# kNN και SVM

## 1 Εισαγωγή

## 1.1 Εισαγωγή στους αλγορίθμους kNN και SVM

Ο k-Nearest Neighbors (kNN) είναι ένας αλγόριθμος μηχανικής μάθησης για τον οποίο η πρόβλεψη για ένα νέο δείγμα βασίζεται στα k κοντινότερα training δείγματα στο συγκεκριμένο δείγμα. Έτσι, η τιμή κλάσης για ένα νέο δείγμα αποφασίζεται κατά πλειοψηφία (majority vote) με βάση τις τιμές κλάσης των k κοντινότερων δειγμάτων του.

Τα SVM είναι ένας αλγόριθμος που αφορά την εύρεση ενός γραμμικού υπερεπίπεδου το οποίο διαχωρίζει τα δεδομένα. Το πρόβλημα έγκειται στη μεγιστοποίηση του περιθωρίου (margin) μεταξύ του υπερεπιπέδου και των δεδομένων:

$$\text{Μεγιστοποίηση } Margin = 2/\|w^2\| \text{ με δεδομένο ότι } f(x) = \begin{cases} 1 & if\ w \cdot x + b \geq 1 \\ -1 & if\ w \cdot x + b \leq 1 \end{cases}$$

Σε περίπτωση που το πρόβλημα δεν επιλύεται γραμμικά, είναι δυνατή η αντιστοίχιση του σετ εκπαίδευσης σε χώρο ανώτερης διάστασης με τη χρήση kernels, ώστε πλέον το σετ εκπαίδευσης να είναι γραμμικά διαχωρίσιμο.

## 1.2 kNN στην Python

Για τον αλγόριθμο kNN στην Python, μπορούμε να χρησιμοποιήσουμε τη βιβλιοθήκη `KNeighborsClassifier`:

```
>>> from sklearn.neighbors import KNeighborsClassifier
```

Για να ταξινομήσουμε μια νέα τιμή τρέχουμε τις εντολές:

```
>>> clf = KNeighborsClassifier(n_neighbors=n_neighbors)
>>> clf.fit(X, y)
>>> clf.predict([new_data])
```

όπου δηλώνουμε τα δεδομένα εκπαίδευσης, το νέο δείγμα και την τιμή του k, ενώ με τη συνάρτηση `predict_proba` δίνονται επιπλέον οι πιθανότητες για την κλάση.

## 1.3 SVM στην Python

Για την κατασκευή ενός SVM στην Python, μπορούμε να χρησιμοποιήσουμε τη βιβλιοθήκη svm:

```
>>> from sklearn import svm
```

Για να κατασκευάσουμε ένα μοντέλο τρέχουμε την εντολή:

```
>>> clf = svm.SVC()
```

```
>>> clf.fit(X, y)
```

όπου με το kernel επιλέγουμε τον πυρήνα του SVM (linear, polynomial, radial) ενώ μπορούμε να ορίσουμε επιπλέον παραμέτρους, όπως τα degree, gamma, coef0.

Έχοντας ένα μοντέλο, για να προβλέψουμε νέες τιμές τρέχουμε την εντολή

```
>>> clf.predict([[2., 2.]])
```

Αν θέλουμε επιπλέον να επιστρέψουμε τις πιθανότητες για κάθε κλάση θα πρέπει να τρέξουμε την εντολή `predict_proba`.

## 2 Κατασκευή Μοντέλου kNN και Κατάταξη Τιμών

Για την κατασκευή ενός μοντέλου kNN θα χρησιμοποιήσουμε ως εφαρμογή τα δεδομένα εκπαίδευσης του παρακάτω πίνακα για ένα πρόβλημα δυαδικής ταξινόμησης.

| **X1** | **X2** | **Y** |
|---|---|---|
| 0.7 | 0.7 | 1 |
| 0.7 | 0.8 | 1 |
| 0.6 | 0.6 | 1 |
| 0.5 | 0.5 | 1 |
| 0.5 | 0.6 | 1 |
| 0.5 | 0.7 | 1 |
| 0.5 | 0.8 | 1 |
| 0.7 | 0.5 | 2 |
| 0.8 | 0.7 | 2 |
| 0.8 | 0.5 | 2 |
| 0.8 | 0.6 | 2 |
| 1.0 | 0.3 | 2 |
| 1.0 | 0.5 | 2 |
| 1.0 | 0.6 | 2 |

Θα απαντήσουμε στα παρακάτω ερωτήματα:

α) Σχεδιάστε τα δεδομένα με διαφορετικά χρώματα/σύμβολα για κάθε κλάση.

