Τοπολογική Ανάλυση Δεδομένων

Εαρινό Εξάμηνο 2026

Φροντιστηριακή Διάλεξη 4: Μηχανική Μάθηση & Τοπολογική Ανάλυση Δεδομένων

Διδάσκοντες:

Αθανάσιος Ανδρικόπουλος

Ιωάννης Γουναρίδης

Επικουρικό Έργο: Αλέξανδρος - Μάριος Αφράτης

Πάτρα, 26 Μαρτίου 2026

Πίνακας Περιεχομένων

Εισαγωγή Βιβλιοθηκών (Importing Libraries)

Κύριες Εισαγωγές

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_openml
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import time
import warnings
warnings.filterwarnings('ignore')

# Ορισμός παραμέτρου ψευδοστοχαστικότητας
np.random.seed(42)
plt.style.use('seaborn-v0_8-whitegrid')
plt.rcParams['figure.figsize'] = [10, 6]

```

Επιβλεπόμενη Μάθηση (Supervised Learning)

Ορισμός

Δίνεται σύνολο εκπαίδευσης $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ *όπου* $\mathbf{x}_i \in \mathbb{R}^d$ *είναι το* ***διάνυσμα χαρακτηριστικών*** *(feature vector) και* $y_i \in \mathcal{Y}$ *η* ***ετικέτα*** *(label). Στόχος: εκμάθηση συνάρτησης* $f : \mathbb{R}^d \to \mathcal{Y}$.

Ορισμός

Ταξινόμηση *(Classification): επιβλεπόμενη μάθηση όπου* $\mathcal{Y} = \{c_1, c_2, \ldots, c_K\}$ *είναι πεπερασμένο διακριτό σύνολο. Κάθε* c_k *ονομάζεται* ***κλάση*** *(class).*

Ορισμός

Το ***χώρος υποθέσεων*** *(hypothesis space)* $\mathcal{H}$ *είναι το σύνολο όλων των δυνατών συναρτήσεων που μπορεί να εκμάθει ο αλγόριθμος:*

- *Γραμμικοί ταξινομητές:* $f(\mathbf{x}) = \mathrm{sign}(\mathbf{w}^T\mathbf{x} + b)$
- *Δέντρα απόφασης: τμηματικά σταθερές συναρτήσεις*

Πίνακας Περιεχομένων

Σύνολο Δεδομένων MNIST (MNIST Dataset)

Περιγραφή (Description)

Το MNIST αποτελείται από **γκρι εικόνες** 28×28 **pixels** χειρόγραφων ψηφίων (0–9). Κάθε εικόνα αναπαρίσταται ως:

- Μητρώο $\mathbf{X} \in \mathbb{R}^{28\times28}$, $X_{ij} \in [0, 255]$ (ένταση pixel)
- ή ισοδύναμα ως εκτεταμένο διάνυσμα $\mathbf{x} \in \mathbb{R}^{784}$

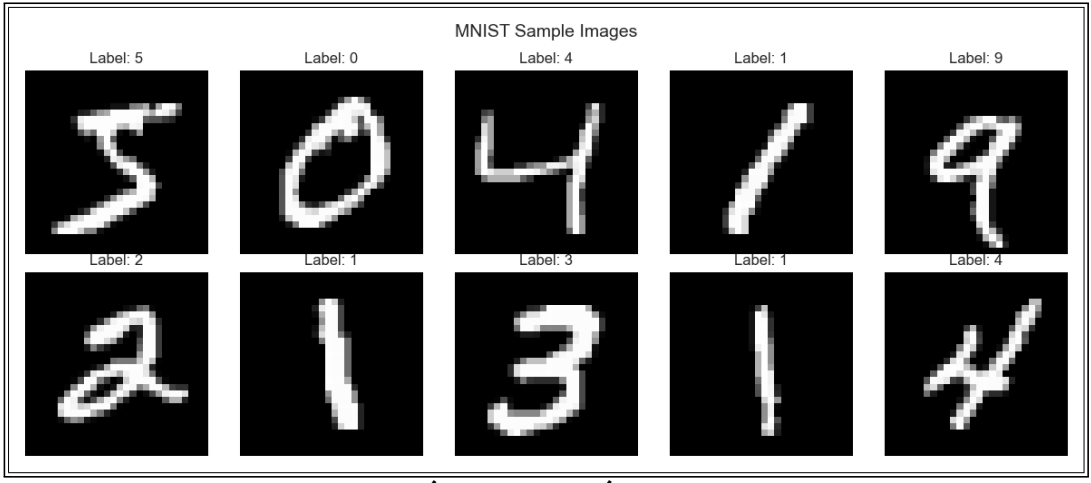
Φόρτωση (Loading)

```
1       mnist = fetch_openml('mnist_784', version=1, as_frame=False)
2       X_full = mnist.data           # shape: (70000, 784)
3       y_full = mnist.target.astype(int)
4       print(f"Dataset loaded: {X_full.shape[0]} samples, {X_full.shape[1]} features")
5
```

Σημείωση

Σύνολο δεδομένων: **70.000 δείγματα**, 784 χαρακτηριστικά (pixels), 10 κλάσεις (ψηφία 0–9).

Οπτικοποίηση (Visualization)



Σχ. 4.1: Σύνολο Δεδομένων MNIST

Επιλογή Υποσυνόλου (Subset Selection)

Επιλογή 5 Οπτικά Όμοιων Ψηφίων

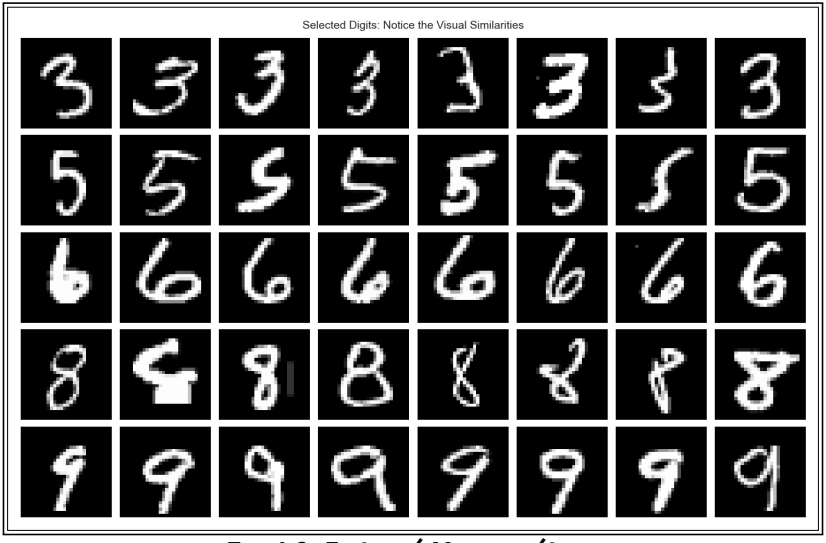
Επιλέγουμε $\mathcal{Y} = \{3, 5, 6, 8, 9\}$ – ψηφία που μοιάζουν οπτικά και δυσκολεύουν την ταξινόμηση.

```
selected_digits = [3, 5, 6, 8, 9]
samples_per_class = 1000   # 1000 δείγματα ανά κλάση

X_list, y_list = [], []
for digit in selected_digits:
mask = y_full == digit
indices = np.random.choice(np.where(mask)[0], samples_per_class, replace=False)
X_list.append(X_full[indices])
y_list.append(y_full[indices])

X = np.vstack(X_list)   # shape: (5000, 784)
y = np.hstack(y_list)   # shape: (5000,)
print(f"Selected dataset: {X.shape[0]} samples, Classes: {selected_digits}")

```

Κίνητρο Χρήσης TDA

Τα ψηφία 6 και 9 έχουν από μία οπή (similar topology), ενώ τα 3 και 5 δεν έχουν οπές. Αυτή η δυσκολία ταξινόμησης επιλύεται αποδοτικά μέσω TDA.

Οπτικοποίηση (Visualization)



Σχ. 4.2: Επιλογή Υποσυνόλου

Κανονικοποίηση Χαρακτηριστικών (Feature Normalization)

Ορισμός

Κανονικοποίηση Min-Max *(Min-Max Normalization): για χαρακτηριστικό x με εύρος* $[x_{\min}, x_{\max}]$*:*

$$\tilde{x} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Για τιμές pixel: $x \in [0, 255] \to \tilde{x} \in [0, 1]$.

Γιατί Κανονικοποίηση;

- Πολλοί αλγόριθμοι (SVM, K-NN) χρησιμοποιούν μετρικές αποστάσεων ευαίσθητες στις κλίμακες
- Η βελτιστοποίηση βαθμίδας (gradient-based optimization) συγκλίνει ταχύτερα με κανονικοποιημένα χαρακτηριστικά
- Αποτρέπει χαρακτηριστικά με μεγάλο εύρος να κυριαρχούν

Παράδειγμα Κανονικοποίησης:

```
1    X_normalized = X / 255.0   # [0,255] -> [0,1]
```

Διαχωρισμός Εκπαίδευσης – Ελέγχου (Train-Test Split)

Ορισμός

Βλάβη Γενικότητας – Σφάλμα Γενίκευσης *(Generalization Error): αναμενόμενο σφάλμα σε προηγουμένως άγνωστες εισοδους (δεδομένα):*

$$R(f) = \mathbb{E}_{(\mathbf{x},y)\sim P}\big[\mathcal{L}(f(\mathbf{x}), y)\big]$$

Ορισμός

Διαχωρισμός εκπαίδευσης-ελέγχου:
$\mathcal{D} = \mathcal{D}_{train} \cup \mathcal{D}_{test},\ \mathcal{D}_{train} \cap \mathcal{D}_{test} = \emptyset.$

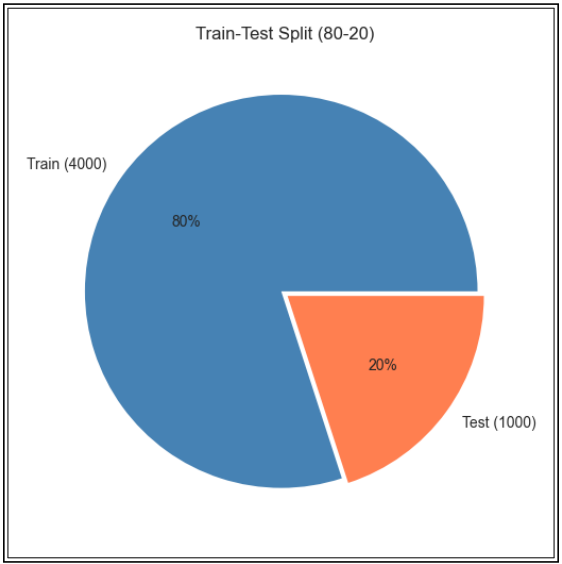
Κώδικας (80% – 20%)

