ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ:

ΥΛΟΠΟΙΗΣΗ ΓΡΑΦΙΚΗΣ ΔΙΕΠΑΦΗΣ ΓΙΑ ΤΗΝ ΔΗΜΙΟΥΡΓΙΑ ΚΑΝΟΝΩΝ ΤΑΞΙΝΟΜΗΣΗΣ

Ονοματεπώνυμο: Θεοντόρ Μέκο

Αριθμός Μητρώου: 1510

Επιβλέπων: Ιωάννης Τσούλος

Επίκουρος Καθηγητής

Άρτα, Οκτώβριος, 2024

GRAPHICAL USER INTERFACE IMPLEMENTATION FOR CREATING CLASSIFICATION RULES

ΠΕΡΙΕΧΟΜΕΝΑ

Δήλωση μη λογοκλοπής 6

ΕΥΧΑΡΙΣΤΙΕΣ..... 7

ΠΕΡΙΛΗΨΗ..... 8

ABSTRACT..... 9

1. Εισαγωγή 10

1.1 Πρότυπα.....10

1.2 Μάθηση με επίβλεψη 11

1.3 Κατηγοριοποίηση Δεδομένων με και χωρίς την Γραμματική Εξέλιξη 14

1.4 Μάθηση συναρτήσεων.....17

1.5 Εισαγωγή στα μοντέλα μηχανικής μάθησης 20

2. Γραμματική Εξέλιξη (Grammatical Evolution - GE) 21

2.1 Γενετικοί Αλγόριθμοι 22

2.2 Γενετικός Προγραμματισμός 24

2.3 Γραμματική εξέλιξη 25

2.4 Κατασκευή κανόνων GenClass 26

2.5 Πλεονεκτήματα και Εφαρμογές της Γραμματικής Εξέλιξης.....27

3. Υλοποίηση της Διεπαφής με τη Χρήση Τεχνολογιών: FastAPI, React και MongoDB 29

3.1 Γραφικές διεπαφές (εμπρός μέρος/frontend) με React/JAVASCRIPT 33

3.2 Εργαλεία υλοποίησης 37

3.3 Τα μέρη της εφαρμογής 41

3.3.1 Δομή πίσω μέρους (- backend) της εφαρμογής. 41

3.3.2 Εμπρός μέρος – Frontend 49

3.4 Πειραματικά αποτελέσματα 60

3.5 Πακετάρισμα εφαρμογής με χρήση Docker.....62

3.5.1 Τι είναι το Docker;.....63

3.5.2 Γιατί είναι χρήσιμο το Docker; 64

3.6 Εξήγησή και οδηγίες εκτέλεσης εφαρμογής με χρήση Docker εντολών 66

4. Παρατηρήσεις σχετικά με την εφαρμογή 71

4.1 Συμπεράσματα 72

4.2 Μελλοντικές επεκτάσεις 73

Τα εργαλεία και οι βιβλιοθήκες που χρησιμοποιήθηκαν για την υλοποίηση:.....73

Πηγές και Βιβλιογραφία.....74

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

Εικόνα 1: Endpoints του πίσω μέρους της εφαρμογής 31
Εικόνα 2: Αρχεία και πακέτα του πίσω μέρους 41
Εικόνα 3: Αρχείο genclassgenerations.py (1) 42
Εικόνα 4: Αρχείο genclassgenerations.py (2) 43
Εικόνα 5: Αρχείο main.py 44
Εικόνα 6: Αρχείο utils.py (1) 45
Εικόνα 7: Αρχείο utils.py (2) 46
Εικόνα 8: Αρχείο genclass.py 46
Εικόνα 9: Αρχείο requirements.txt 47
Εικόνα 10: Dashboard της εφαρμογής 47
Εικόνα 11: Επιλογές και παράμετροι του genclass 48
Εικόνα 12: Αρχείο ConfussionMatrix.js 49
Εικόνα 13: Αρχείο ErrorChart.js 50
Εικόνα 14: Αρχείο FileManagement.js 50
Εικόνα 15: Αρχείο GenClassForm.js (1) 51
Εικόνα 16: Αρχείο GenClassForm.js (2) 52
Εικόνα 17: Αρχείο GenclassHelp.js 53
Εικόνα 18: Αρχείο GenClassOutput.js 54
Εικόνα 19: Αρχείο useFiles.js 55
Εικόνα 20: Αρχείο useGenClass.js 56
Εικόνα 21: Αρχείο api.js 57
Εικόνα 22: Αρχείο theme.js 57
Εικόνα 23: Αρχείο parseGenClassOutput.js 58
Εικόνα 24: Αρχείο App.js (1) 59
Εικόνα 25: Αρχείο App.js (2) 59

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για την συγγραφή της εργασίας περιλαμβάνονται στη βιβλιογραφία.

Μέκο Θεοντόρ

Υπογραφή

ΕΥΧΑΡΙΣΤΙΕΣ

Θέλω να ευχαριστήσω τον επιβλέπων καθηγητή κ. Ιωάννη Τσούλο για την υποστήριξη που έλαβα καθ'όλη την διάρκεια εκπόνησης της πτυχιακής μου εργασίας.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία εστιάζει στην ανάπτυξη μιας γραφικής διεπαφής για τη δημιουργία και βελτιστοποίηση κανόνων ταξινόμησης μέσω της Γραμματικής Εξέλιξης (Grammatical Evolution - GE). Ο σκοπός της είναι να διευκολύνει την αλληλεπίδραση των χρηστών με τους κανόνες ταξινόμησης και να προσφέρει ένα εργαλείο που να επιτρέπει τη δημιουργία εξηγήσιμων μοντέλων, τα οποία είναι κατανοητά και διαφανή. Η χρήση τεχνολογιών όπως το FastAPI, το React και το MongoDB ενισχύει την αποδοτικότητα και ευελιξία της εφαρμογής. Επιπλέον, η εφαρμογή χρησιμοποιεί το Docker για να εξασφαλίσει συνέπεια και ευκολία στην ανάπτυξη σε διαφορετικά περιβάλλοντα.

ABSTRACT

This thesis focuses on the development of a graphical interface for the creation and optimization of classification rules using Grammatical Evolution (GE). The purpose is to facilitate user interaction with classification rules and provide a tool that allows the creation of explainable models that are understandable and transparent. By utilizing technologies such as FastAPI, React, and MongoDB, the application is designed to enhance efficiency and flexibility. In addition, the application utilizes Docker to ensure consistency and ease of deployment across different environments.

1. Εισαγωγή

Με την εκθετική γρήγορη εξάπλωση των δεδομένων, η ανάγκη για πιο αποδοτικά και εξηγήσιμα μοντέλα ταξινόμησης έχει αναδειχθεί ως κεντρικό ζήτημα στην έρευνα των υπολογιστικών συστημάτων. Οι επιστήμες της πληροφορικής στρέφονται όλο και περισσότερο σε μεθόδους που όχι μόνο θα διαχειρίζονται την πολυπλοκότητα των δεδομένων, αλλά προσφέρουν και σαφή ερμηνεία των αποτελεσμάτων τους. Σε αυτό το πλαίσιο, η γραμματική εξέλιξη (Grammatical Evaluation – GE) και ο γενετικός προγραμματισμός (Genetic Programming - GE) παρέχουν καινοτόμες λύσεις για την ανάπτυξη επεξηγηματικών κανόνων ταξινόμησης. Οι κανόνες αυτοί καθοδηγούνται από γραμματικές και εξελίσσονται δυναμικά μέσω εξελικτικών αλγορίθμων, προσφέροντας βελτιώσεις σε κάθε γενιά. Η παρούσα πτυχιακή εργασία εστιάζει στην ανάπτυξη μιας γραφικής διεπαφής που επιτρέπει τη δημιουργία και βελτιστοποίηση κανόνων ταξινόμησης με έμφαση στη διαφάνεια και την ευκολία κατανόησης από τους χρήστες. Η διεπαφή αυτή έχει σκοπό να διευκολύνει τόσο την αλληλεπίδραση όσο και την ερμηνεία των αποτελεσμάτων, καθιστώντας τα μοντέλα πιο κατανοητά.

1.1 Πρότυπα

Γραμματική Εξέλιξη (Grammatical Evolution)

Η **γραμματική εξέλιξη (Grammatical Evolution - GE)** αποτελεί μια βασική υπολογιστή μέθοδο που αξιοποιεί τους κανόνες τις γραμματικής για τη δημιουργία προγραμμάτων. Τα προγράμματα επιλύουν προβλήματα ταξινόμησης και ανάλυσης δεδομένων με τρόπο που διασφαλίζει τη συντακτική ορθότητα και την εξηγήσιμη λήψη των αποφάσεων. Η μέθοδος χρησιμοποιεί τη γραμματική αναπαράσταση BNF (Backus-Naur Form), η οποία καθορίζει την σύνταξη έγκυρου κώδικα μέσω των προκαθορισμένων κανόνων και συμβολισμών. Αυτή η αναπαράσταση εγγυάται ότι τα προγράμματα που παράγονται είναι συντακτικά ορθά και μπορούν να εφαρμοστούν σε διάφορα προγραμματιστικά περιβάλλοντα.