β) Χρησιμοποιώντας τον kNN με k = 1, σε ποια κλάση θα κατατάσσατε μία νέα παρατήρηση με τιμές (X1, X2) = (0.7, 0.4);

γ) Επαναλάβετε το ερώτημα (β) χρησιμοποιώντας k = 5.

## 2.1 Εισαγωγή Δεδομένων και Βιβλιοθηκών

Αρχικά διαβάζουμε τα δεδομένα και φορτώνουμε τις απαραίτητες βιβλιοθήκες:

```
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>> knndata = pd.read_csv("./knndata.txt")
```

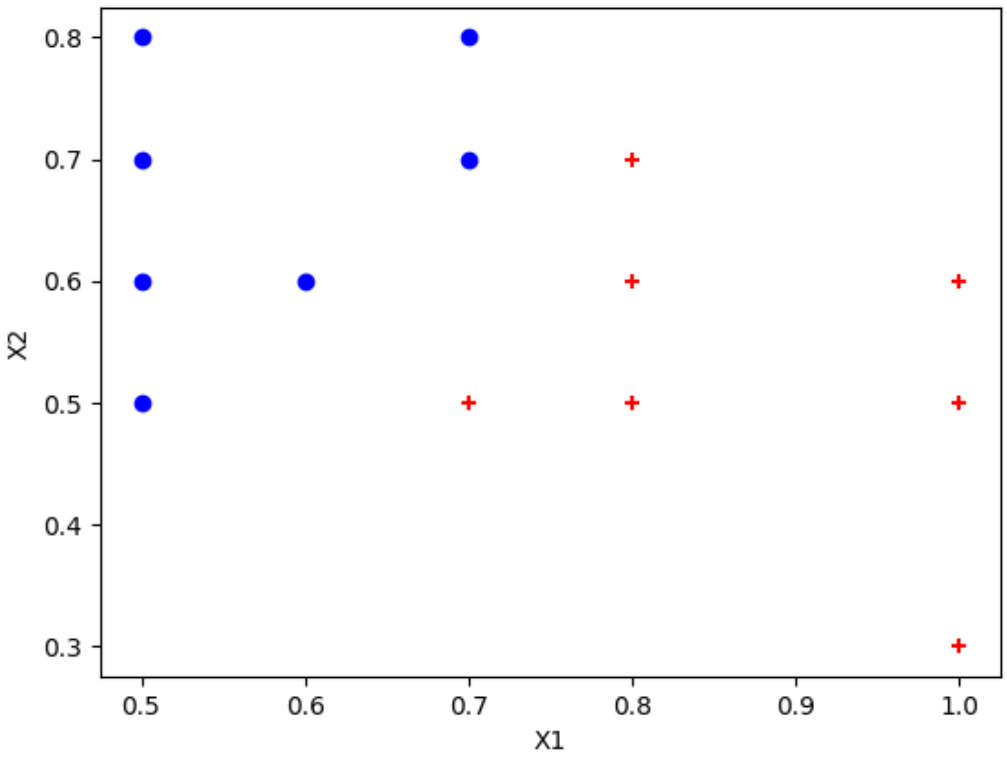
## 2.2 Εφαρμογή Αλγορίθμου kNN

Χωρίζουμε αρχικά το dataset όπως παρακάτω:

```
>>> X = knndata.loc[:, ["X1", "X2"]]
>>> y = knndata.Y
```

Για το ερώτημα (α) εκτελούμε :

```
>>> plt.scatter(X[(y == 2)].X1, X[(y == 2)].X2, c="red", marker="+")
>>> plt.scatter(X[(y == 1)].X1, X[(y == 1)].X2, c="blue", marker="o")
>>> plt.show()
```



Για k = 1 (ερώτημα (β)), αρκεί να βρούμε την κοντινότερη παρατήρηση (ευκλείδεια απόσταση) στην (0.7, 0.4), η οποία είναι η (0.7, 0.5) που ανήκει στην κλάση 2. Οπότε εκτελώντας τις εντολές:

```
>>> from sklearn.neighbors import KNeighborsClassifier
>>> clf = KNeighborsClassifier(n_neighbors=1)
>>> clf = clf.fit(X, y)
>>> print(clf.predict([[0.7, 0.4]]))
>>> print(clf.predict_proba([[0.7, 0.4]]))
```

προκύπτει ότι η νέα παρατήρηση ανήκει στην κλάση 2 με πιθανότητα 1.