```
1           X_train, X_test, y_train, y_test =
2           train_test_split(
3           X_normalized, y,
4           test_size=0.2,
5           random_state=42,
6           stratify=y)
7           # Training set: 4000 samples
8           # Test set:     1000 samples
9
```

Διαισθητικά

Η εκπαίδευση θυμίζει μελέτη παλαιών θεμάτων εξεταστικής και ο έλεγχος είναι σαν τελική εξέταση με νέες ερωτήσεις.

Οπτικοποίηση (Visualization)



Σχ. 4.3: Διαχωρισμός Εκπαίδευσης – Ελέγχου

Μείωση Διαστατάσεων για Οπτικοποίηση (PCA)

Ορισμός

Ανάλυση Κύριων Συνιστωσών *(Principal Component Analysis – PCA): βρίσκει ορθογώνιες κατευθύνσεις μέγιστης διακύμανσης. Για κεντραρισμένο πίνακα* $\mathbf{X} \in \mathbb{R}^{n \times d}$*:*

$$\mathbf{X} = \mathbf{U}\mathbf{V}^T, \quad \mathbf{Z} = \mathbf{X}\mathbf{V}_k \in \mathbb{R}^{n \times k}$$

Για οπτικοποίηση: $k = 2$*.*

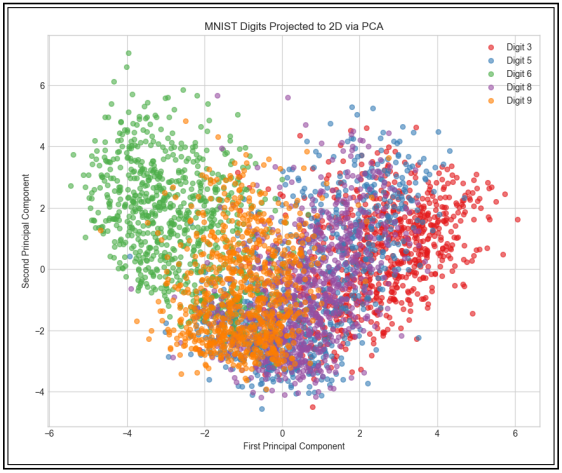
Κώδικας PCA

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_train_2d = pca.fit_transform(
X_train)
print(f"Variance explained: "
f"{pca.explained_variance_ratio_
  .sum()*100:.1f}%")

```

Ερμηνεία

Τα ψηφία 6 και 9 (μία οπή) ομαδοποιούνται μαζί, όπως και τα 3 και 5 (χωρίς οπές) – η τοπολογική δομή επηρεάζει την κατανομή!

Οπτικοποίηση (Visualization)



Σχ. 4.4: Μείωση Διαστατάσεων για Οπτικοποίηση (PCA)

Πίνακας Περιεχομένων

Ορισμός

Πίνακας Σύγχυσης *(Confusion Matrix): $C \in \mathbb{R}^{K \times K}$ όπου C_{ij} = αριθμός δειγμάτων με αληθινή ετικέτα i που προβλέφθηκαν ως j.*

- ***Διαγώνιες εγγραφές*** *C_{ii}: σωστές προβλέψεις (True Positives κλάσης i)*
- ***Εκτός διαγωνίου*** *C_{ij} $(i \neq j)$: σφάλματα ταξινόμησης*

Ορισμός

Ακρίβεια *(Accuracy):*

$$Accuracy = \frac{\text{Αριθμός σωστών προβλέψεων}}{\text{Συνολικός αριθμός προβλέψεων}} = \frac{\sum_c TP_c}{n}$$

Προσοχή*: η ακρίβεια μπορεί να είναι παραπλανητική σε μη ισορροπημένα σύνολα δεδομένων (imbalanced datasets).*

Ορισμός

Ακρίβεια ανά κλάση *(Precision):*

$$Precision_c = \frac{TP_c}{TP_c + FP_c}$$

Από όλα τα δείγματα που προβλέφθηκαν ως c, ποιο ποσοστό όντως ανήκει σε c;

Ορισμός

Ανάκληση *(Recall):*

$$Recall_c = \frac{TP_c}{TP_c + FN_c}$$

Από όλα τα δείγματα που ανήκουν σε c, ποιο ποσοστό εντοπίστηκε σωστά;

Ορισμός

F1-Score*:*

$$F1_c = 2 \cdot \frac{Precision_c \cdot Recall_c}{Precision_c + Recall_c}$$

Αρμονικός μέσος ακρίβειας και ανάκλησης – τιμωρεί οριακές τιμές.

Συμβολισμός

- TP_c: True Positives (σωστά ως c)
- FP_c: False Positives (λάθος ως c)
- FN_c: False Negatives (c προβλεφθέν αλλού)

Βοηθητικές Συναρτήσεις Αξιολόγησης (Evaluation Helper Functions)

```python
from sklearn.metrics import precision_score, recall_score, f1_score

results = {'Algorithm': [], 'Accuracy': [], 'Precision': [],
  'Recall': [], 'F1-Score': [], 'Training Time (s)': []}

def print_metrics(y_true, y_pred, name):
"""Εκτύπωση όλων των μετρικών αξιολόγησης για ένα μοντέλο."""
acc  = accuracy_score(y_true, y_pred)
prec = precision_score(y_true, y_pred, average='weighted')
rec  = recall_score(y_true, y_pred, average='weighted')
f1   = f1_score(y_true, y_pred, average='weighted')
print(f"\n{name} - Evaluation Metrics:")
print(f"Accuracy:  {acc:.4f} ({acc*100:.2f}%)")
print(f"Precision: {prec:.4f}")
print(f"Recall:    {rec:.4f}")
print(f"F1-Score:  {f1:.4f}")
return acc, prec, rec, f1

```

Πίνακας Περιεχομένων

K-Πλησιέστεροι Γείτονες – Θεωρία (K-Nearest Neighbors)

Ορισμός

Για ερώτημα $\mathbf{x}_q$, έστω $N_k(\mathbf{x}_q)$ οι k πλησιέστεροι γείτονες στο εκπαιδευτικό σύνολο. Η πρόβλεψη K-NN:

$$\hat{y} = \arg\max_{c \in \mathcal{Y}} \sum_{(\mathbf{x}_i, y_i) \in N_k(\mathbf{x}_q)} \begin{cases} 1 & \text{αν } y_i = c \\ 0 & \text{αν } y_i \neq c \end{cases}$$

Κοινές μετρικές αποστάσεων (distance metrics):

- ***Ευκλείδεια***: $d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_2$
- ***Manhattan***: $d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_1$

Ιδιότητες

- **Μη-παραμετρικός** (non-parametric): χωρίς παραδοχές για κατανομή
- **Lazy learning**: χωρίς ρητή φάση εκπαίδευσης

Διαισθητικά

"Είσαι ο μέσος όρος των *k* κοντινότερων φίλων σου." (Ταξινόμηση με πλειοψηφική ψηφοφορία γειτόνων).

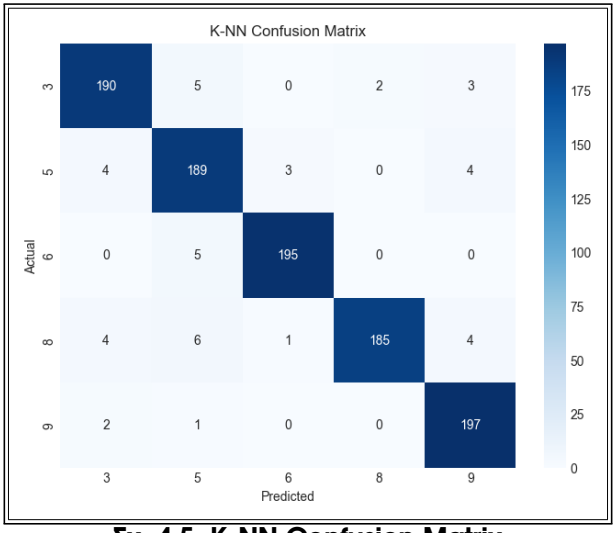
K-NN: Κώδικας & Αποτελέσματα

```python
from sklearn.neighbors import KNeighborsClassifier

# Εκπαίδευση K-NN με k=5
knn = KNeighborsClassifier(n_neighbors=5)
start = time.time()
knn.fit(X_train, y_train)
knn_time = time.time() - start

knn_pred = knn.predict(X_test)

# Εκτύπωση μετρικών
knn_acc, knn_prec, knn_rec, knn_f1 = print_metrics(y_test, knn_pred, 'K-NN (k=5)')
print(f"Training Time: {knn_time:.4f}s")

# Αποθήκευση αποτελεσμάτων για σύγκριση
results['Algorithm'].append('K-NN')
results['Accuracy'].append(knn_acc)
results['Precision'].append(knn_prec)
results['Recall'].append(knn_rec)
results['F1-Score'].append(knn_f1)
results['Training Time (s)'].append(knn_time)

```

Ερμηνεία

Ο K-NN επιτυγχάνει υψηλή απόδοση (≈ 95–96%). Τα περισσότερα σφάλματα εμφανίζονται μεταξύ οπτικά όμοιων ζευγών: $3 \leftrightarrow 5$ και $3 \leftrightarrow 8$.

