

Νευρωνικά Δίκτυα - Βαθιά Μάθηση

2^η Υποχρεωτική Εργασία
Παπαδόπουλος Παναγιώτης 10697 ΗΜΜΥ

Σετ Δεδομένων

Για την ανάπτυξη ενός **Support Vector Machine** επιλέχθηκε η βάση δεδομένων Cifar-10. Αποτελείται από 60.000 εικόνες ανάλυσης 32x32 πίξελ με κωδικοποίηση RGB που ανήκουν σε 10 κλάσεις, με 6.000 εικόνες ανά κλάση. Υπάρχουν 50.000 εικόνες εκπαίδευσης και 10.000 εικόνες ελέγχου (testing). Οι εικόνες κανονικοποιούνται ώστε οι τιμές των πίξελ να είναι εντός του εύρους [0,1], να έχουν μηδενική μέση τιμή και μοναδική διακύμανση.

Εργαλεία

Για την υλοποίηση της μηχανής επιλέχθηκε η γλώσσα προγραμματισμού *python* και η βιβλιοθήκη *keras* για την ανάπτυξή της. Άλλες βιβλιοθήκες που χρησιμοποιήθηκαν είναι οι εξής:

- *pumpry*, για τον εύκολο χειρισμό δεδομένων
- *sklearn*, για συμπληρωματικές λειτουργίες στην προεπεξεργασία και ανάλυση των δεδομένων
- *seaborn*, για την παρουσίαση διαγραμμάτων της μορφής heatmap
- *time*, για την μέτρηση του χρόνου εκπαίδευσης
- *matplotlib*, για την παρουσίαση διαγραμμάτων
- *tensorflow*, για την εκπαίδευση του *SVM*

Ανάπτυξη της μηχανής

Η κατηγοριοποίηση που θα εξετασατεί είναι **One-vs-One**, η οποία εξετάζει κάθε ένα δείγμα έναντι ένα άλλο για κάθε άλλο δείγμα. Θα πραγματοποιηθεί *Grid Search* για την εύρεση των βέλτιστων παραμέτρων. Τα παρακάτω πειράματα διεξάχθηκαν με 60 μέγιστες επαναλήψεις, ανοχή (*tolerance*) 10^{-3} , 5-πτυχιακή διασταυρούμενη επικύρωση (*5-fold cross validation*) και μέτρο του καλύτερου μοντέλου ήταν το *mean test score*. Τέλος, στο μοντέλο με την καλύτερη απόδοση γίνεται συνολικός έλεγχος με τα δεδομένα ελέγχου που αναφέρθηκαν παραπάνω.

Γραμμικό kernel:

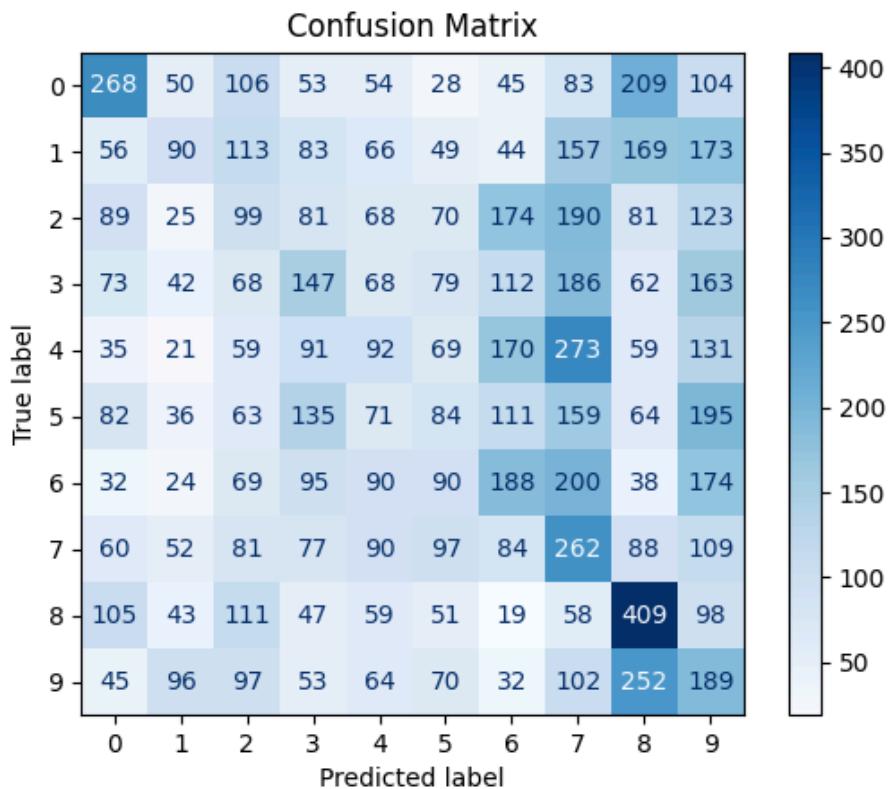
Έγιναν δοκιμές για γραμμικό *kernel* με τιμές για την παράμετρο *c* : [0.01, 0.1, 1, 10, 100].

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

Αποτελέσματα για κάθε τιμή της παραμέτρου c και $fold$ του *cross validation*.

Συνολικός χρόνος: 21 λεπτά

Είναι εμφανές ότι η ευστοχία παραμένει σταθερή ανεξαρτήτως της τιμής του c . Επιλέχθηκε τελικός έλεγχος για $c = 0.01$ με ευστοχία ελέγχου 0.1828. Ακολουθεί πίνακας σύγχυσης για τον καλύτερο εκτιμητή (*best estimator*).

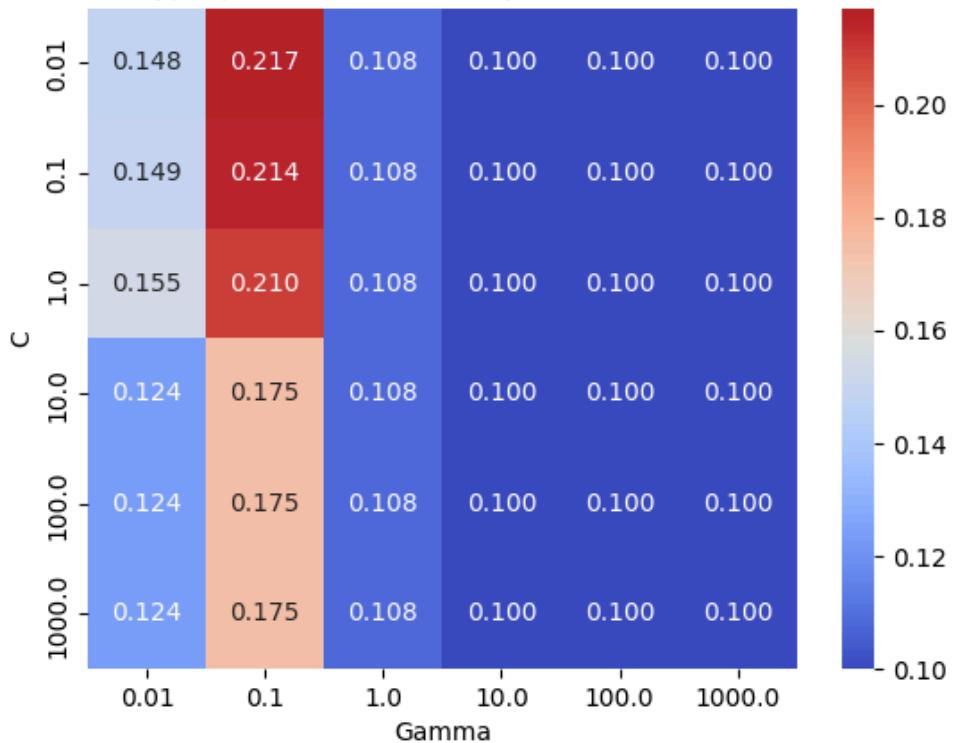


Confusion matrix for linear kernel with $c = 0.01$

RBF kernel:

Έγιναν δοκιμές για *RBF kernel* με διάφορες τιμές για τις παραμέτρους c και γ . Τα αποτελέσματα φαίνονται στον παρακάτω πίνακα.

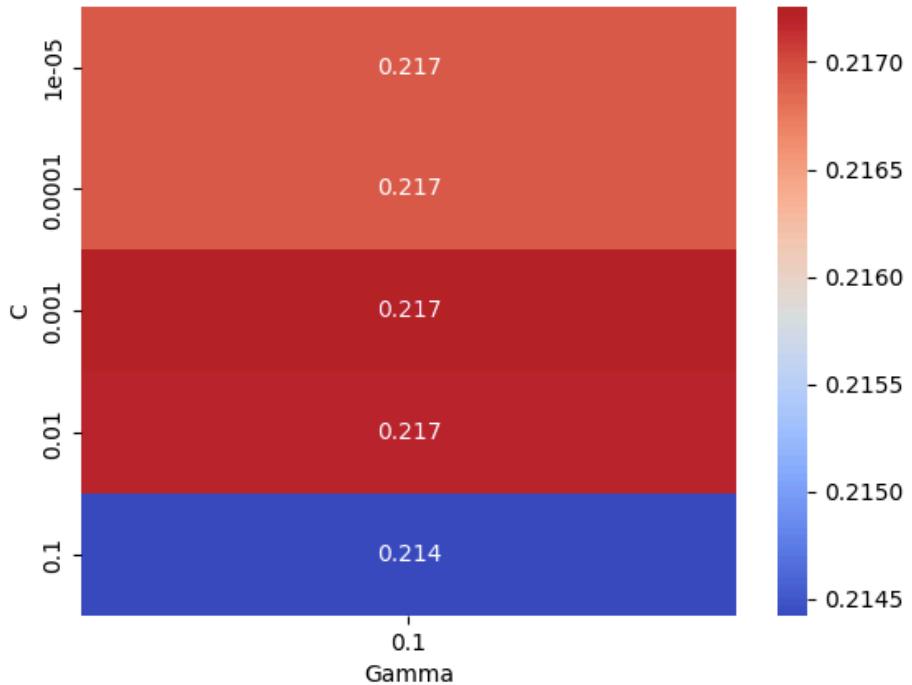
Hyperparameter Heatmap (C vs. Gamma)



Heatmap της ευστοχίας της μηχανής για διάφορες τιμές των παραμέτρων
Συνολικός χρόνος: 90 λεπτά

Εμφανώς η ευστοχία είναι μεγαλύτερη για $c = 0.01$ και $\gamma = 0.1$. Επειδή η ευστοχία αυξάνεται καθώς το c μικραίνει, γίνεται περαιτέρω έλεγχος για την συμπεριφορά της ευστοχίας σε μικρότερες τιμές c .

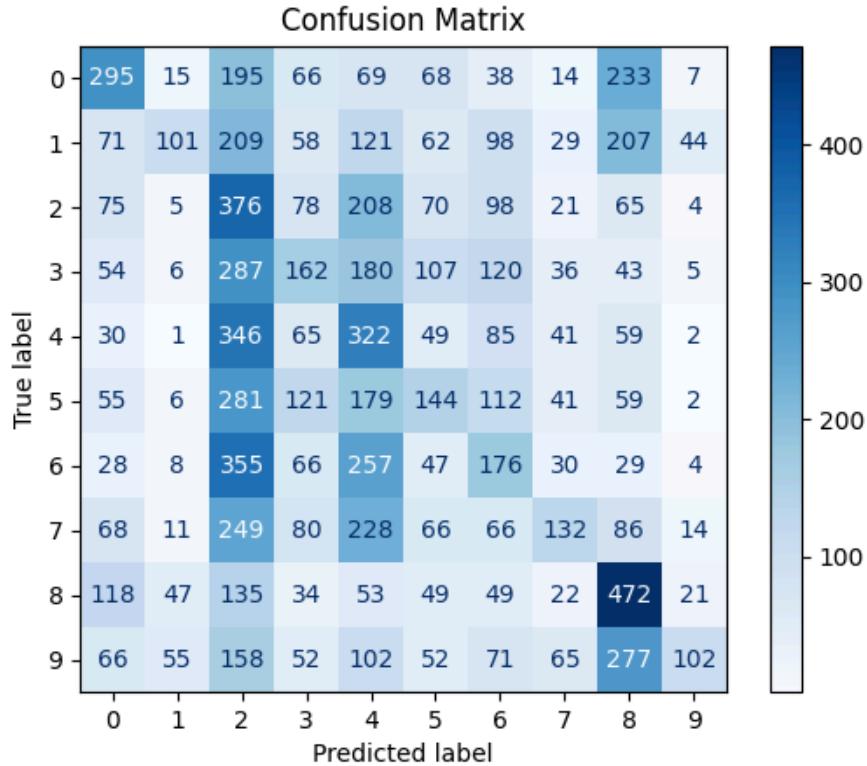
Hyperparameter Heatmap (C vs. Gamma)



Heatmap της ευστοχίας της μηχανής για διάφορες τιμές των παραμέτρων
Συνολικός χρόνος: 14 λεπτά

Λόγω στρογγυλοποίησης δεν είναι ξεκάθαρο στο διάγραμμα, ωστόσο η ευστοχία ήταν μέγιστη για $c = 0.001$.

Επομένως, για *RBF kernel* ο ιδανικός συνδυασμός παραμέτρων είναι $c = 0.001$ και $\gamma = 0.1$ με ευστοχία 0.2282. Ακολουθεί πίνακας σύγχυσης.

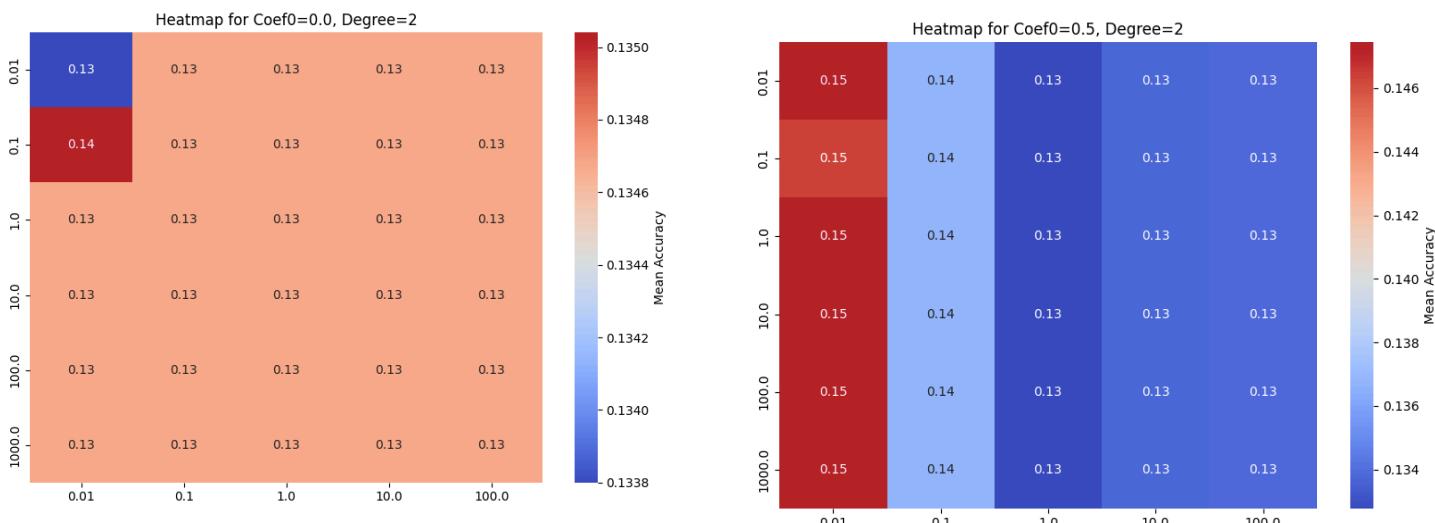


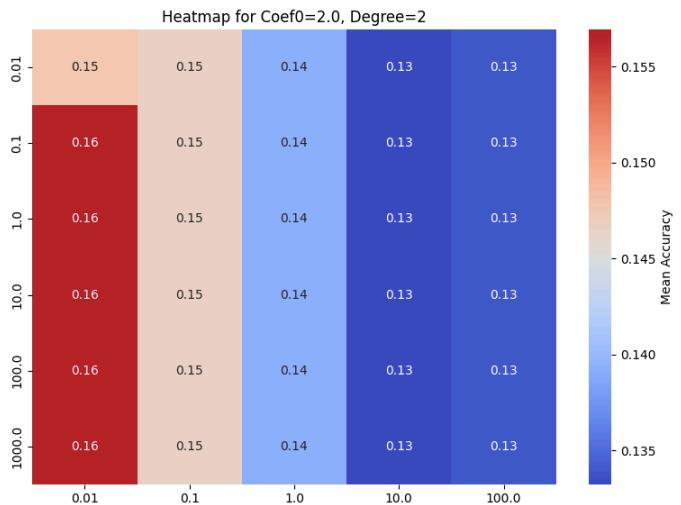
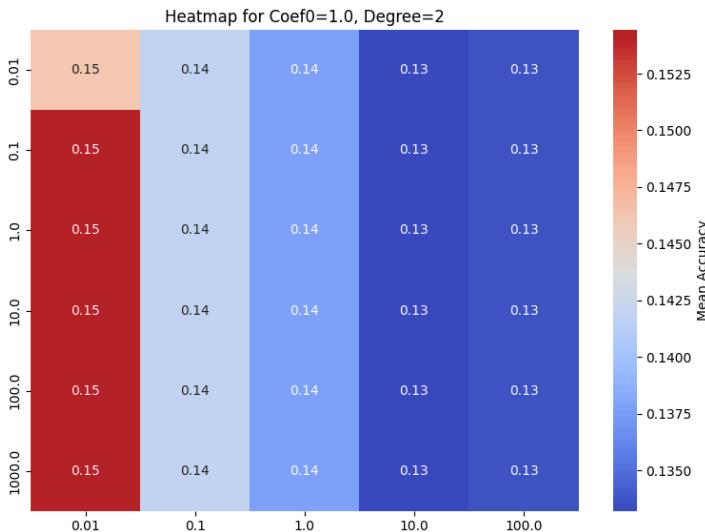
Confusion matrix for RBF kernel with $c = 0.001$ and $\gamma = 0.1$

Polynomial kernel:

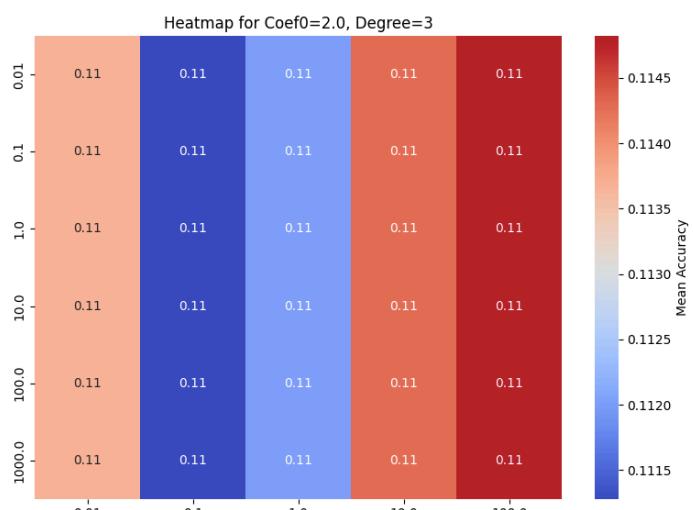
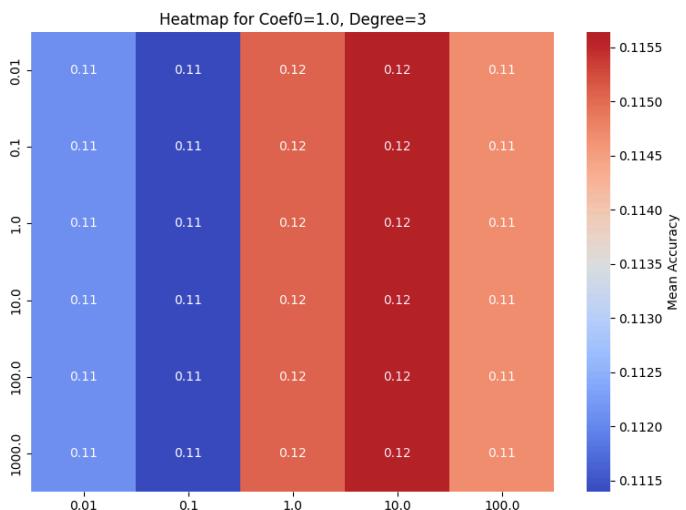
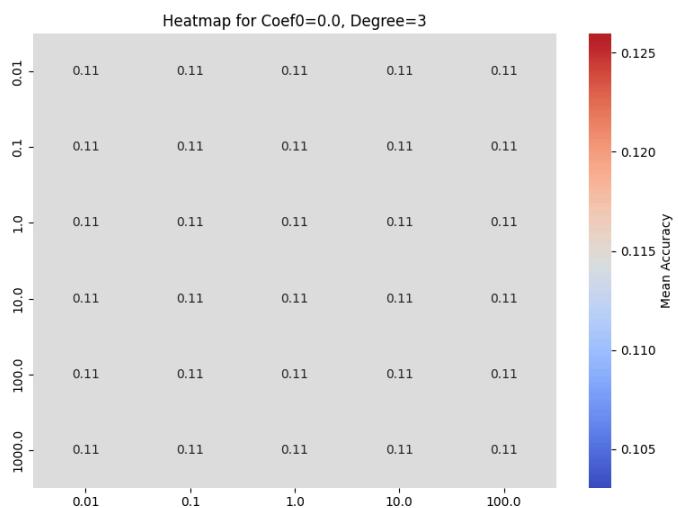
Έγιναν δοκιμές για διάφορες τιμές των παραμέτρων c και γ για συνδυασμούς βαθμού και σταθερού συντελεστή των πολυωνύμων. Στον κάθετο άξονα μετράται η παράμετρος c και στον οριζόντιο άξονα η παράμετρος γ . Συνολικός χρόνος εκπαίδευσης: 26 ώρες.

Για πολυώνυμα 2^{ov} βαθμού:

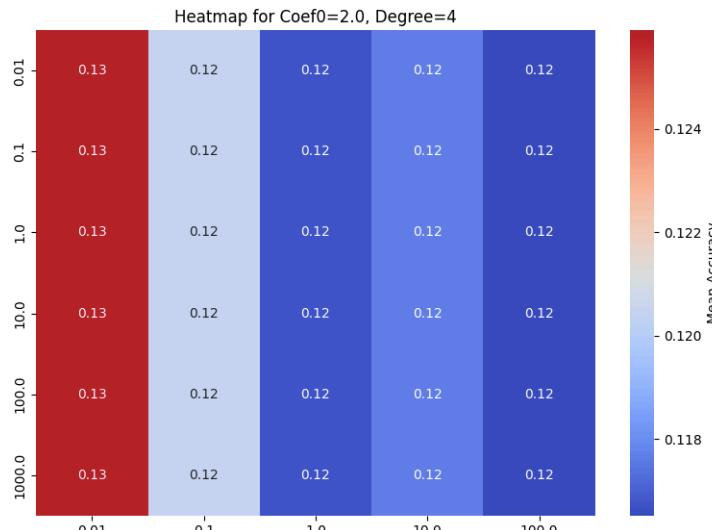
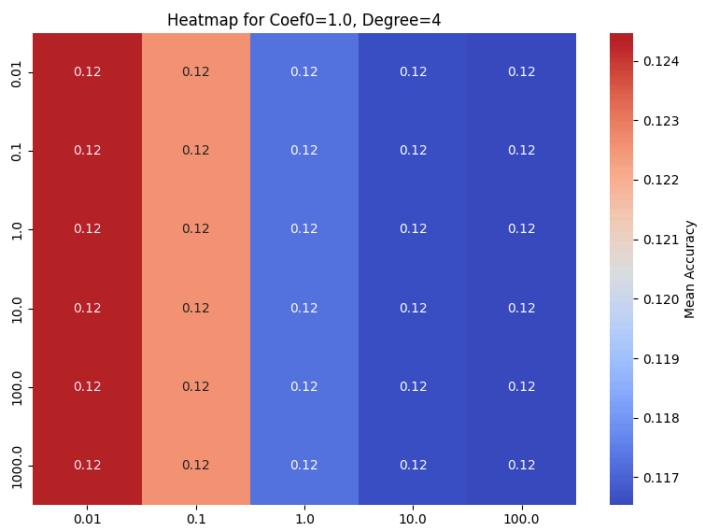
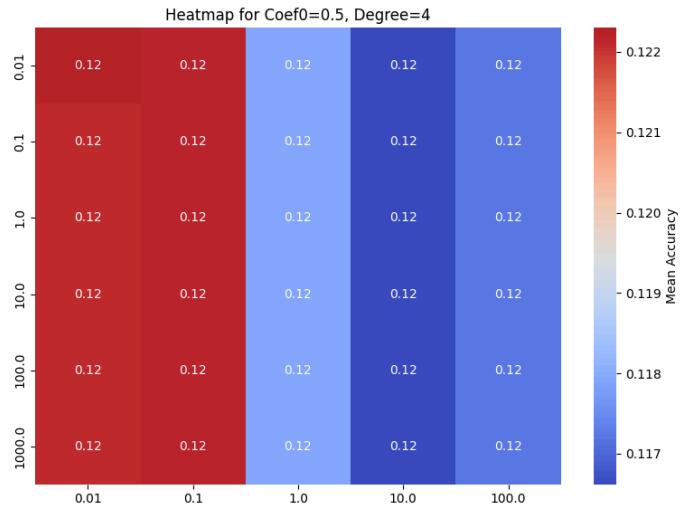
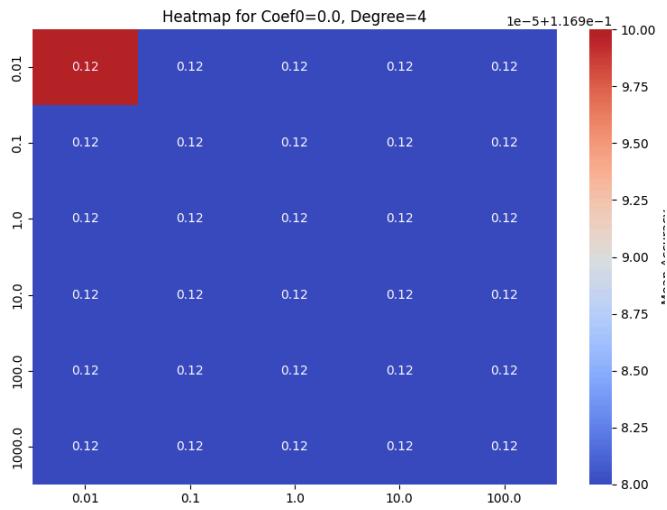




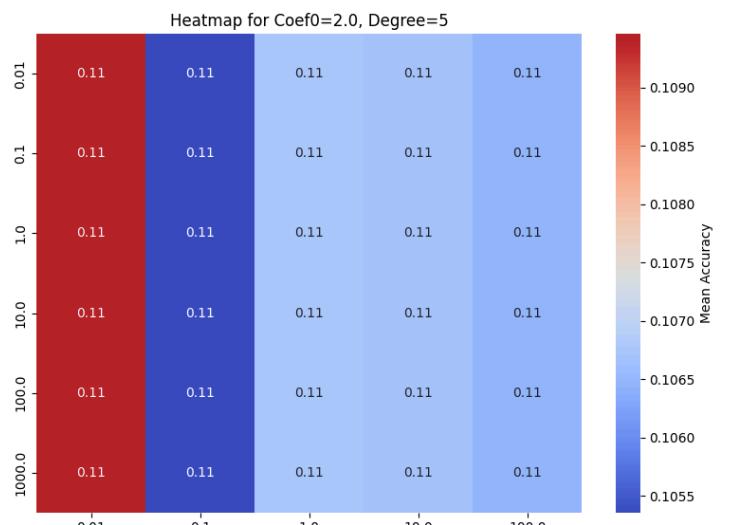
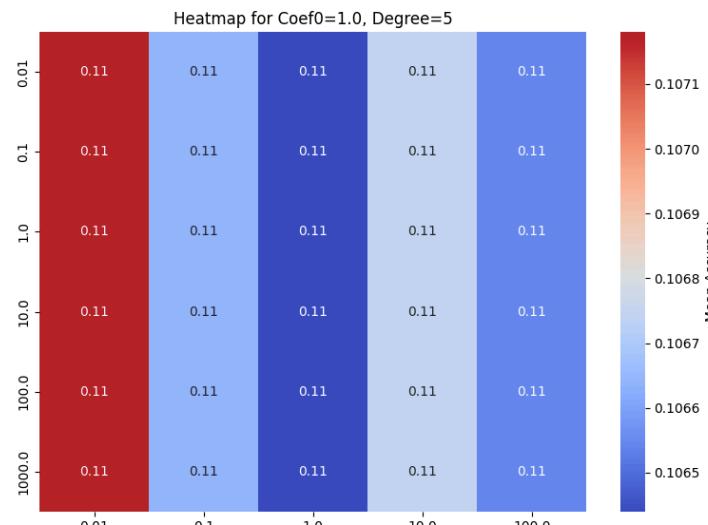
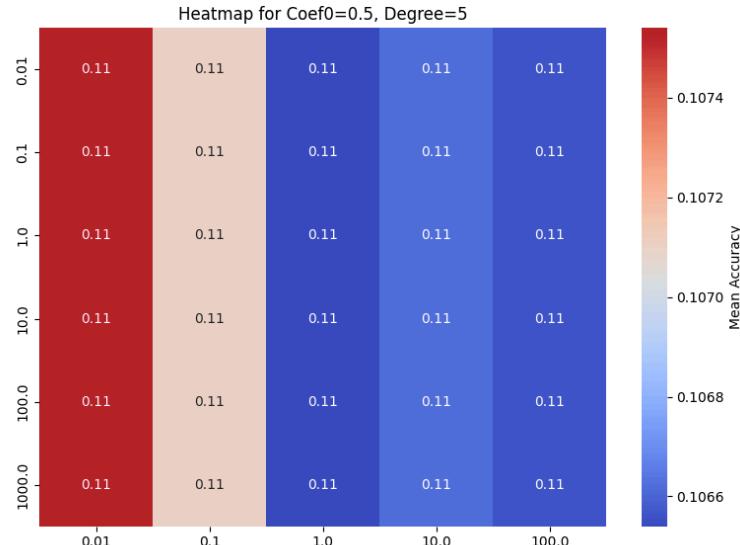
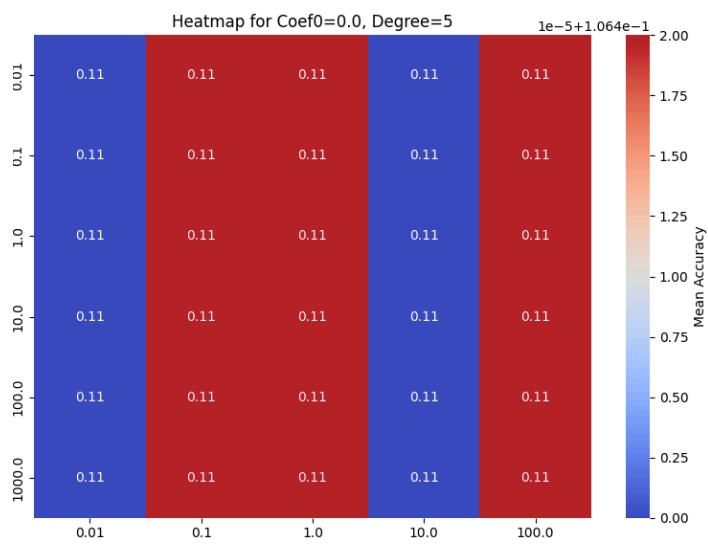
Για πολυώνυμα 3^{ον} βαθμού:



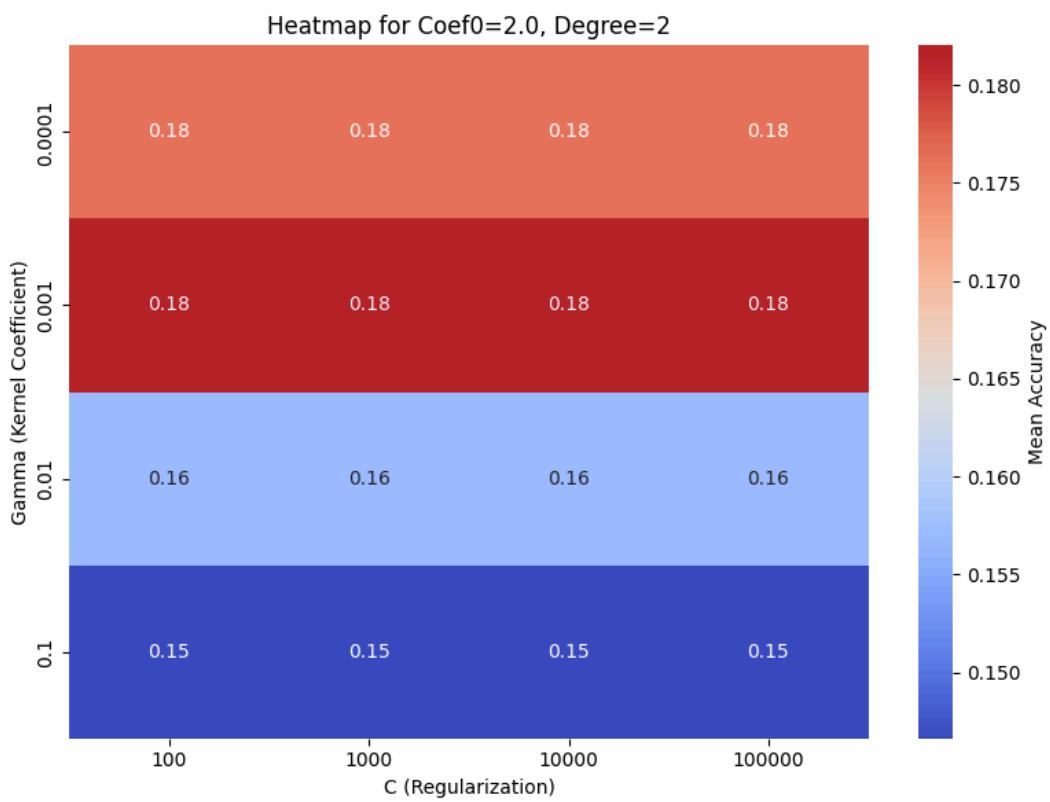
Για πολυωνύμια 4^{ov} βαθμού:



Για πολυωνύμια 5^{ov} βαθμού:

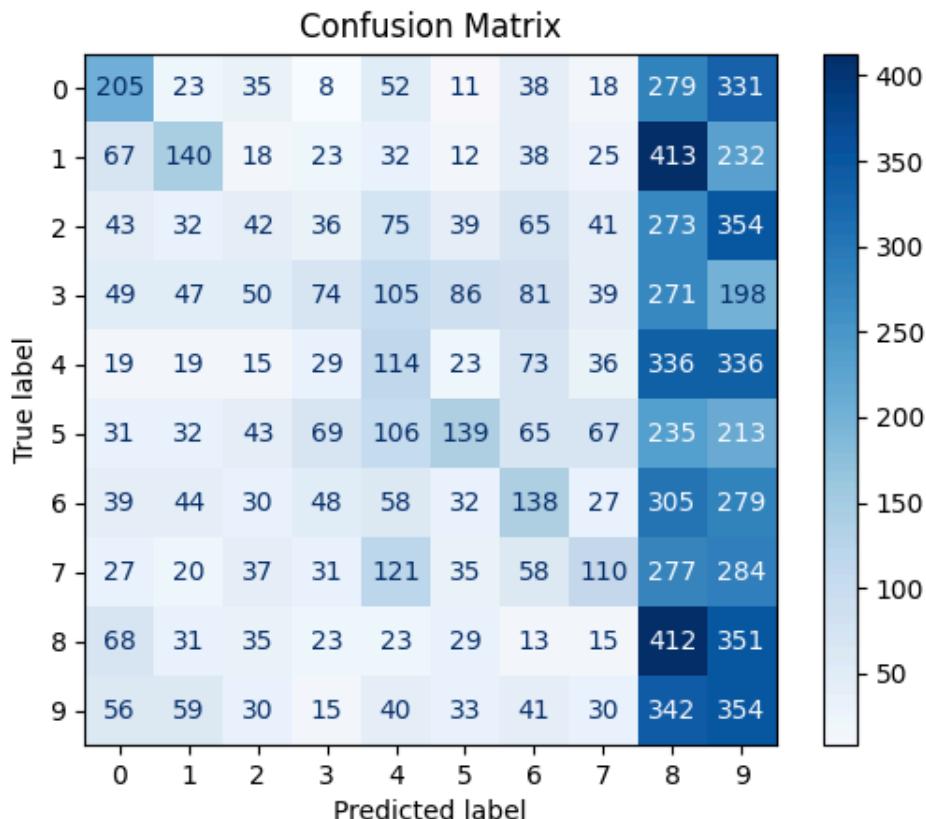


Το πολυωνυμικό kernel 2^{ov} βαθμού με σταθερά 2 είχε την μεγαλύτερη ευστοχία. Άρα γίνεται περαιτέρω διερεύνηση.



Heatmap της ευστοχίας της μηχανής για διάφορες τιμές των παραμέτρων γ και c
Συνολικός χρόνος: 69 λεπτά

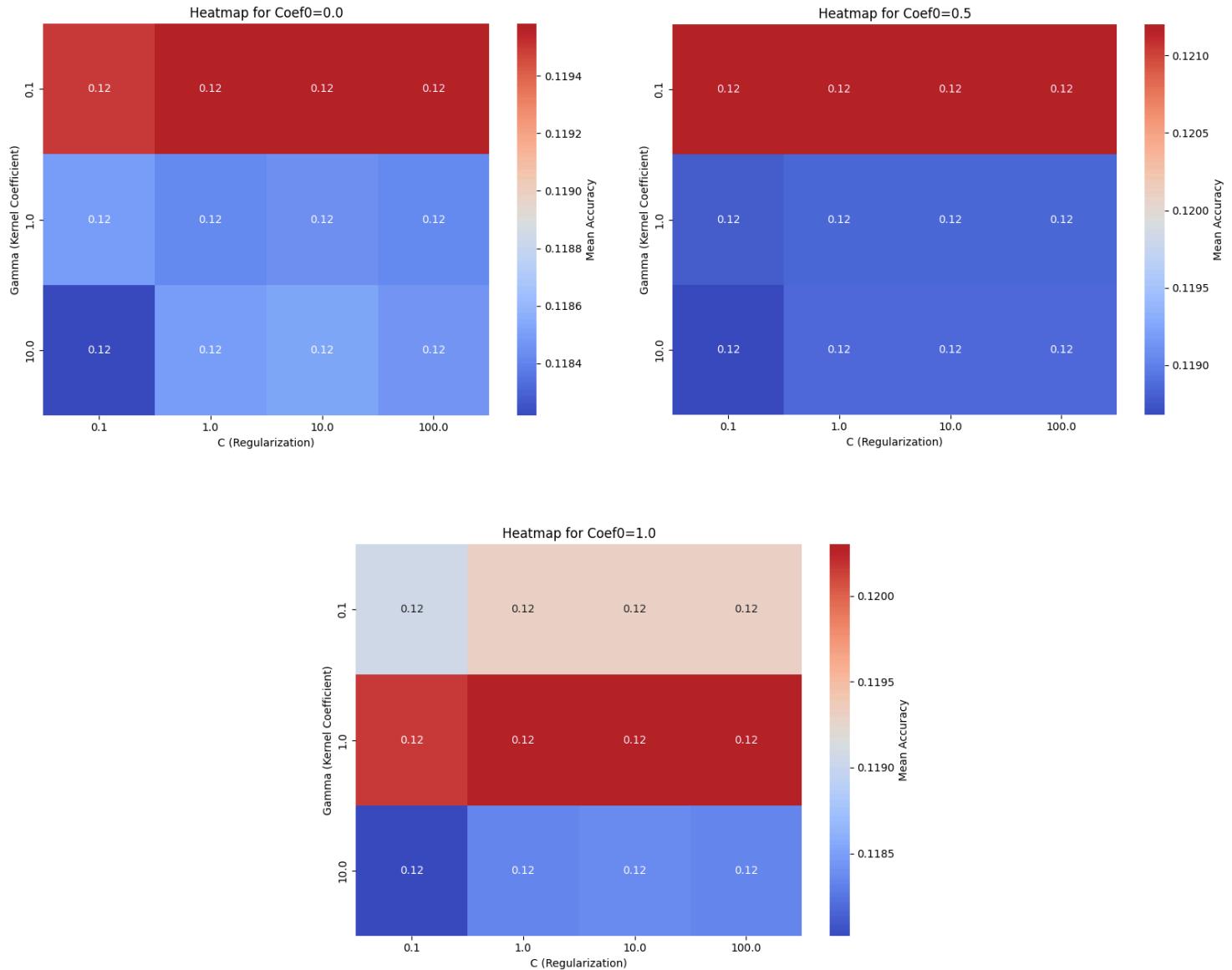
Είναι εμφανές ότι η ευστοχία μεγιστοποιείται για $c = 100$ and $\gamma = 0.001$. Ακολουθεί ο πίνακας σύγχυσης.



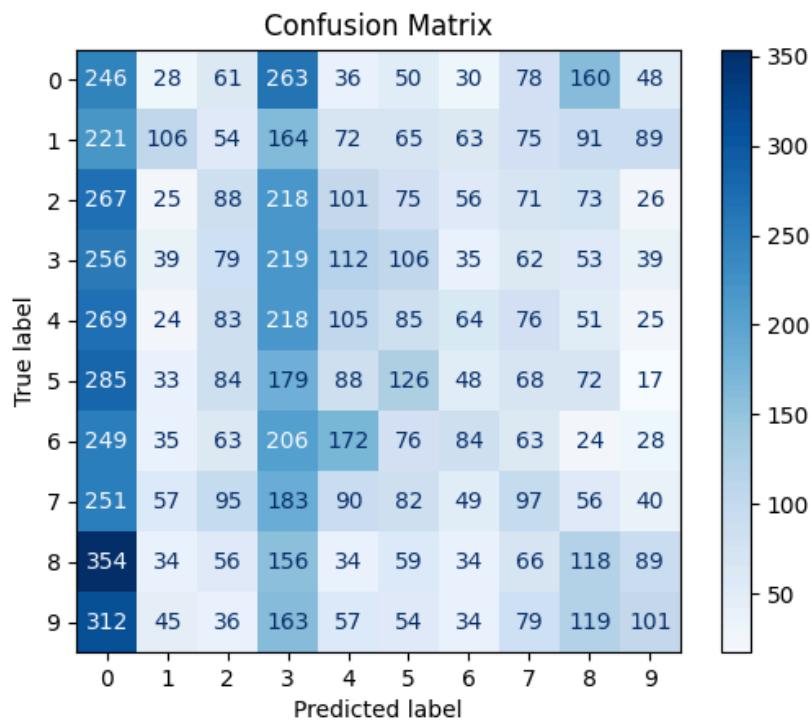
Confusion matrix for Polynomial kernel of 2nd degree with constant of 2, $c = 100$ and $\gamma = 0.001$

Sigmoid kernel:

Έγιναν δοκιμές για διάφορες τιμές των παραμέτρων c και γ συναρτήσει του σταθερού συντελεστή. Συνολικός χρόνος: 137 λεπτά

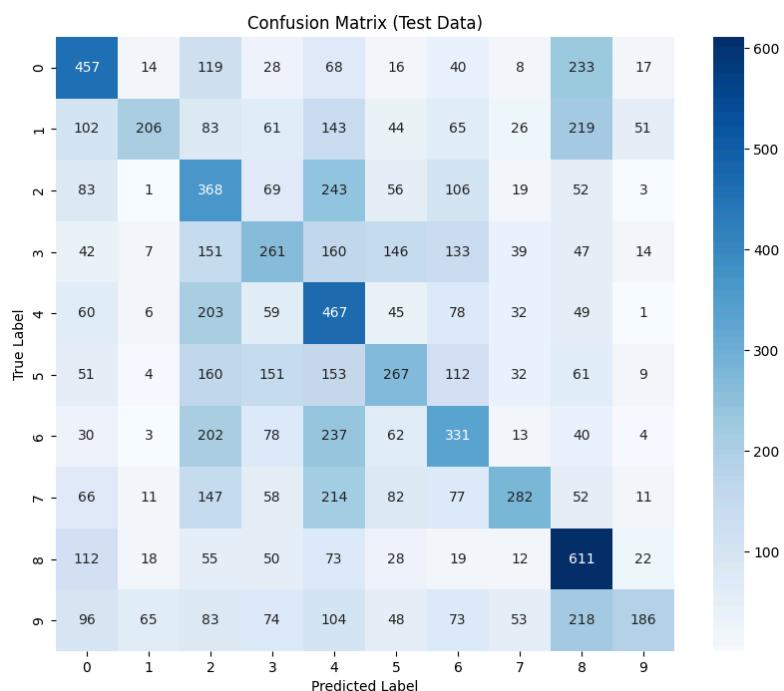


Ο καλύτερος συνδυασμός είναι για $c = 0.1$, $\gamma = 0.1$, $coef = 0.5$ με διάγραμμα σύγχυσης:



Confusion matrix for Sigmoid kernel of with constant of
 0.5 , $c = 0.1$ and $\gamma = 0.1$

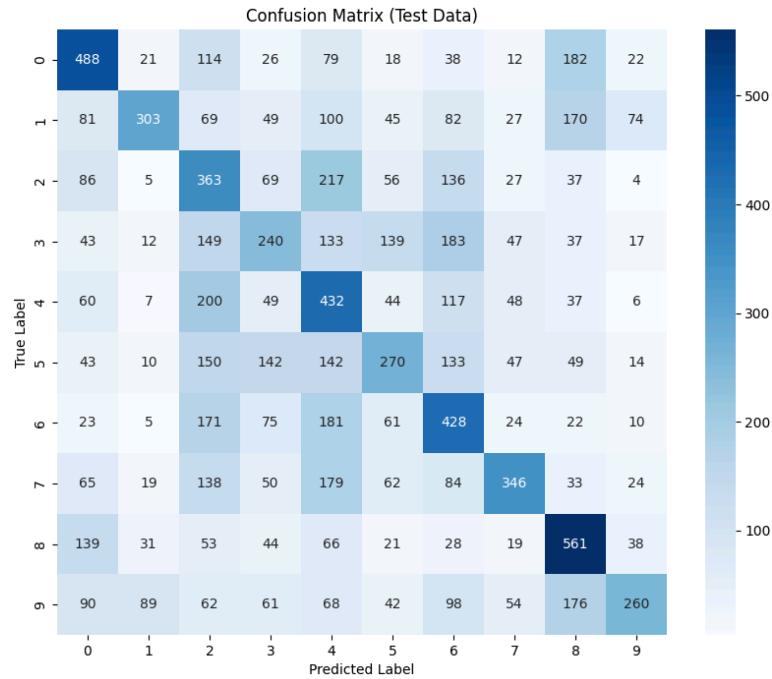
Σύμφωνα με τα παραπάνω πειράματα, η μέγιστη ευστοχία εντοπίστηκε για *RBF kernel* με παραμέτρους $c = 0.001$ και $\gamma = 0.1$. Άρα, εκπαιδεύεται περαιτέρω για 4000 επαναλήψεις και προκύπτει:



Confusion matrix for *rbf kernel* with $c = 0.001$ and
 $\gamma = 0.1$ without PCA

Ο συνολικός χρόνος εκπαίδευσης ήταν 74 λεπτά και επιτεύχθηκε ευστοχία 34.36 %. Γίνονται δοκιμές με μείωση διάστασης μέσω *PCA*.

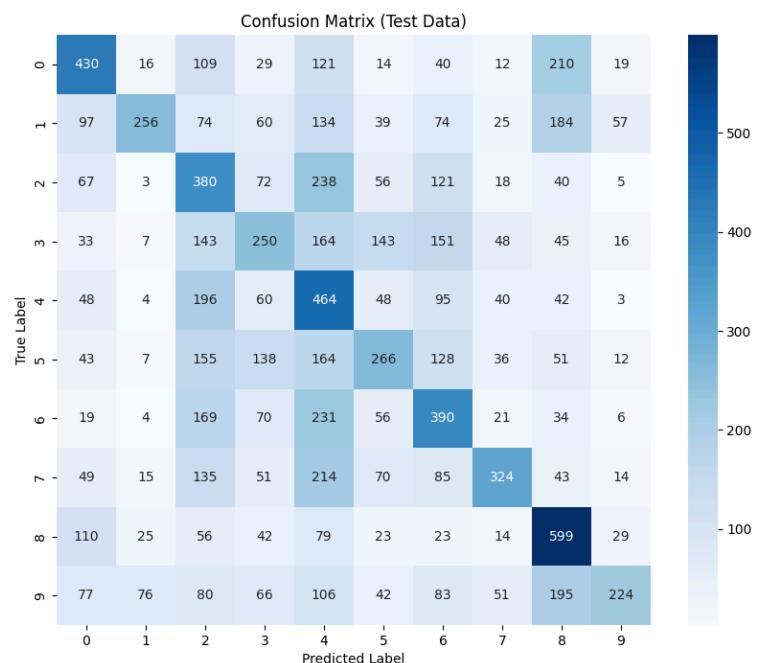
- Για διατήρηση 90 % της πληροφορίας:



Confusion matrix for *rbf kernel* with $c = 0.001$
and $\gamma = 0.1$ with *PCA*

με συνολικό χρόνο εκπαίδευσης 7 λεπτά και ευστοχία 36.91 % .

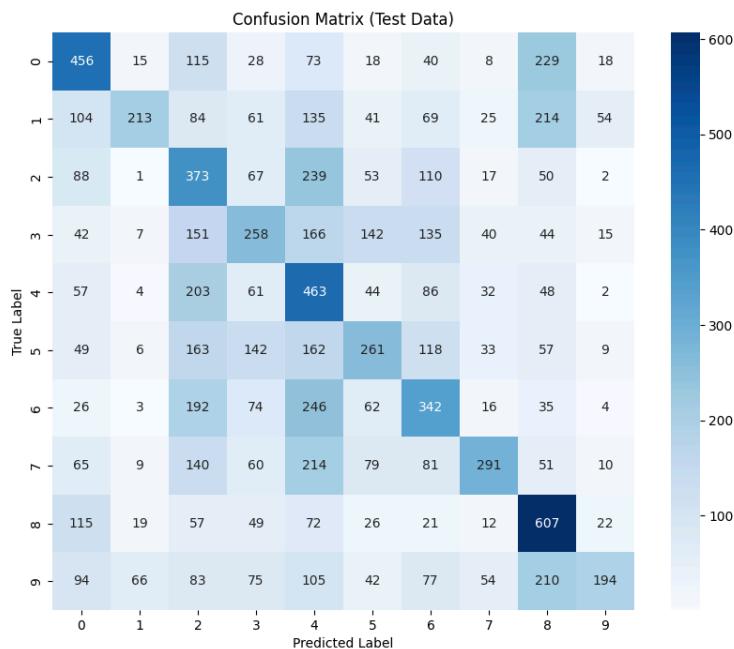
- Για διατήρηση 95 % της πληροφορίας:



Confusion matrix for *rbf kernel* with $c = 0.001$
and $\gamma = 0.1$ with *PCA*

με συνολικό χρόνο εκπαίδευσης 13 λεπτά και ευστοχία 35.85 % .

- Για διατήρηση 99 % της πληροφορίας:



Confusion matrix for *rbf kernel* with $c = 0.001$
and $\gamma = 0.1$ with *PCA*

με συνολικό χρόνο εκπαίδευσης 27 λεπτά και ευστοχία 34.58 % .

Τελικά, για *rbf kernel* με $c = 0.001$, $\gamma = 0.1$ και 90 % διατήρηση της πληροφορίας μέσω *PCA* επιτυγχάνεται η μέγιστη ευστοχία 36.91 % .

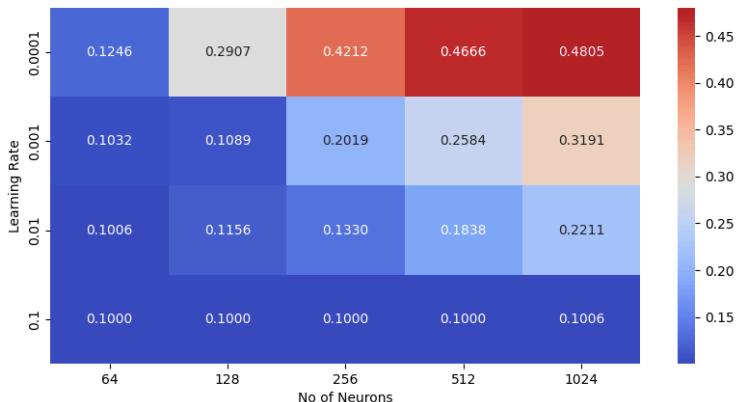
Σύγκριση με άλλους κατηγοριοποιητές

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

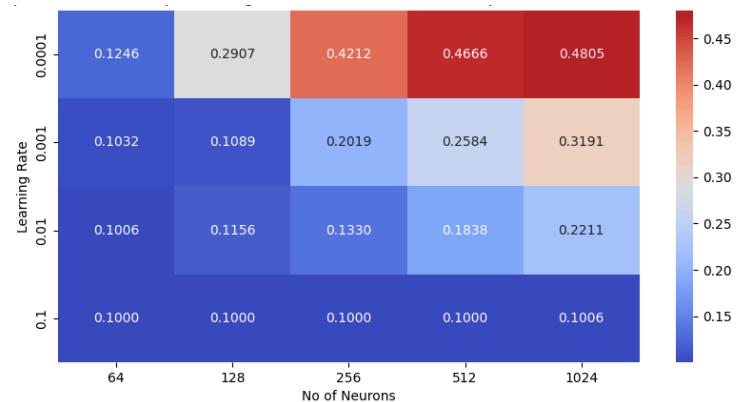
Accuracy (%)	No PCA	PCA (90 % retention)
KNN (1 Neighbor)	35	39
KNN (3 Neighbors)	33	37
NCC	28	28

Για το *MLP* με ένα κρυφό επίπεδο και *Hinge loss* για την βελτιστοποίηση έγιναν διάφορες δοκιμές. Δοκιμάστηκαν διαφορετικά πλήθη νευρώνων στο κρυφό επίπεδο (64, 128, 256, 512, 1024), διαφορετικοί ρυθμοί εκμάθησης (0.1, 0.01, 0.001, 0.0001) και batch sizes (32, 64, 128, 256) με συνολικό χρόνο εκπαίδευσης 6 ώρες.

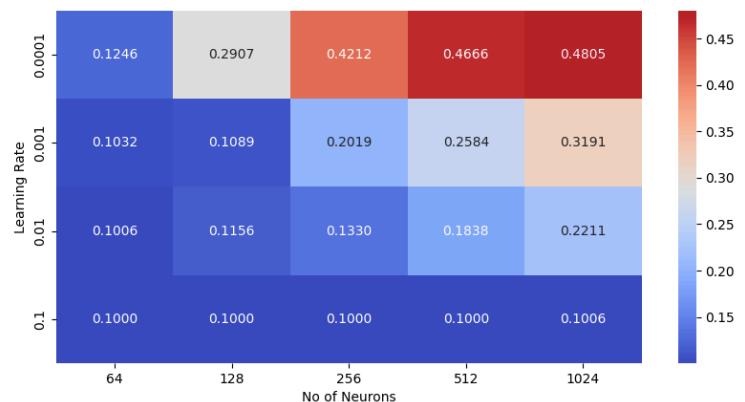
Hyperparameter Heatmap for *Adadelta* optimizer & batch size 32



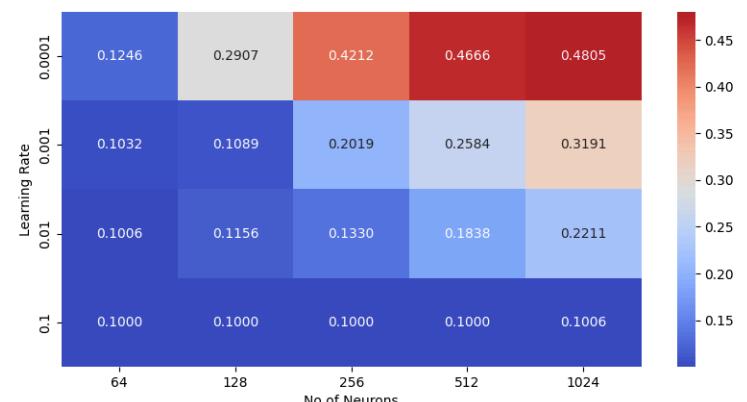
Hyperparameter Heatmap for *Adadelta* optimizer & batch size 64



Hyperparameter Heatmap for *Adadelta* optimizer & batch size 128



Hyperparameter Heatmap for *Adadelta* optimizer & batch size 256

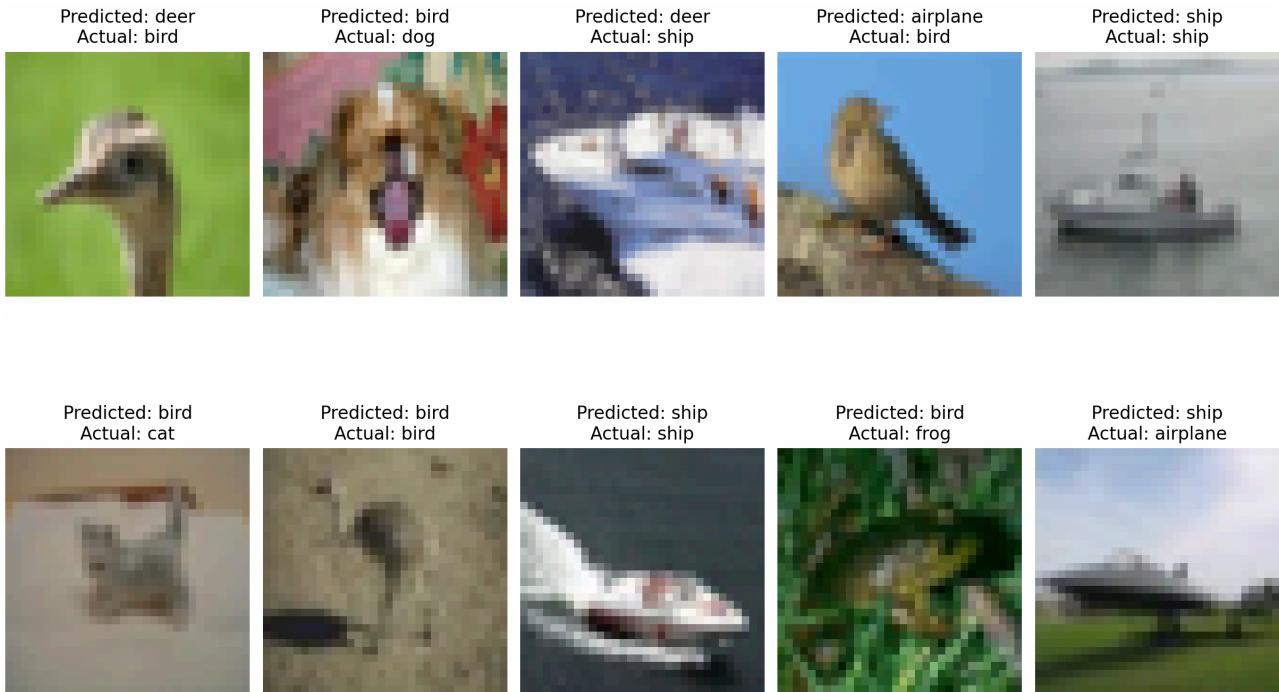


Φαίνεται ότι η ευστοχία ελέγχου υπερβαίνει το 48 % , ποσοστό το οποίο ξεπερνά το 36.91 % του *SVM*.

Αποτελέσματα:

Η υψηλότερη απόδοση του *MLP* οφείλεται στην ικανότητά του να μαθαίνει σύνθετα χαρακτηριστικά απευθείας από τα δεδομένα. Το kNN με $k=1$ υπερέχει του $k=3$, καθώς εκμεταλλεύεται τον πλησιέστερο γείτονα, αποφεύγοντας την επηρεασμότητα από θορυβώδη ή παραπλανητικά δεδομένα. Αντίθετα, το $k=3$ επηρεάζεται από γείτονες άλλων κατηγοριών, με αποτέλεσμα απόδοση παρόμοια με αυτή του *SVM*, το οποίο περιορίζεται από την επιλογή υπερπαραμέτρων και την πολυπλοκότητα του CIFAR-10. Ο NCC παρουσιάζει τη χαμηλότερη απόδοση, καθώς βασίζεται στη γραμμική προσέγγιση των χαρακτηριστικών, η οποία είναι ανεπαρκής για τη συγκεκριμένη βάση δεδομένων.

Έλεγχος SVM:



Παραδείγματα κατηγοριοποίησης *SVM*

Η λεπτομερής ανάλυση των αποτελεσμάτων κατηγοριοποίησης από το μοντέλο μας αποκαλύπτει σημαντικά διδάγματα για τις δυνατότητες και τα όρια του αλγορίθμου στο συγκεκριμένο σύνολο δεδομένων. Η εσφαλμένη κατηγοριοποίηση ενός πουλιού ως ελάφι, για παράδειγμα, μπορεί να έχει προκύψει λόγω των οπτικών ομοιοτήτων στο χρώμα και στην υφή του φόντου, που είναι χαρακτηριστικά φυσικών τοπίων όπως τα δάση. Τέτοιες ομοιότητες μπορεί να προκαλέσουν στο μοντέλο την παρανόηση ότι πρόκειται για μια αντίστοιχη οντότητα.

Ακόμη, η αναγνώριση ενός σκύλου ως πουλί ενδέχεται να προέρχεται από τη διαστρέβλωση του σχήματος του ζώου στην εικόνα, που μπορεί να έχει προκαλέσει αμφισημίες στην αντίληψη του μοντέλου. Τέτοιες ασάφειες στην εικονική αναπαράσταση μπορεί να οδηγήσουν το μοντέλο σε λανθασμένες ερμηνείες της εικόνας.

Επίσης, το πουλί που προβλέπεται λανθασμένα ως γάτα μπορεί να έχει συμβεί λόγω της θολής εικόνας ή ομοιοτήτων στο χρώμα και την υφή των φόντων που τα συνδέουν.

Αντιθέτως, το ελάφι που προβλέπεται σωστά ως ελάφι ενδεχομένως οφείλεται στην αναγνώριση τυπικών χαρακτηριστικών του ελαφιού, όπως το σχήμα του σώματος ή το χρώμα.

Κώδικας:

Στο αρχείο για την εκπαίδευση του *SVM* (*PanagiotisPapadopoulosErgasia2_10697.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 pixels 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.

Στο αρχείο για την εκπαίδευση του *MLP*
(*PanagiotisPapadopoulosErgasia2_10697_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.

Στο αρχείο με τους υπόλοιπους κατηγοριοποιητές (*PanagiotisPapadopoulosErgasia2_10697_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.

Στο αρχείο με τα παραδείγματα κατηγοριοποίησης (*PanagiotisPapadopoulosErgasia2_10697_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.