Για k = 5 (ερώτημα (γ)), θα βρούμε τις 5 πιο κοντινές παρατηρήσεις, που είναι οι (0.7, 0.5), (0.8, 0.5), (0.6, 0.6), (0.8, 0.6), (0.7, 0.7) που ανήκουν στις κλάσεις 2, 2, 1, 2, 1 αντίστοιχα. Άρα αφού τα 3/5 των παρατηρήσεων αυτών ανήκουν στην κλάση 2 εκτελώντας τις εντολές:

```
>>> clf = KNeighborsClassifier(n_neighbors=5)
>>> clf = clf.fit(X, y)
>>> print(clf.predict([[0.7, 0.4]]))
>>> print(clf.predict_proba([[0.7, 0.4]]))
```

προκύπτει ότι η νέα παρατήρηση ανήκει στην κλάση 2 με πιθανότητα 0.6.

## 3 Κατασκευή Μοντέλου SVM και Εφαρμογή με Cross-validation

Για την κατασκευή ενός μοντέλου SVM θα χρησιμοποιήσουμε ως εφαρμογή τα δεδομένα εκπαίδευσης του αρχείου που δίνεται (alldata.txt) για ένα πρόβλημα δυαδικής ταξινόμησης.

Ένα summary των δεδομένων είναι το παρακάτω:

```
       X1                    X2                   y
 Min.   :-3.25322007   Min.   :-4.664161   Min.   :1.0
 1st Qu.:-0.70900886   1st Qu.: 0.625361   1st Qu.:1.0
 Median :-0.01162501   Median : 1.641370   Median :1.5
 Mean   : 0.03493570   Mean   : 1.939284   Mean   :1.5
 3rd Qu.: 0.73337179   3rd Qu.: 3.115350   3rd Qu.:2.0
 Max.   : 3.63957363   Max.   : 9.784076   Max.   :2.0
```

Αρχικά, εισάγουμε τα δεδομένα και τα διαχωρίζουμε σε training set και test set:

```
>>> alldata = pd.read_csv("./alldata.txt")
>>> xtrain = alldata.loc[0:600, ["X1", "X2"]]
>>> ytrain = alldata.loc[0:600, "y"]
>>> xtest = alldata.loc[600:800, ["X1", "X2"]]
>>> ytest = alldata.loc[600:800, "y"]
```

Στη συνέχεια, θα απαντήσουμε στα παρακάτω ερωτήματα:

α) Σχεδιάστε τα training data με διαφορετικά χρώματα/σύμβολα για κάθε κλάση.

β) Εφαρμόστε τον SVM με RBF kernel και την παράμετρο gamma ίση με 1 και σχεδιάστε το ύπερ-επίπεδο στο σχήμα του ερωτήματος (α).

γ) Επαναλάβετε το ερώτημα (β) για τιμές του gamma ίσες με 0.01 και με 100. Τι παρατηρείτε;

Χρησιμοποιώντας για την παράμετρο gamma τις τιμές

```
gammavalues = [0.01, 0.1, 1, 10, 100, 1000, 10000, 100000, 1000000]
```

θα απαντήσουμε επιπλέον στα παρακάτω ερωτήματα:

δ) Υπολογίστε το σφάλμα εκπαίδευσης (training error) και το σφάλμα ελέγχου (testing error) στο training set και στο test set αντίστοιχα.

ε) Σχεδιάστε τα δύο σφάλματα στο ίδιο διάγραμμα. Τι παρατηρείτε;

στ) Εφαρμόστε 10-fold cross validation για να βρείτε την καλύτερη τιμή για το gamma.

## 3.1 Σχεδίαση Δεδομένων

Για το ερώτημα (α) εκτελούμε:

```
>>> plt.scatter(xtrain[(ytrain == 2)].X1, xtrain[(ytrain == 2)].X2,
c="red", marker="+")
>>> plt.scatter(xtrain[(ytrain == 1)].X1, xtrain[(ytrain == 1)].X2,
c="blue", marker="o")
>>> plt.show()
```

## 3.2 Εφαρμογή Αλγορίθμου SVM και Σχεδίαση Υπερεπίπεδου

Για τα ερωτήματα (β), (γ), κατασκευάζουμε αρχικά ένα grid πάνω στο οποίο θα σχεδιάσουμε τα υπερεπίπεδα:

```
>>> import numpy as np
>>> X1 = np.arange (min(xtrain.X1.tolist()), max(xtrain.X1.tolist()), 0.01)
>>> X2 = np.arange (min(xtrain.X2.tolist()), max(xtrain.X2.tolist()), 0.01)
xx, yy = np.meshgrid(X1, X2)
```

Για το (β) κάνουμε train το svm για gamma = 1 και το εφαρμόζουμε στα σημεία του grid:

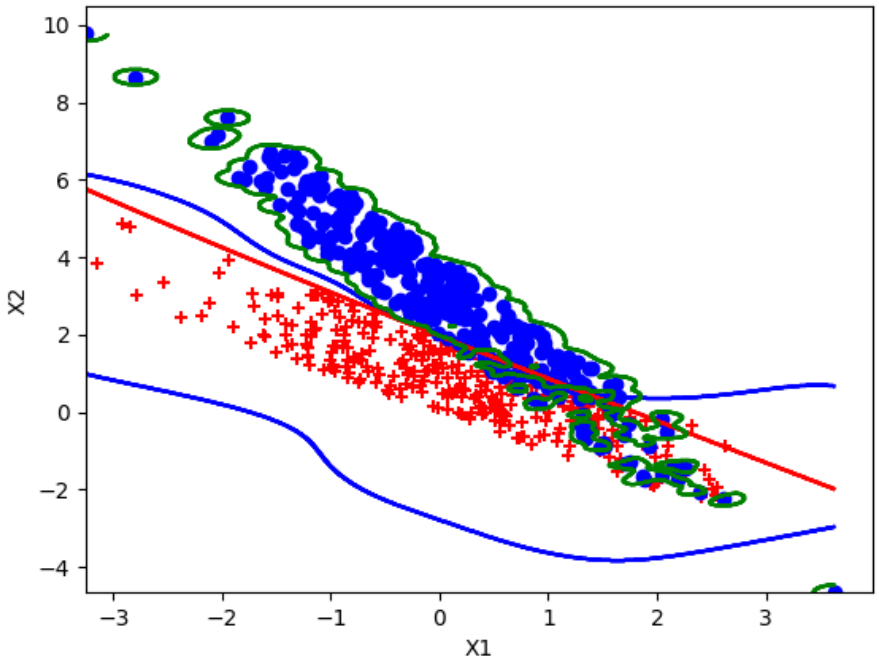
```
>>> from sklearn import svm
>>> clf = svm.SVC(kernel="rbf", gamma=1)
>>> clf = clf.fit(xtrain, ytrain)
>>> pred = clf.predict(np.c_[xx.ravel(), yy.ravel()])
>>> pred = pred.reshape(xx.shape)
>>> plt.contour(xx, yy, pred, colors="blue")
```

Όμοια για gamma ίσο με 0.01 και για gamma ίσο με 100 (ερώτημα (γ)), προκύπτει:

```
>>> clf = svm.SVC(kernel="rbf", gamma=0.01)
>>> clf = clf.fit(xtrain, ytrain)
>>> pred = clf.predict(np.c_[xx.ravel(), yy.ravel()])
>>> pred = pred.reshape(xx.shape)
>>> plt.contour(xx, yy, pred, colors="red")

>>> clf = svm.SVC(kernel="rbf", gamma=100)
>>> clf = clf.fit(xtrain, ytrain)
>>> pred = clf.predict(np.c_[xx.ravel(), yy.ravel()])
>>> pred = pred.reshape(xx.shape)
>>> plt.contour(xx, yy, pred, colors="green")
```

Έτσι, τελικά προκύπτει το παρακάτω διάγραμμα:




## 3.3 Σχεδίαση Καμπυλών Training και Testing Error

Για το ερώτημα (δ) εκτελούμε μια for για τις τιμές του gamma και υπολογίζουμε το σφάλμα εκπαίδευσης για κάθε τιμή:

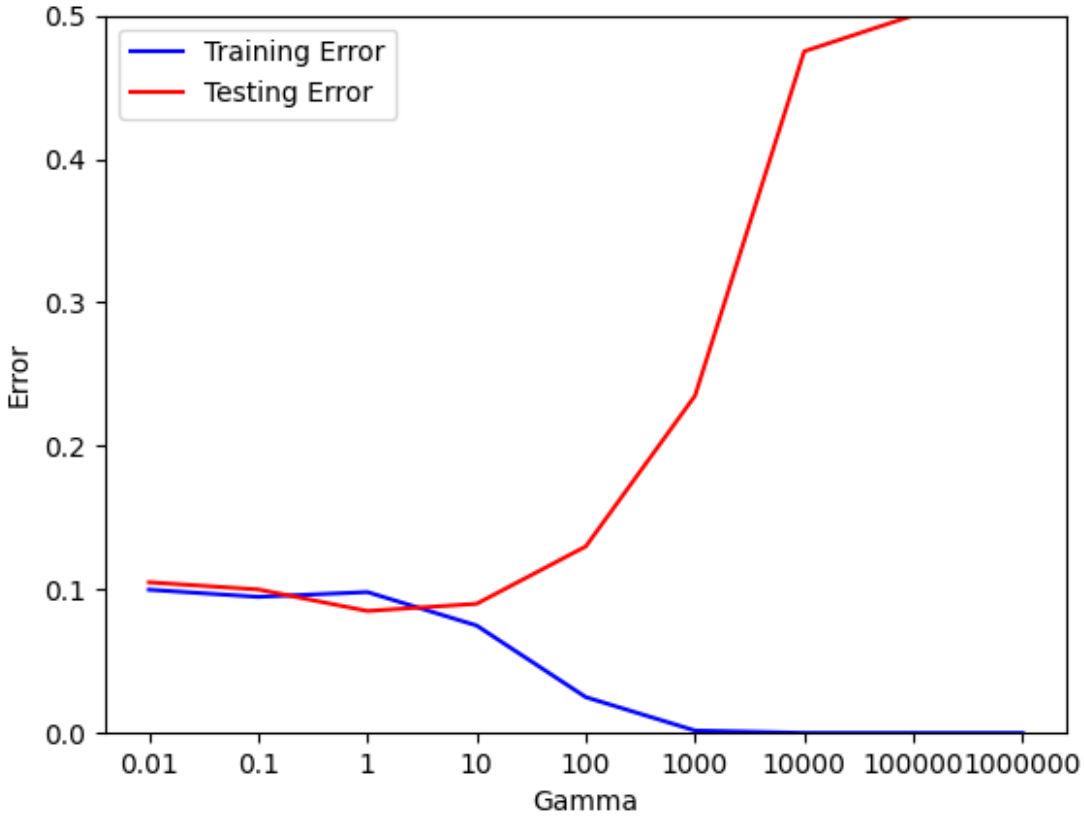
```
>>> gammavalues = [0.01, 0.1, 1, 10, 100, 1000, 10000, 100000,
1000000]


>>> trainingError = []
>>> testingError = []
>>> for gamma in gammavalues:
>>>     clf = svm.SVC(kernel="rbf", gamma=gamma)
>>>     clf = clf.fit(xtrain, ytrain)
>>>     pred = clf.predict(xtrain)
>>>     trainingError.append(1 - accuracy_score(ytrain, pred))
>>>     pred = clf.predict(xtest)
>>>     testingError.append(1 - accuracy_score(ytest, pred))
```

Σχεδιάζουμε τα δύο σφάλματα στο ίδιο διάγραμμα με τις παρακάτω εντολές:

```
>>> plt.plot(trainingError, c="blue")
>>> plt.plot(testingError, c="red")
>>> plt.ylim(0, 0.5)
>>> plt.xticks(range(len(gammavalues)), gammavalues)
>>> plt.legend(["Training Error", "Testing Error"])
>>> plt.xlabel("Gamma")
>>> plt.ylabel("Error")
>>> plt.show()
```

Οπότε προκύπτει το παρακάτω διάγραμμα:



## 3.4 Εφαρμογή k-fold Cross Validation

Θα εφαρμόσουμε k-fold cross validation για να υπολογίσουμε τη βέλτιστη τιμή για την παράμετρο gamma (ερώτημα (στ)). Δημιουργούμε ένα βρόχο επανάληψης για το gamma, και εντός του βρόχου δημιουργούμε τα folds.

```
>>> from sklearn.model_selection import cross_val_score
>>> accuracies = []
>>> for gamma in gammavalues:
>>>     clf = svm.SVC(kernel="rbf", gamma=gamma)
>>>     scores = cross_val_score(clf, xtrain, ytrain, cv=10)
>>>     accuracies.append(scores.mean())
```

Για κάθε fold υπολογίζουμε το accuracy και τελικά επιλέγουμε το gamma με το μέγιστο accuracy:

```
>>> print("Best gamma: ", gammavalues[np.argmax(accuracies)])
```