Η BNF (Backus-Naur Form) προσφέρει μεγάλη ευελιξία, καθώς μπορεί να προσαρμοστεί σε διαφορετικές γλώσσες προγραμματισμού και σε ποικίλα προβλήματα. Αυτή η ευχέρεια την καθιστά ιδανική για την ανάπτυξη λύσεων που διατηρούν συντακτική ορθότητα ανεξάρτητα από το περιβάλλον της εφαρμογής. Ένα από τα κύρια πλεονεκτήματα της BNF είναι η ικανότητά της να διασφαλίζει τη συντακτική ορθότητα των προγραμμάτων, κάτι που

παίζει ρόλο στην παραγωγή έγκυρων και αξιόπιστων αποτελεσμάτων. Η συντακτική ακρίβεια αυτή ενισχύει την αξιοπιστία των λύσεων που προκύπτουν. Ο συνδυασμός της BNF με την γραμματική εξέλιξη δημιουργεί κανόνες που είναι εύκολα κατανοητοί από τους ανθρώπους, ωθώντας την επεξηγησιμότητα των μοντέλων. Αυτή η διαφάνεια στη λειτουργία των μοντέλων καθιστά τις αποφάσεις πιο κατανοητές και αξιόπιστες. Η γραμματική εξέλιξη ξεχωρίζει για την ικανότητά της να παράγει εξηγήσιμα μοντέλα αποφάσεων. Αυτή η διαφάνεια είναι ιδιαίτερης σημασίας σε τομείς όπως η βιολογία και η χρηματοοικονομική ανάλυση, όπου η κατανόηση των μοντέλων παίζει καθοριστικό ρόλο.

Γενετικοί Αλγόριθμοι (Genetic Algorithms - GAs)

Οι **γενετικοί αλγόριθμοι** αποτελούν αναπόσπαστο κομμάτι της γραμματικής εξέλιξης και της διαδικασίας ταξινόμησης δεδομένων, καθώς χρησιμοποιούνται για τη βελτιστοποίηση των λύσεων μέσω εξελικτικών τεχνικών. Μέσω των διαδικασιών επιλογής, μετάλλαξης και διασταύρωσης, οι γενετικοί αλγόριθμοι εντοπίζουν τις βέλτιστες λύσεις και τις βελτιώνουν διαρκώς με την πάροδο του χρόνου, οδηγώντας σε συνεχώς αναπτυσσόμενες και αποδοτικές λύσεις. Ο συνδυασμός των γενετικών αλγορίθμων με τη γραμματική εξέλιξη παράγει πιο αποτελεσματικές λύσεις σε προβλήματα ταξινόμησης, ιδιαίτερα όταν οι παραδοσιακές μέθοδοι δεν επαρκούν για την επίλυση σύνθετων ζητημάτων.

1.2 Μάθηση με επίβλεψη

Η μάθηση με επίβλεψη αποτελεί μια βασική μέθοδο στη μηχανική μάθηση, όπου το μοντέλο εκπαιδεύεται με σύνολα δεδομένων που περιλαμβάνουν τόσο τις εισόδους όσο και τις αντίστοιχες εξόδους. Ο στόχος είναι το μοντέλο να μάθει να συσχετίζει τις εισόδους με τις σωστές εξόδους. Ο πρωταρχικός στόχος αυτής της μεθόδου είναι το μοντέλο να μάθει να συσχετίζει τις εισόδους με τις αντίστοιχες εξόδους, ώστε να μπορεί στη συνέχεια να προβλέπει σωστά τις εξόδους για νέες άγνωστες εισόδους. Αυτή η διαδικασία μπορεί να εφαρμοστεί σε πληθώρα προβλημάτων, όπως η ταξινόμηση, η παλινδρόμηση και η αναγνώριση μοτίβων.

Κύρια συστατικά της μάθησης με επίβλεψη:

Δεδομένα Εκπαίδευσης (Training Data): Το σύνολο των δεδομένων εκπαίδευσης αποτελείται από ζεύγη εισόδου-εξόδου, γνωστά και ως ετικέτες (**labels)**. Αυτά τα δεδομένα είναι καθοριστικής σημασίας για τη διαδικασία μάθησης του μοντέλου.

- **Παράδειγμα**: Ένα σύνολο δεδομένων για ένα πρόβλημα ταξινόμησης εικόνων μπορεί να περιλαμβάνει εικόνες ως εισόδους και τις αντίστοιχες κατηγορίες (όπως "γάτα" ή "σκύλος") ως εξόδους.

Ετικέτες (Labels): Οι ετικέτες αναπαριστούν τις επιθυμητές εξόδους που συνοδεύουν κάθε είσοδο στα δεδομένα εκπαίδευσης. Καθορίζουν την κατηγορία ή την τιμή που το μοντέλο καλείται να προβλέψει με ακρίβεια.

- **Παράδειγμα**: Στο πρόβλημα ταξινόμησης εικόνων, μια ετικέτα μπορεί να υποδεικνύει αν μια εικόνα περιέχει μια γάτα ή έναν σκύλο.

Αλγόριθμοι Μάθησης (Learning Algorithms): Οι αλγόριθμοι αυτοί είναι οι μηχανισμοί που επιτρέπουν στο μοντέλο να προβλέπει τις εξόδους, χρησιμοποιώντας τα δεδομένα εκπαίδευσης για συσχετίσει τις εισόδους με τις αντίστοιχες εξόδους.

Κάποιοι δημοφιλείς αλγόριθμοι:

Νευρωνικά Δίκτυα (Neural Networks): Χρησιμοποιούνται κυρίως σε προβλήματα αναγνώρισης εικόνων και επεξεργασίας φυσικής γλώσσας, όπου η πολυπλοκότητα των δεδομένων απαιτεί σύνθετα μοντέλα για ακριβείς προβλέψεις.

Υποστηρικτικοί Διανυσματικοί Μηχανισμοί (Support Vector Machines - SVMs): Χρησιμοποιούνται κυρίως για ταξινόμηση και παλινδρόμηση, βοηθώντας στον εντοπισμό του βέλτιστου διαχωριστικού υπερεπίπεδου μεταξύ διαφορετικών κατηγοριών δεδομένων.

Δέντρα Αποφάσεων (Decision Trees): Χρησιμοποιούνται για προβλήματα ταξινόμησης και παλινδρόμησης, βασίζομενα σε κανόνες "αν-τότε". Είναι εύκολα κατανοητά και ιδιαίτερα χρήσιμα για την ερμηνεία των προβλέψεων.

Γραμμική Παλινδρόμηση (Linear Regression): Χρησιμοποιείται για την πρόβλεψη συνεχών τιμών, βασιζόμενη σε γραμμικές σχέσεις ανάμεσα στα χαρακτηριστικά των εισόδων.

Συνάρτηση Κόστους (Cost Function): Η συνάρτηση κόστους μετρά την απόκλιση ανάμεσα στις προβλέψεις του μοντέλου και τις πραγματικές τιμές. Ο στόχος είναι να ελαχιστοποιηθεί αυτή η απόκλιση επιτρέποντας στο μοντέλο να βελτιώνει την ακρίβειά του.

- **Η μέση τετραγωνική απόκλιση (Mean Squared Error - MSE)** είναι μια κοινή συνάρτηση κόστους για προβλήματα παλινδρόμησης. Μετρά την τετραγωνική

διαφορά ανάμεσα στις προβλεπόμενες και τις πραγματικές τιμές, δίνοντας μεγαλύτερη έμφαση στις μεγαλύτερες αποκλίσεις.

Βελτιστοποίηση (Optimization): Η διαδικασία κατά την οποία το μοντέλο βελτιώνεται μέσω της ελαχιστοποίησης της συνάρτησης κόστους. Ο στόχος είναι να φτάσει σε ένα σημείο όπου οι προβλέψεις του πλησιάζουν όσο το δυνατόν περισσότερο τις πραγματικές τιμές.

- **Συνήθης μέθοδος βελτιστοποίησης**: Η πιο κοινή μέθοδος είναι η **Στοχαστική Βαθμίδα Κατάβασης (Stochastic Gradient Descent - SGD)**, η οποία προσαρμόζει τις παραμέτρους του μοντέλου με βάση τον υπολογισμό του κόστους για μικρά υποσύνολα δεδομένων.