Οπτικοποίηση (Visualization)



Σχ. 4.5: K-NN Confusion Matrix

K-NN: Επίδραση της Παραμέτρου k

Συμβιβασμός Πόλωσης – Διακύμανσης (Bias-Variance Tradeoff)

- **Μικρό** k: χαμηλή πόλωση (bias), υψηλή διακύμανση (variance) – ευαίσθητο στο θόρυβο
- **Μεγάλο** k: υψηλή πόλωση, χαμηλή διακύμανση – υπερεξομάλυνση (over-smoothing)

Μέθοδος Αγκώνα (Elbow Method)

Σχεδιάζουμε το ποσοστό σφάλματος $1 - \text{accuracy}$ ως συνάρτηση του k.
Αναζητούμε το σημείο "αγκώνα" όπου το σφάλμα σταματά να μειώνεται σημαντικά.

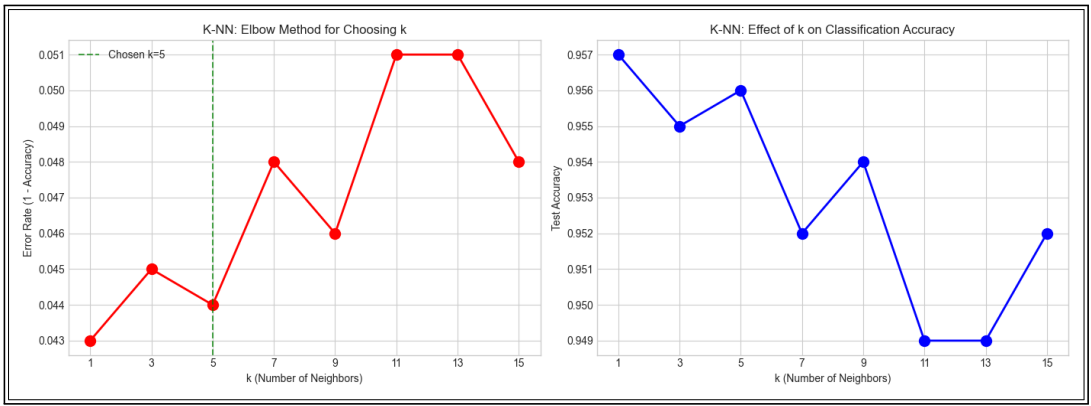
Κώδικας

```
k_values = [1, 3, 5, 7, 9, 11, 13, 15]
k_errors = []
for k in k_values:
knn_temp = KNeighborsClassifier(n_neighbors=k)
knn_temp.fit(X_train, y_train)
acc = accuracy_score(y_test, knn_temp.predict(X_test))
k_errors.append(1 - acc)
print(f"k={k:2d}: Accuracy={acc:.4f}, Error Rate={1-acc:.4f}")

```

Ερμηνεία

Για αυτό το σύνολο δεδομένων, $k = 1$ έως $k = 5$ δίνουν παρόμοιη απόδοση (≈ 95–96%) – επιλέγουμε $k = 5$ ως ισορροπία τοπικής δομής και ανθεκτικότητας στο θόρυβο.

Οπτικοποίηση (Visualization)



Σχ. 4.6: K-NN: Επίδραση της Παραμέτρου *k*

Naive Bayes: Θεωρητικό Υπόβαθρο

Ορισμός

Θεώρημα Bayes:

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})}$$

Ορισμός

Ταξινομητής Naive Bayes: *υποθέτει* ***δεσμευμένη ανεξαρτησία*** *(conditional independence) χαρακτηριστικών δοθείσης κλάσης:*

$$P(\mathbf{x}|y) = \prod_{j=1}^{d} P(x_j|y)$$

Πρόβλεψη: $\hat{y} = \arg\max_c P(y{=}c) \prod_j P(x_j|y{=}c)$

Ορισμός

Gaussian Naive Bayes: *κάθε* $P(x_j|y = c)$ *ακολουθεί κατανομή Gauss:*

$$P(x_j|y{=}c) = \frac{1}{\sqrt{2\pi\sigma_{jc}^2}} \exp\left(-\frac{(x_j - \mu_{jc})^2}{2\sigma_{jc}^2}\right)$$

Διαισθητικά

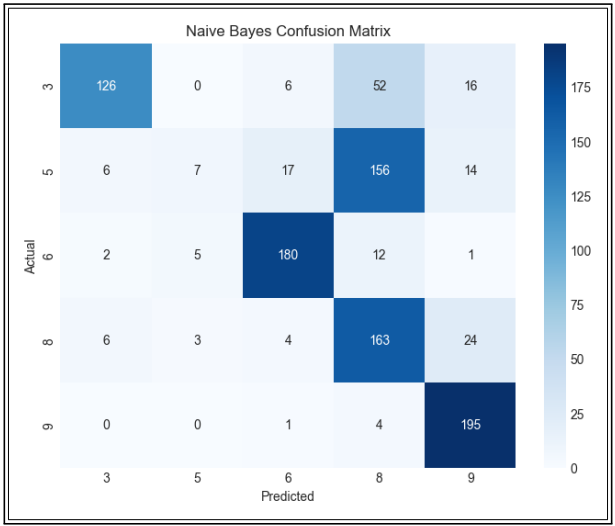
"Ποια είναι η πιθανότητα αυτό να είναι 3, δεδομένου πώς μοιάζουν τα τυπικά 3;"
Παρά την αφελή υπόθεση ανεξαρτησίας, αποδίδει καλά στην πράξη.

Naive Bayes: Κώδικας & Αποτελέσματα

```
from sklearn.naive_bayes import GaussianNB

# Εκπαίδευση Gaussian Naive Bayes
nb = GaussianNB()
start = time.time()
nb.fit(X_train, y_train)
nb_time = time.time() - start

nb_pred = nb.predict(X_test)

# Εκτύπωση μετρικών
nb_acc, nb_prec, nb_rec, nb_f1 = print_metrics(y_test, nb_pred, 'Naive Bayes')
print(f"Training Time: {nb_time:.4f}s (very fast!)")

# Αποθήκευση αποτελεσμάτων
results['Algorithm'].append('Naive Bayes')
results['Accuracy'].append(nb_acc)

```

Χαρακτηριστικά Naive Bayes

- **Πλεονέκτημα**: εξαιρετικά γρήγορη εκπαίδευση – υπολογίζει μόνο μέσους και διακυμάνσεις ανά χαρακτηριστικό και κλάση
- **Μειονέκτημα**: η υπόθεση ανεξαρτησίας χαρακτηριστικών σπάνια ισχύει για pixels (γειτονικά pixels συσχετίζονται)



Σχ. 4.7: Naive Bayes Confusion Matrix

Λογιστική Παλινδρόμηση (Logistic Regression)

Ορισμός

Λογιστική Παλινδρόμηση *(δυαδική περίπτωση): μοντελοποιεί την πιθανότητα κλάσης 1 μέσω της* ***λογιστικής (sigmoid) συνάρτησης****:*

$$P(y=1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}}$$

Για K κλάσεις: ***softmax*** *πολυκλαδική επέκταση.*

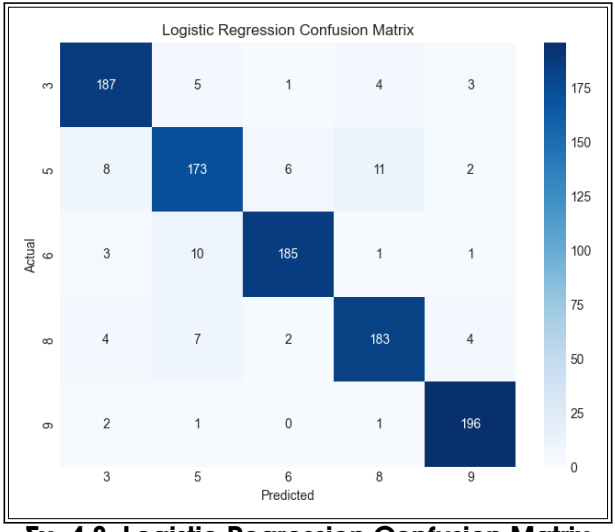
Κώδικας

```
from sklearn.linear_model \
import LogisticRegression

lr = LogisticRegression(
max_iter=1000,
random_state=42)
start = time.time()
lr.fit(X_train, y_train)
lr_time = time.time() - start
lr_pred = lr.predict(X_test)
lr_acc, lr_prec, lr_rec, lr_f1\
= print_metrics(y_test, lr_pred,
'Logistic Regression')

```

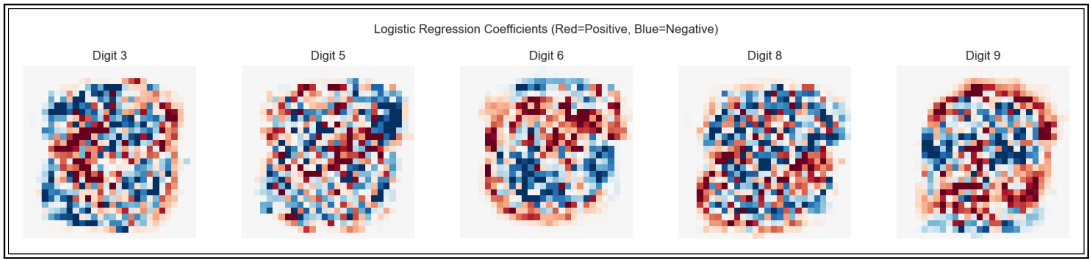
Οπτικοποίηση Συντελεστών (Coefficient Visualization)

Κάθε κλάση έχει διάνυσμα βαρών $\mathbf{w}_k \in \mathbb{R}^{784}$. Αναδιαμόρφωση σε 28×28 αποκαλύπτει ποιες περιοχές pixels συμβάλλουν σε κάθε κλάση.

```
    for i, digit in enumerate(selected_digits):
    coef = lr.coef_[i].reshape(28, 28)
    # Οπτικοποίηση με cmap='RdBu'  κόκκινοθετικό(, μπλεαρνητικό)

```



Σχ. 4.8: Logistic Regression Confusion Matrix



Σχ. 4.9: Συντελεστές Λογιστικής Παλινδρόμησης (Κόκκινο = Θετικό, Μπλε = Αρνητικό)

Ορισμός

Δέντρο Απόφασης: *αναδρομικώς διαμερίζει το χώρο χαρακτηριστικών με διαχωρισμούς παράλληλους στους άξονες. Σε κάθε εσωτερικό κόμβο:*

$$(j^*, t^*) = \arg\min_{j,t} \mathcal{L}(\mathcal{D}_L) + \mathcal{L}(\mathcal{D}_R)$$

