ДОДАТОК А

Вихідний код програми

```
void FCM::CalculateKernelPCA()
{
	vector <Element> TransfData;
	TransfData.resize(numberElements);
	for (int i = 0; i < numberElements; i++)
		TransfData[i].SetSize(dimensions);
	for (int i = 0; i < dimensions; i++)
	 if (i != valueClass)
		{
			double tempSum = 0;
		for (int j = 0; j < numberElements; j++)
				tempSum += Data[j].GetValue(i);
			tempSum /= numberElements;
			for (int j = 0; j < numberElements; j++)
				TransfData[j].SetValue(i,  (Data[j].GetValue(i)  –
tempSum) / (this->Max[i] – this->Min[i]));
		}
	int KernelDimensions = (dimensions – 1) * 2.5;
	vector <Element> KernelData, KernelCentroids;

 KernelData.resize(numberElements);
  for (Element& i : KernelData)
  {
	i.SetSize(KernelDimensions);
  }

	KernelCentroids.resize(KernelDimensions);
	for (Element& i : KernelCentroids)
	{
		i.SetSize(dimensions);
	}
```

```
srand((unsigned)time(0));
for (Element& i : KernelCentroids)
        for (int j = 0; j < i.GetSize(); j++)
        {
                i.SetValue(j, ((rand() % 100) – 50.0) / 100);
        }
for (int i = 0; i < KernelData.size(); i++)
        for (int j = 0; j < KernelData[i].GetSize(); j++)
        {
                KernelData[i].SetValue(j, pow(2.71828, -1 *
KernelCentroids[j].Distance(TransfData[i]) / (2 * 0.35 * 0.35)));
        }
vector <Element> TransfKernelData;
    TransfKernelData.resize(numberElements);
    for (int i = 0; i < numberElements; i++)
            TransfKernelData[i].SetSize(KernelDimensions);
     vector <double> KernelMax, KernelMin;
     KernelMax.resize(KernelDimensions);
     KernelMin.resize(KernelDimensions);
     for (int i = 0; i < KernelDimensions; i++)
     {
             KernelMax[i] = -10000;
             KernelMin[i] = 10000;
     }
     for (int i = 0; i < KernelDimensions; i++)
             for (int j = 0; j < numberElements; j++)
             {
                     KernelMax[i] = max(KernelMax[i],
KernelData[j].GetValue(i));
                     KernelMin[i] = min(KernelMin[i],
KernelData[j].GetValue(i));
             }
      for (int i = 0; i < KernelDimensions; i++)
      {
             double tempSum = 0;
```

```
                for (int j = 0; j < numberElements; j++)
                        tempSum += KernelData[j].GetValue(i);
                tempSum /= numberElements;
                for (int j = 0; j < numberElements; j++)
                    TransfKernelData[j].SetValue(i,  (KernelData[j].GetValue(i)  –
tempSum /*– KernelMin[i]*/) / (KernelMax[i] – KernelMin[i]));
        }
        Element W;
        W.SetSize(KernelDimensions);
        int counter = -1;
        int globalCounter = -1;
        int stopper = 1;
        string outpFirst = "1";
        string outpSecond = "2";
        string outpThird = "3";
        do
        {
                ofstream ofsFirst(outpFirst + ".txt");
              ofstream ofsSecond(outpSecond + ".txt");
                ofstream ofsThird(outpThird + ".txt");
                stopper *= 4;
                double tempNorm = 0;
                for (double i = 0; i < KernelDimensions; i++)
                {
                        W.SetValue(i, (rand() % 100));
                        if (W.GetValue(i) == 0)
                                W.SetValue(i, 10);
                        tempNorm += W.GetValue(i) * W.GetValue(i);
                 }
                 for (double i = 0; i < KernelDimensions; i++)
                          W.SetValue(i, W.GetValue(i) / sqrt(tempNorm));
                 Element Wnew(W);
                 vector <double> Y;
                 Y.resize(numberElements);
                 for (int i = 0; i < numberElements; i++)
```

```
                {
                        Y[i] = 0;
                        for (int j = 0; j < KernelDimensions; j++)
                                Y[i]                    +=                    W.GetValue(j)
*TransfKernelData[i].GetValue(j));
                }
                bool stop = false;
                do
                {
                        counter = (counter + 1) % numberElements;
                        if (counter == 0)
                                globalCounter++;
                        Element Wtemp(W);
                        for (int i = 0; i < KernelDimensions; i++)
                                Wtemp.SetValue(i, W.GetValue(i) + fuzz /*(1/
(counter + 1))*/ * Y[counter] * (TransfKernelData[counter].GetValue(i) –
W.GetValue(i) * Y[counter]));

                for (int i = 0; i < numberElements; i++)
                        {
                                Y[i] = 0;
                                for (int j = 0; j < KernelDimensions; j++)
                                        Y[i]     +=     Wtemp.GetValue(j)     *
(TransfKernelData[i].GetValue(j));

                        }

        double testDist = W.Distance(Wtemp);
                //if (testDist < 0.0000005)
                //        stop = true;
                if (globalCounter >= stopper)
                        stop = true;
                for (int k = 0; k < KernelDimensions; k++)
                        W.SetValue(k, Wtemp.GetValue(k));
        } while (!stop);
```

```
double acc = 0;
for (int i = 0; i < numberElements; i++)
{
        Element Wtemp(W);
        for (int j = 0; j < KernelDimensions; j++)
                Wtemp.SetValue(j, Wtemp.GetValue(j) * Y[i]);

 double tempRes = 0;
                for (int j = 0; j < KernelDimensions; j++)
                        if (j != valueClass)
                        tempRes                                 +=
TransfKernelData[i].GetValue(j) * TransfKernelData[i].GetValue(j);
                tempRes = sqrt(tempRes);
                acc  +=  TransfKernelData[i].Distance(Wtemp)  /
tempRes;
}

    acc /= numberElements;
        this->accuracy += acc;

  for (int i = 0; i < numberElements; i++)
        {
                if (Data[i].GetValue(valueClass) == 1)
                        ofsFirst << Y[i] << endl;
                if (Data[i].GetValue(valueClass) == 2)
                        ofsSecond << Y[i] << endl;
                if (Data[i].GetValue(valueClass) == 3)
                        ofsThird << Y[i] << endl;
        }
        if (acc < 0.8)
        {
                for (int i = 0; i < numberElements; i++)
                        for (int j = 0; j < KernelDimensions; j++)
                                if (j != valueClass)
                                {
```

```
TransfKernelData[i].SetValue(j, TransfKernelData[i].GetValue(j) –
W.GetValue(j)* Y[i]);
                                        }
                        double fuckSum = 0;
                        for (int i = 0; i < W.GetSize(); i++)
                                fuckSum += W.GetValue(i) * W.GetValue(i);
                        fuckSum = fuckSum;
                }
        } while (this->accuracy < 0.9);
}
```

ДОДАТОК Б
Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	__ с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата		
Розробив		Григор`єв Д.С.			Ядерне послідовне кластерування потоків даних на основі еволюційних нейронних мереж	Шифр групи: СШІм-19-2; Код напр./спец.: 122
Перевірив		Шевченко О.Ю.				
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ
Затв.		Філатов В.О.				