Αξιολόγηση (Evaluation): Το τελικό στάδιο της μάθησης με επίβλεψη, όπου το μοντέλο δοκιμάζεται σε ένα σύνολο δεδομένων που δεν χρησιμοποιήθηκε κατά την εκπαίδευση (test data). Με αυτόν τον τρόπο, αξιολογείται η ικανότητά του να γενικεύει σε νέα δεδομένα.

- **Συνολική ορθότητα (Accuracy)**: Η αναλογία των σωστών προβλέψεων του μοντέλου σε σχέση με το σύνολο των προβλέψεων που έκανε.
- **Ακρίβεια (Precision)**: Το ποσοστό των αληθών θετικών προβλέψεων σε σχέση με το σύνολο των προβλέψεων που έγιναν θετικές.
- **Ανάκληση (Recall)**: Το ποσοστό των πραγματικών θετικών παραδειγμάτων που εντοπίστηκαν σωστά από το μοντέλο.
- **Μέση Τετραγωνική Απόκλιση (MSE)**: Χρησιμοποιείται ευρέως στα προβλήματα παλινδρόμησης, μετρώντας τη μέση απόκλιση των προβλέψεων του μοντέλου από τις πραγματικές τιμές, υψωμένη στο τετράγωνο.

Παραδείγματα Αλγορίθμων Μάθησης με Επίβλεψη:

Γραμμική Παλινδρόμηση (Linear Regression): Χρησιμοποιείται για πρόβλεψη συνεχών τιμών, όπως η τιμή ενός σπιτιού, λαμβάνοντας υπόψη χαρακτηριστικά όπως το μέγεθος και η τοποθεσία.

Δέντρα Αποφάσεων (Decision Trees): Χρησιμοποιούνται τόσο για την επίλυση προβλημάτων ταξινόμησης όσο και παλινδρόμησης. Ένα παράδειγμα εφαρμογής είναι η ταξινόμηση ενός email ως "spam" ή "not spam."

Νευρωνικά Δίκτυα (Neural Networks): Εφαρμόζονται σε πολύπλοκα προβλήματα, όπως η αναγνώριση εικόνων και η επεξεργασία φυσικής γλώσσας. Ένα παράδειγμα είναι η αναγνώριση χειρόγραφων ψηφίων.

Υποστηρικτικοί Διανυσματικοί Μηχανισμοί (Support Vector Machines - SVM): Χρησιμοποιούνται για ταξινόμηση και παλινδρόμηση, όπως η ταξινόμηση δεδομένων υγείας για τη διάγνωση ασθενειών.

1.3 Κατηγοριοποίηση Δεδομένων με και χωρίς την Γραμματική Εξέλιξη

Η κατηγοριοποίηση των δεδομένων είναι ένας από τους κύριους στόχους της μηχανικής μάθησης, είτε γίνεται μέσω της **Γραμματικής Εξέλιξης (Grammatical Evolution - GE)** είτε μέσω άλλων παραδοσιακών μεθόδων. Σκοπός της κατηγοριοποίησης είναι η δημιουργία μοντέλων που προβλέπουν κατηγορίες ή ετικέτες δεδομένων (**labels**) με βάση τα χαρακτηριστικά των εισόδων. Ωστόσο, ο τρόπος υλοποίησης της κατηγοριοποίησης διαφέρει ανάλογα με την επιλεγμένη προσέγγιση.

Κατηγοριοποίηση Δεδομένων με Γραμματική Εξέλιξη (GE)

Η **Γραμματική Εξέλιξη (GE)** είναι μια μέθοδος που στηρίζεται σε γενετικούς αλγόριθμους και χρησιμοποιεί τη γραμματική **BNF (Backus-Naur Form)** για την παραγωγή κανόνων κατηγοριοποίησης. Αυτή η προσέγγιση είναι ιδιαίτερα χρήσιμη σε περιπτώσεις που απαιτούν την κατασκευή εξηγήσιμων μοντέλων, τα οποία όχι μόνο ταξινομούν δεδομένα, αλλά παρέχουν και μια σαφή και κατανοητή δομή για τους κανόνες ταξινόμησης. Η κατηγοριοποίηση με τη χρήση της GE περιλαμβάνει τα εξής βασικά στοιχεία:

Χρήση Γραμματικής: Η Γραμματική Εξέλιξη βασίζεται σε μια προκαθορισμένη γραμματική, συνήθως σε μορφή **BNF (Backus-Naur Form)**, η οποία ορίζει τους κανόνες και τη δομή των παραγόμενων ταξινομητικών μοντέλων. Αυτό επιτρέπει τη δημιουργία εξηγήσιμων μοντέλων, τα οποία μπορούν να αναλυθούν και να κατανοηθούν εύκολα από τους χρήστες.

Παραγωγή Κανόνων Ταξινόμησης: Οι ταξινομητικοί κανόνες προκύπτουν από μια εξελικτική διαδικασία, στην οποία τα χρωμοσώματα που αναπαριστούν λύσεις (προγράμματα) αξιολογούνται με βάση την καταλληλόλητά τους (fitness). Αυτά τα

χρωμοσώματα προσαρμόζονται και βελτιώνονται μέσω γενετικών τεχνικών, όπως η **διασταύρωση (crossover)** και η **μετάλλαξη (mutation)**.

Επεξηγηματικότητα των Μοντέλων: Οι κανόνες που παράγονται μέσω της **Γραμματικής Εξέλιξης** είναι γραμμένοι με τρόπο που μπορεί να κατανοηθεί εύκολα από τους χρήστες, καθιστώντας τα μοντέλα εξαιρετικά εξηγήσιμα και ιδανικά για εφαρμογές όπου η διαφάνεια είναι κρίσιμη, όπως στην υγεία ή τα οικονομικά.

Προσαρμοστικότητα και Βελτιστοποίηση: Μέσω της εξελικτικής διαδικασίας, τα μοντέλα που δημιουργούνται προσαρμόζονται εύκολα σε διαφορετικά σύνολα δεδομένων, καθιστώντας τα ευέλικτα και ικανά να βελτιστοποιηθούν για μεγαλύτερη ακρίβεια στην ταξινόμηση.

Παραδείγματα εφαρμογών της **Γραμματικής Εξέλιξης** στην κατηγοριοποίηση δεδομένων περιλαμβάνουν την ταξινόμηση βιολογικών δεδομένων, χρηματοοικονομικών δεικτών και χημικών ενώσεων, όπου η επεξηγηματικότητα των μοντέλων είναι ζωτικής σημασίας.

Κατηγοριοποίηση Δεδομένων Χωρίς Γραμματική Εξέλιξη

Η κατηγοριοποίηση δεδομένων χωρίς τη χρήση της **Γραμματικής Εξέλιξης** βασίζεται σε παραδοσιακές μεθόδους μηχανικής μάθησης, όπως τα **νευρωνικά δίκτυα**, οι **Υποστηρικτικοί Διανυσματικοί Μηχανισμοί (SVM)** και τα **Δέντρα Απόφασης (Decision Trees)**. Αντί να χρησιμοποιούν κάποια γραμματική για τη δημιουργία κανόνων, αυτές οι μέθοδοι βασίζονται σε αλγοριθμικές προσεγγίσεις που μαθαίνουν από τα δεδομένα. Ορισμένα από τα χαρακτηριστικά της κατηγοριοποίησης δεδομένων χωρίς γραμματική εξέλιξη περιλαμβάνουν:

Αυτόματη Εκμάθηση Μοτίβων: Στις παραδοσιακές μεθόδους, τα μοντέλα μαθαίνουν να αναγνωρίζουν μοτίβα στα δεδομένα μέσω αλγοριθμικής ανάλυσης και βελτιστοποίησης. Για παράδειγμα, τα **νευρωνικά δίκτυα** μαθαίνουν από τα δεδομένα εισόδου και δημιουργούν προβλέψεις χρησιμοποιώντας πολυεπίπεδες δομές 'νευρώνων', χωρίς την ανάγκη για επεξηγήσιμους κανόνες ταξινόμησης.

Υψηλή Απόδοση αλλά Περιορισμένη Επεξηγηματικότητα: Παρόλο που αυτές οι μέθοδοι είναι εξαιρετικά αποδοτικές σε προβλήματα κατηγοριοποίησης, όπως η αναγνώριση εικόνας ή η επεξεργασία φυσικής γλώσσας (**NLP**), η επεξηγηματικότητα των μοντέλων είναι

συνήθως περιορισμένη. Για παράδειγμα, ένα **νευρωνικό δίκτυο** μπορεί να παρέχει υψηλή ακρίβεια, αλλά δεν προσφέρει κατανοητούς κανόνες ή εξηγήσεις για τις αποφάσεις που έλαβε

Αλγοριθμική Απόδοση: Στις παραδοσιακές μεθόδους μηχανικής μάθησης, η κατηγοριοποίηση δεδομένων εξαρτάται σε μεγάλο βαθμό από την απόδοση του αλγορίθμου. Ορισμένοι αλγόριθμοι, όπως οι **Υποστηρικτικοί Διανυσματικοί Μηχανισμοί (SVM)** και τα **Δέντρα Απόφασης (Decision Trees)**, παρέχουν πιο κατανοητά μοντέλα σε σχέση με τα **νευρωνικά δίκτυα**, αλλά η κατανόηση αυτών των μοντέλων παραμένει πιο πολύπλοκη συγκριτικά με την επεξηγησιμότητα της γραμματικής εξέλιξης..

Μαζική Εκπαίδευση σε Μεγάλα Σύνολα Δεδομένων: Οι παραδοσιακές μέθοδοι μηχανικής μάθησης είναι καλύτερα προσαρμοσμένες για τη μαζική εκπαίδευση σε μεγάλα σύνολα δεδομένων. Τα **νευρωνικά δίκτυα**, και ιδιαίτερα τα **βαθιά νευρωνικά δίκτυα (deep learning)**, έχουν επιδείξει εξαιρετική απόδοση σε προβλήματα όπως η αναγνώριση εικόνων και η φωνητική αναγνώριση. Ωστόσο, δεν παρέχουν σαφή και κατανοητά πρότυπα λήψης αποφάσεων.

Στοιχείο	**Κατηγοριοποίηση με Γραμματική Εξέλιξη (GE)**	**Κατηγοριοποίηση Χωρίς Γραμματική Εξέλιξη (Παραδοσιακές Μέθοδοι)**
Επεξηγηματικότητα	Πολύ υψηλή - Παράγει κανόνες σε μορφή BNF	Χαμηλή - Τα μοντέλα δεν παρέχουν πάντα επεξηγήσεις για τις αποφάσεις
Ευελιξία	Υψηλή - Μπορεί να προσαρμοστεί σε πολλές γλώσσες και προβλήματα	Υψηλή - Παραδοσιακές μέθοδοι προσαρμόζονται σε πολλαπλά δεδομένα και προβλήματα
Ακρίβεια	Εξαρτάται από τα δεδομένα και τη γραμματική	Συνήθως υψηλότερη σε μεγάλα δεδομένα, ειδικά με βαθιά μάθηση
Βελτιστοποίηση και Απόδοση	Βασισμένη σε εξελικτικές διαδικασίες, προσαρμόζεται δυναμικά	Εξαρτάται από τον αλγόριθμο (π.χ. SVM, Decision Trees, Neural Networks)
Προσαρμοστικότητα	Εύκολη προσαρμογή σε διαφορετικά δεδομένα μέσω αλλαγής της γραμματικής	Απαιτεί προσαρμογή μοντέλου και επανεκπαίδευση

Σύγκριση Κατηγοριοποίησης με και χωρίς Γραμματική Εξέλιξη

1.4 Μάθηση συναρτήσεων

Η **μάθηση συναρτήσεων** αποτελεί θεμελιώδη τομέα της μηχανικής μάθησης, όπου το μοντέλο μαθαίνει να αναπαριστά τη σχέση μεταξύ εισόδων και εξόδων μέσω μιας συνάρτησης. Ο στόχος είναι η ανάπτυξη μοντέλων που, βασισμένα σε δεδομένα εκπαίδευσης, μπορούν να προβλέψουν την τιμή εξόδου για νέες, άγνωστες εισόδους. Αυτή

η διαδικασία είναι ιδιαίτερα χρήσιμη σε προβλήματα **παλινδρόμησης (regression)** και **ταξινόμησης (classification)**. Υπάρχουν διάφοροι τρόποι υλοποίησης της μάθησης συναρτήσεων, τόσο μέσω παραδοσιακών μεθόδων μηχανικής μάθησης όσο και μέσω εξελικτικών τεχνικών, όπως η **Γραμματική Εξέλιξη (GE)Παραδοσιακή Μάθηση Συναρτήσεων.** Η παραδοσιακή μάθηση συναρτήσεων χρησιμοποιεί συγκεκριμένους αλγορίθμους και μοντέλα που "μαθαίνουν" από τα δεδομένα, προσαρμόζοντας τις παραμέτρους τους για να προσεγγίσουν τη σωστή συνάρτηση. Τα πιο σημαντικά μοντέλα περιλαμβάνουν:

Γραμμική Παλινδρόμηση (Linear Regression): Η **Γραμμική Παλινδρόμηση** προσπαθεί να μάθει μια γραμμική σχέση μεταξύ των εισόδων και εξόδων. Η συνάρτηση έχει τη μορφή **y = wX + b**, όπου **w** είναι τα βάρη και **b** το σταθερό μέλος

Νευρωνικά Δίκτυα (Neural Networks): Τα **νευρωνικά δίκτυα** αποτελούν μια πιο πολύπλοκη προσέγγιση στη μάθηση συναρτήσεων. Μέσω της δομής τους, μαθαίνουν να προσαρμόζουν τα βάρη και τις συνδέσεις μεταξύ των νευρώνων, ώστε να αναπαριστούν τη συνάρτηση που συνδέει τις εισόδους με τις εξόδους.

Υποστηρικτικοί Διανυσματικοί Μηχανισμοί (Support Vector Machines - SVMs): Οι **Υποστηρικτικοί Διανυσματικοί Μηχανισμοί (SVMs)** μαθαίνουν μια συνάρτηση που διαχωρίζει τις κατηγορίες δεδομένων σε προβλήματα ταξινόμησης. Στην παλινδρόμηση, οι **SVMs** χρησιμοποιούνται για την εκμάθηση συναρτήσεων που προσεγγίζουν τις τιμές εξόδου βάσει των δεδομένων εισόδου.

Δέντρα Απόφασης (Decision Trees): Τα **Δέντρα Απόφασης (Decision Trees)** μαθαίνουν μια συνάρτηση με τη μορφή κανόνων "αν-τότε", όπου κάθε διακλάδωση αναπαριστά μια συνθήκη που οδηγεί σε μια συγκεκριμένη πρόβλεψη για την τιμή εξόδου.

Αυτές οι παραδοσιακές μέθοδοι έχουν χρησιμοποιηθεί ευρέως για τη μάθηση συναρτήσεων σε διάφορους τομείς, όπως η ανάλυση δεδομένων, η οικονομία και οι φυσικές επιστήμες.

Μάθηση Συναρτήσεων μέσω Γραμματικής Εξέλιξης (GE)

Η **Γραμματική Εξέλιξη (Grammatical Evolution - GE)** προσφέρει μια πιο ευέλικτη προσέγγιση στη μάθηση συναρτήσεων, συνδυάζοντας εξελικτικές τεχνικές με τη χρήση γραμματικών για την παραγωγή μοντέλων. Αντί να απαιτεί από τον χρήστη να κατασκευάσει

απευθείας μια μαθηματική συνάρτηση, η **GE** χρησιμοποιεί γενετικούς αλγόριθμους για την εξελικτική δημιουργία προγραμμάτων που αναπαριστούν τη συνάρτηση.

Η διαδικασία μάθησης συναρτήσεων μέσω της γραμματικής εξέλιξης περιλαμβάνει τα εξής στάδια:

Ορισμός Γραμματικής: Η γραμματική ορίζει τους κανόνες με βάση τους οποίους θα κατασκευαστεί η συνάρτηση. Μπορεί να περιλαμβάνει τύπους συναρτήσεων, πράξεις και κανόνες που καθορίζουν τη μαθηματική σχέση μεταξύ εισόδων και εξόδων.

Χρήση Γενετικών Αλγορίθμων: Η **Γραμματική Εξέλιξη** χρησιμοποιεί γενετικούς αλγόριθμους για την εξελικτική παραγωγή λύσεων. Αυτές οι λύσεις, με τη μορφή χρωμοσωμάτων, περιέχουν τους κανόνες που καθορίζουν τη συνάρτηση.

Αξιολόγηση Καταλληλόλητας (Fitness Evaluation): Κάθε συνάρτηση που παράγεται μέσω της **GE** αξιολογείται με βάση την απόδοσή της. Η **συνάρτηση καταλληλόλητας (fitness function)** μετρά την ακρίβεια των προβλέψεων της συνάρτησης για τα δεδομένα εισόδου.

Βελτιστοποίηση Συναρτήσεων: Η εξελικτική διαδικασία συνεχίζεται μέχρι να παραχθεί μια συνάρτηση που πληροί τα κριτήρια καταλληλόλητας. Μέσω γενετικών λειτουργιών, όπως η διασταύρωση και η μετάλλαξη, οι παραγόμενες λύσεις βελτιώνονται συνεχώς.

Πλεονεκτήματα της Μάθησης Συναρτήσεων μέσω Γραμματικής Εξέλιξης (ΓΕ):

Ευελιξία στη Μάθηση

Η **γραμματική εξέλιξη** μπορεί να προσαρμοστεί σε διαφορετικά σύνολα δεδομένων και προβλήματα, χωρίς να απαιτεί εκ των προτέρων γνώση της δομής της συνάρτησης.

Επεξηγηματικότητα

Οι συναρτήσεις που παράγονται από τη **ΓΕ** μπορούν να αναπαρασταθούν με βάση τους κανόνες της γραμματικής, καθιστώντας τις πιο επεξηγήσιμες και κατανοητές.

Βελτιστοποίηση Σύνθετων Συναρτήσεων

Η ΓΕ μπορεί να παράγει πολύπλοκες συναρτήσεις που είναι δύσκολο να παραχθούν με παραδοσιακές μεθόδους. Αυτή η ικανότητα την καθιστά ιδανική για την αντιμετώπιση προβλημάτων όπου η σχέση μεταξύ των δεδομένων δεν είναι γραμμική ή απλή

1.5 Εισαγωγή στα μοντέλα μηχανικής μάθησης

Τα **μοντέλα μηχανικής μάθησης** είναι αλγοριθμικές δομές που επιτρέπουν στους υπολογιστές να μαθαίνουν από δεδομένα και να κάνουν προβλέψεις ή να παίρνουν αποφάσεις χωρίς να προγραμματίζονται για συγκεκριμένες εργασίες. Υπάρχουν πολλοί τύποι μοντέλων μηχανικής μάθησης, καθένας από τους οποίους είναι κατάλληλος για διαφορετικά είδη προβλημάτων και δεδομένων.

Κατηγορίες Μοντέλων Μηχανικής Μάθησης

Εποπτευόμενη Μάθηση (Supervised Learning): Χρησιμοποιείται όταν τα δεδομένα εκπαίδευσης περιλαμβάνουν γνωστές εξόδους (ετικέτες). Το μοντέλο μαθαίνει να συσχετίζει τις εισόδους με τις αντίστοιχες εξόδους.

Παραδείγματα Μοντέλων:

- **Γραμμική Παλινδρόμηση (Linear Regression)**: Χρησιμοποιείται για την πρόβλεψη συνεχών τιμών.
- **Δέντρα Απόφασης (Decision Trees)**: Χρησιμοποιούν κανόνες για την πρόβλεψη των ετικετών.
- **Υποστήριξη Διανυσματικών Μηχανών (SVM)**: Βρίσκει το βέλτιστο διαχωριστικό υπερεπίπεδο για την ταξινόμηση.
- **Μη Εποπτευόμενη Μάθηση (Unsupervised Learning):** Χρησιμοποιείται όταν τα δεδομένα δεν έχουν γνωστές εξόδους. Το μοντέλο αναζητά δομές ή σχέσεις μέσα στα δεδομένα.

Παραδείγματα Μοντέλων:

- **Αλγόριθμος Ομαδοποίησης (Clustering)**: Ομαδοποιεί δεδομένα σε ομάδες.
- **Ανάλυση Κύριων Συνιστωσών (PCA)**: Μειώνει τις διαστάσεις των δεδομένων για την εύρεση μοτίβων..

Ενισχυτική Μάθηση (Reinforcement Learning): Επιτρέπει στο μοντέλο να λαμβάνει αποφάσεις μέσω αλληλεπίδρασης με το περιβάλλον, μαθαίνοντας από τις ανταμοιβές που λαμβάνει. **Παράδειγμα**: Χρησιμοποιείται σε ρομποτική και σε παιχνίδια όπως το σκάκι, όπου ο αλγόριθμος επιδιώκει να μεγιστοποιήσει τη μακροπρόθεσμη ανταμοιβή.

Μοντέλα Βασισμένα σε Σύνολα (Ensemble Models): Συνδυάζουν πολλά μοντέλα για τη βελτίωση των προβλέψεων.

Παραδείγματα Μοντέλων:

- **Random Forest**: Συνδυάζει πολλά **Δέντρα Απόφασης** για τη βελτίωση της απόδοσης.
- **Gradient Boosting**: Εκπαιδεύει μοντέλα διαδοχικά, επικεντρώνοντας στα σφάλματα των προηγούμενων μοντέλων.

Νευρωνικά Δίκτυα και Βαθιά Μάθηση (Neural Networks & Deep Learning): Τα **νευρωνικά δίκτυα** και τα **βαθιά νευρωνικά δίκτυα** χρησιμοποιούν στρώματα "νευρώνων" για να μάθουν σύνθετα μοτίβα από τα δεδομένα.

Παραδείγματα:

- **Απλά Νευρωνικά Δίκτυα (Feedforward Neural Networks)**: Χρησιμοποιούνται για βασικές προβλέψεις.
- **Συνεκτικά Νευρωνικά Δίκτυα (CNNs)**: Κατάλληλα για προβλήματα αναγνώρισης εικόνας.
- **Αναδρομικά Νευρωνικά Δίκτυα (RNNs)**: Χρησιμοποιούνται για την ανάλυση ακολουθιών, όπως στην επεξεργασία φυσικής γλώσσας.

2. Γραμματική Εξέλιξη (Grammatical Evolution - GE)

Η **Γραμματική Εξέλιξη (Grammatical Evolution - GE)** είναι μια εξελικτική τεχνική γενετικού προγραμματισμού που χρησιμοποιεί γενετικούς αλγόριθμους για την αυτόματη παραγωγή προγραμμάτων. Η κεντρική ιδέα της είναι η χρήση γραμματικών, συνήθως σε μορφή **Backus-Naur Form (BNF)**, για τη δημιουργία λύσεων σε προβλήματα προγραμματισμού. Αυτός ο συνδυασμός των κανόνων γραμματικής με εξελικτικούς αλγόριθμους καθιστά τη **GE** εξαιρετικά ευέλικτη και ισχυρή, επιτρέποντάς της να προσαρμόζεται σε πλήθος εφαρμογών, από τη βιολογία έως τα χρηματοοικονομικά.

Ορισμός Γραμματικής Εξέλιξης:

Η **Γραμματική Εξέλιξη** χρησιμοποιεί γραμματικές για να καθοδηγήσει την εξελικτική διαδικασία των γενετικών αλγορίθμων. Οι λύσεις αναπαρίστανται ως συμβολοσειρές (γενετικοί κώδικες), οι οποίες αποκωδικοποιούνται χρησιμοποιώντας μια γραμματική για την παραγωγή έγκυρων προγραμμάτων ή εκφράσεων.

Κύρια Συστατικά και Διαδικασία:

Γραμματική (Grammar): Η **γραμματική**, συνήθως σε μορφή **BNF**, ορίζει το σύνολο των κανόνων που χρησιμοποιούνται για την παραγωγή έγκυρων προγραμμάτων ή εκφράσεων. Παρέχει τη δομή που επιτρέπει την κατασκευή προγραμμάτων με σαφείς κανόνες, καθορίζοντας τι είναι έγκυρο ή όχι σε κάθε στάδιο της εξέλιξης.

Χρωμοσώματα: Τα **χρωμοσώματα** αναπαρίστανται ως ακολουθίες ακεραίων αριθμών, όπου κάθε αριθμός αντιστοιχεί σε έναν κανόνα της γραμματικής. Αυτά τα χρωμοσώματα λειτουργούν ως "συνταγές" για την παραγωγή προγραμμάτων..

Αρχικοποίηση Πληθυσμού: Ένας αρχικός πληθυσμός από τυχαία χρωμοσώματα δημιουργείται, όπου κάθε χρωμόσωμα αντιπροσωπεύει μια πιθανή λύση στο πρόβλημα.

Αποκωδικοποίηση Χρωμοσωμάτων: Κάθε χρωμόσωμα αποκωδικοποιείται με τη χρήση της γραμματικής για την παραγωγή ενός έγκυρου προγράμματος ή μιας έκφρασης, που στη συνέχεια αξιολογείται.

Αξιολόγηση Καταλληλόλητας (Fitness Evaluation): Κάθε πρόγραμμα ή έκφραση που παράγεται, αξιολογείται με μια **συνάρτηση καταλληλόλητας (fitness function)**, η οποία μετρά πόσο καλά εκπληρώνει το πρόγραμμα το επιθυμητό αποτέλεσμα.

Επιλογή, Διασταύρωση και Μετάλλαξη: Οι βασικοί γενετικοί τελεστές που εφαρμόζονται περιλαμβάνουν την επιλογή των καλύτερων λύσεων, τη **διασταύρωση** (συνδυασμός δύο γονέων για δημιουργία απογόνων) και τη **μετάλλαξη** (τυχαία τροποποίηση γονιδίων για εισαγωγή ποικιλίας).

Νέα Γενιά: Οι απόγονοι αντικαθιστούν τον παλιό πληθυσμό, και η διαδικασία επαναλαμβάνεται για πολλές γενιές με στόχο τη βελτίωση των λύσεων.

Κριτήριο Τερματισμού (Termination Criterion): Η διαδικασία συνεχίζεται έως ότου ικανοποιηθεί κάποιο **κριτήριο τερματισμού**, όπως η ολοκλήρωση ενός καθορισμένου αριθμού γενεών ή η επίτευξη της επιθυμητής τιμής καταλληλόλητας.

2.1 Γενετικοί Αλγόριθμοι

Οι **γενετικοί αλγόριθμοι (Genetic Algorithms - GAs)** αποτελούν μια μέθοδο βελτιστοποίησης που βασίζεται στις αρχές της φυσικής επιλογής και της γενετικής, όπως

διατυπώθηκαν από τον **Charles Darwin**. Οι **GAs** αναζητούν τη βέλτιστη λύση μέσα σε έναν μεγάλο χώρο πιθανών λύσεων και χρησιμοποιούνται σε προβλήματα όπου η αναζήτηση του βέλτιστου είναι περίπλοκη ή δεν μπορεί να επιτευχθεί με παραδοσιακές μεθόδους.

Κύρια Συστατικά και Διαδικασία:

Αναπαράσταση Χρωμοσωμάτων: Τα **χρωμοσώματα** αναπαρίστανται συνήθως ως συμβολοσειρές από δυαδικά ψηφία (**bits**), αλλά μπορούν να χρησιμοποιηθούν και άλλες μορφές, όπως πραγματικοί αριθμοί ή πιο σύνθετες δομές δεδομένων. Κάθε χρωμόσωμα αντιπροσωπεύει μια πιθανή λύση στο πρόβλημα.

Αρχικοποίηση Πληθυσμού: Δημιουργείται ένας αρχικός πληθυσμός από χρωμοσώματα, τα οποία μπορεί να είναι είτε τυχαία είτε να δημιουργηθούν με ευρετικές μεθόδους.

Αξιολόγηση Καταλληλόλητας (Fitness Evaluation): Κάθε χρωμόσωμα αξιολογείται με μια **συνάρτηση καταλληλόλητας**, η οποία μετρά την ποιότητα της λύσης που αντιπροσωπεύει. Αυτή η αξιολόγηση είναι κρίσιμη για τη διαδικασία επιλογής των καλύτερων χρωμοσωμάτων.

Επιλογή (Selection): Οι καλύτερες λύσεις επιλέγονται για αναπαραγωγή και δημιουργία απογόνων. Κοινές μέθοδοι επιλογής περιλαμβάνουν τη **ρουλέτα (roulette wheel selection)**, το **τουρνουά (tournament selection)** και την **επιλογή κατάταξης (rank selection)**.

Γενετικοί Τελεστές:

- **Διασταύρωση (Crossover)**: Συνδυάζει δύο γονείς για τη δημιουργία απογόνων. Παραδείγματα περιλαμβάνουν τη **μονοσημειακή (single-point crossover)** και τη **δισημειακή (two-point crossover)** διασταύρωση.
- **Μετάλλαξη (Mutation)**: Η **μετάλλαξη** τροποποιεί τυχαία τα γονίδια των χρωμοσωμάτων για να διατηρηθεί η γενετική ποικιλία και να αποτραπεί η πρόωρη σύγκλιση σε τοπικά βέλτιστα..
- **Νέα Γενιά (New Generation)**: Οι απόγονοι αντικαθιστούν τον παλιό πληθυσμό και η διαδικασία επαναλαμβάνεται, με στόχο τη συνεχή βελτίωση των λύσεων.
- **Κριτήριο Τερματισμού (Termination Criterion)**: Η διαδικασία συνεχίζεται μέχρι να ικανοποιηθεί κάποιο **κριτήριο τερματισμού**, όπως η ολοκλήρωση ενός αριθμού γενεών ή η επίτευξη μιας συγκεκριμένης τιμής καταλληλόλητας.

2.2 Γενετικός Προγραμματισμός

Ο **γενετικός προγραμματισμός (Genetic Programming - GP)** είναι μια εξελικτική μέθοδος υπολογιστικής νοημοσύνης και μηχανικής μάθησης που χρησιμοποιείται για την αυτόματη δημιουργία προγραμμάτων υπολογιστών με σκοπό την επίλυση συγκεκριμένων προβλημάτων. Εμπνέεται από τις αρχές της φυσικής επιλογής και της γενετικής, όπως διατυπώθηκαν από τον **Κάρολο Δαρβίνο**. Στον γενετικό προγραμματισμό, οι υποψήφιες λύσεις αναπαρίστανται ως προγράμματα που εξελίσσονται μέσω διαδικασιών όπως η **αναπαραγωγή**, η **διασταύρωση (crossover)**, η **μετάλλαξη (mutation)** και η **επιλογή (selection)**. Η διαδικασία συνεχίζεται μέχρι να βρεθεί μια ικανοποιητική λύση ή να εξαντληθεί ο προκαθορισμένος αριθμός γενεών.

Διαδικασία του Γενετικού Προγραμματισμού

Αρχικοποίηση Πληθυσμού: Δημιουργείται ένας αρχικός πληθυσμός τυχαίων προγραμμάτων, τα οποία συνήθως αναπαρίστανται ως δέντρα (**tree structures**) με λειτουργίες στους εσωτερικούς κόμβους και δεδομένα ή σταθερές στα φύλλα..

Αξιολόγηση Καταλληλόλητας (Fitness Evaluation): Κάθε πρόγραμμα αξιολογείται με μια **συνάρτηση καταλληλόλητας (fitness function)**, η οποία μετρά την απόδοσή του στην επίλυση του προβλήματος. Η συνάρτηση αυτή μπορεί να βασίζεται σε κριτήρια όπως η ακρίβεια, η ταχύτητα ή η κατανάλωση πόρων..

Επιλογή (Selection): Τα καλύτερα προγράμματα επιλέγονται με βάση την καταλληλόλητά τους για να συμμετάσχουν στην επόμενη γενιά. Κοινές μέθοδοι επιλογής περιλαμβάνουν την επιλογή τουρνουά (tournament selection) και την επιλογή με ρουλέτα (roulette wheel selection).

Γενετικές Λειτουργίες:

- **Αναπαραγωγή (Reproduction)**: Η **αναπαραγωγή (reproduction)** αφορά την αντιγραφή των καλύτερων προγραμμάτων στην επόμενη γενιά χωρίς αλλαγές.
- **Διασταύρωση (Crossover)**: Η **διασταύρωση (crossover)** συνδυάζει δύο προγράμματα-γονείς για τη δημιουργία νέων προγραμμάτων-απογόνων, μέσω της ανταλλαγής τμημάτων των δέντρων των γονέων.
- **Μετάλλαξη (Mutation)**: Η **μετάλλαξη (mutation)** επιφέρει τυχαίες αλλαγές σε ένα μέρος του προγράμματος για τη δημιουργία ποικιλίας και την αποφυγή της πρόωρης

σύγκλισης. Η μετάλλαξη μπορεί να αφορά την αλλαγή ενός κόμβου ή μιας ολόκληρης υποδομής του δέντρου.

Επανάληψη (Iteration): Οι διαδικασίες αξιολόγησης, επιλογής και εφαρμογής γενετικών λειτουργιών επαναλαμβάνονται για πολλές γενιές, με στόχο τη συνεχή βελτίωση των προγραμμάτων.

Τερματισμός (Termination): Η διαδικασία του γενετικού προγραμματισμού τερματίζεται όταν επιτευχθεί το επιθυμητό επίπεδο απόδοσης, όταν δεν παρατηρείται περαιτέρω βελτίωση ή όταν συμπληρωθεί ο προκαθορισμένος αριθμός γενεών. Το καλύτερο πρόγραμμα που βρέθηκε θεωρείται η λύση στο πρόβλημα.

Εφαρμογές του Γενετικού Προγραμματισμού

Ο γενετικός προγραμματισμός έχει εφαρμογές σε πολλούς τομείς, όπως:

- Βελτιστοποίηση και προσαρμογή αλγορίθμων
- Αυτόματη παραγωγή κώδικα
- Ανάλυση και εξόρυξη δεδομένων
- Ρομποτική και αυτόνομα συστήματα
- Χρηματοοικονομικά μοντέλα και προβλέψεις
- Βιοπληροφορική και ανακάλυψη φαρμάκων.