Ορισμός

Νοθεία Gini: $G(\mathcal{D}) = 1 - \sum_{k=1}^{K} p_k^2$

Εντροπία *(Entropy)*: $H(\mathcal{D}) = -\sum_{k=1}^{K} p_k \log_2 p_k$

Ορισμός

Σημασία Χαρακτηριστικού *(Feature Importance)*:

$$Imp(j) = \sum_{\text{κόμβοι } t \text{ που διαχωρίζουν σε } j} n_t \cdot \Delta\mathcal{L}_t$$

Υπερπροσαρμογή (Overfitting)

Τα δέντρα μεγάλου βάθους απομνημονεύουν σε προβληματικό βαθμό τα δεδομένα εκπαίδευσης:

$$R_{train}(f) \ll R_{test}(f)$$

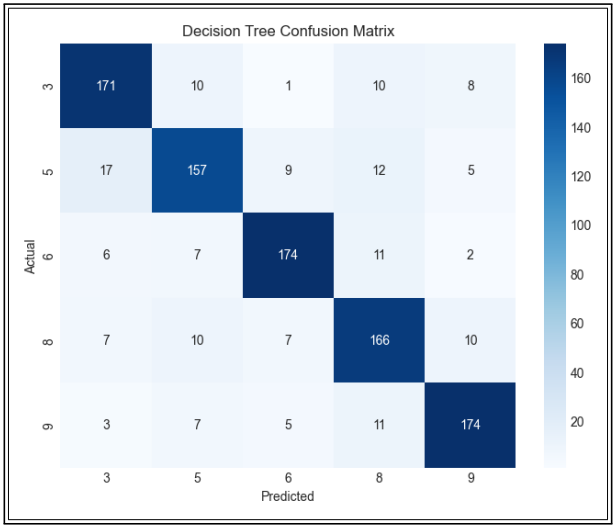
Δέντρα Απόφασης: Κώδικας & Υπερπροσαρμογή

```python
from sklearn.tree import DecisionTreeClassifier

# Εκπαίδευση Δέντρου Απόφασης (max_depth=10)
dt = DecisionTreeClassifier(max_depth=10, random_state=42)
dt.fit(X_train, y_train)
dt_pred = dt.predict(X_test)
dt_acc, dt_prec, dt_rec, dt_f1 = print_metrics(y_test, dt_pred, 'Decision Tree')

# Απόδειξη υπερπροσαρμογής για διάφορα βάθη
depths = [3, 5, 10, 15, None]
for depth in depths:
dt_temp = DecisionTreeClassifier(max_depth=depth, random_state=42)
dt_temp.fit(X_train, y_train)
train_acc = accuracy_score(y_train, dt_temp.predict(X_train))
test_acc  = accuracy_score(y_test,  dt_temp.predict(X_test))
depth_str = str(depth) if depth else 'None (unrestricted)'
print(f"Depth={depth_str}: Train={train_acc:.4f}, Test={test_acc:.4f}")

```

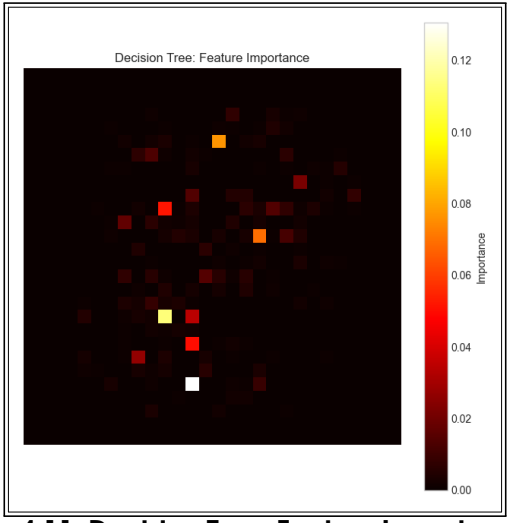
Παρατήρηση: Υπερπροσαρμογή

Καθώς το βάθος αυξάνεται, η ακρίβεια εκπαίδευσης προσεγγίζει 100% ενώ η ακρίβεια ελέγχου κορυφώνεται γύρω στο βάθος 10–15 και στη συνέχεια μειώνεται. Αυτό είναι το χαρακτηριστικό σύμπτωμα υπερπροσαρμογής.

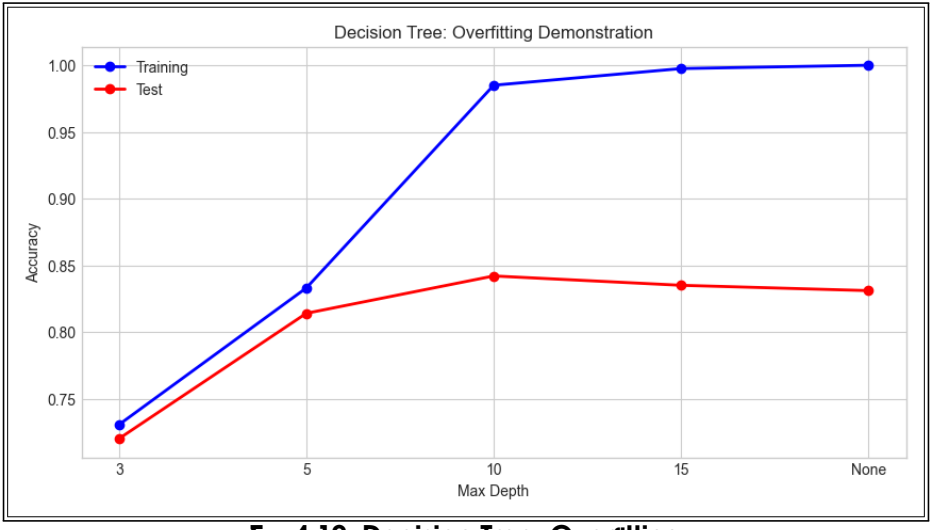
Οπτικοποίηση (Visualization)



Σχ. 4.10: Decision Tree Confusion Matrix



Σχ. 4.11: Decision Tree: Feature Importance



Σχ. 4.12: Decision Tree: Overfitting

Τυχαίο Δάσος (Random Forest)

Ορισμός

Μέθοδος Ensemble: *συνδυάζει M βασικούς εκμαθητές για μείωση διακύμανσης:*

$$\hat{y} = \arg\max_{c \in \mathcal{Y}} \sum_{m=1}^{M} \mathbb{1}[f_m(\mathbf{x}) = c]$$

Τυχαίο Δάσος (Random Forest): ensemble δέντρων απόφασης, κάθε δέντρο εκπαιδεύεται σε:

1. **Bootstrap δείγμα**: τυχαίο δείγμα με επανάθεση
2. **Τυχαίο υποσύνολο χαρακτηριστικών**: σε κάθε διαχωρισμό, μόνο $\sqrt{d}$ τυχαία χαρακτηριστικά

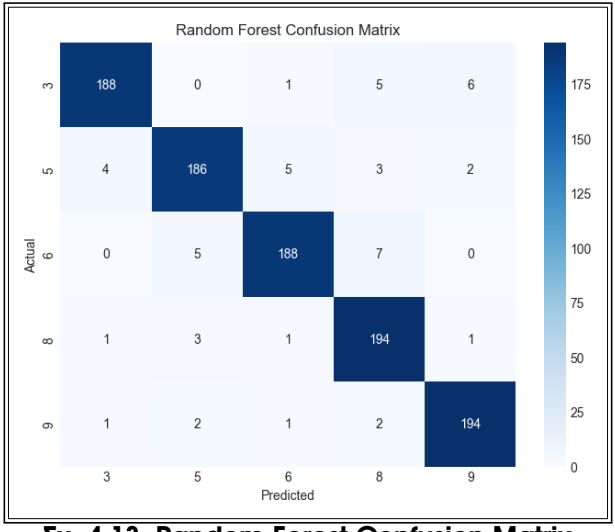
Θεώρημα

Μείωση Διακύμανσης: *για M ανεξάρτητους εκτιμητές με διακύμανση* σ^2*:*

$$\mathrm{Var}\left(\frac{1}{M}\sum_{m=1}^{M} f_m\right) = \frac{\sigma^2}{M}$$

Διαισθητικά

"Σοφία του πλήθους": πολλά ελαφρώς διαφορετικά δέντρα ψηφίζουν, εξισορροπώντας τα μεμονωμένα σφάλματα.



Σχ. 4.13: Random Forest Confusion Matrix

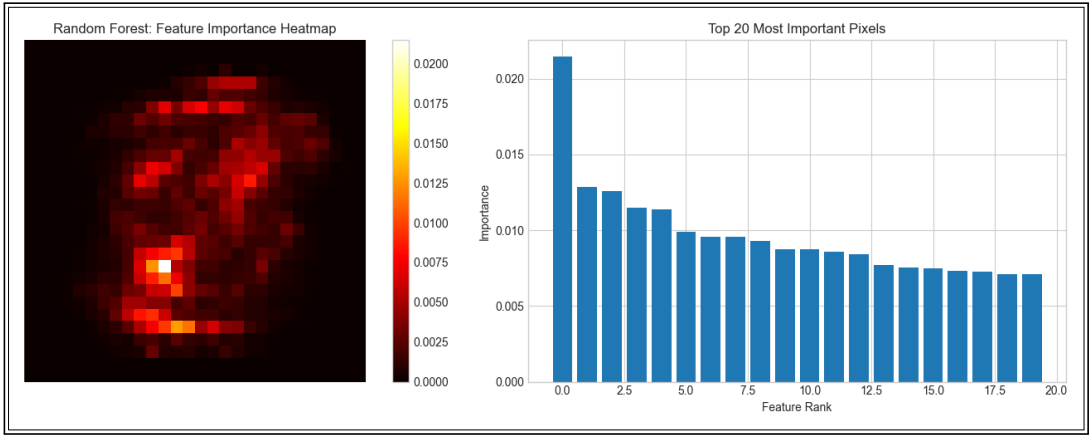
Τυχαίο Δάσος: Κώδικας & Σημασία Χαρακτηριστικών

```python
from sklearn.ensemble import RandomForestClassifier

# Εκπαίδευση Τυχαίου Δάσους (100 δέντρα)
rf = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
start = time.time()
rf.fit(X_train, y_train)
rf_time = time.time() - start

rf_pred = rf.predict(X_test)
rf_acc, rf_prec, rf_rec, rf_f1 = print_metrics(y_test, rf_pred, 'Random Forest')

# Σύγκριση μονό δέντρο vs δάσος
print(f"Single Decision Tree:       {dt_acc:.4f}")
print(f"Random Forest (100 trees):  {rf_acc:.4f}")
print(f"Improvement:                {(rf_acc - dt_acc)*100:.2f}%")

# Ανάκτηση σημασίας χαρακτηριστικών
rf_importance = rf.feature_importances_   # shape: (784,)
# Αναδιαμόρφωση σε (28,28) για οπτικοποίηση

```

Σημασία Χαρακτηριστικών

Η χαρτογράφηση σημασίας του Τυχαίου Δάσους είναι πιο ομαλή σε σχέση με μονό δέντρο, με κέντρο βάρους στην κεντρική περιοχή της εικόνας όπου γράφεται το ψηφίο. Ένα μικρό σύνολο pixels (~ 20–30) φέρει το μεγαλύτερο μέρος της προβλεπτικής ισχύος.

Οπτικοποίηση (Visualization)



Σχ. 4.14: Σημασία Χαρακτηριστικών & Κορυφαία 20 Σημαντικά Pixels

Ορισμός

SVM Σκληρού Περιθωρίου *(Hard-Margin SVM): εύρεση υπερεπιπέδου που μεγιστοποιεί το περιθώριο* $\frac{2}{\|\mathbf{w}\|}$*:*

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{υπό } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$$

Ορισμός

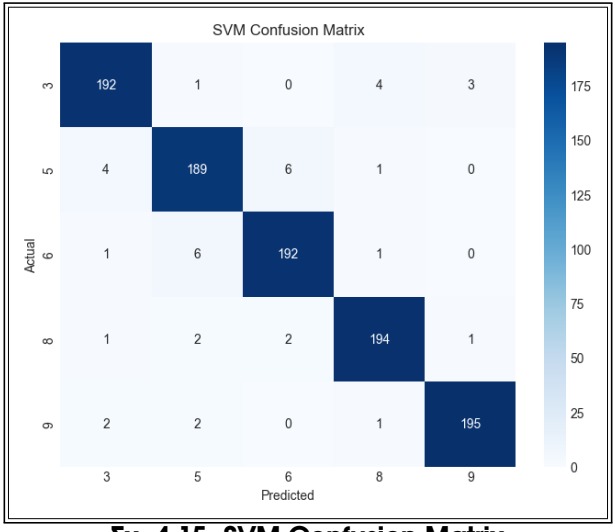
SVM Μαλακού Περιθωρίου *(Soft-Margin SVM) με μεταβλητές χαλάρωσης* $\xi_i \geq 0$*:*

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i$$

Ορισμός

Τέχνασμα Πυρήνα *(Kernel Trick): αντιστοίχιση σε υψηλότερες διαστάσεις μέσω συνάρτησης πυρήνα* $k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$*.*

- ***Γραμμικός****:* $k = \mathbf{x}^T\mathbf{x}'$
- ***RBF****:* $k = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$
- ***Πολυωνυμικός****:* $k = (\mathbf{x}^T\mathbf{x}' + c)^d$



Σχ. 4.15: SVM Confusion Matrix

SVM: Κώδικας & Αποτελέσματα

```python
from sklearn.svm import SVC

# Εκπαίδευση SVM με πυρήνα RBF
svm = SVC(kernel='rbf', random_state=42)
start = time.time()
svm.fit(X_train, y_train)
svm_time = time.time() - start

svm_pred = svm.predict(X_test)
svm_acc, svm_prec, svm_rec, svm_f1 = print_metrics(y_test, svm_pred, 'SVM')
print(f"Training Time: {svm_time:.4f}s")

# Αποθήκευση αποτελεσμάτων
results['Algorithm'].append('SVM')
results['Accuracy'].append(svm_acc)
results['Precision'].append(svm_prec)
results['Recall'].append(svm_rec)
results['F1-Score'].append(svm_f1)
results['Training Time (s)'].append(svm_time)

```

Ερμηνεία

Το SVM με πυρήνα RBF επιτυγχάνει κορυφαία απόδοση σε αυτό το σύνολο δεδομένων. Η ανά-κλάση ανάλυση δείχνει F1-scores 94–98%, με το ψηφίο 8 (δύο οπές – διακριτή τοπολογία) να επιτυγχάνει τις υψηλότερες βαθμολογίες.

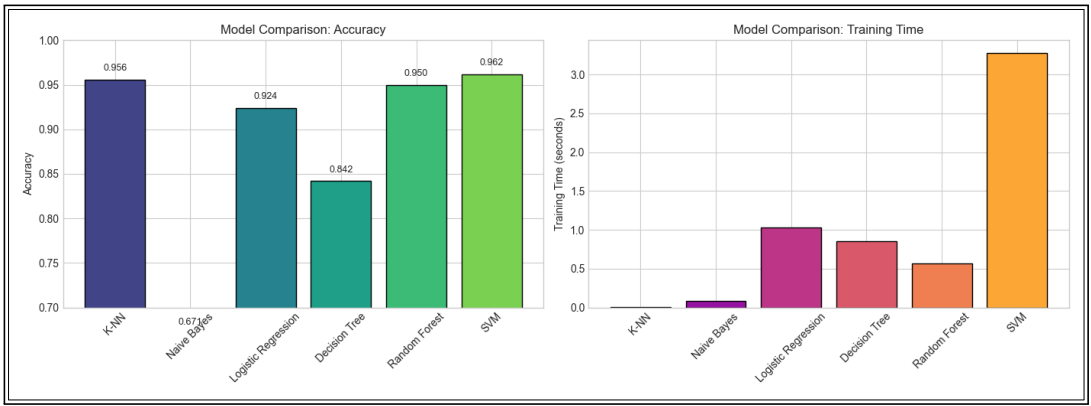
Πίνακας Περιεχομένων

Πίνακας Σύνοψης Απόδοσης

```
    comparison_df = pd.DataFrame(results)
    comparison_df = comparison_df.sort_values('Accuracy', ascending=False)
    print(comparison_df.to_string(index=False))
```

Αλγόριθμος	Accuracy	Precision	Recall	F1-Score
SVM (RBF)	≈ 0.97	≈ 0.97	≈ 0.97	≈ 0.97
Random Forest	≈ 0.96	≈ 0.96	≈ 0.96	≈ 0.96
K-NN ($k = 5$)	≈ 0.96	≈ 0.96	≈ 0.96	≈ 0.96
Logistic Regression	≈ 0.91	≈ 0.91	≈ 0.91	≈ 0.91
Decision Tree	≈ 0.88	≈ 0.88	≈ 0.88	≈ 0.88
Naive Bayes	≈ 0.78	≈ 0.79	≈ 0.78	≈ 0.78

Οπτικοποίηση (Visualization)



Σχ. 4.16: Σύγκριση Μοντέλων

Οδηγός Επιλογής Αλγορίθμου (Algorithm Selection Guidelines)

Κριτήρια Επιλογής

Κριτήριο	**Καλύτερη Επιλογή**
Υψηλότερη ακρίβεια	SVM, Random Forest
Ταχύτερη εκπαίδευση	Naive Bayes
Ερμηνευσιμότητα (interpretability)	Decision Tree, Logistic Regression
Σημασία χαρακτηριστικών	Random Forest, Decision Tree
Μη-γραμμικά όρια απόφασης	SVM (RBF), Random Forest
Μεγάλα δεδομένα	Logistic Regression, Naive Bayes

Κεντρικό Συμπέρασμα

Δεν υπάρχει καθολικά "καλύτερος" αλγόριθμος (No Free Lunch Theorem). Η επιλογή εξαρτάται από τη φύση των δεδομένων, τους υπολογιστικούς πόρους και τις εκάστοτε απαιτήσεις.

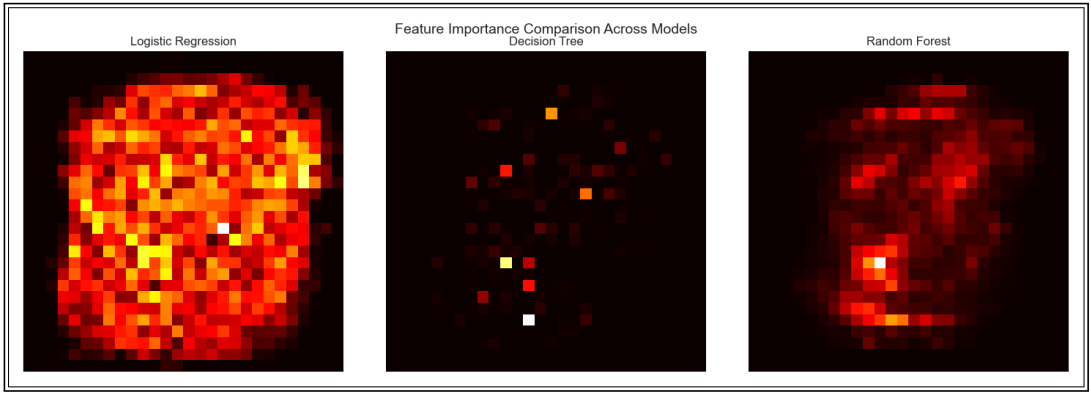
Πίνακας Περιεχομένων

Σύγκριση Σημασίας Χαρακτηριστικών (Feature Importance Comparison)

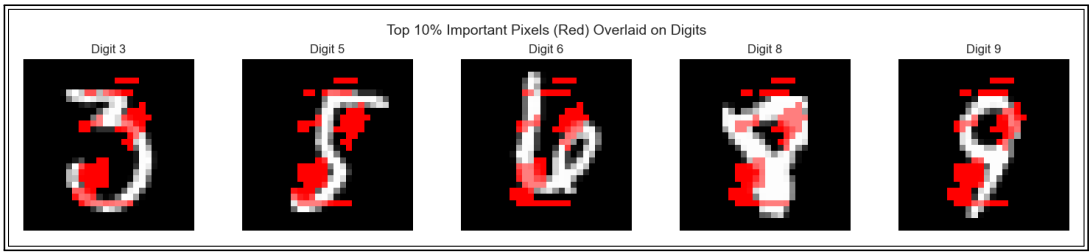
Ανάλυση για τρία μοντέλα

```
    # Logistic Regression: μέσος όρος απόλυτης τιμής συντελεστών ανά pixel
    lr_imp = np.abs(lr.coef_).mean(axis=0)   # shape: (784,)

    # Decision Tree: ενσωματωμένη σημασία
    dt_importance = dt.feature_importances_  # shape: (784,)

    # Random Forest: μέσος όρος σημασίας ανά δέντρο
    rf_importance = rf.feature_importances_  # shape: (784,)

    # Επικάλυψη σημαντικότερων pixels σε δείγματα
    importance_mask = rf_importance.reshape(28, 28) > np.percentile(rf_importance, 90)
    # Τα κόκκινα pixels: κορυφαίο 10% σε σημασία

```

Οπτικοποίηση (Visualization)



Σχ. 4.17: Σύγκριση Τριών Μοντέλων



Σχ. 4.18: Κορυφαία 10% Σημαντικών Pixels (Κόκκινο) Επικαλυμμένο στα Ψηφία

Σύνοψη Βασικών Ευρημάτων

- **Κεντρική εστίαση**: όλα τα μοντέλα συμφωνούν ότι η κεντρική περιοχή (όπου γράφεται το ψηφίο) φέρει τη μεγαλύτερη πληροφορία. Γωνίες και άκρες (μαύρο φόντο) συμβάλλουν ελάχιστα.
- **Διαφορές μοντέλων**: η Logistic Regression κατανέμει ομαλά, το Decision Tree είναι αραιό, το Random Forest δίνει ομαλή χαρτογράφηση.
- **Διακριτές περιοχές**: σημαντικά pixels αντιστοιχούν σε περιοχές όπου τα ψηφία διαφέρουν – περιοχές βρόχων (6, 8, 9) και καμπύλες (3, 5).

Βασικά Συμπεράσματα

1. **Επιβλεπόμενη μάθηση** απαιτεί: διαχωρισμό εκπαίδευσης/ελέγχου, ομαλοποίηση, αξιολόγηση με σωστές μετρικές.
2. **Χαρακτηριστικά δεδομένων καθορίζουν τον αλγόριθμο**: τα pixels MNIST έχουν συσχετίσεις – το Naive Bayes υποφέρει, τα SVM/RF ευδοκιμούν.
3. **Υπερπροσαρμογή** ελέγχεται με περιορισμό πολυπλοκότητας (βάθος δέντρου, regularization).
4. **Ανάλυση σημασίας χαρακτηριστικών** αποκαλύπτει ποια pixels "βλέπουν" τα μοντέλα.

Σύνδεση με TDA

Η ανάλυση σημασίας pixel υπονοεί ότι τα μοντέλα ανακαλύπτουν εμμέσως **τοπολογικά χαρακτηριστικά** (βρόχοι, οπές). Η TDA μπορεί να **εξάγει ρητά** αυτά τα χαρακτηριστικά και να τα χρησιμοποιήσει ως είσοδο στους παραπάνω ταξινομητές.

Κύριες Εισαγωγές

```
1       import numpy as np
2       import gudhi as gd
3       import gudhi.representations
4       import matplotlib.pyplot as plt
5
```

Πίνακας Περιεχομένων

Περιγραφή Συστήματος (System Description)

Χρησιμοποιούμε δυναμικό σύστημα που εξαρτάται από παράμετρο $r > 0$:

$$x_{n+1} = x_n + r\, y_n(1 - y_n) \pmod 1$$

$$y_{n+1} = y_n + r\, x_{n+1}(1 - x_{n+1}) \pmod 1$$

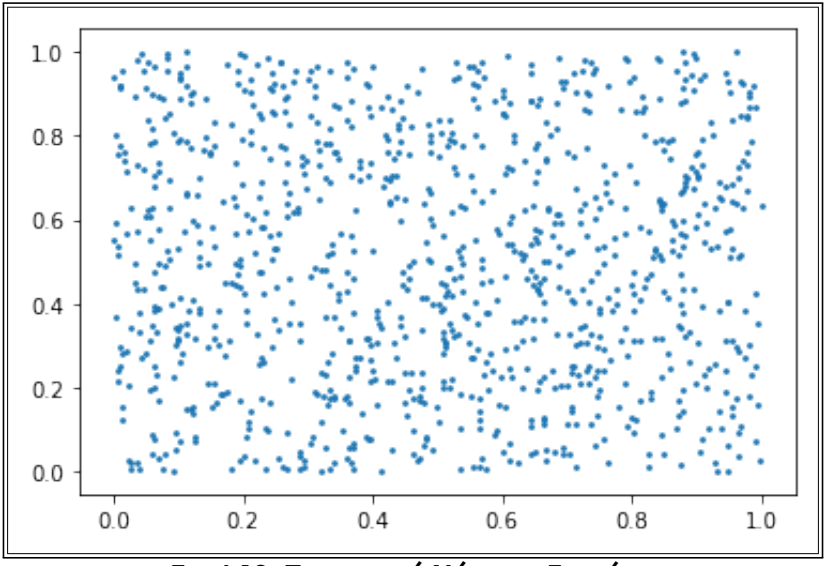
Για διαφορετικές τιμές *r*, η τροχιά εμφανίζει **διαφορετική τοπολογική δομή**.

Παραγωγή Νέφους Σημείων

```
num_pts = 1000
r       = 3.5

X = np.empty([num_pts, 2])
x, y = np.random.uniform(), np.random.uniform()
for i in range(num_pts):
X[i,:] = [x, y]
x = (X[i,0] + r*X[i,1]*(1-X[i,1])) % 1.
y = (X[i,1] + r*x*(1-x)) % 1.

```

Βασική Παρατήρηση

Για $r = 3.5$: το νέφος φαίνεται τυχαίο.
Για $r = 4.1$: εμφανίζεται **οπή στο κέντρο** – σημαντικό τοπολογικό χαρακτηριστικό. Η τιμή *r* καθορίζει την τοπολογία!



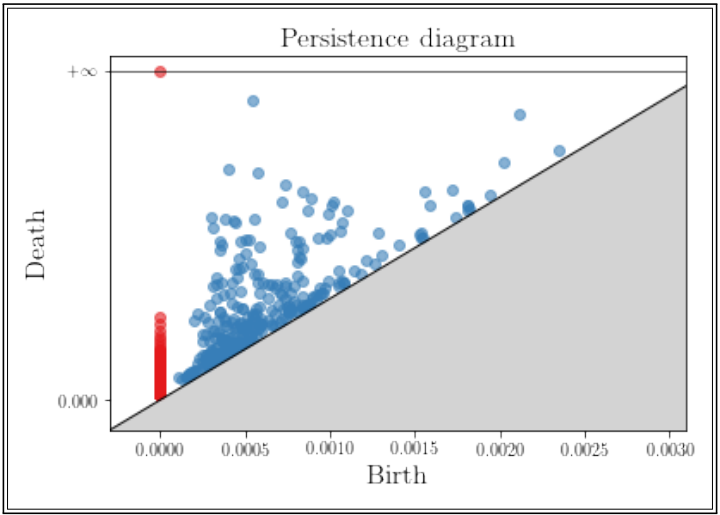
Σχ. 4.19: Παραγωγή Νέφους Σημείων

Παραγωγή Διαγραμμάτων Επιμονής (Computing Persistence Diagrams)

Alpha Φιλτράρισμα με GUDHI

```
# Κατασκευή Alpha συμπλέγματος και υπολογισμός επιμονής
acX = gd.AlphaComplex(points=X).create_simplex_tree()
dgmX = acX.persistence()

# Οπτικοποίηση διαγράμματος επιμονής
gd.plot_persistence_diagram(dgmX)

# Δεύτερο νέφος σημείων (r=4.1 -- έχει τοπολογία!)
r = 4.1
Y = np.empty([num_pts, 2])
x, y = np.random.uniform(), np.random.uniform()
for i in range(num_pts):
Y[i,:] = [x, y]
x = (Y[i,0] + r*Y[i,1]*(1-Y[i,1])) % 1.
y = (Y[i,1] + r*x*(1-x)) % 1.

acY = gd.AlphaComplex(points=Y).create_simplex_tree()
dgmY = acY.persistence()
```

Σύγκριση: Για $r = 3.5$: τυχαίο νέφος, χωρίς έντονα τοπολογικά χαρακτηριστικά στο διάγραμμα επιμονής. Για $r = 4.1$: εμφανής οπή στη διάσταση 1, υποδηλώνει **δακτύλιο** στη δομή της τροχιάς.



Σχ. 4.20: Παραγωγή Διαγράμματος Επιμονής

Πίνακας Περιεχομένων

Τοπίο Επιμονής (Persistence Landscape)

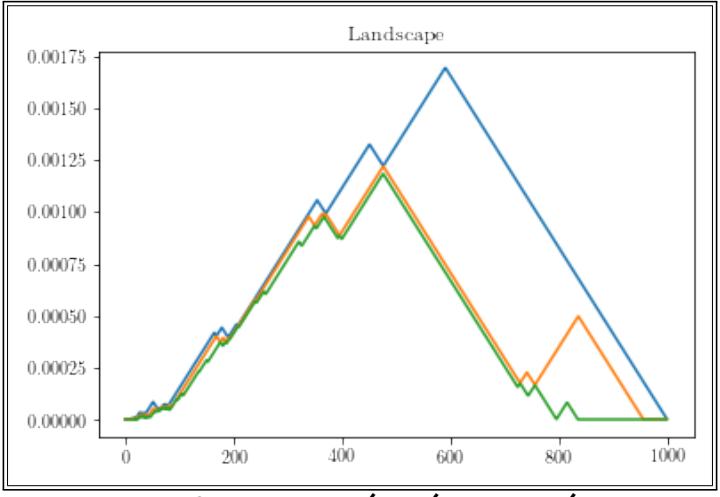
Ορισμός

*Το **τοπίο επιμονής** (persistence landscape) λαμβάνεται:*

1. *Περιστροφή του διαγράμματος επιμονής κατά $-\pi/4$ (η διαγώνιος γίνεται άξονας x)*
2. *Τοποθέτηση **σκηνικών συναρτήσεων** (tent functions) σε κάθε σημείο*
3. *Το k-ιοστό τοπίο (landscape) είναι η k-ιοστή μεγαλύτερη τιμή μεταξύ όλων των σκηνικών συναρτήσεων*
4. *Μετατροπή σε διάνυσμα με δειγματοληψία σε ομοιόμορφα σημεία του άξονα x*

Κώδικας

```
LS = gd.representations.Landscape(resolution=1000)
L  = LS.fit_transform([acX.persistence_intervals_in_dimension(1)])
# L[0] έχει μήκος 3000 (3 τοπία x 1000 σημεία)
# Οπτικοποίηση:
plt.plot(L[0][:1000])    # ο1 τοπίο
plt.plot(L[0][1000:2000]) # ο2 τοπίο
plt.plot(L[0][2000:3000]) # ο3 τοπίο
plt.title("Landscape")
```



Σχ. 4.21: Παραγωγή Τοπίου Επιμονής

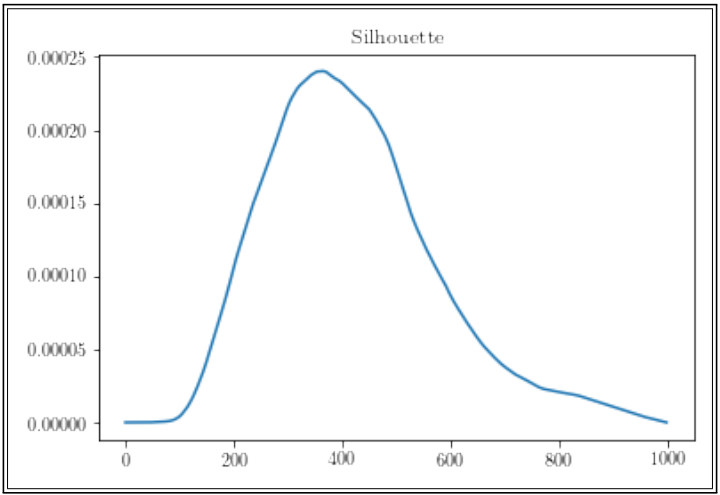
Σιλουέτα Επιμονής (Persistence Silhouette)

Ορισμός

*Η **σιλουέτα επιμονής** (persistence silhouette) αποτελεί παραλλαγή του τοπίου: λαμβάνει **σταθμισμένο μέσο** όρο των σκηνικών συναρτήσεων αντί του k-ιοστού μέγιστου. Κάθε σκηνική συνάρτηση σταθμίζεται ανάλογα με την απόσταση του αντίστοιχου σημείου από τη διαγώνιο.*

Κώδικας

```
# Στάθμιση: τετράγωνο της επιμονής (death - birth)^1
SH = gd.representations.Silhouette(
resolution=1000,
weight=lambda x: np.power(x[1]-x[0], 1)
)
sh = SH.fit_transform([acX.persistence_intervals_in_dimension(1)])
plt.plot(sh[0])
plt.title("Silhouette")

```



Σχ. 4.22: Παραγωγή Σιλουέτας Επιμονής

Εικόνα Επιμονής (Persistence Image)

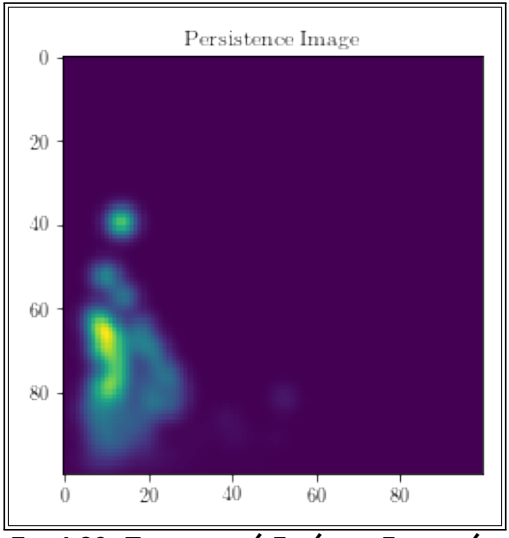
Ορισμός

*Η **εικόνα επιμονής** (persistence image) λαμβάνεται:*

1. *Περιστροφή κατά $-\pi/4$*
2. *Κέντρωση Γκαουσιανών συναρτήσεων σε κάθε σημείο (σταθμισμένες από παραμετρική συνάρτηση – συνήθως τετράγωνο απόστασης από τη διαγώνιο)*
3. *Άθροιση όλων των Γκαουσιανών – δίνει 2Δ συνάρτηση*
4. *Pixelization σε εικόνα (rasterization)*

Κώδικας

```
1       PI = gd.representations.PersistenceImage(
2       bandwidth=1e-4,
3       weight=lambda x: x[1]**2,   # τετράγωνο death
4       im_range=[0, .004, 0, .004],
5       resolution=[100, 100]        # 100x100 pixels
6       )
7       pi = PI.fit_transform([acX.persistence_intervals_in_dimension(1)])
8       # pi[0] shape: (10000,) -- αναδιαμόρφωση σε (100,100)
9
```



Σχ. 4.23: Παραγωγή Εικόνας Επιμονής

Επιπλέον Μέθοδοι Αναπαράστασης (Additional Representations)

Διαθέσιμες Μέθοδοι στο `gudhi.representations`

- **Καμπύλη Betti** (Betti Curve): μετρά τον αριθμό ζωντανών χαρακτηριστικών σε κάθε τιμή του φίλτρου
- **Μιγαδικό Πολυώνυμο** (Complex Polynomial): χαρτογράφηση σε χώρο πολυωνύμων
- **Τοπολογικό Διάνυσμα** (Topological Vector): συμπίεση πληροφορίας σε διάνυσμα σταθερού μήκους

Κοινό Πλαίσιο scikit-learn

Όλες οι μέθοδοι ακολουθούν το API του scikit-learn: `fit()`, `transform()`, `fit_transform()`. Αυτό επιτρέπει απρόσκοπτη ενσωμάτωση με μηχανική μάθηση.

Πίνακας Περιεχομένων

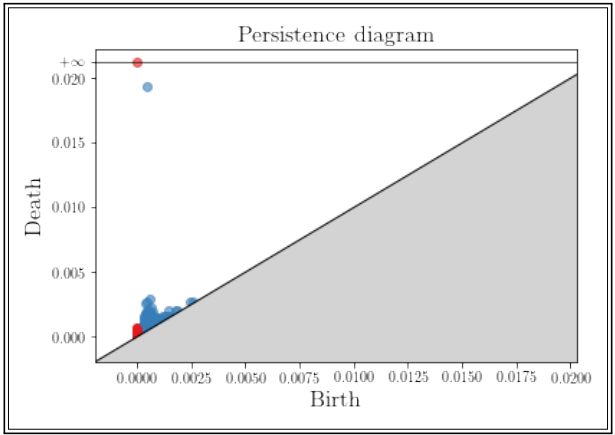
Πυρήνες Διαγραμμάτων Επιμονής (Persistence Diagram Kernels)

Ορισμός

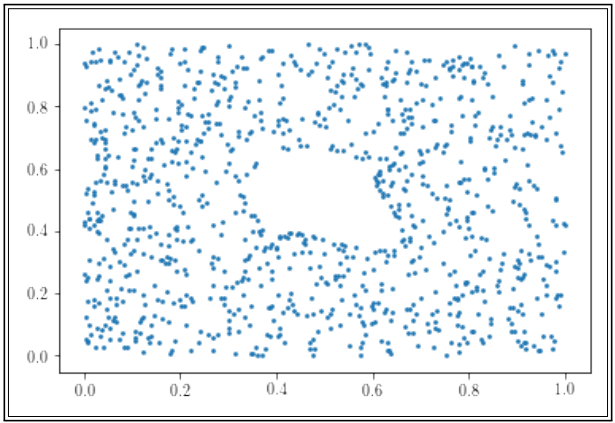
*Ένας **πυρήνας** (kernel) k δέχεται ζεύγος διαγραμμάτων D, D' και επιστρέφει πραγματικό αριθμό ώστε:*

$$k(D, D') = \langle \Phi(D), \Phi(D') \rangle_{\mathcal{H}}$$

*για κάποιον **υπονοούμενο χώρο Hilbert** (implicit Hilbert space) $\mathcal{H}$ και συνεχή συνάρτηση $\Phi : \mathcal{D} \to \mathcal{H}$.*



Σχ. 4.24: Διάγραμμα Επιμονής



Σχ. 4.25: Επίδειξη Ομολογίας Δεδομένων

Διαθέσιμοι Πυρήνες & Βασικό Πλεονέκτημα

Διαθέσιμοι Πυρήνες στο GUDHI

- **Sliced Wasserstein Kernel** (SW): βασισμένος σε προβολές Wasserstein
- **Persistence Weighted Gaussian Kernel** (PWG): σταθμισμένες Γκαουσιανές σε χώρο διαγραμμάτων
- **Persistence Scale Space Kernel** (PSS): stable multi-scale kernel
- **Persistence Fisher Kernel** (PF): βασισμένος σε απόκλιση Fisher

Πλεονέκτημα

Πολλοί αλγόριθμοι (SVM, kernel PCA) απαιτούν μόνο ζευγαρωτά εσωτερικά γινόμενα. Με τους πυρήνες, μπορούμε να τους εφαρμόσουμε **απευθείας σε διαγράμματα επιμονής**!

Υπολογισμός Πυρήνων & Αποστάσεων (Computing Kernels & Distances)

```
    # Persistence Weighted Gaussian Kernel
    PWG = gd.representations.PersistenceWeightedGaussianKernel(
    bandwidth=0.01, kernel_approx=None,
    weight=lambda x: np.arctan(np.power(x[1], 1)))
    PWG.fit([acX.persistence_intervals_in_dimension(1)])
    pwg = PWG.transform([acY.persistence_intervals_in_dimension(1)])
    print("PWG kernel is " + str(pwg[0][0]))

    # Persistence Scale Space Kernel
    PSS = gd.representations.PersistenceScaleSpaceKernel(bandwidth=1.)
    PSS.fit([acX.persistence_intervals_in_dimension(1)])
    pss = PSS.transform([acY.persistence_intervals_in_dimension(1)])

    # Persistence Fisher Kernel
    PF = gd.representations.PersistenceFisherKernel(
    bandwidth_fisher=.001, bandwidth=.001, kernel_approx=None)
    PF.fit([acX.persistence_intervals_in_dimension(1)])

    # Sliced Wasserstein Kernel
    SW = gd.representations.SlicedWassersteinKernel(bandwidth=.1, num_directions=100)
    SW.fit([acX.persistence_intervals_in_dimension(1)])

```

Αποστάσεις Bottleneck & Wasserstein

```
1   # Απόσταση Bottleneck μέσω( gudhi.representations)
2   BD = gd.representations.BottleneckDistance(epsilon=.001)
3   BD.fit([acX.persistence_intervals_in_dimension(1)])
4   bd = BD.transform([acY.persistence_intervals_in_dimension(1)])
5   print("Bottleneck distance is " + str(bd[0][0]))
6
7   # Απόσταση Wasserstein
8   WD = gd.representations.WassersteinDistance(internal_p=2, order=2)
9   WD.fit([acX.persistence_intervals_in_dimension(1)])
10  wd = WD.transform([acY.persistence_intervals_in_dimension(1)])
11  print("Wasserstein distance is " + str(wd[0][0]))
12
```

Συνοπτικός Πίνακας Μεθόδων

Μέθοδος	**Τύπος**	**Παραμέτρος**
Persistence Landscape (LS)	Vectorization	`resolution`
Silhouette (SH)	Vectorization	`resolution, weight`
Persistence Image (PI)	Vectorization	`bandwidth, resolution`
Sliced Wasserstein (SW)	Kernel	`bandwidth, num_directions`
PWG	Kernel	`bandwidth, weight`
Bottleneck (BD)	Distance	`epsilon`
Wasserstein (WD)	Distance	`internal_p, order`

Πίνακας Περιεχομένων

Δημιουργία Πλήρους Συνόλου Δεδομένων (Full Dataset Generation)

5 Κλάσεις – Διαφορετικές Τιμές *r*

```
    num_diag_per_class = 10   # Διαγράμματα ανά κλάση
    dgms, labs = [], []

    for idx, radius in enumerate([2.5, 3.5, 4., 4.1, 4.3]):
    for _ in range(num_diag_per_class):
    labs.append(idx)
    X = np.empty([num_pts, 2])
    x, y = np.random.uniform(), np.random.uniform()
    for i in range(num_pts):
    X[i,:] = [x, y]
    x = (X[i,0] + radius * X[i,1] * (1-X[i,1])) % 1.
    y = (X[i,1] + radius * x * (1-x)) % 1.
    ac  = gd.AlphaComplex(points=X).create_simplex_tree(max_alpha_square=1e12)
    dgm = ac.persistence()
    dgms.append(ac.persistence_intervals_in_dimension(1))

```

Δομή Συνόλου Δεδομένων

5 κλάσεις $\times$ 10 διαγράμματα = 50 διαγράμματα επιμονής συνολικά. Κάθε κλάση αντιστοιχεί σε διαφορετική τιμή *r* (διαφορετική τοπολογική δομή τροχιάς).

Διαχωρισμός Εκπαίδευσης – Ελέγχου (Train-Test Split)

```
test_size           = 0.2
perm                = np.random.permutation(len(labs))
limit               = int(test_size * len(labs))
test_sub, train_sub = perm[:limit], perm[limit:]

train_labs  = np.array(labs)[train_sub]
test_labs   = np.array(labs)[test_sub]
train_dgms  = [dgms[i] for i in train_sub]
test_dgms   = [dgms[i] for i in test_sub]
```

Σύνθεση Συνόλων

- **Εκπαιδευτικό**: 80% – 40 διαγράμματα
- **Ελέγχου**: 20% – 10 διαγράμματα

Ολοκληρωμένη Προσέγγιση Μέσω Scikit-Learn

Στάδια – Βήματα

Η προσέγγιση δομείται ως εξής:

1. **DiagramSelector**: εξαγωγή σημείων με πεπερασμένες συντεταγμένες (non-essential points)
2. **DiagramScaler**: κλιμάκωση ή μη στο μοναδιαίο τετράγωνο
3. **TDA (Αναπαράσταση)**: vectorization ή kernel method με GUDHI
4. **Estimator (Ταξινομητής)**: αλγόριθμος ταξινόμησης scikit-learn

Βασική Δομή Προσέγγισης

```
1       from sklearn.preprocessing import MinMaxScaler
2       from sklearn.pipeline      import Pipeline
3       from sklearn.svm           import SVC
4
5       pipe = Pipeline([
6       ("Separator", gd.representations.DiagramSelector(limit=np.inf, point_type="finite")),
7       ("Scaler",    gd.representations.DiagramScaler(scalers=[([0,1], MinMaxScaler())])),
8       ("TDA",       gd.representations.PersistenceImage()),
9       ("Estimator", SVC())
10      ])
```

Πλέγμα Παραμέτρων (Parameter Grid)

```
param = [
# Sliced Wasserstein kernel SVM
{"Scaler__use": [False],
  "TDA":          [gd.representations.SlicedWassersteinKernel()],
  "TDA__bandwidth": [0.1, 1.0], "TDA__num_directions": [20],
  "Estimator":    [SVC(kernel="precomputed", gamma="auto")]},

# Persistence Weighted Gaussian kernel SVM
{"Scaler__use": [False],
  "TDA":          [gd.representations.PersistenceWeightedGaussianKernel()],
  "TDA__bandwidth": [0.1, 0.01],
  "TDA__weight": [lambda x: np.arctan(x[1]-x[0])],
  "Estimator":    [SVC(kernel="precomputed", gamma="auto")]},

# Persistence Image + SVM
{"Scaler__use": [True],
  "TDA":          [gd.representations.PersistenceImage()],
  "TDA__resolution": [[5,5], [6,6]], "TDA__bandwidth": [0.01, 0.1, 1.0, 10.0],
  "Estimator":    [SVC()]},

# Persistence Landscape + Random Forest
{"Scaler__use": [True],
  "TDA":          [gd.representations.Landscape()], "TDA__resolution": [100],
  "Estimator":    [RandomForestClassifier()]},

# Bottleneck Distance + K-NN
{"Scaler__use": [False],
  "TDA":          [gd.representations.BottleneckDistance()], "TDA__epsilon": [0.1],
  "Estimator":    [KNeighborsClassifier(metric="precomputed")]}
]
```

Διασταυρούμενη Επικύρωση & Αποτελέσματα (Cross-Validation & Results)

GridSearchCV – 3-fold Cross-Validation

```
from sklearn.model_selection import GridSearchCV

model = GridSearchCV(pipe, param, cv=3)

# Εκπαίδευση -- μπορεί να αργεί ειδικά( k-NN + Bottleneck)
model = model.fit(train_dgms, train_labs)

# Βέλτιστες παράμετροι
print(model.best_params_)

# Αξιολόγηση
print("Train accuracy = " + str(model.score(train_dgms, train_labs)))
print("Test accuracy  = " + str(model.score(test_dgms,  test_labs)))

```

Αποτέλεσμα

Random Forest + Persistence Landscape ανέδειξε την καλύτερη απόδοση για αυτό το μικρό σύνολο δεδομένων. Ακρίβεια ελέγχου $\approx 70\%$ – αξιόλογο για μόλις 50 διαγράμματα εκπαίδευσης! Η απόδοση βελτιώνεται σημαντικά με μεγαλύτερα σύνολα δεδομένων.

Ερμηνεία Αποτελεσμάτων (Interpretation of Results)

Γιατί Random Forest + Landscape;

- Το τοπίο επιμονής παρέχει **σταθερή, πλούσια αναπαράσταση** της τοπολογικής δομής σε πολλαπλές κλίμακες
- Το Τυχαίο Δάσος εκμεταλλεύεται αποτελεσματικά **πολυδιάστατα διανύσματα** (1000 σημεία ανά τοπίο)
- Ισχυρός συνδυασμός για **μικρά σύνολα δεδομένων** με πολυδιάστατες αναπαραστάσεις

Συμβουλές Πρακτικής Εφαρμογής

- Χρήση `GridSearchCV` για αυτόματη βελτιστοποίηση υπερπαραμέτρων
- Αρχικά δοκιμάστε **Persistence Image + SVM** (καλή ισορροπία ταχύτητας/απόδοσης)
- Για εφαρμογές πραγματικού χρόνου: **Silhouette** ή **Betti Curve** (ταχύτεροι)
- Αποφυγή k-NN + Bottleneck σε μεγάλα σύνολα (υπολογιστικά ακριβό)

Πίνακας Περιεχομένων

Σύνοψη Εννοιών (Summary of Concepts)

- **Supervised Learning**: MNIST, preprocessing, train-test split, PCA
- **Μετρικές**: accuracy, precision, recall, F1-score, confusion matrix
- **Αλγόριθμοι**: K-NN, Naive Bayes, Logistic Regression, Decision Trees, Random Forest, SVM
- **Σημασία χαρακτηριστικών**: κεντρική εστίαση σε pixels – υπονοούμενη τοπολογική πληροφορία
- **Τροχιές δυναμικών συστημάτων** ως γεννήτρια διαγραμμάτων επιμονής
- **Αναπαραστάσεις**: Landscape, Silhouette, Persistence Image
- **Πυρήνες**: SW, PWG, PSS, PF – για kernel-SVM
- **Ολοκληρωμένη Προσέγγιση Μέσω Scikit-Learn** + `GridSearchCV` για αυτόματη επιλογή μεθόδου

Κεντρικό Μήνυμα

Η TDA παρέχει **ρητές τοπολογικές αναπαραστάσεις** δεδομένων που μπορούν να ενσωματωθούν **πλήρως** σε παραδοσιακούς αλγορίθμους μηχανικής μάθησης.

Σας ευχαριστώ θερμά για την προσοχή σας!

Ερωτήσεις;

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS