Теоретичні відомості

PNN (Probabilistic Neural Network)

В мережі PNN зразки класифікуються за допомогою оцінки їхньої близькості до сусідніх зразків.

Одним із методів, на основі якого можна класифікувати зразок, є обчислення центроїда кожного класу (тобто середньої характеристики розміщення усіх зразків класу). Такий метод чудово підійде для розподілів, подібних до рис. 1, але може призводити до помилок при розподілах, подібних до рис. 2.
Інший популярний метод заснований на використанні "найближчого сусіда". Такий метод теж не завжди працюватиме (рис. 3)

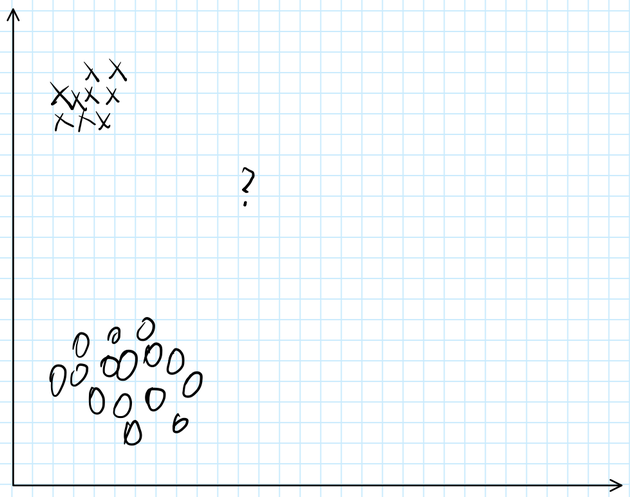


рис. 1

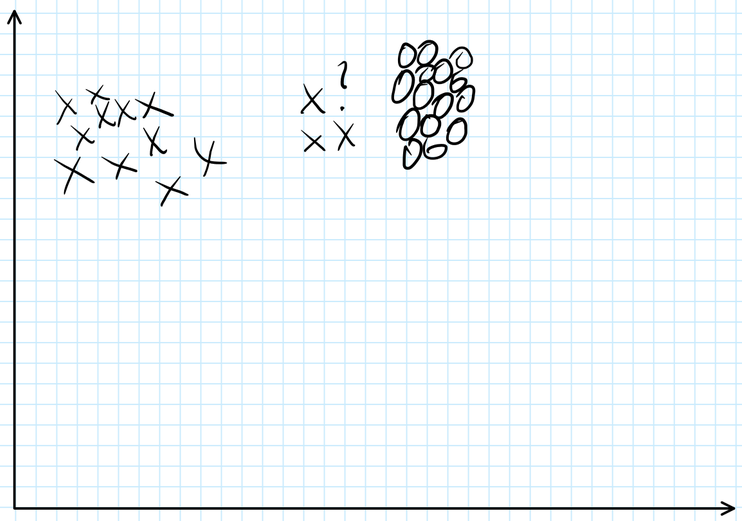
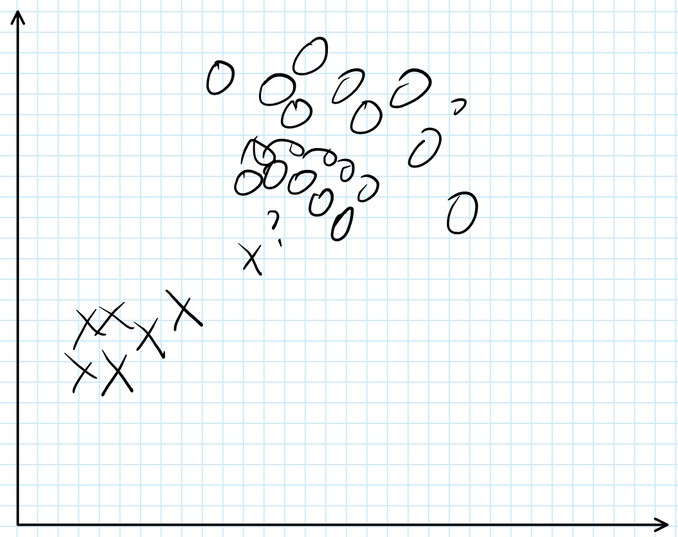


рис. 2



PNN мережа проводитиме класифікацію ефективно в усіх трьох випадках - на відміну від представлених методів, PNN враховує густину розподілу елементів одного класу.

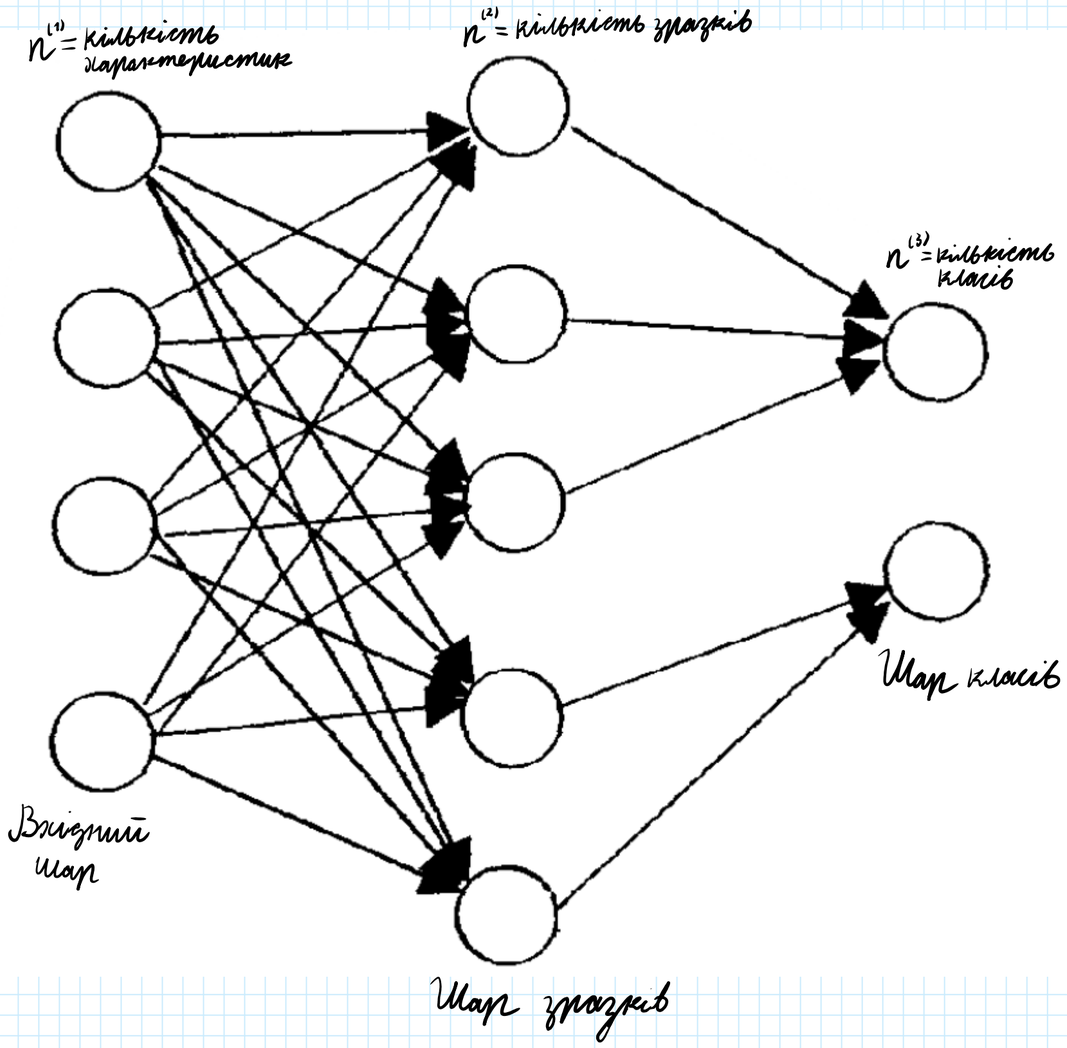
Функція Гауса:

$$g(x) = \sum_{i=1}^{n} e^{\left(\frac{-\|x - x_i\|^2}{\delta^2}\right)}$$

x – вектор характеристик досліджуваного зразка
x_i – вектор характеристик i-того зразка певного класу
δ – параметр розтягування функції.

Значення функції Гауса - оцінка імовірності належності x до класу.

Приклад PNN мережі (4 характеристики, 5 зразків, 2 класи)



Архітектура PNN

Вхідний шар:

Вхідний шар розподіляє характеристики вхідного зразка на шар зразків. Кожен нейрон вхідного шару пов'язаний із кожним нейроном шару зв'язків.

Шар зразків:

Шар зразків має по одному нейрону для кожного зразка. Вагові коефіцієнти вхідних сигналів нейрону дорівнюють значенням характеристик відповідного зразка.

Функція активації нейрону шару зразків:

$$O_j = e^{\frac{-\sum_{i=1}^{n}(w_{ij} - x_i)^2}{\delta^2}}$$

, j: індекс нейрону шару
$\vec{x}$: вхідний зразок
$\vec{w_j}$: j-тий зразок

Шар класів:

Шар класів має по одному нейрону для кожного класу. Кожен нейрон шару класів має зв'язки лише з нейронами зв'язків, які належать до класу. Всі вагові коефіцієнти між шаром зразків та шаром класів рівні 1.

Функція активації нейрону шару класів є сумою вхідних сигналів нейрону.

PNN мережа не потребує навчання, подібного до навчання мереж із оберненим поширенням сигналу. Всі параметри мережі PNN визначаються безпосередньо навчальними даними:
Кількість нейронів вхідного шару = кількість характеристик зразків
Кількість нейронів шару зразків = кількість зразків
Кількість нейронів шару класів = кількість класів
Значення сігми має значний вплив на результати мережі PNN, зазвичай підбирається експериментальним шляхом.

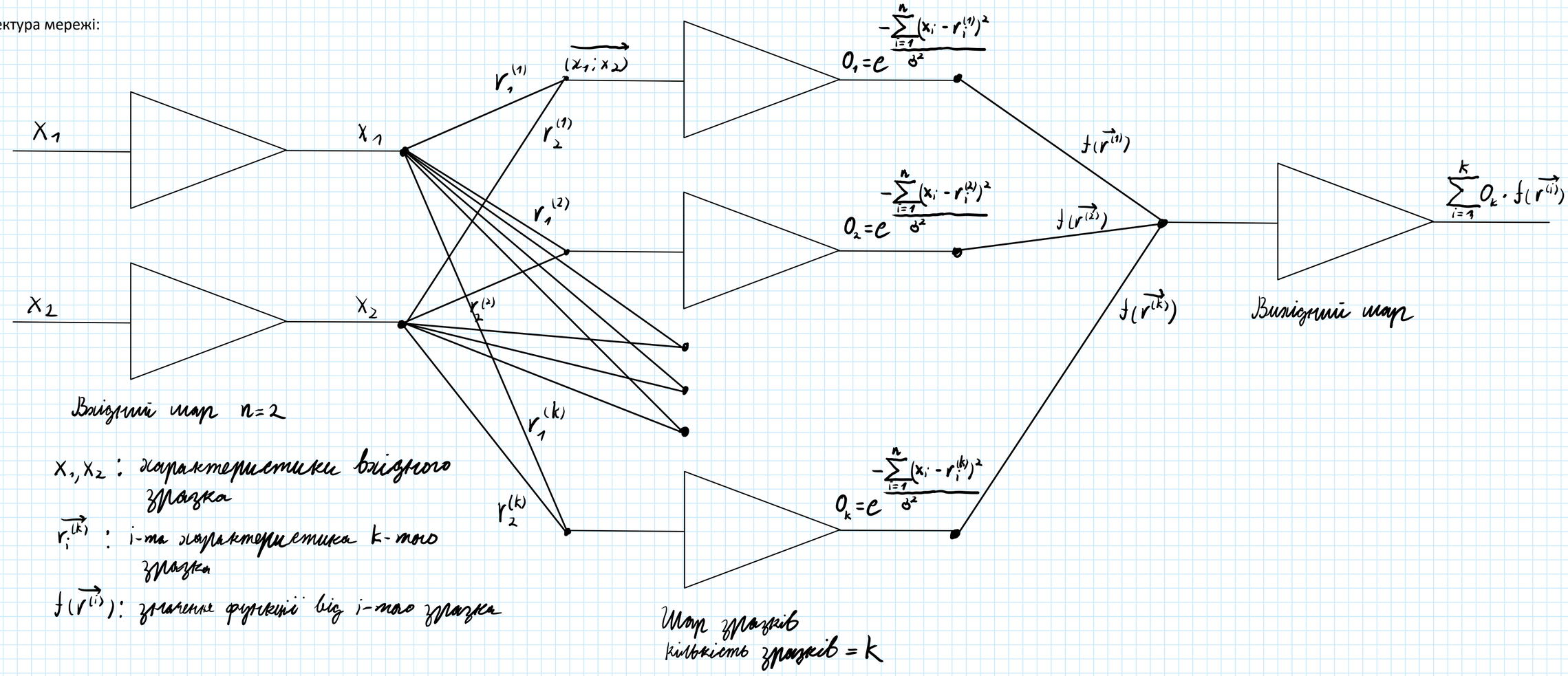
Список використаних джерел:

Роберт Каллан
Основні концепції нейронних мереж

Реалізація

В роботі було реалізовано PNN мережу мовою C. Посилання на репозиторій: https://github.com/Bohdan628318ylypchenko/MLDL-Lab2.git

Архітектура мережі:




```
PS J:\repos\MLDL\Lab2\x64\Release> .\Lab2.exe
Usage:
 [n]ew sigma a b segment_count;
 [s]ave path;
 [l]oad path;
 [p]rint;
 s[i]gma sigma;
 [r]un x y;
 [e]xit;
 [u]sage;

Command: n 0.045 0 0.4 5

Command: p 100
sigma = 0.045000; rfv_count = 25
ref #0: x = 0.000000, y = 0.000000 | f value #0: 0.000000
ref #1: x = 0.000000, y = 0.080000 | f value #1: 0.080000
ref #2: x = 0.000000, y = 0.160000 | f value #2: 0.160000
ref #3: x = 0.000000, y = 0.240000 | f value #3: 0.240000
ref #4: x = 0.000000, y = 0.320000 | f value #4: 0.320000
ref #5: x = 0.080000, y = 0.000000 | f value #5: 0.080000
ref #6: x = 0.080000, y = 0.080000 | f value #6: 0.160000
ref #7: x = 0.080000, y = 0.160000 | f value #7: 0.240000
ref #8: x = 0.080000, y = 0.240000 | f value #8: 0.320000
ref #9: x = 0.080000, y = 0.320000 | f value #9: 0.400000
ref #10: x = 0.160000, y = 0.000000 | f value #10: 0.160000
ref #11: x = 0.160000, y = 0.080000 | f value #11: 0.240000
ref #12: x = 0.160000, y = 0.160000 | f value #12: 0.320000
ref #13: x = 0.160000, y = 0.240000 | f value #13: 0.400000
ref #14: x = 0.160000, y = 0.320000 | f value #14: 0.480000
ref #15: x = 0.240000, y = 0.000000 | f value #15: 0.240000
ref #16: x = 0.240000, y = 0.080000 | f value #16: 0.320000
ref #17: x = 0.240000, y = 0.160000 | f value #17: 0.400000
ref #18: x = 0.240000, y = 0.240000 | f value #18: 0.480000
ref #19: x = 0.240000, y = 0.320000 | f value #19: 0.560000
ref #20: x = 0.320000, y = 0.000000 | f value #20: 0.320000
ref #21: x = 0.320000, y = 0.080000 | f value #21: 0.400000
ref #22: x = 0.320000, y = 0.160000 | f value #22: 0.480000
ref #23: x = 0.320000, y = 0.240000 | f value #23: 0.560000
ref #24: x = 0.320000, y = 0.320000 | f value #24: 0.640000

Command: r 0.32 0.32
result = 0.688360

Command: r 0.33 0.15
result = 0.486145

Command: r 0.16 0.24
result = 0.470728

Command: r 0.08 0.12
result = 0.197265

Command:
```

Демонстрація роботи реалізації PNN мережі

Створення нової мережі. Sigma = 0.045, a = 0, b = 0.4, segment_count = 5

Новостворена мережа. Мережа апроксимує функцію в межах квадрату (0, 0) (0, 0.4) (0.4, 0) (0.4, 0.4)
Кількість зразків = segment_count ^ 2

За рахунок експериментально підібраного значення сігми
маємо мережу, що обчислює "адекватні" значення як для відомих прикладів,
так і для довільних прикладів.

Код реалізації

pnn.h

```
#pragma once

#include <stdio.h>

typedef struct
{
	double x;
	double y;
} v2;

typedef struct
{
	double sigma;
	int rfv_count;
	v2 * refs;
	double * f_vals;
} pnn;

void pnn_new(double sigma,
		double a, double b, int segment_count,
		pnn * nn,
		double(*f)(double, double));

double pnn_run(pnn * nn, v2 * x);

void pnn_fread(pnn * nn, FILE * stream);

void pnn_fwrite(pnn * nn, FILE * stream);

void pnn_fprint(pnn * nn, FILE * stream, int head_count);
```

pnn.c

```
#include "pnn.h"

#include <stdlib.h>
#include <math.h>

static double act(v2 * x, v2 * ref, double sigma);
static double eucl2(v2 * a, v2 * b);

void pnn_new(double sigma,
		double a, double b, int segment_count,
		pnn * nn,
		double(*f)(double, double))
{
	int rfv_count = segment_count * segment_count;

	nn->sigma = sigma;
	nn->rfv_count = rfv_count;
	nn->refs = (v2 *)malloc(sizeof(v2) * rfv_count);
	nn->f_vals = (double *)malloc(sizeof(double) * rfv_count);

	double delta = (b - a) / (double)segment_count;
	double x, y;
	int k;
	for (int i = 0; i < segment_count; i++)
	{
		x = a + (double)i * delta;
		for (int j = 0; j < segment_count; j++)
		{
			y = a + (double)j * delta;

			k = i * segment_count + j;

			nn->refs[k].x = x;
			nn->refs[k].y = y;
			nn->f_vals[k] = f(x, y);
		}
	}
}

double pnn_run(pnn * nn, v2 * x)
{
	double result = 0;

	for (int i = 0; i < nn->rfv_count; i++)
		result += act(x, &(nn->refs[i]), nn->sigma) * nn->f_vals[i];

	return result;
}

static double act(v2 * x, v2 * ref, double sigma)
{
	return exp((-1.0 * (eucl2(x, ref)) / (sigma * sigma)));
}

static double eucl2(v2 * a, v2 * b)
{
	double x = b->x - a->x;
	double y = b->y - a->y;
	return x * x + y * y;
}
```

pnn_io.c

```
#include "pnn.h"

#include <stdio.h>
#include <stdlib.h>

void pnn_fread(pnn * nn, FILE * stream)
{
	fread(&(nn->sigma), sizeof(double), 1, stream);

	int rfv_count;
	fread(&rfv_count, sizeof(int), 1, stream);
	nn->rfv_count = rfv_count;

	nn->refs = (v2 *)malloc(sizeof(v2) * rfv_count);
	nn->f_vals = (double *)malloc(sizeof(double) * rfv_count);
	fread(nn->refs, sizeof(v2), rfv_count, stream);
	fread(nn->f_vals, sizeof(double), rfv_count, stream);
}

void pnn_fwrite(pnn * nn, FILE * stream)
{
	fwrite(&(nn->sigma), sizeof(double), 1, stream);
	fwrite(&(nn->rfv_count), sizeof(int), 1, stream);

	fwrite(nn->refs, sizeof(v2), nn->rfv_count, stream);
	fwrite(nn->f_vals, sizeof(double), nn->rfv_count, stream);
}

void pnn_fprint(pnn * nn, FILE * stream, int head_count)
{
	fprintf(stream, "sigma = %lf; rfv_count = %d\n", nn->sigma, nn->rfv_count);

	for (int i = 0; (i < nn->rfv_count) && (i < head_count); i++)
		fprintf(stream, "ref #%d: x = %lf, y = %lf | f value #%d: %lf\n",
			i, nn->refs[i].x, nn->refs[i].y, i, nn->f_vals[i]);
}
```