2.3 Γραμματική εξέλιξη

Η **γραμματική εξέλιξη (Grammatical Evolution - GE)** είναι μια εξελικτική τεχνική που χρησιμοποιεί γραμματικές για την αυτόματη δημιουργία προγραμμάτων υπολογιστών. Σε αντίθεση με τον κλασικό γενετικό προγραμματισμό, η **GE** βασίζεται σε μια ανεξάρτητη αναπαράσταση των λύσεων, η οποία μεταφράζεται σε έγκυρα προγράμματα μέσω μιας προκαθορισμένης γραμματικής. Αυτή η προσέγγιση προσφέρει μεγαλύτερη ευελιξία και δυνατότητα χρήσης διαφορετικών γλωσσών προγραμματισμού.

Διαδικασία της Γραμματικής Εξέλιξης

Ορισμός Γραμματικής: Η διαδικασία ξεκινά με τον ορισμό μιας γραμματικής, η οποία καθορίζει τη σύνταξη των προγραμμάτων. Η γραμματική περιγράφεται μέσω κανόνων παραγωγής (production rules).

Κωδικοποίηση και Αρχικοποίηση Πληθυσμού: Ένας πληθυσμός υποψηφίων λύσεων κωδικοποιείται ως συμβολοσειρές (**strings**) ή χρωμοσώματα, τα οποία αναπαριστούν επιλογές από τους κανόνες της γραμματικής.

Αποκωδικοποίηση: Οι συμβολοσειρές αποκωδικοποιούνται σε έγκυρα προγράμματα, σύμφωνα με τους κανόνες παραγωγής της γραμματικής.

Αξιολόγηση Καταλληλόλητας (Fitness Evaluation): Τα προγράμματα αξιολογούνται με μια συνάρτηση καταλληλόλητας, η οποία μετρά την απόδοσή τους στην επίλυση του προβλήματος.

Επιλογή (Selection): Οι καλύτερες συμβολοσειρές επιλέγονται με βάση την καταλληλότητά τους για συμμετοχή στην επόμενη γενιά.

Γενετικές Λειτουργίες:

- **Αναπαραγωγή (Reproduction)**: Η **αναπαραγωγή** αφορά την αντιγραφή των καλύτερων συμβολοσειρών στην επόμενη γενιά.
- **Διασταύρωση (Crossover)**: Η **διασταύρωση** συνδυάζει δύο γονικές συμβολοσειρές για τη δημιουργία νέων απογόνων.
- **Μετάλλαξη (Mutation)**: Η **μετάλλαξη** αφορά τυχαία αλλαγή σε μέρος της συμβολοσειράς για την εισαγωγή ποικιλίας.

Επανάληψη και Τερματισμός: Η διαδικασία επαναλαμβάνεται για πολλές γενιές μέχρι να επιτευχθεί μια ικανοποιητική λύση ή να εξαντληθεί ο προκαθορισμένος αριθμός γενεών.

2.4 Κατασκευή κανόνων GenClass

Η κατασκευή κανόνων μέσω της **Γραμματικής Εξέλιξης (GE)** αποτελεί μια από τις βασικές εφαρμογές της για την κατηγοριοποίηση δεδομένων και τη δημιουργία ταξινομητικών μοντέλων. Το κύριο πλεονέκτημα της μεθόδου είναι η παραγωγή κανόνων που είναι κατανοητοί και επεξηγήσιμοι από τον χρήστη, σε αντίθεση με πιο σύνθετα

μοντέλα, όπως τα **νευρωνικά δίκτυα**, τα οποία, αν και πολύ αποδοτικά, συχνά χαρακτηρίζονται ως "μαύρα κουτιά".

Διαδικασία Κατασκευής Κανόνων

Η διαδικασία κατασκευής κανόνων μέσω της γραμματικής εξέλιξης-**ΓΕ** περιλαμβάνει τα εξής βήματα:

1. **Ορισμός Γραμματικής**: Οι κανόνες κατασκευάζονται με βάση τη γραμματική που ορίζεται μέσω της **BNF (Backus-Naur Form)**. Η γραμματική περιλαμβάνει τους επιτρεπτούς τελεστές, τα δεδομένα εισόδου και τις πιθανές δομές των προγραμμάτων.

2. **Εξελικτική Διαδικασία**: Η **γραμματική εξέλιξη** χρησιμοποιεί τη δομή των χρωμοσωμάτων για να αναπαραστήσει τους κανόνες ταξινόμησης. Κάθε χρωμόσωμα αποτελεί μια συμβολοσειρά που αναπαριστά τις επιλογές από τη γραμματική. Τεχνικές όπως η **διασταύρωση (crossover)** και η **μετάλλαξη (mutation)** επιτρέπουν τη συνεχή δημιουργία και βελτίωση των κανόνων.

3. **Αξιολόγηση καταλληλόλητας (Fitness Evaluation)**: Μετά από κάθε γενιά, οι κανόνες αξιολογούνται με βάση την ακρίβειά τους στην ταξινόμηση δεδομένων. Οι καλύτεροι κανόνες επιλέγονται για την επόμενη γενιά.

4. **Βελτιστοποίηση και Επεξηγησιμότητα**: Η **γραμματική εξέλιξη** βελτιστοποιεί όχι μόνο την ακρίβεια των κανόνων αλλά και την επεξηγησιμότητά τους. Οι κανόνες είναι κατανοητοί από τον άνθρωπο, καθιστώντας τη **GE** ιδανική για εφαρμογές όπου η διαφάνεια είναι σημαντική, όπως στην υγειονομική περίθαλψη, τη χρηματοοικονομική ανάλυση και τη νομική τεκμηρίωση.

2.5 Πλεονεκτήματα και Εφαρμογές της Γραμματικής Εξέλιξης

Η **Γραμματική Εξέλιξη (GE)** ξεχωρίζει για τα σημαντικά πλεονεκτήματα που προσφέρει σε σύγκριση με άλλες μεθόδους μηχανικής μάθησης. Παρόλο που παραδοσιακές μέθοδοι, όπως τα **νευρωνικά δίκτυα** ή οι **Υποστηρικτικοί Διανυσματικοί Μηχανισμοί (SVMs)**, επιδεικνύουν εξαιρετική απόδοση σε προβλήματα ταξινόμησης και παλινδρόμησης, η **GE** προσφέρει επιπλέον την εξηγήσιμη φύση των μοντέλων της.

Πλεονεκτήματα της ΓΕ

Επεξηγησιμότητα: Τα μοντέλα που παράγονται μέσω της **GE** είναι κατανοητά από τον άνθρωπο. Σε αντίθεση με τα **νευρωνικά δίκτυα**, που συχνά θεωρούνται "μαύρα κουτιά", οι κανόνες της GE είναι επεξηγήσιμοι, καθιστώντας τους ιδιαίτερα χρήσιμους σε πεδία όπου απαιτείται διαφάνεια..

Ευελιξία: Η **GE** προσφέρει μεγάλη ευελιξία στη δομή των μοντέλων, καθώς προσαρμόζεται σε διάφορα σύνολα δεδομένων και προβλήματα. Μπορεί επίσης να χρησιμοποιηθεί για τη δημιουργία κανόνων σε πολλές γλώσσες προγραμματισμού.

Δυναμική Βελτιστοποίηση: Η εξελικτική φύση της **Γραμματικής Εξέλιξης (GE)** επιτρέπει τη συνεχή βελτιστοποίηση των κανόνων, αυξάνοντας την ακρίβεια ταξινόμησης με κάθε νέα γενιά. Αυτή η διαδικασία εξασφαλίζει ότι οι λύσεις προσαρμόζονται στα δεδομένα, προσφέροντας μεγαλύτερη ακρίβεια και απόδοση..

Εφαρμογές της ΓΕ

Η **Γραμματική Εξέλιξη (GE)** έχει εφαρμοστεί σε πολλούς τομείς, όπως:

- **Βιοπληροφορική**: Η GE χρησιμοποιείται για την κατηγοριοποίηση βιολογικών δεδομένων, όπως η ανάλυση γονιδίων, όπου η επεξηγησιμότητα των μοντέλων είναι κρίσιμη.
- **Χρηματοοικονομική Ανάλυση**: Η GE εφαρμόζεται για τη δημιουργία μοντέλων πρόβλεψης χρηματοοικονομικών δεδομένων, όπως οι δείκτες της αγοράς. Οι εξηγήσιμοι κανόνες βοηθούν στη λήψη στρατηγικών επενδυτικών αποφάσεων..
- **Αυτόματη Σύνθεση Μουσικής**: Η GE έχει χρησιμοποιηθεί σε δημιουργικές εφαρμογές μουσικής σύνθεσης, όπου οι κανόνες της γραμματικής καθοδηγούν τη δημιουργία πρωτότυπων μουσικών κομματιών.

3. Υλοποίηση της Διεπαφής με τη Χρήση Τεχνολογιών: FastAPI, React και MongoDB

Για την υλοποίηση της γραφικής διεπαφής της εφαρμογής που εξυπηρετεί τη δημιουργία κανόνων ταξινόμησης με βάση την **Γραμματική Εξέλιξη (Grammatical Evolution - GE)**, χρησιμοποιήθηκαν οι τεχνολογίες **FastAPI** για το πίσω μέρος (backend) και **React** για το εμπρός μέρος (frontend).

FastAPI ως Υπηρεσία Πίσω Μέρους (Backend Service)

Το **FastAPI** είναι ένα σύγχρονο framework για την ανάπτυξη εφαρμογών ιστού και αποτελεί κορυφαία επιλογή για τη δημιουργία **RESTful APIs**. Η αρχιτεκτονική **RESTful (Representational State Transfer)** επιτρέπει την αποτελεσματική αλληλεπίδραση μεταξύ πελάτη και εξυπηρετητή μέσω του πρωτοκόλλου **HTTP**, καθιστώντας το ιδανικό για εφαρμογές που απαιτούν ταχύτητα, απόδοση και ευελιξία.

Πλεονεκτήματα του FastAPI στη Δημιουργία RESTful Εφαρμογών

1. **Υψηλή Απόδοση και Ταχύτητα**: Το **FastAPI** είναι σχεδιασμένο για υψηλές επιδόσεις, προσφέροντας ταχύτητα ισοδύναμη με λύσεις όπως το **Node.js** και το **Go**. Αυτό επιτυγχάνεται χάρη στη χρήση σύγχρονων τεχνικών της Python, όπως οι **async/await**, που επιτρέπουν την ασύγχρονη εκτέλεση πολλών αιτημάτων ταυτόχρονα χωρίς να επιβαρύνεται η απόδοση της εφαρμογής.

2. **Αυτόματη Δημιουργία και Τεκμηρίωση API**: Το **FastAPI** υποστηρίζει αυτόματη δημιουργία τεκμηρίωσης API, χρησιμοποιώντας πρότυπα όπως το **OpenAPI** και το **JSON Schema**. Αυτό επιτρέπει στους προγραμματιστές να έχουν πλήρη τεκμηρίωση που διευκολύνει την κατανόηση και ανάπτυξη του API..

3. **Υποστήριξη RESTful Αρχιτεκτονικής**: Το **FastAPI** είναι σχεδιασμένο για τη δημιουργία **RESTful APIs** με απλό και επεκτάσιμο τρόπο. Οι προγραμματιστές μπορούν να δημιουργήσουν τυπικές λειτουργίες HTTP (**GET, POST, PUT, DELETE**) και να διαχειριστούν πόρους εύκολα, καθιστώντας το ιδανικό για εφαρμογές που απαιτούν λειτουργίες **CRUD (Create, Read, Update, Delete)**.

4. **Επικύρωση Δεδομένων:** Το **FastAPI** χρησιμοποιεί τη βιβλιοθήκη **Pydantic** της Python, η οποία επιτρέπει την αυτόματη επικύρωση και μετατροπή των δεδομένων

εισόδου. Αυτό διασφαλίζει ότι οι εισερχόμενες αιτήσεις περιέχουν έγκυρα και σωστά μορφοποιημένα δεδομένα, ενισχύοντας την ασφάλεια και τη σταθερότητα της εφαρμογής.

5. **Ασύγχρονη Επεξεργασία (Asynchronous Programming)**: Το **FastAPI** υποστηρίζει ασύγχρονη εκτέλεση, επιτρέποντας τη διαχείριση πολλαπλών αιτημάτων ταυτόχρονα, χωρίς καθυστερήσεις. Αυτή η λειτουργία είναι ιδιαίτερα χρήσιμη για εφαρμογές με υψηλό όγκο αιτημάτων, βοηθώντας τις να κλιμακώνονται πιο εύκολα και αποτελεσματικά.

6. **Ευελιξία και Επεκτασιμότητα**: Το **FastAPI** προσφέρει μεγάλη ευελιξία και επεκτασιμότητα, επιτρέποντας στους προγραμματιστές να προσαρμόζουν και να επεκτείνουν τις εφαρμογές τους με ελάχιστες αλλαγές στον κώδικα.

7. **Χρήση Αντιπροσωπεύσεων (Representations)**: Στο πλαίσιο της **RESTful** αρχιτεκτονικής, το **FastAPI** επιτρέπει την ανταλλαγή δεδομένων μέσω μορφών όπως **JSON**, **XML** ή **HTML**, με δυνατότητα παραμετροποίησης των απαντήσεων ανάλογα με τις ανάγκες του πελάτη..

RESTful Εφαρμογές και FastAPI: Τι Προσφέρει;

Οι **RESTful APIs** με FastAPI είναι πολύ αποδοτικές λόγω της υποστήριξης των βασικών αρχών της RESTful αρχιτεκτονικής:

- **Stateless Communication**: Κάθε αίτημα από τον πελάτη προς τον εξυπηρετητή είναι ανεξάρτητο, και δεν αποθηκεύονται πληροφορίες σχετικά με προηγούμενα αιτήματα. Αυτό βελτιώνει την απόδοση και την ασφάλεια.

- **Καθαρή Χρήση Πόρων (Resources)**: Κάθε στοιχείο ή δεδομένο που διαχειρίζεται η εφαρμογή έχει μοναδική διεύθυνση URL. Οι πόροι διαχειρίζονται μέσω τυπικών HTTP μεθόδων, όπως **GET**, **POST**, και **DELETE**.

- **Ασφάλεια και Επικύρωση**: Το FastAPI διευκολύνει την ενσωμάτωση μηχανισμών αυθεντικοποίησης και εξουσιοδότησης, καθώς και την ασφαλή επικύρωση των δεδομένων για την προστασία των εφαρμογών.

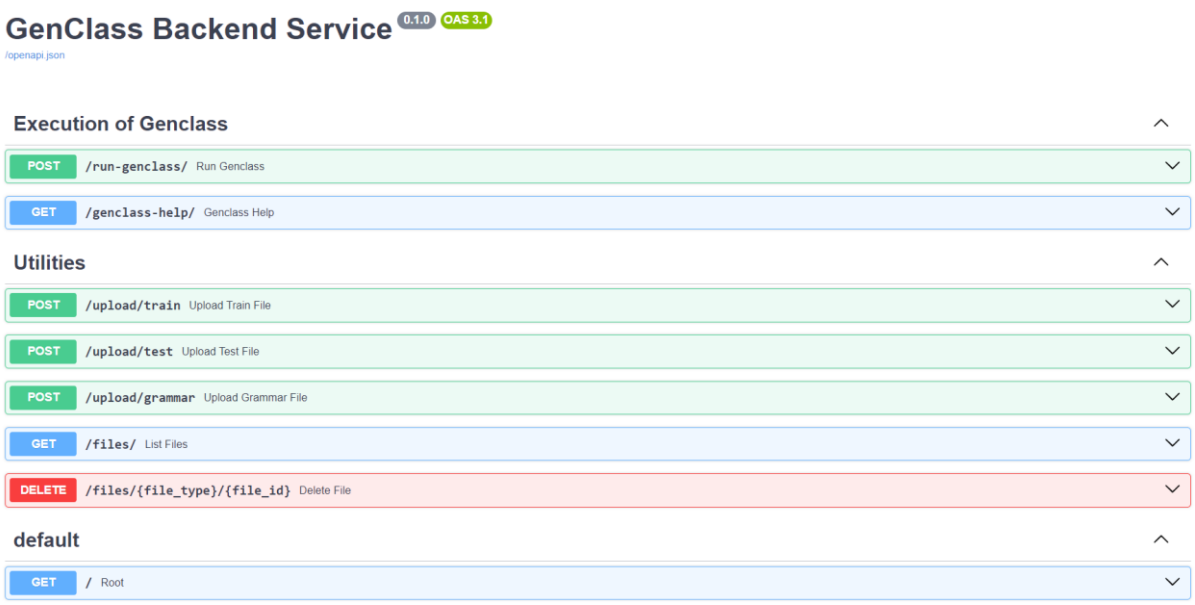
Ενέργειες και Λειτουργίες του GenClass Backend

POST / run-genclass / : Εκτέλεση της διαδικασίας GenClass

Αυτό το ακραίο σημείο(- endpoint) χρησιμοποιείται για την εκκίνηση και την εκτέλεση της υπηρεσίας **GenClass**, η οποία χρησιμοποιεί τη **Γραμματική Εξέλιξη** για τη δημιουργία και βελτιστοποίηση κανόνων ταξινόμησης. