
        top_indexes.add(index);
    }

}
```

Here I sort the eigenvalues that I found in previous step.
And pick the top **attribute_number_to_reduce**(number of attributes in new set) .

Then I create matrix from eigenvectors that correspond to my picked eigenvalues. In this matrix each row is one eigenvector. And I multiply this matrix with my data to get new data set.

```java
public static void createAmatrix_and_multiply()
{
    double A_matrix[][]=new double[top_indexes.size()][number_of_attributes];
    for(int i=0; i<top_indexes.size(); i++)
    {
        for(int j=0; j<eigenvector.getRowDimension(); j++)
        {
            A_matrix[i][j]=eigenvector.get(top_indexes.get(i),j);
        }
    }

    double data_matrix[][]=new double[data.size()][data.get(0).size()];

    for(int i=0; i<data.size(); i++)
    {
        for(int j=0; j<data.get(0).size(); j++)
        {
            data_matrix[i][j]=data.get(i).get(j);
        }
    }
    Matrix m1=new Matrix(A_matrix);
    Matrix m3=new Matrix(data_matrix);
    Matrix m2=m3.transpose();
    Matrix m4=m1.times(m2);
    final_result=m4.transpose();

}
```

2.2 Running the code

I also provided ReadMe file in SourceCode folder. Since I used Jama library you have to compile the code with Jama. Put your **"arff"** file to same folder with source code and provide name as first argument. In my code I took the location of file like this **"./"+filename** so putting the "**arff"** file to the same folder with source-code and providing name like **"Data_WithoutLabels.arff"** as **first** argument is enough. Then as **second** argument provide number that you want your attributes to be reduced to. It will generate **reduced.arff** file in the same folder, this will be your new data set with reduced number of attributes.

2.3 Testing

I reduced the number of attributes in **"Data_WithoutLabels.arff"** from 5 to 3 and saved it in **"reduced3.arff".** Then I reduced number of attributes in **"reduced3.arff"** from 3 to 2 and saved it in **"reduced2.arff"** and compared their result all of them gave same perfectly separated 6 clusters so I think my code works perfectly.

2.Comparison with Weka and result obtained.

In this section I would like to compare the result that we obtained with help of Weka and result that we obtained with my implementation.

In Result folder there exist **reduced3.arff** our new data set with 3 attribute which we obtained from our original data **Data_WithoutLabels.arff.** And also **reduced2.arff** with 2 attributes that we obtained from **reduced3.arff.** In their .xlsx files also provided below. You can see that their attributes were enough to cluster them into 6 clusters.

k	Within cluster sum of squared errors		No	x	y	z	w	t	d Instance
				Final Cluster Centroids					
1	810.6925101916		1	12	6.0068	5.0143	4.002	5.0028	800
2	529.8816088737		2	5.0211	9.0041	11	9.9992	12	300
3	238.0119660525		3	6.9958	0.988	9.0006	11.007	3.008	400
4	145.8994653014		4	0.9645	2.0302	3.0318	1.0137	7.0218	100
5	75.4857954602		5	7.9877	4.01	8.0142	7.0015	9.0089	200
6	3.5446067797		6	10	7.9932	1.0127	4.9812	9.991	200
7	3.4598251315		7						
8	3.1993469413		8						
9	3.1247513846		9						
10	3.0013260538		10						
11	2.9649892881		11						
12	2.9313331396		12						
13	2.8575638188		13						
14	2.7925235051		14						
15	2.7338235933		15						

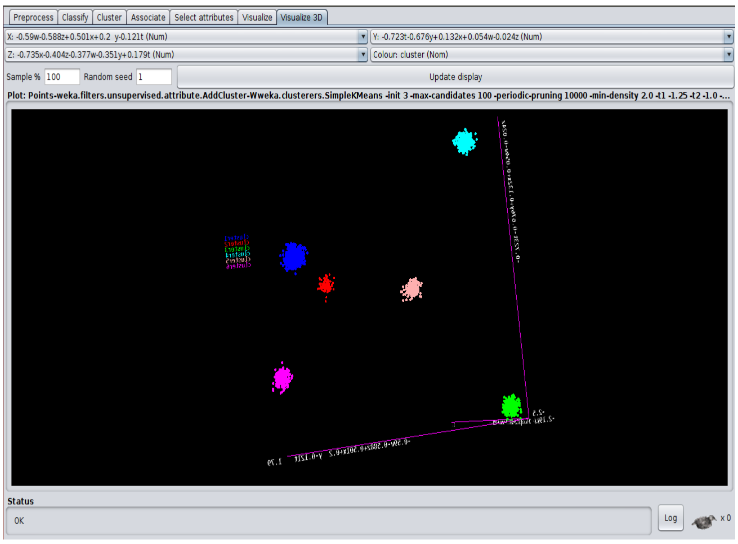
k	Within cluster sum of squared errors		No	x	y	z	w	t	d Instance
				Final Cluster Centroids					
1	430.5284426215		1	1.443	7.079	-3.907			800
2	247.451346436		2	5.3028	18.141	-5.394			300
3	143.290481669		3	7.7874	7.6675	-1.499			200
4	37.2159727541		4	-0.778	8.9933	-10.9			400
5	19.3807744298		5	2.2138	6.1416	1.6274			100
6	1.7498477129		6	2.0715	11.321	-4.664			200
7	1.7126925851		7						
8	1.6571569012		8						
9	1.5689932246		9						
10	1.512211756		10						
11	1.4028013272		11						
12	1.1731091214		12						
13	1.1567508553		13						
14	1.1236086968		14						
15	1.1160063369		15						

Here the 1st result is from our original data **Data_WithoutLabels.arff** with 5 attributes, 2nd result is from our **reduced3.arff** with 3 attributes. And last one is from **reduced2.arff** with 2 attributes. I want you to pay attention to the number of instances, as you can see in all three result our total 20000 data divided to clusters as 800,300,400,200,200,100.

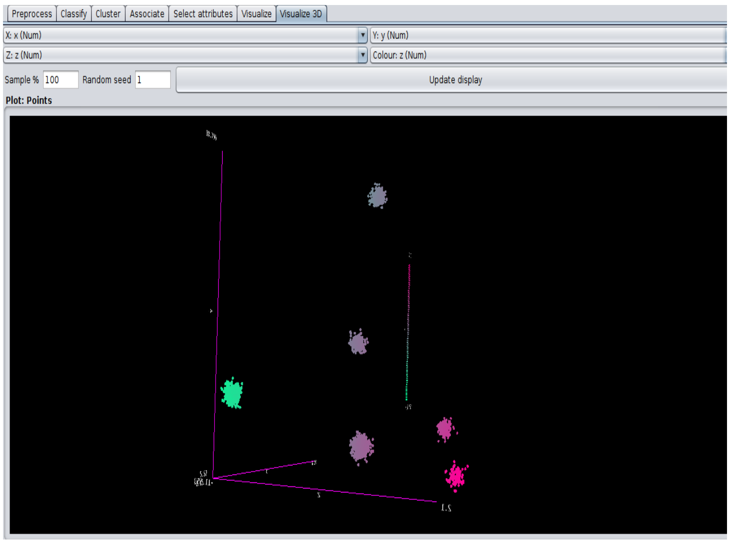
Below I provided plots for all three cases.

A	B	C	D	E	F	G	H	I	J
k	Within cluster sum of squared errors		No	x	y	z	w	t	d Instance
				Final Cluster Centroids					
1	307.782522851		1	3.1407	-3.764				800
2	174.8839918157		2	9.4638	-4.983				300
3	61.2596334962		3	2.4068	-11.07				400
4	24.1883531726		4	1.9875	1.5089				200
5	18.0941628601		5	4.5957	1.3813				100
6	1.4476248795		6	5.8439	-4.841				200
7	1.3381515387		7						
8	1.0724617869		8						
9	1.020239743		9						
10	0.9781228718		10						
11	0.8533624204		11						
12	0.789996291		12						
13	0.7690044855		13						
14	0.7349526715		14						
15	0.7152868286		15						

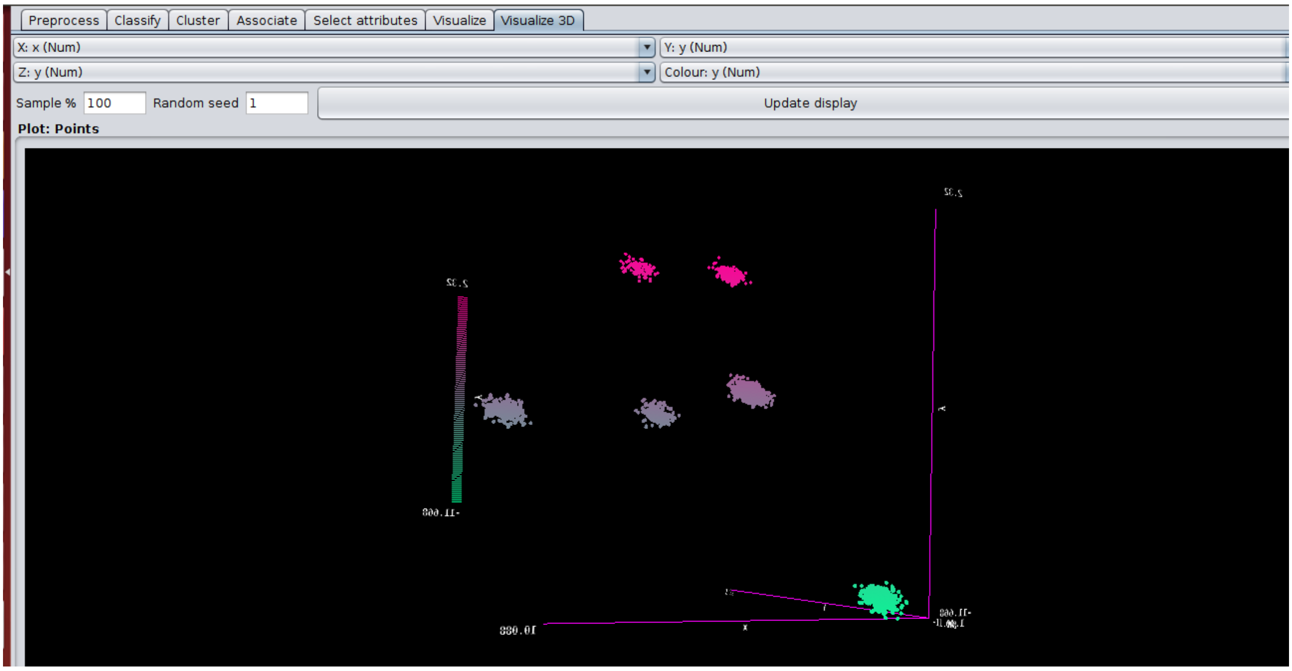
Plot of data when I reduce number of attributes to 3 with help of Weka



Plot of data when I reduce number of attributes to 3 with help of my implementation



Plot of data when number of attributes reduced to 2 with help of my implementation.



2.Conclusion

As conclusion we can clearly see from results obtained and from plots we could reduce number of attributes to 3 and to 2. This implementation was implemented with help of my lecture notes.