Νευρωνικά Δίκτυα - Βαθιά Μάθηση
Εργασία 2η
Τουλκερίδης Νικόλαος 10718 ΗΜΜΥ

Σετ Δεδομένων
Η ανάπτυξη ενός Support Vector Machine έγινε χρησιμοποιώντας τη βάση δεδομένων CIFAR-10. Αυτή περιλαμβάνει 60.000 εικόνες με ανάλυση 32x32 πίξελ και χρωματική κωδικοποίηση RGB, ταξινομημένες σε 10 κατηγορίες, με 6.000 εικόνες ανά κατηγορία. Από αυτές, οι 50.000 χρησιμοποιούνται για εκπαίδευση και οι 10.000 για έλεγχο. Οι εικόνες κανονικοποιούνται ώστε οι τιμές των πίξελ να κυμαίνονται στο διάστημα [0,1], με μηδενική μέση τιμή και μοναδιαία διακύμανση.

Εργαλεία
Η μηχανή υλοποιήθηκε με τη χρήση της γλώσσας προγραμματισμού **Python** και της βιβλιοθήκης **Keras**. Επιπλέον, αξιοποιήθηκαν οι παρακάτω βιβλιοθήκες:

- **NumPy**: για εύκολο χειρισμό δεδομένων.
- **Scikit-learn**: για την προεπεξεργασία και ανάλυση δεδομένων.
- **Seaborn**: για τη δημιουργία διαγραμμάτων τύπου heatmap.
- **Time**: για τη μέτρηση του χρόνου εκπαίδευσης.
- **Matplotlib**: για την απεικόνιση γραφημάτων.
- **TensorFlow**: για την εκπαίδευση του SVM.

Ανάπτυξη της μηχανής
Η κατηγοριοποίηση που εξετάζεται είναι η μέθοδος **One-vs-One**, όπου κάθε δείγμα συγκρίνεται με κάθε άλλο δείγμα. Για την εύρεση των βέλτιστων παραμέτρων χρησιμοποιήθηκε η μέθοδος **Grid Search**. Τα πειράματα διεξήχθησαν με έως 60 επαναλήψεις, ανοχή (tolerance) 10^{-3}, 5-πτυχιακή διασταυρούμενη επικύρωση (5-fold cross-validation) και κριτήριο επιλογής του καλύτερου μοντέλου τον μέσο βαθμό δοκιμής (mean test score). Τέλος, το μοντέλο με την καλύτερη απόδοση υποβλήθηκε σε συνολικό έλεγχο με τα δεδομένα ελέγχου.

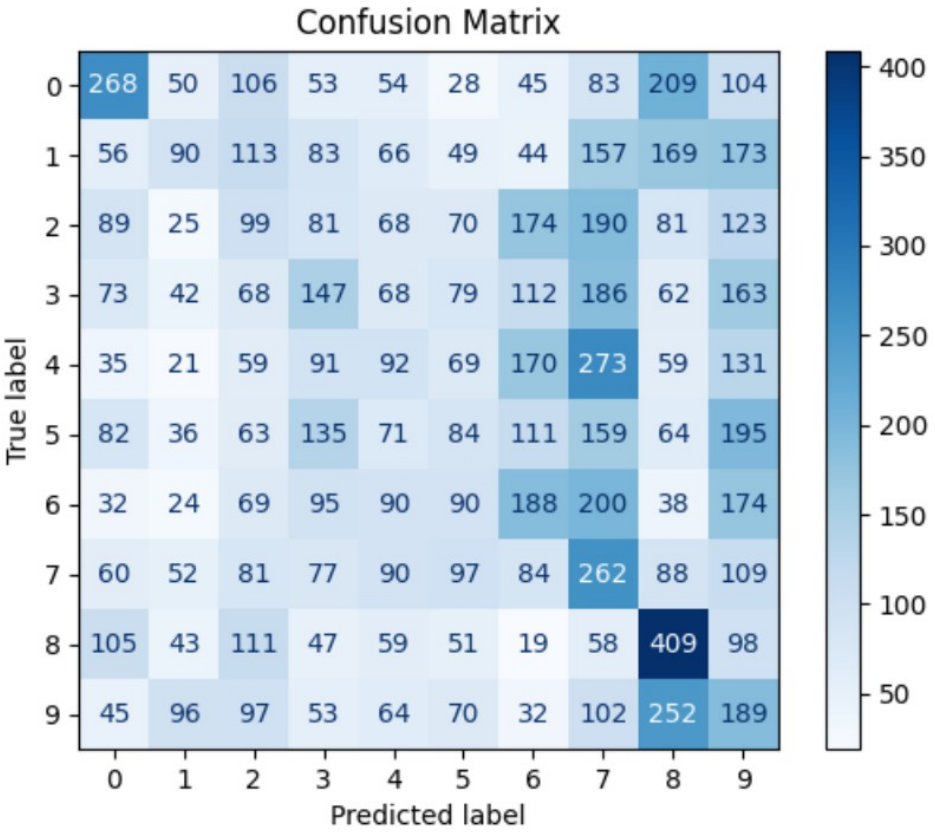
Γραμμικό kernel:
Έγιναν δοκιμές για γραμμικό kernel με τιμές για την παράμετρο c: [0.01, 0.1, 1, 10, 100]. Αποτελέσματα για κάθε τιμή της παραμέτρου c και fold του cross validation.

param_C	param_kernel	params	split0_test_score	split1_test_score
0.01	linear	{'C': 0.01, 'kernel': 'linear'}	0.1817	0.1666
0.10	linear	{'C': 0.1, 'kernel': 'linear'}	0.1817	0.1666
1.00	linear	{'C': 1, 'kernel': 'linear'}	0.1817	0.1666
10.00	linear	{'C': 10, 'kernel': 'linear'}	0.1817	0.1666
100.00	linear	{'C': 100, 'kernel': 'linear'}	0.1817	0.1666

split2_test_score	split3_test_score	split4_test_score	mean_test_score	std_test_score
0.1766	0.2038	0.1795	0.18164	0.012225
0.1766	0.2038	0.1795	0.18164	0.012225
0.1766	0.2038	0.1795	0.18164	0.012225
0.1766	0.2038	0.1795	0.18164	0.012225
0.1766	0.2038	0.1795	0.18164	0.012225

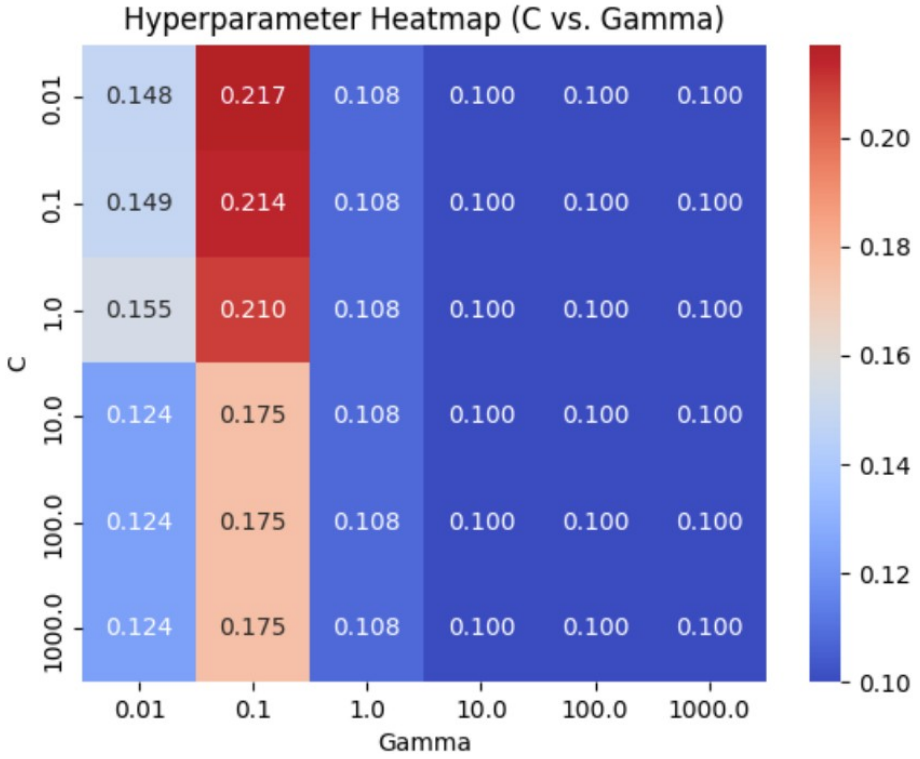
Συνολικός χρόνος: 19 λεπτά

Η **ευστοχία** παρατηρήθηκε να παραμένει σταθερή ανεξάρτητα από την τιμή του c. Για τον τελικό έλεγχο επιλέχθηκε **c=0.01**, με ευστοχία ελέγχου **0.1828.** Στη συνέχεια, παρουσιάζεται ο πίνακας σύγχυσης για τον καλύτερο εκτιμητή (best estimator).



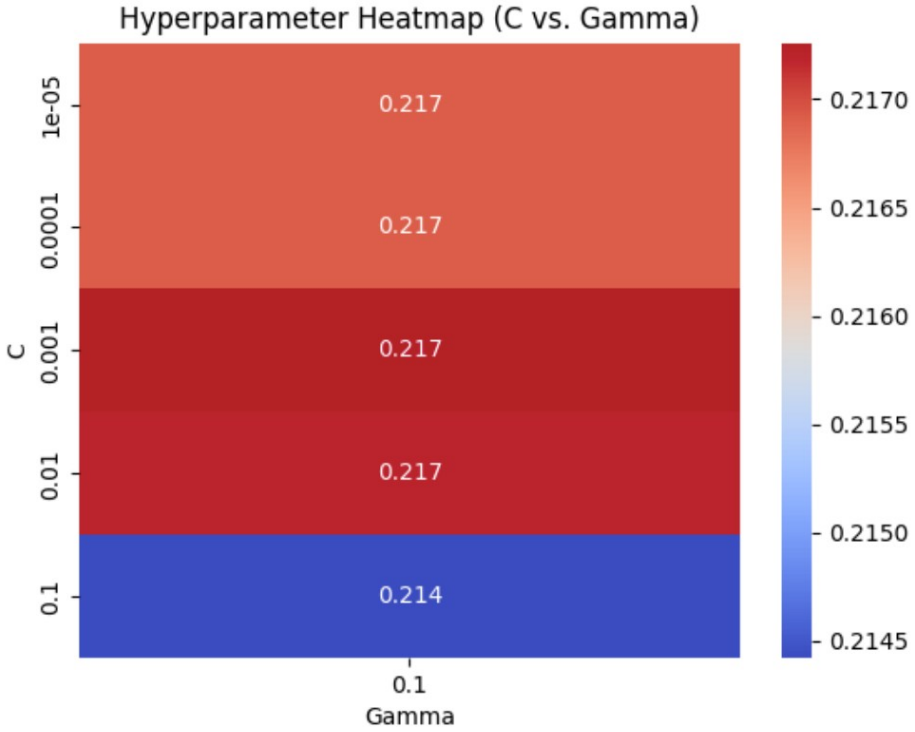
Πίνακας σύγχυσης για linear kernel με c = 0.01

RBF kernel:
Έγιναν δοκιμές για RBF kernel με κάποιες τιμές για τις παραμέτρους c και γ . Στον παρακάτω πίνακα φαίνονται τα αποτελέσματα.



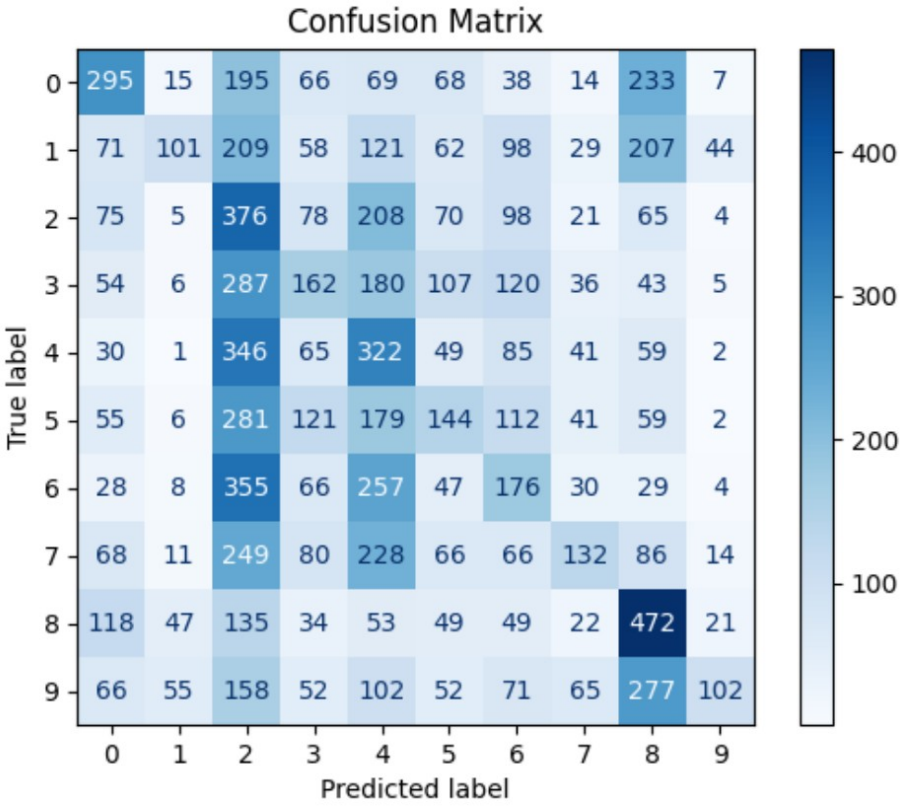
Heatmap της ευστοχίας της μηχανής
Συνολικός χρόνος: 85 λεπτά

Η ευστοχία είναι σαφώς υψηλότερη για c=0.01 και γ=0.1. Δεδομένου ότι η ευστοχία αυξάνεται όσο το c μειώνεται, πραγματοποιήθηκε επιπλέον έλεγχος για να μελετηθεί η συμπεριφορά της ακρίβειας σε ακόμη μικρότερες τιμές του c.



Heatmap της ευστοχίας της μηχανής
Συνολικός χρόνος: 10 λεπτά

Παρόλο που το διάγραμμα δεν το δείχνει ξεκάθαρα λόγω στρογγυλοποίησης, η μέγιστη ακρίβεια επιτεύχθηκε για c=0.001. Επομένως, για τον **RBF kernel**, ο ιδανικός συνδυασμός παραμέτρων είναι **c=0.001 και γ=0.1**, με ακρίβεια **0.2282**. Παρακάτω βρίσκεται ο πίνακας σύγχυσης.

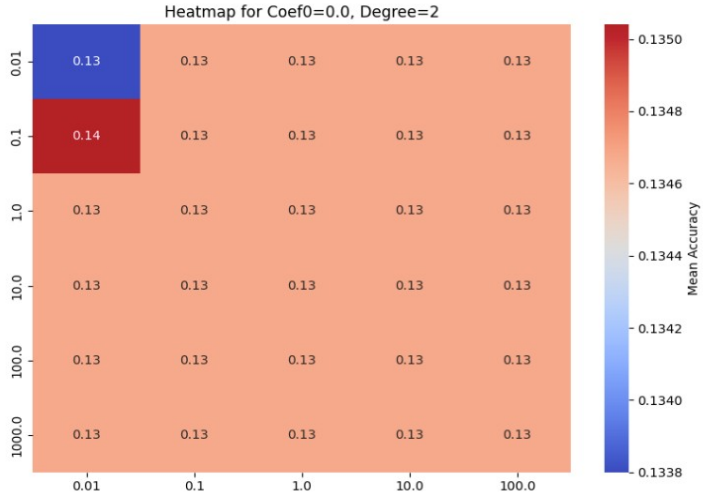


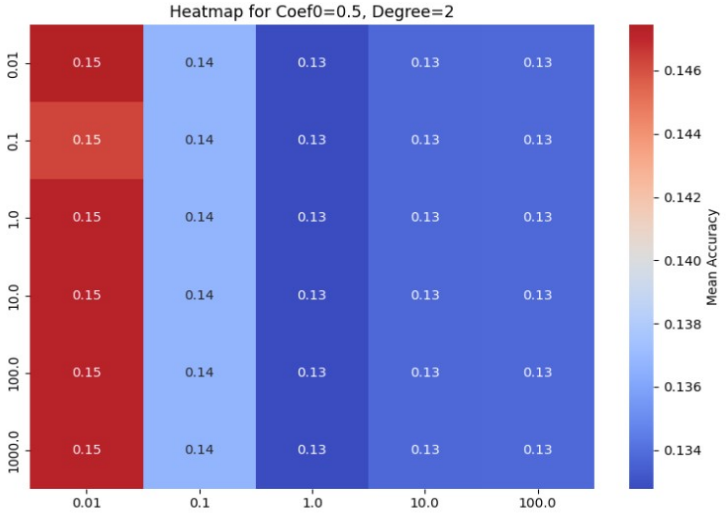
Πίνακας σύγχυσης για RBF kernel με c = 0.001 and γ = 0.1

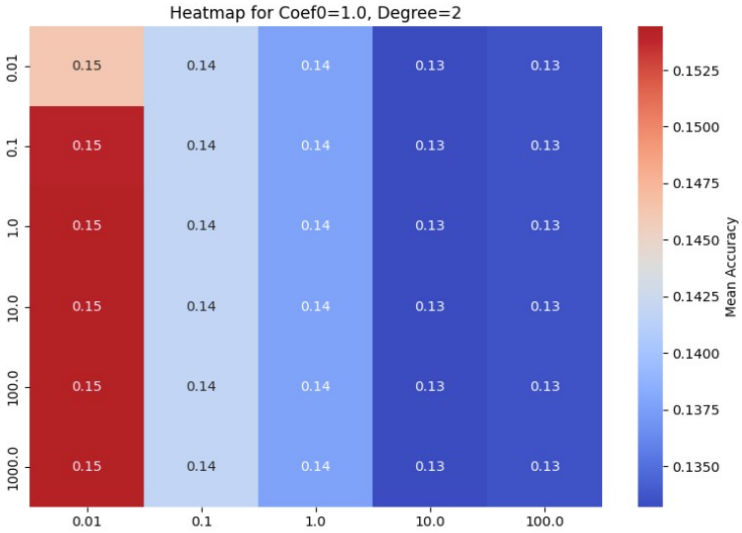
Polynomial kernel

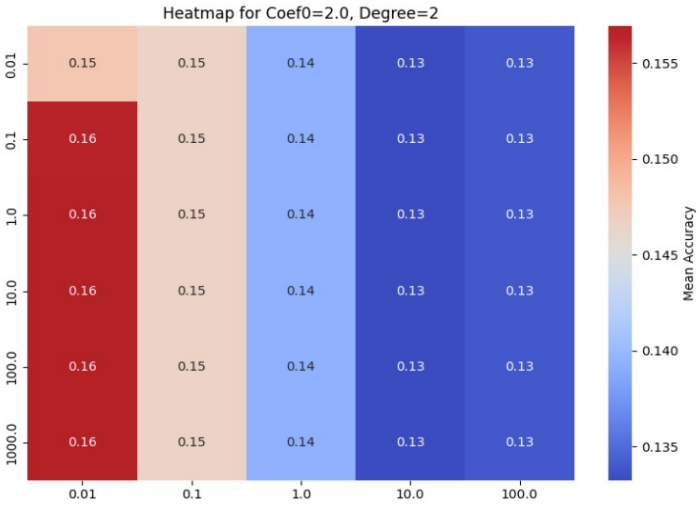
Δοκιμάστηκαν διάφορες τιμές για τις παραμέτρους c και γ, εξετάζοντας και διάφορους συνδυασμούς βαθμού και σταθερού συντελεστή των πολυωνύμων. Στο διάγραμμα, ο κάθετος άξονας αναπαριστά την παράμετρο c, ενώ ο οριζόντιος άξονας την παράμετρο γ. Ο συνολικός χρόνος εκπαίδευσης ήταν 24 ώρες.

Για πολυώνυμα 2ου βαθμού:

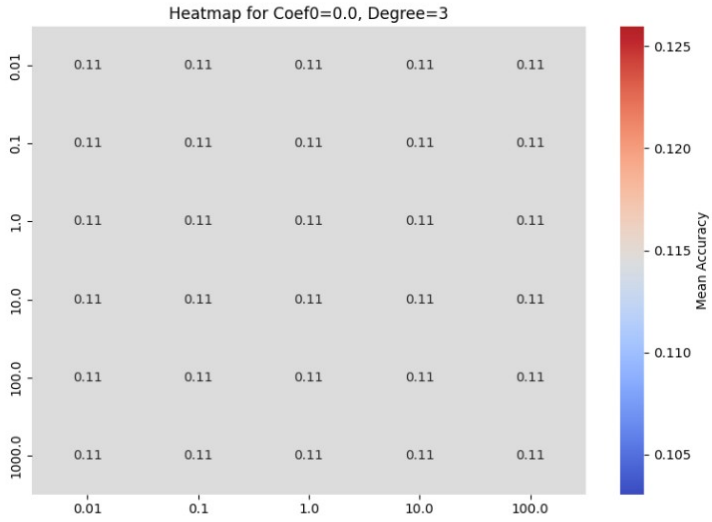


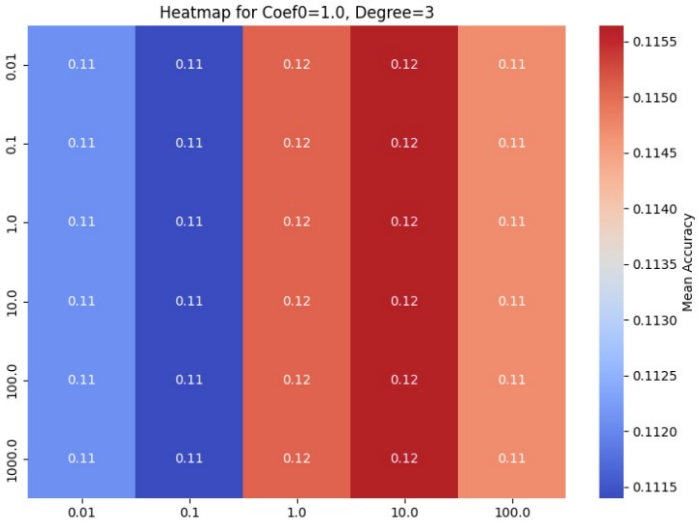


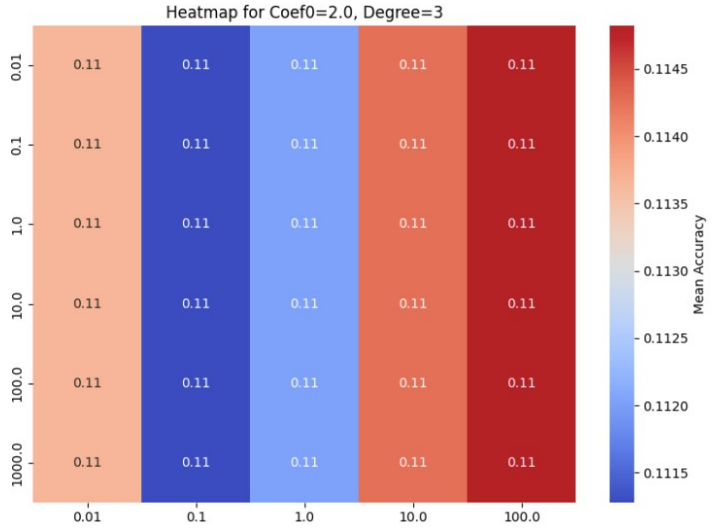




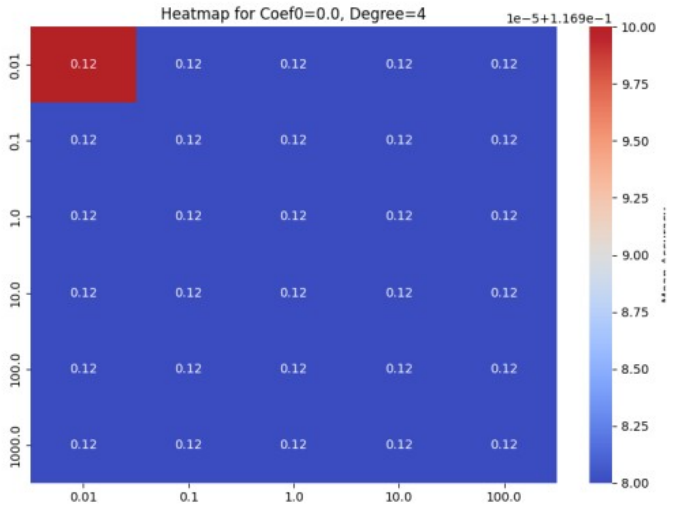
Για πολυώνυμα 3ου βαθμού:

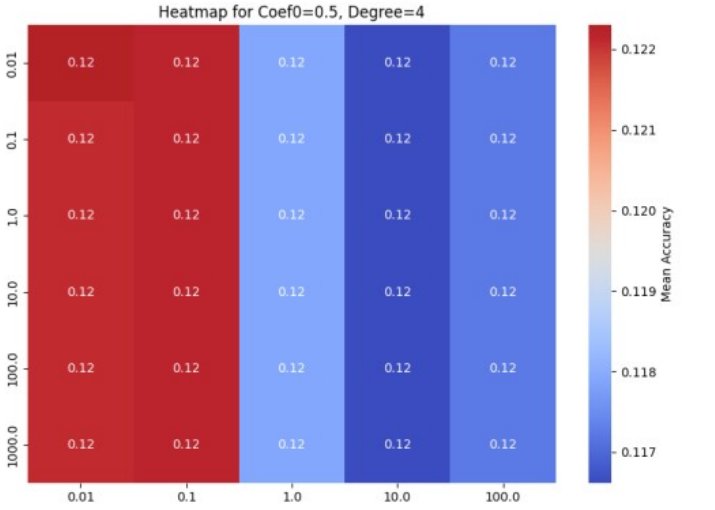


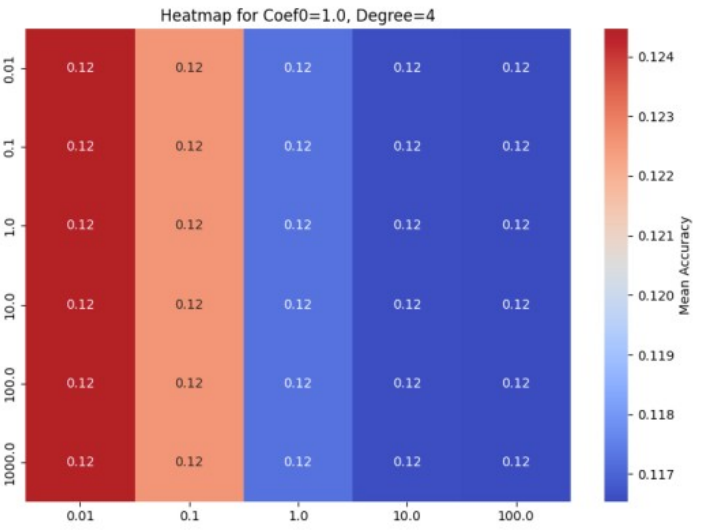


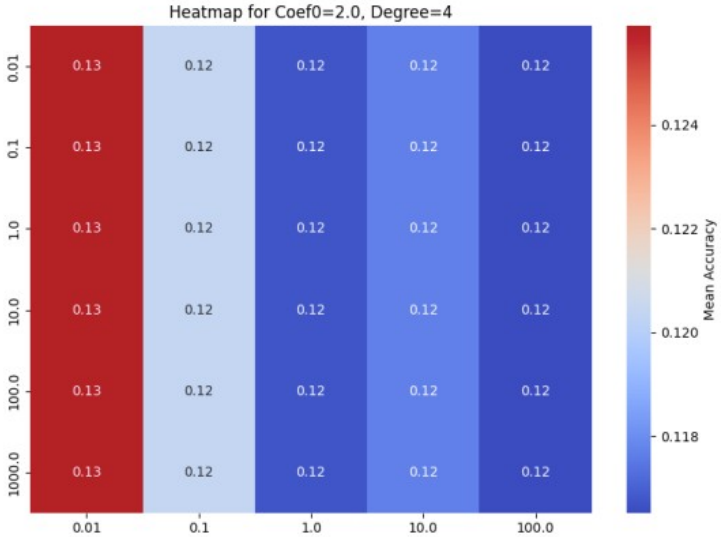


Για πολυώνυμα 4ου βαθμού:

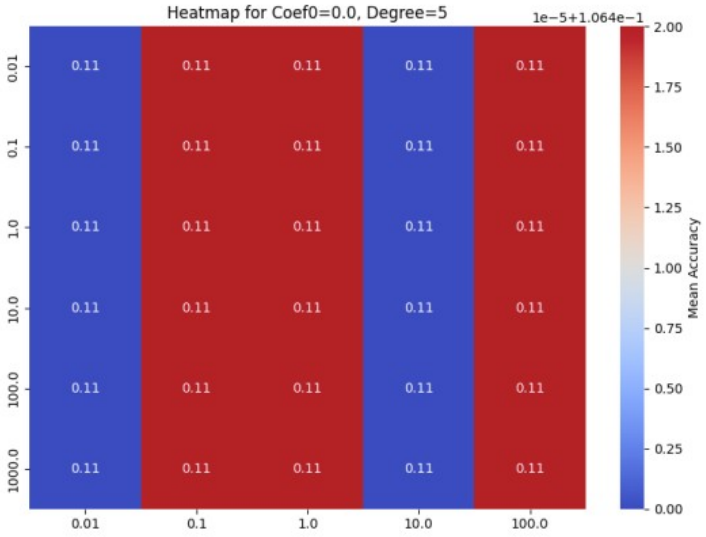


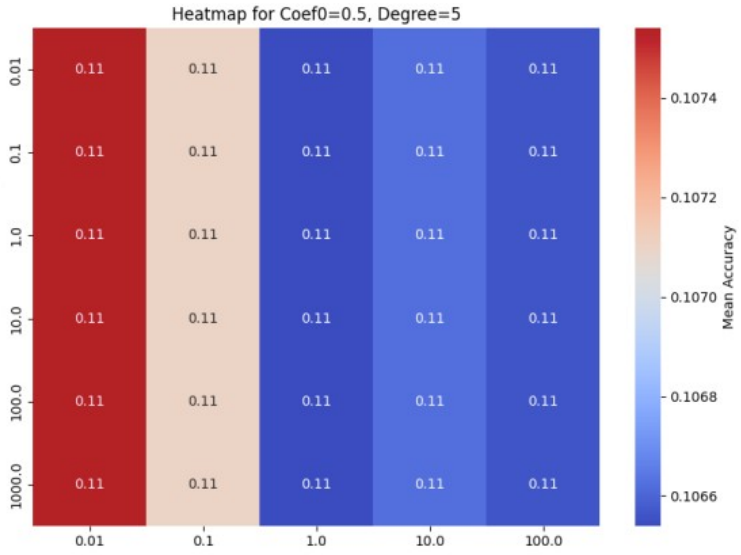


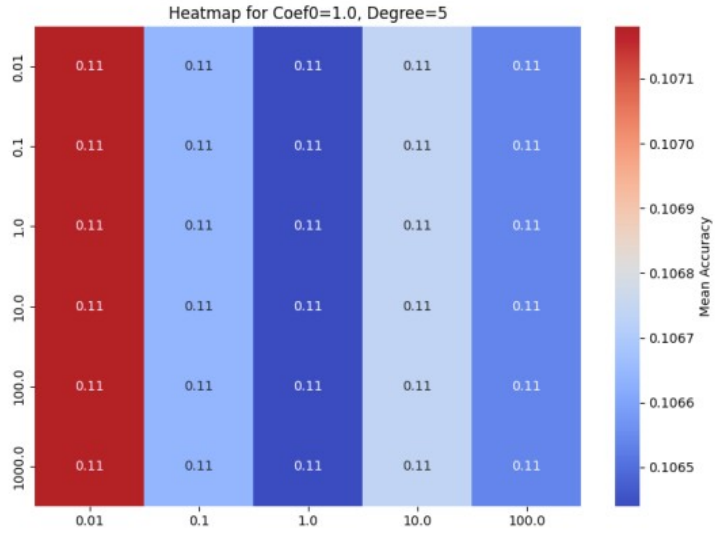


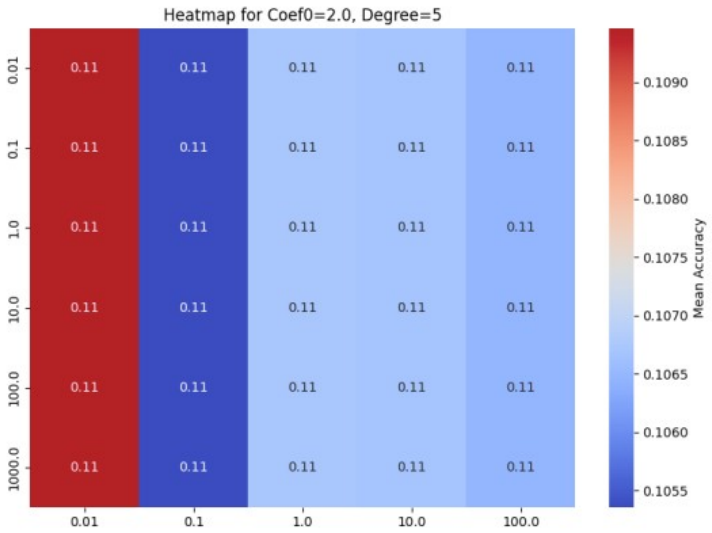


Για πολυώνυμα 5ου βαθμού:

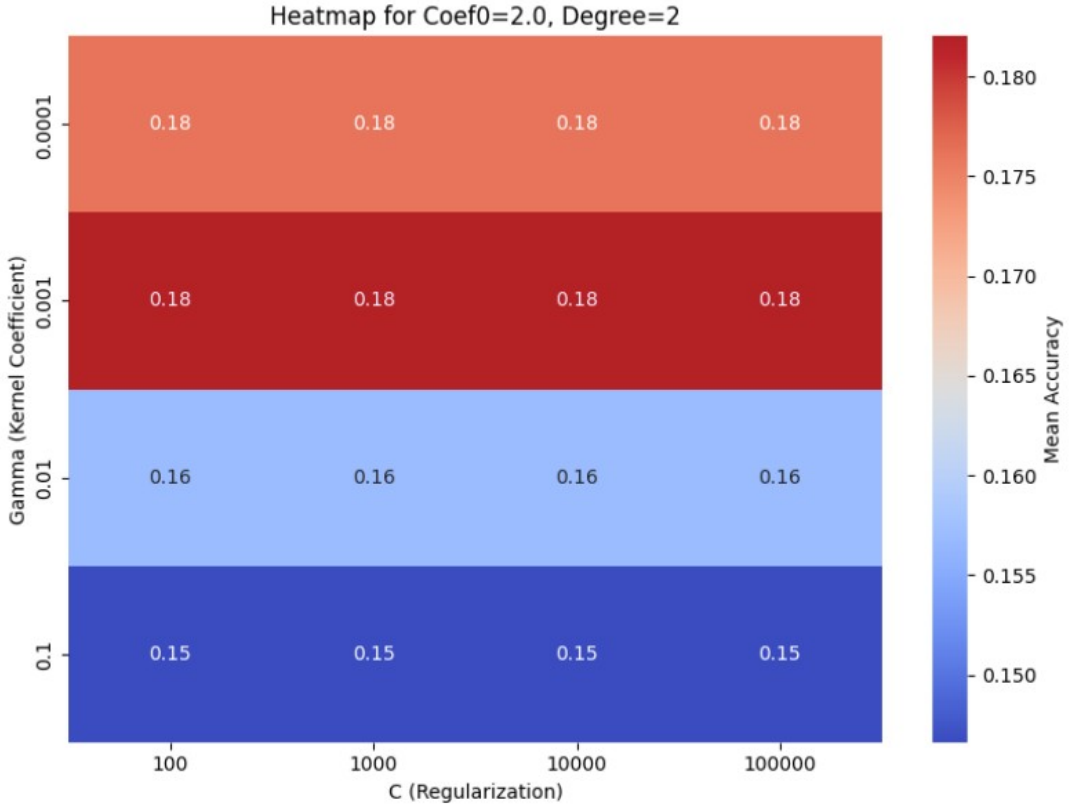








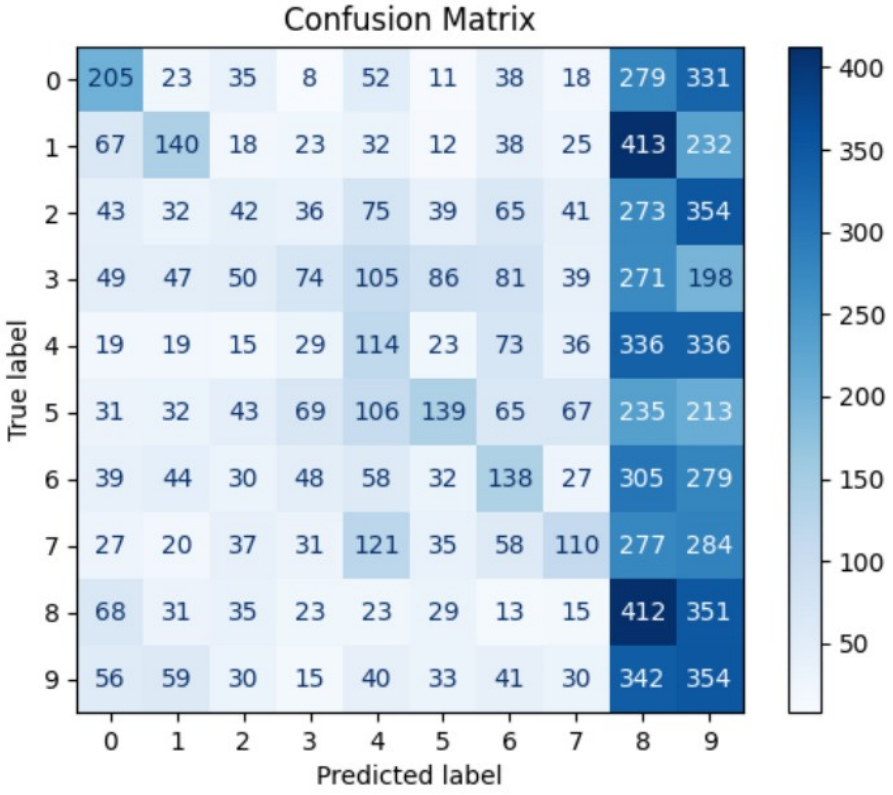
Ο πολυωνυμικός kernel **δεύτερου βαθμού** με σταθερά **2** παρουσίασε τη μεγαλύτερη ευστοχία. Επομένως, πραγματοποιήθηκε επιπλέον διερεύνηση για αυτόν τον συνδυασμό:



Heatmap της ευστοχίας της μηχανής για διάφορες τιμές των παραμέτρων γ και c
Συνολικός χρόνος: 60 λεπτά

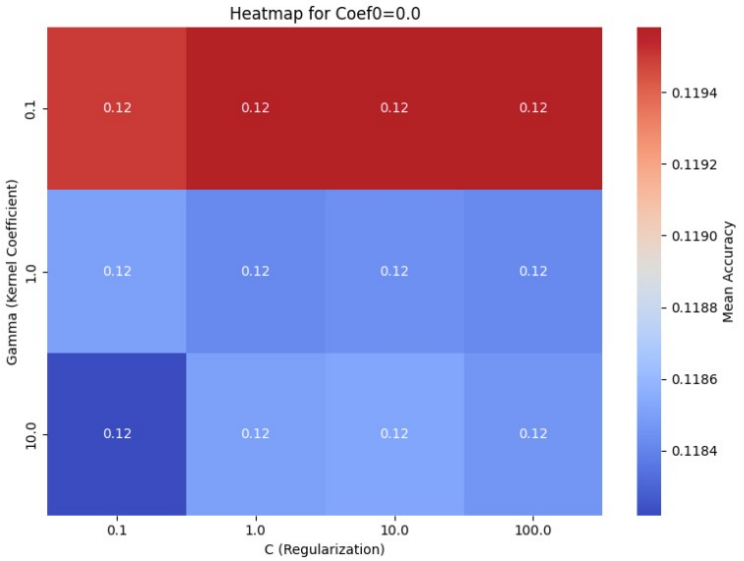
Η **ευστοχία** μεγιστοποιείται για **c=100** και **γ = 0.001**. Στη συνέχεια, παρουσιάζεται ο πίνακας σύγχυσης.

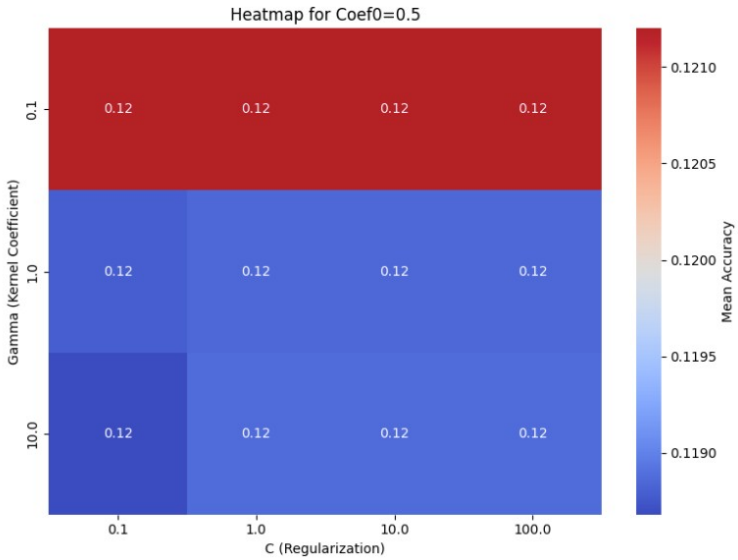
Πίνακας σύγχυσης για Πολυωνυμικό Kernel 2ου βαθμού με σταθέρα 2, c = 100 και γ = 0.001

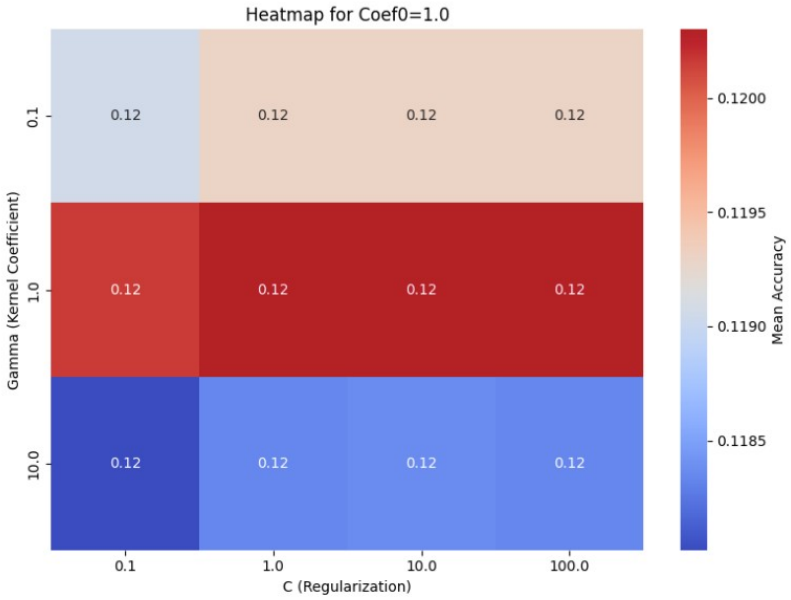


Sigmoid kernel:

Δοκιμάστηκαν διάφορες τιμές για τις παραμέτρους c και γ σε σχέση με τον σταθερό συντελεστή. Ο συνολικός χρόνος για τις δοκιμές ήταν 125 λεπτά.

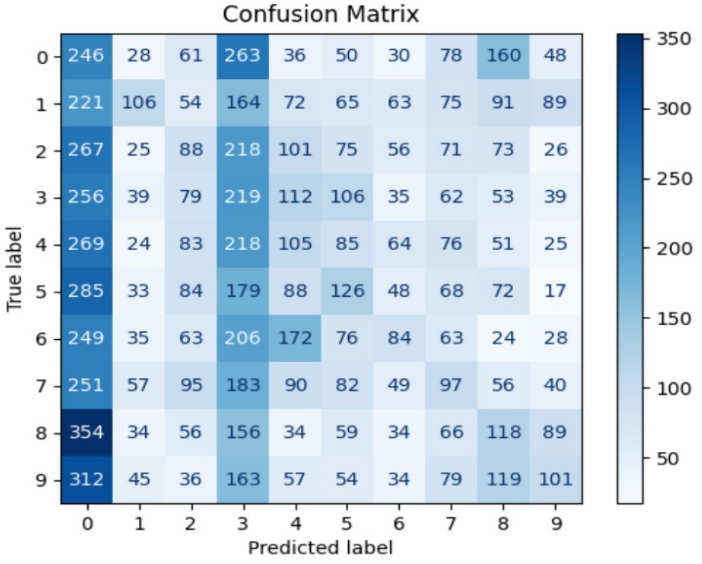




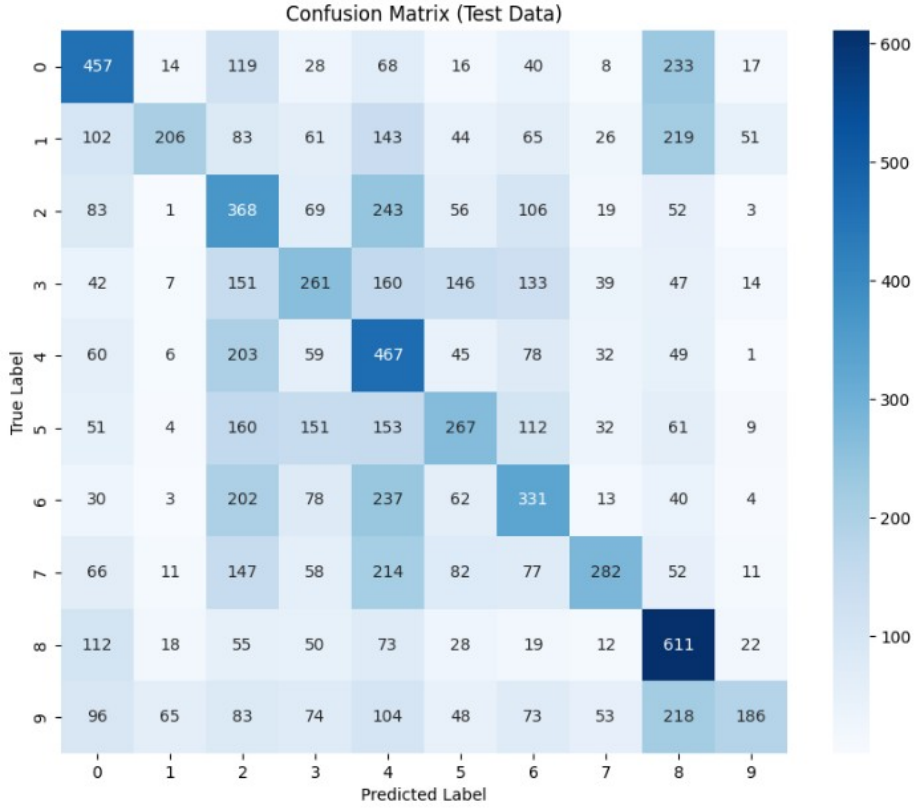


Ο καλύτερος συνδυασμός είναι για **c = 0.1, γ = 0.1, coef = 0.5** με διάγραμμα σύγχυσης:

Πίνακας σύγχυσης για Sigmoid kernel με σταθερά 0.5, c = 0.1 και γ = 0.1



Βάσει των παραπάνω πειραμάτων, η μέγιστη ακρίβεια επιτεύχθηκε με τον **RBF kernel** και παραμέτρους **c=0.001** και **γ=0.1**. Συνεπώς, πραγματοποιήθηκε περαιτέρω εκπαίδευση για 4000 επαναλήψεις, και τα αποτελέσματα είναι τα εξής:

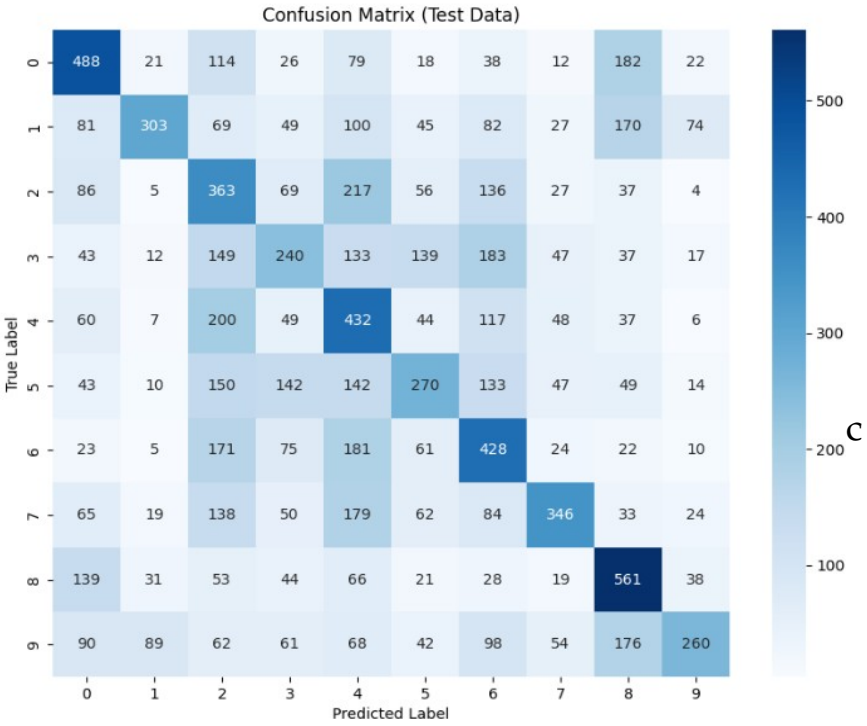


Πίνακας σύγχυσης για RBF kernel με c = 0.001 και γ = 0.1 χωρίς PCA

Ο συνολικός χρόνος εκπαίδευσης ήταν 69 λεπτά, με επιτευχθείσα ακρίβεια **34.36%**. Στη συνέχεια, πραγματοποιήθηκαν δοκιμές με μείωση διαστάσεων χρησιμοποιώντας τη μέθοδο PCA.

→ Για διατήρηση 90 % της πληροφορίας,

συνολικό χρόνο εκπαίδευσης 6 λεπτά και ευστοχία **36.91 %:**

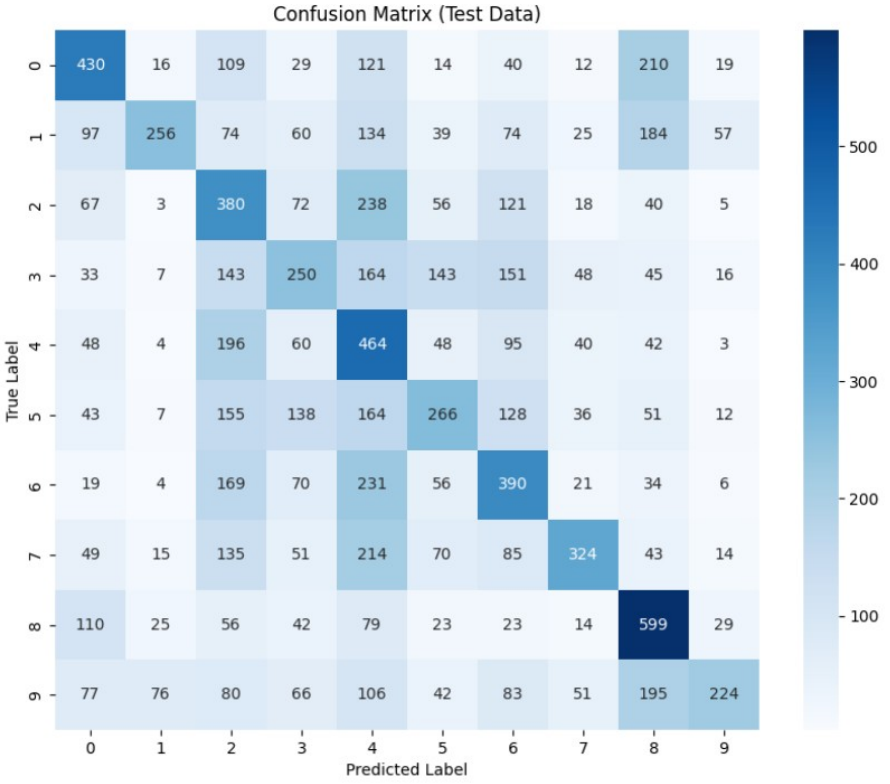


Confusion matrix for RBF kernel with = 0.001 and γ = 0.1 with PCA = 0.90

→ Για διατήρηση 95 % της πληροφορίας,

με συνολικό χρόνο εκπαίδευσης 12 λεπτά και ευστοχία **35.85 %:**

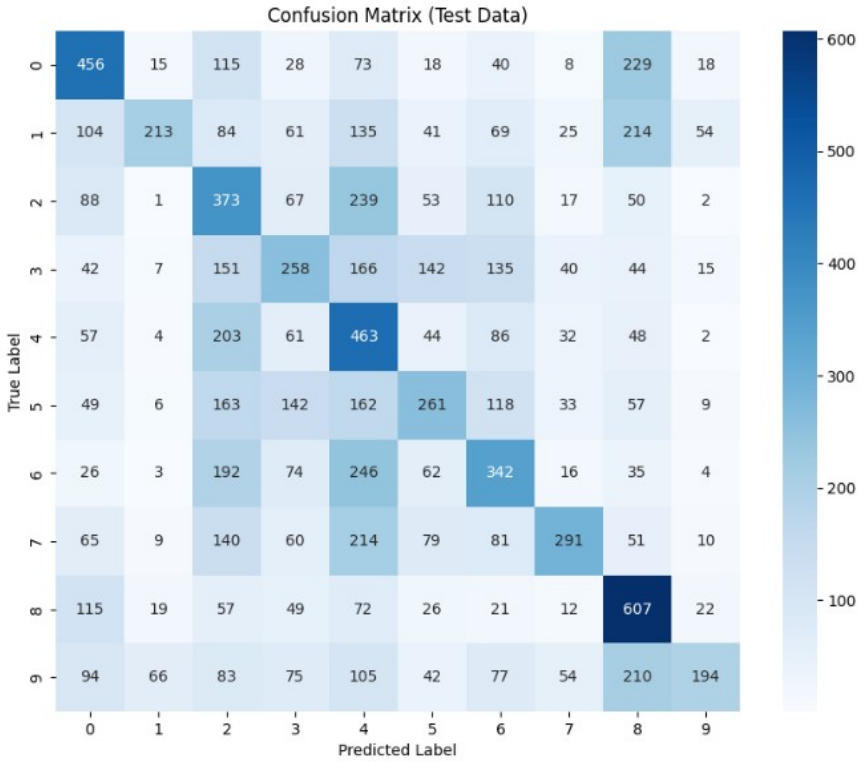
Confusion matrix for RBF kernel with c = 0.001 and γ = 0.1 with PCA = 0.95



→ Για διατήρηση 99 % της πληροφορίας,

με συνολικό χρόνο εκπαίδευσης 25 λεπτά και ευστοχία **34.58 %:**

Confusion matrix for RBF kernel with c = 0.001 and γ = 0.1 with PCA = 0.99



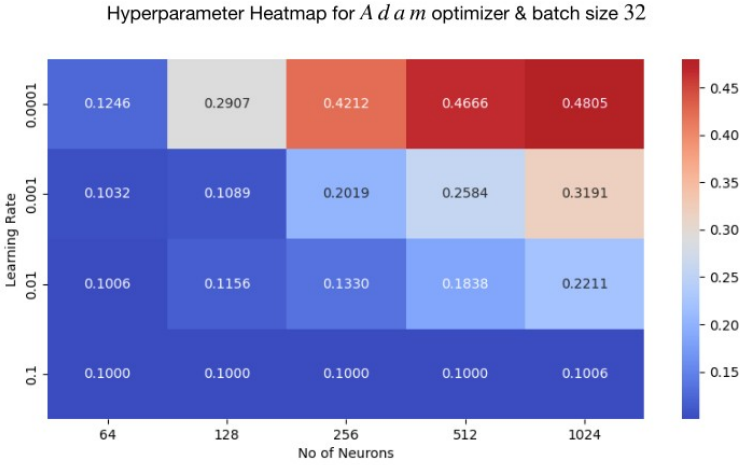
Τελικά, για RBF kernel με c = 0.001 , γ = 0.1 και μείωση διαστάσεων μέσω PCA που διατηρεί το 90 % της πληροφορίας, επιτεύχθηκε η μέγιστη ευστοχία **36.91 %** .

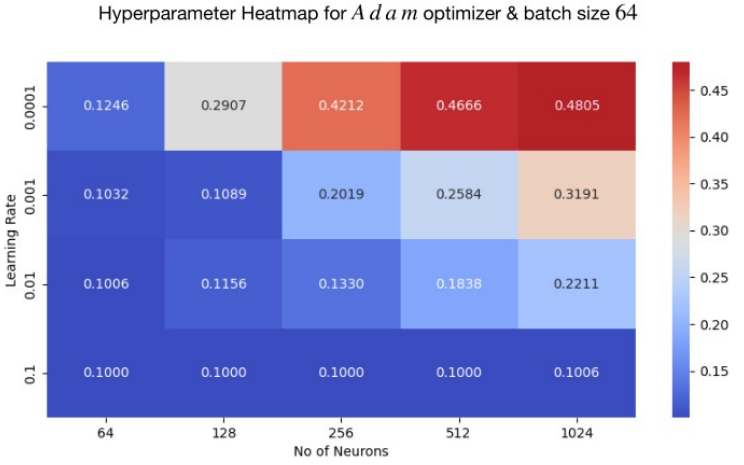
Σύγκριση με άλλους κατηγοριοποιητές

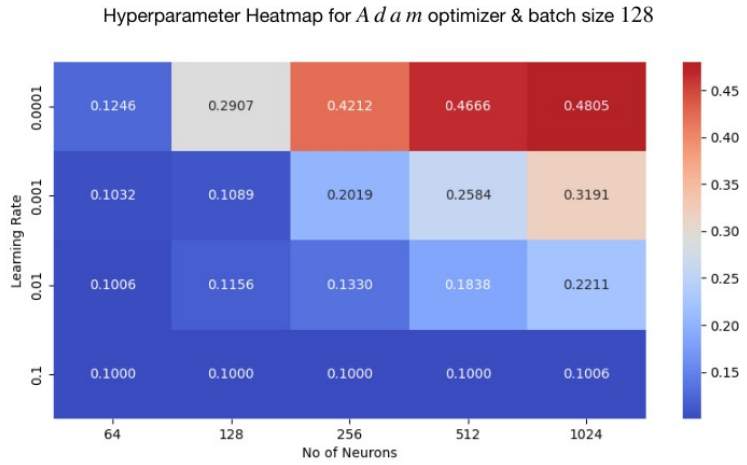
Για κατηγοριοποίηση με 1 και 3 πλησιέστερου γείτονα (Nearest Neighbor) και πλησιέστερου κέντρου κλάσης (Nearest Class Centroid) προκύπτει:

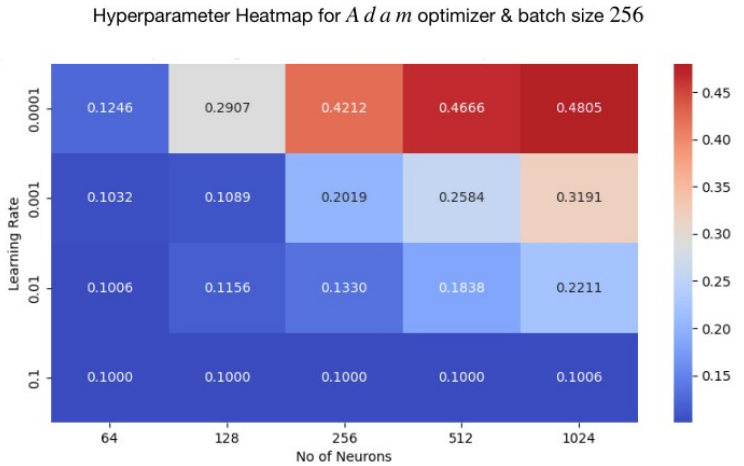
Accuracy (%)	No PCA	PCA (90 % retention)
KNN (1 Neighbor)	35	39
KNN (3 Neighbors)	33	37
NCC	28	28

Για το **MLP** με ένα κρυφό επίπεδο και Hinge loss ως συνάρτηση βελτιστοποίησης, πραγματοποιήθηκαν διάφορες δοκιμές. Εξετάστηκαν διαφορετικοί αριθμοί νευρώνων στο κρυφό επίπεδο (64, 128, 256, 512, 1024), ποικίλοι ρυθμοί εκμάθησης (0.1, 0.01, 0.001, 0.0001), καθώς και μεγέθη batch (32, 64, 128, 256). Ο συνολικός χρόνος εκπαίδευσης ανήλθε στις **5** ώρες.







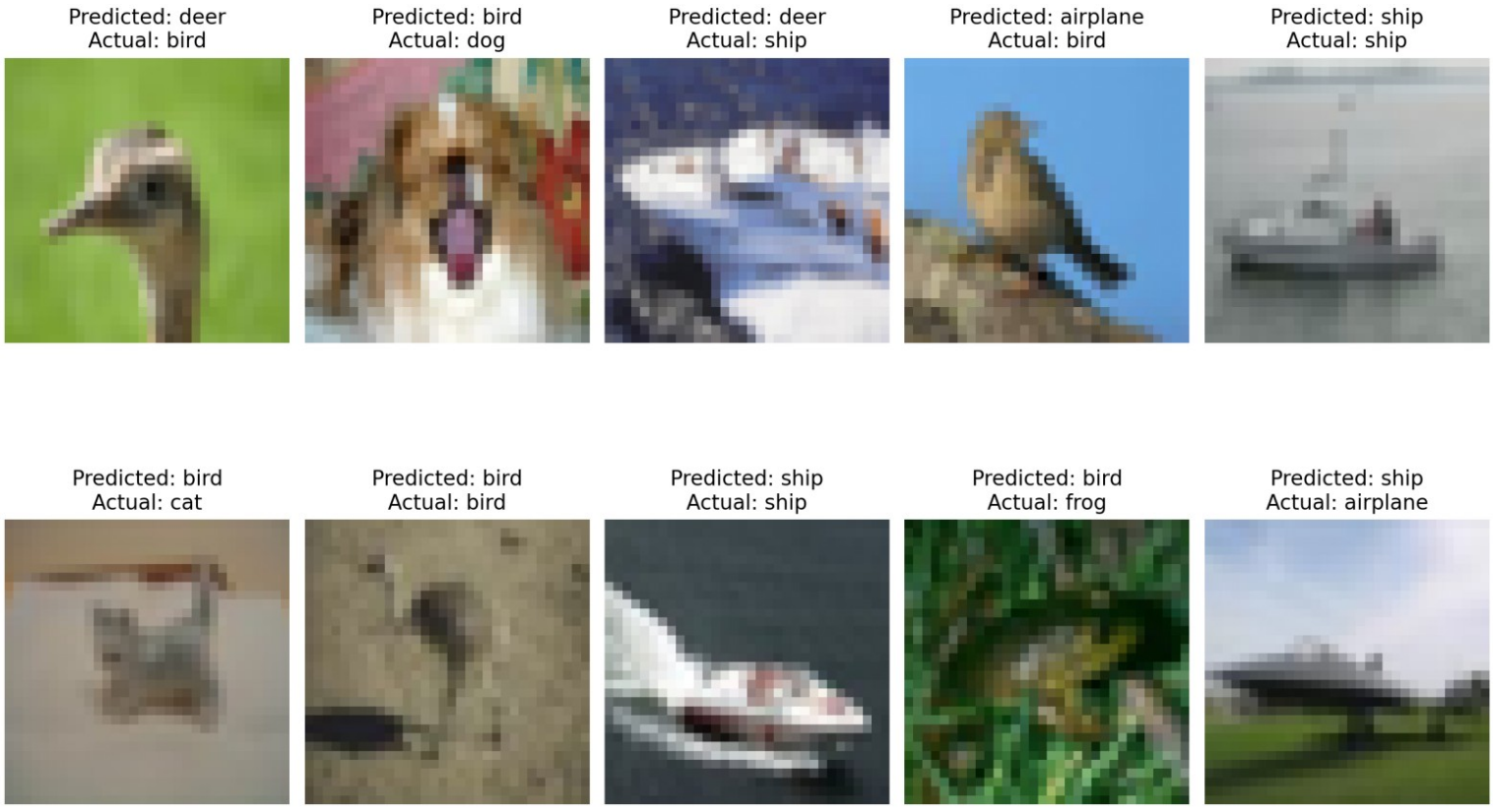


Φαίνεται ότι η ευστοχία ελέγχου υπερβαίνει το 48 % , ποσοστό το οποίο **ξεπερνά** το 36.91 % του SVM.

Αποτελέσματα:

Η καλύτερη απόδοση του MLP αποδίδεται στην ικανότητά του να μαθαίνει σύνθετα χαρακτηριστικά απευθείας από τα δεδομένα. Το kNN με k=1 υπερέχει του k=3, καθώς βασίζεται στον πλησιέστερο γείτονα, αποφεύγοντας την ευαισθησία σε θορυβώδη ή παραπλανητικά δεδομένα. Αντίθετα, το k=3 επηρεάζεται από γείτονες άλλων κατηγοριών, οδηγώντας σε απόδοση παρόμοια με αυτή του SVM, το οποίο επηρεάζεται από την επιλογή υπερπαραμέτρων και την πολυπλοκότητα του CIFAR-10. Ο NCC έχει τη χαμηλότερη απόδοση, επειδή βασίζεται στη γραμμική προσέγγιση των χαρακτηριστικών, η οποία είναι ανεπαρκής για τη συγκεκριμένη βάση δεδομένων.

Έλεγχος SVM:



Παραδείγματα κατηγοριοποίησης SVM

Η ανάλυση των αποτελεσμάτων κατηγοριοποίησης από το μοντέλο μας αναδεικνύει σημαντικά συμπεράσματα σχετικά με τις δυνατότητες και τους περιορισμούς του αλγορίθμου στο συγκεκριμένο σύνολο δεδομένων. Για παράδειγμα, η εσφαλμένη κατηγοριοποίηση ενός πουλιού ως ελάφι μπορεί να οφείλεται σε οπτικές ομοιότητες, όπως το χρώμα και η υφή του φόντου, χαρακτηριστικά που συναντώνται σε φυσικά τοπία, όπως δάση. Αυτές οι ομοιότητες ενδέχεται να προκαλέσουν στο μοντέλο τη λανθασμένη εντύπωση ότι ανήκουν στην ίδια κατηγορία.

Παρομοίως, η ταξινόμηση ενός σκύλου ως πουλί μπορεί να αποδοθεί σε παραμορφώσεις στο σχήμα του ζώου μέσα στην εικόνα, που δημιουργούν αμφισημίες στην ερμηνεία του μοντέλου. Τέτοιου είδους αναπαραστάσεις μπορούν να οδηγήσουν σε εσφαλμένες προβλέψεις.

Επιπλέον, ένα πουλί που ταξινομείται λανθασμένα ως γάτα πιθανόν να επηρεάζεται από θολές εικόνες ή από ομοιότητες στο χρώμα και την υφή του φόντου, που ενώνουν τις δύο κατηγορίες.

Αντίθετα, η σωστή κατηγοριοποίηση ενός ελαφιού ως ελάφι είναι πιθανό να προκύπτει από την αναγνώριση χαρακτηριστικών που είναι χαρακτηριστικά της κατηγορίας, όπως το σχήμα του σώματος ή το χρώμα του.

Κώδικας:

ToulkeridisNikolaosErgasia2_10718.py:

- Εισαγωγή Δεδομένων

```
# Load the CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Normalize pixel values to [0, 1]
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Flatten the images into 1D vectors (32x32x3 -> 3072)
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)
```

Τα δεδομένα εισάγονται και χωρίζονται σε δεδομένα εκπαίδευσης (X_train, y_train) και δεδομένα δοκιμών (X_test, y_test). Οι τιμές των pixel κανονικοποιούνται στο διάστημα [0, 1], ενώ οι εικόνες μετασχηματίζονται από 32x32x3 πίνακες σε μονοδιάστατα διανύσματα 3072 χαρακτηριστικών.

- Τυποποίηση Δεδομένων

```
print('Standardizing data...')
# Standardize features to have mean 0 and variance 1
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Reduce dimensions using PCA (keeping 99% of the variance)
pca = PCA(n_components=0.99)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

Η τυποποίηση των δεδομένων γίνεται για να αποκτήσουν μέση τιμή 0 και διακύμανση 1, εξασφαλίζοντας ομοιομορφία στα χαρακτηριστικά. Επιπλέον, εφαρμόζεται PCA για μείωση της διάστασης των δεδομένων, διατηρώντας το 99% της πληροφορίας.

• Εκπαίδευση SVM

```
# Toggle grid search mode
gridSearch = False

if not gridSearch:

    # Train and evaluate a single SVM model
    start_time = time.time()

    # Create the SVM model
    svm = SVC(kernel='rbf', C=0.001, gamma = 0.1, max_iter=4000)
    svm.fit(X_train, y_train)


    # Step 8: Evaluate the final model on the test data
    y_pred_test = svm.predict(X_test)
    print("Test Accuracy:", accuracy_score(y_test, y_pred_test))
    print("Classification Report (Test):")
    print(classification_report(y_test, y_pred_test))

   # Measure training time
    end_time = time.time()
    print("training time: " + str(-start_time + end_time))

    # Step 9: Plot the confusion matrix for test results
    cm = confusion_matrix(y_test, y_pred_test)
    plt.figure(figsize=(10, 8))
```

```
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=range(10),
yticklabels=range(10))
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.title("Confusion Matrix (Test Data)")
    plt.show()

    joblib.dump(svm, 'svm_model.pkl')
```

Ο χρήστης έχει τη δυνατότητα να επιλέξει είτε την απλή εκπαίδευση ενός SVM με προκαθορισμένες παραμέτρους είτε την αναζήτηση βέλτιστων παραμέτρων μέσω Grid Search. Εάν η επιλογή του Grid Search είναι απενεργοποιημένη (gridSearch = False), εκπαιδεύεται ένα μοντέλο SVM με πυρήνα RBF και συγκεκριμένες τιμές για τις παραμέτρους C και gamma.

• Ορισμός πλέγματος υπερπαραμέτρων

```
# gridIndex
# Determines which grid search will be executed
# Value 0: linear Kernel
# Value 1: rbf Kernel
# Value 2: polynomial Kernel
# Value 3: sigmoid Kernel

# The actual parameter grid is shown below
gridIndex = 3

# Define the parameter grid
param_grid = [
    {
        'kernel': ['linear'],
        'C': [0.01, 0.1, 1, 10, 100]
    },
    {
        'kernel': ['rbf'],
        'C': [0.00001, 0.0001, 0.001, 0.01, 0.1],
        'gamma': [0.1]
    },
    {
        'kernel': ['poly'],
        'C': [100, 1000, 10000, 100000],
        'gamma': [0.0001, 0.001, 0.01, 0.1],
        'degree': [2],
        'coef0': [2.0]
    },
    {
        'kernel': ['sigmoid'],
        'C': [0.1, 1, 10, 100],
        'gamma': [0.1, 1, 10],
        'coef0': [0, 0.5, 1]
    }
]
```

Αν επιλέγει αναζήτηση βέλτιστων παραμέτρων μέσω Grid Search, ο χρήστης επιλέγει το είδος SVM που θα εκπαιδευτούν από το πλέγμα υπερπαραμέτρων.

• Εκπαίδευση του μοντέλου

```
start_time = time.time()

# Train an SVM classifier with RBF kernel
svm = SVC(max_iter=60)
```

```
svm.fit(X_train, y_train)

cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=42)

# Create the GridSearchCV object
grid_search = GridSearchCV(svm, param_grid[gridIndex], cv=cv, n_jobs=-1)
# Fit the model
grid_search.fit(X_train, y_train)
```

Η GridSearchCV εκπαιδεύει το μοντέλο SVM με κάθε συνδυασμό παραμέτρων και υπολογίζει την ακρίβεια για κάθε συνδυασμό, χρησιμοποιώντας διασταυρωμένη επικύρωση. Για κάθε επανάληψη της cross-validation, τα δεδομένα εκπαίδευσης χωρίζονται σε νέο σύνολο εκπαίδευσης και επικύρωσης.

• Αποτελέσματα και επιλογή του καλύτερου μοντέλου

```
end_time = time.time()
print("training time: " + str(end_time - start_time))

results = grid_search.cv_results_

# Print the best parameters found
print(f"Best Parameters: {grid_search.best_params_}")

# Get the best model from grid search
best_model = grid_search.best_estimator_

# Evaluate the model on the test set
test_accuracy = best_model.score(X_test, y_test)
print(f"Test Accuracy: {test_accuracy:.4f}")


# Make predictions on the test set
y_pred = grid_search.best_estimator_.predict(X_test)
y_pred = svm.predict(X_test)

# Evaluate the SVM classifier
print("Classification report on validation data:")
print(classification_report(y_test, y_pred))
```

Αφού ολοκληρωθεί η αναζήτηση, τα αποτελέσματα αποθηκεύονται και το καλύτερο μοντέλο επιστρέφεται.

• Οπτικοποίηση αποτελεσμάτων

```
cm = confusion_matrix(y_test, y_pred)
ConfusionMatrixDisplay(confusion_matrix=cm).plot(cmap="Blues")
plt.title("Confusion Matrix")
plt.show()

if(param_grid == 1):
    # Convert the GridSearchCV results to a DataFrame
    results_df = pd.DataFrame(grid_search.cv_results_)
    pd.set_option('display.max_rows', None)  # Show all rows
    pd.set_option('display.max_columns', None)  # Show all columns
    pd.set_option('display.width', None)  # Adjust the display width to fit the
data
    pd.set_option('display.max_colwidth', None)  # Show full column content
    print(results_df)

    # Create a pivot table to visualize the mean_test_score for C and gamma
    pivot_table = results_df.pivot_table(
```

```
        index="param_C",           # Row labels (C values)
        columns="param_gamma",     # Column labels (gamma values)
        values="mean_test_score"   # Values to display (mean test score)
    )

    # Visualize the pivot table as a heatmap
    sns.heatmap(pivot_table, annot=True, fmt=".3f", cmap="coolwarm")
    plt.title("Hyperparameter Heatmap (C vs. Gamma)")
    plt.xlabel("Gamma")
    plt.ylabel("C")
    plt.show()
```

Τα αποτελέσματα μπορούν να απεικονιστούν με confusion matrix για να φανούν οι τα ισχυρά και αδύναμα σημεία του των καλύτερων παραμέτρων ανά κλάση και heatmaps για να αναδειχθεί η επίδραση των παραμέτρων στην ακρίβεια.

→ ToulkeridisNikolaosErgasia2_10718_MLP.py

• Φόρτωση και Προεπεξεργασία Δεδομένων

```
# Load and preprocess CIFAR-10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize the images
x_train = x_train / 255.0
x_test = x_test / 255.0

# Split into training and validation sets
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train,
test_size=0.2, random_state=42)

# Convert labels to categorical (one-hot encoding)
y_train = to_categorical(y_train, num_classes=10)
y_val = to_categorical(y_val, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

Φορτώνονται και προετοιμάζονται τα δεδομένα CIFAR-10. Οι εικόνες κανονικοποιούνται στο διάστημα [0, 1], και τα δεδομένα χωρίζονται σε training, validation και test sets. Παράλληλα, οι ετικέτες κατηγοριοποίησης μετατρέπονται σε μορφή one-hot encoding.

• Ορισμός Υπερπαραμέτρων

```
EPOCHS = 60

# Define hyperparameters
neurons = [64, 128, 256, 512, 1024]
learning_rates = [0.1, 0.01, 0.001, 0.0001]
batch_sizes = [32, 64, 128, 256]
```

~ neurons: Αριθμός νευρώνων στο κρυφό επίπεδο.
~ learning_rates: Ρυθμός εκμάθησης για τον βελτιστοποιητή.
~ batch_sizes: Μέγεθος batch κατά την εκπαίδευση.

• Εκπαίδευση και Αποθήκευση Αποτελεσμάτων

```
resultsAdam = []
start_time = time.time()
```

```
# Train the model for different combinations of hyperparameters
for k in range (len(batch_sizes)):
    for j in range (len(learning_rates)):
        for i in range (len(neurons)):

            # Create the MLP model
            model = Sequential([
                Flatten(input_shape=(32, 32, 3)),
                Dense(neurons[i], activation='relu'),
                Dense(10, activation='linear')
            ])

            # Define the optimizer and the loss function
            optimizer = Adam(learning_rate=learning_rates[j])
            model.compile(optimizer=optimizer, loss='categorical_hinge',
metrics=['accuracy'])

            # Train the model
            history = model.fit(x_train, y_train, epochs=EPOCHS,
batch_size=batch_sizes[k], validation_data=(x_val, y_val), verbose=0)

            # Evaluate on the test set
            test_loss, test_acc = model.evaluate(x_test, y_test)
            print(f"Test Accuracy: {test_acc:.2f}")
            test_loss, test_accuracy = model.evaluate(x_test, y_test)

            # Store the results
            resultsAdam.append({'learning_rate': learning_rates[j], 'neurons':
neurons[i], 'batch_size': batch_sizes[k], 'test_accuracy': test_accuracy})

end_time = time.time()
total_time = end_time - start_time
print(f'Total time: {total_time:.2f} ')
```

Για κάθε συνδυασμό υπερπαραμέτρων, εκπαιδεύεται ένα MLP μοντέλο, καταγράφεται η ακρίβεια στο test set και τα αποτελέσματα αποθηκεύονται σε λίστα.

- Δημιουργία Heatmaps

```
results_df = pd.DataFrame(resultsAdam)
print(results_df)

# Generate heatmaps for each batch size
batch_values = results_df['batch_size'].unique()

for batch in batch_values:
    subset = results_df[(results_df['batch_size'] == batch)]

    pivot_table = results_df.pivot_table(
        index="learning_rate",                    # Row labels (C values)
        columns="neurons",                        # Column labels (gamma
values)
        values="test_accuracy"                    # Values to display (mean
test score)
    )

    # Create the heatmap
    sns.heatmap(pivot_table, annot=True, fmt=".4f", cmap="coolwarm")
    plt.title(f"Hyperparameter Heatmap (Learning Rate vs. No of Neurons) for
optimizer Adam & batch size {batch}")
    plt.xlabel("No of Neurons")
    plt.ylabel("Learning Rate")
    plt.show()
```

Τα αποτελέσματα αποθηκεύονται σε ένα DataFrame και οπτικοποιούνται μέσω heatmaps. Κάθε heatmap δείχνει την ακρίβεια στο test set για συγκεκριμένα μεγέθη batch.

→ ToulkeridisNikolaosErgasia_2_10718_KNN_NCC.py

• Φόρτωση και Προεπεξεργασία Δεδομένων

```
# Load CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Normalize the pixel values to the range [0, 1]
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Transform data into 1D array
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

pca = PCA(n_components=0.9)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

• Ταξινόμηση με KNN για 1 Γείτονα

```
knn = KNeighborsClassifier(n_neighbors=1, metric='euclidean')
knn.fit(X_train, y_train.ravel())
y_pred = knn.predict(X_test)
acc1 = accuracy_score(y_test, y_pred)
print(f"Accuracy for 1 neighbour: {acc1:.2f}")
```

Ορίζεται n_neighbors=1, γεγονός που σημαίνει ότι η ταξινόμηση κάθε δείγματος βασίζεται στον κοντινότερο γείτονα στο χώρο χαρακτηριστικών.

• Ταξινόμηση με KNN για 3 Γείτονες

```
knn = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
knn.fit(X_train, y_train.ravel())
y_pred = knn.predict(X_test)
acc3 = accuracy_score(y_test, y_pred)
print(f"Accuracy for 3 neighbours: {acc3:.2f}")
```

Εδώ, το κάθε δείγμα ταξινομείται σύμφωνα με την πλειοψηφία των τριών πιο κοντινών γειτόνων.

• Ταξινόμηση με NCC

```
# Train NearestCentroid classifier
ncc = NearestCentroid(metric='euclidean')
ncc.fit(X_train, y_train.ravel())

# Predict and evaluate
y_pred = ncc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"NCC Accuracy: {accuracy:.2f}")
```

Η μέθοδος Nearest Class Centroid βασίζεται στη μέση τιμή των χαρακτηριστικών κάθε κλάσης. Για κάθε δείγμα, υπολογίζεται η απόσταση από τα κεντροειδή των κλάσεων, και το δείγμα ταξινομείται στην κλάση με την πλησιέστερη απόσταση.

→ ToulkeridisNikolaosErgasia2_10718_Test.py

Η αρχικοποίηση και προεπεξεργασία των δεδομένων είναι ακριβώς ίδια με αυτή στην εκπαίδευση.

• Φόρτωση Εκπαιδευμένου Μοντέλου SVM

```
# Load the trained SVM model
svm_model = joblib.load("svm_model.pkl")
```

Φορτώνεται το εκπαιδευμένο μοντέλο SVM από ένα αρχείο που έχει αποθηκευτεί στον δίσκο. Αυτό επιτρέπει τη χρήση του μοντέλου χωρίς να χρειάζεται νέα εκπαίδευση.

• Επιλογή Τυχαίων Εικόνων για Τεστ

```
# Choose some random images from the test set
num_images = 10  # Number of images to display
random_indices = np.random.choice(X_test.shape[0], num_images, replace=False)
sample_images = original_test_images[random_indices]
sample_labels = y_test[random_indices]
sample_transformed = X_test[random_indices]
```

Επιλέγεται τυχαία δείγματα εικόνων από το σετ δοκιμών για οπτικοποίηση και πρόβλεψη. Αυτό επιτρέπει την αξιολόγηση του μοντέλου σε δείγματα που δεν έχουν χρησιμοποιηθεί στην εκπαίδευση.

• Πρόβλεψη Κατηγοριών με το SVM

```
# Predict the classes using the SVM
predicted_classes = svm_model.predict(sample_transformed)
```

Προβλέπονται τις κατηγορίες των επιλεγμένων εικόνων χρησιμοποιώντας το μοντέλο SVM. Αυτό αποτελεί το βασικό βήμα για την αξιολόγηση της ακρίβειας του μοντέλου σε πραγματικές συνθήκες.

- Οπτικοποίηση Αποτελεσμάτων

```
# Plot the results
plt.figure(figsize=(15, 8))
for i in range(num_images):
    plt.subplot(2, 5, i + 1)
    plt.imshow(sample_images[i])
    plt.title(
        f"Predicted: {class_names[predicted_classes[i]]}\nActual:
{class_names[sample_labels[i][0]]}"
    )
    plt.axis("off")
plt.tight_layout()
plt.show()
```

Παρουσιάζονται οι εικόνες και οι σχετικές προβλέψεις, επισημαίνοντας την προβλεπόμενη και την πραγματική κατηγορία κάθε εικόνας. Αυτό βοηθά στην οπτική αξιολόγηση της απόδοσης του μοντέλου.