

Pattern Recognition – HW 3 – Yağız DÖNER – 141044062

Exercise 1 :

Exercise 1 ;

Yağız DÖNER - 141044062

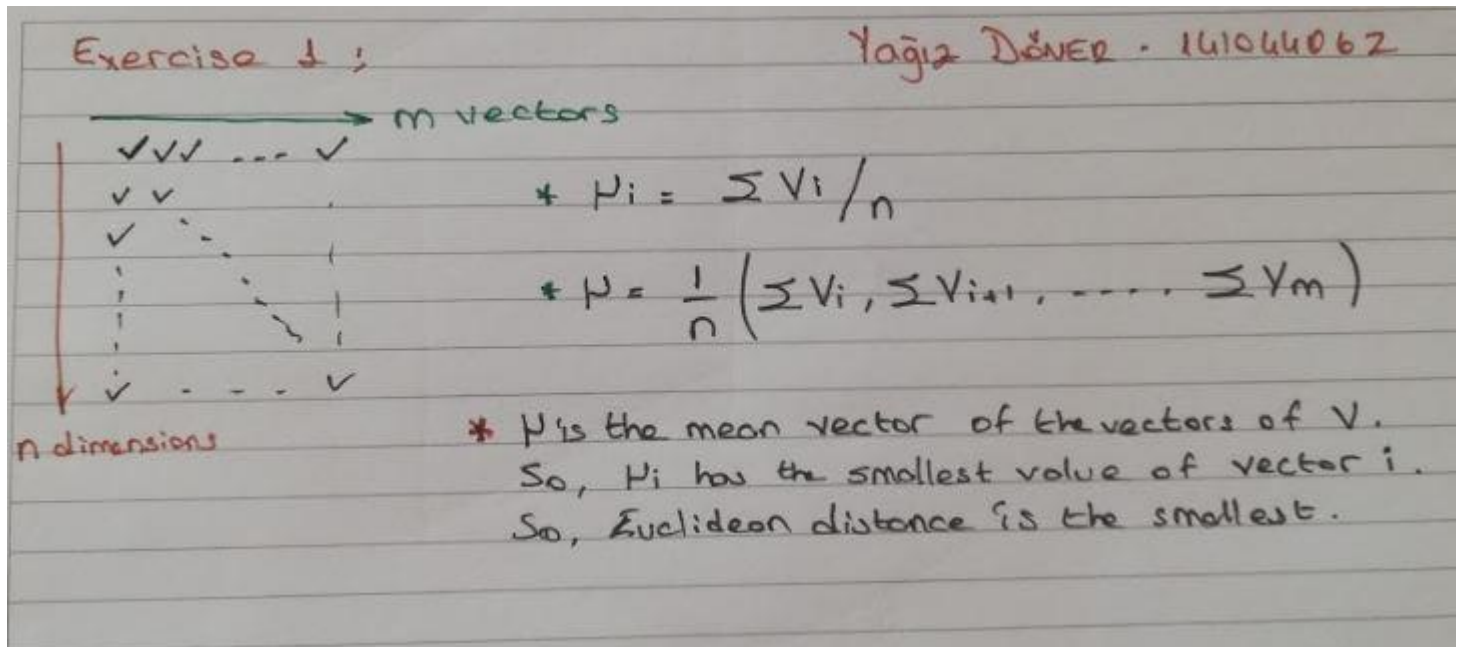
m vectors

n dimensions

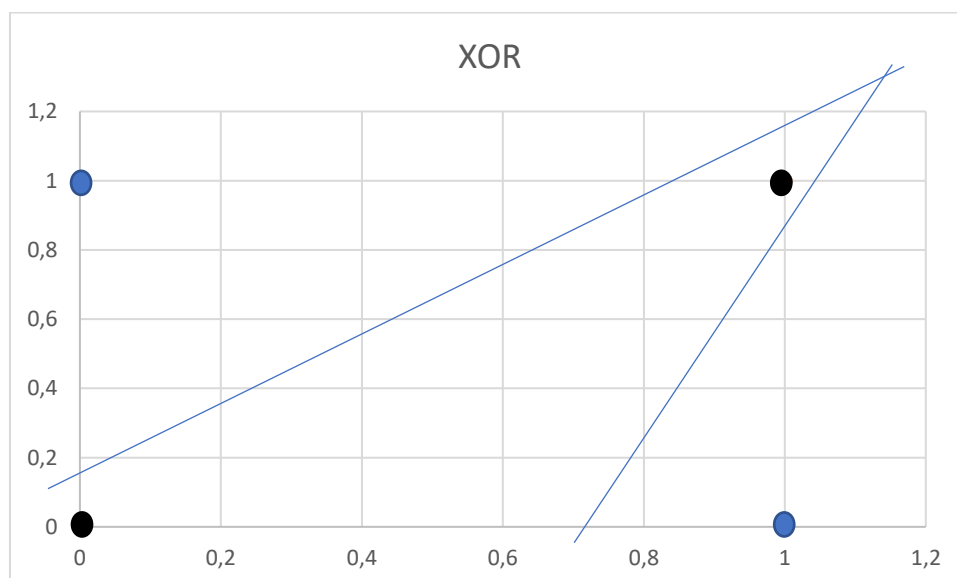
* $\mu_i = \sum V_i / n$

* $\mu = \frac{1}{n} (\sum V_1, \sum V_2, \dots, \sum V_m)$

* μ is the mean vector of the vectors of V .
So, μ_i has the smallest value of vector i .
So, Euclidean distance is the smallest.



Exercise 2 :



Shown in this figure is the illustration of XOR function that two classes, 0 for black dot and 1 for blue dot, cannot be separated with a single line. So, XOR is not a linearly separable classification problem.

Exercise 3 :

I used Python's Scikit-Learn Library for find the 5-fold cross validation and grid search.

What is Scikit-Learn?

Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms. It's built upon some of the technology you might already be familiar with, like NumPy, pandas, and Matplotlib!

The functionality that scikit-learn provides include:

- **Regression**, including Linear and Logistic Regression
- **Classification**, including K-Nearest Neighbors
- **Clustering**, including K-Means and K-Means++
- **Model selection**
- **Preprocessing**, including Min-Max Normalization

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, KFold
4 from sklearn.svm import SVC
5 from sklearn.linear_model import LogisticRegression
6
7 bankdata = pd.read_csv("bill_authentication.csv")
8 X = bankdata.drop('Class', axis=1)
9 y = bankdata['Class']
10 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 0)
11
12 linearClassifier = SVC(kernel='linear')
13 linearClassifier.fit(X_train, y_train)
14
15 print("\n -- LINEAR -- \n")
16 laccuracies = cross_val_score(estimator = linearClassifier, X = X_train, y = y_train, cv = 5)
17 print("Ortalama deger (mean): %",round(laccuracies.mean()*100,2))
18 print("std: %",round(laccuracies.std()*100))
19
20 print("\n -- POLYNOMIAL with Degree 8 -- \n")
21 poly8Classifier = SVC(kernel='poly', degree=8, gamma='auto')
22 poly8Classifier.fit(X_train, y_train)
23
24 p8accuracies = cross_val_score(estimator = poly8Classifier, X = X_train, y = y_train, cv = 5)
25 print("Ortalama deger (mean): %",round(p8accuracies.mean()*100,2))
26 print("std: %",round(p8accuracies.std()*100))
27
28 print("\n -- POLYNOMIAL with Degree 35 -- \n")
29 poly35Classifier = SVC(kernel='poly', degree=35, gamma='auto')
30 poly35Classifier.fit(X_train, y_train)
31
32 p35accuracies = cross_val_score(estimator = poly35Classifier, X = X_train, y = y_train, cv = 5)
33 print("Ortalama deger (mean): %",round(p35accuracies.mean()*100,2))
34 print("std: %",round(p35accuracies.std()*100))
35
36 print("\n -- GAUSSIAN -- \n")
37 GausClassifier = SVC(kernel='rbf', gamma='auto')
38 GausClassifier.fit(X_train, y_train)
39
40 gaccuracies = cross_val_score(estimator = GausClassifier, X = X_train, y = y_train, cv = 5)
41 print("Ortalama deger (mean): %",round(gaccuracies.mean()*100,2))
42 print("std: %",round(gaccuracies.std()*100))
```

Pic. 3.1 – The Corss Validation Part of My Code

There are four kernels which is linear, RBF, and polynomials with degrees of 8 and 35.

```

D:\Ders Notlari\4 - Pattern Recognition\HW3>python de.py

-- LINEAR --

[0.97727273 0.97727273 0.98630137 1.          0.99543379]
Ortalama deęer (mean): % 98.73
std: % 1.0

-- POLYNOMIAL with Degree 8 --

[0.98181818 0.97272727 0.97260274 0.96803653 0.96803653]
Ortalama deęer (mean): % 97.26
std: % 1.0

-- POLYNOMIAL with Degree 35 --

[0.45          0.69090909 0.44748858 0.44748858 0.71689498]
Ortalama deęer (mean): % 55.06
std: % 13.0

-- GAUSSIAN --

[1. 1. 1. 1. 1.]
Ortalama deęer (mean): % 100.0
std: % 0.0

```

Pic. 3.2 – The Output of My Code

If we compare the performance of the different types of kernels we can clearly see that RBF(Gaussian) kernel achieved a perfect %100 prediction rate. So, the Gaussian kernel performed slightly better. Linear kernel and Polynomial kernel (with smaller degree) were similar to each other. But, the performance of the Polynomial kernel decreased when the degree of the kernel increased.

```

44 logistic = LogisticRegression()
45 penalty = ['l1','l2']
46 C = np.logspace(0,4,10)
47 hyperparameters = dict(C=C,penalty = penalty)
48 clf = GridSearchCV(logistic,hyperparameters,cv=5,verbose=0)
49 best_model = clf.fit(X_train,y_train)
50
51 kfold = KFold(n_splits=5,random_state=0)
52 model = LogisticRegression(C=best_model.best_estimator_.get_params()['C'],
53                             penalty = best_model.best_estimator_.get_params()['penalty'])
54
55 results = cross_val_score(model,X_train,y_train,cv=kfold)
56 print("Ortalama deęer (mean): %",round(results.mean()*100,2))
57 print("Iyilesme Derecesi = %", (1-accuracies-results).mean()*100)

```

Pic. 3.3 – The Grid Search Part of My Code

When i performed the grid search to the linear kernel, i got a -%0.01 decrease according to the 5-fold cross validation result in linear kernel.

Optional : There are two identical vectors. One is a transformed version of X and the other one is a transformed version of x dash. This is the inner product and it happens to be in infinite-dimensional space, because you are summing for 0 until infinity.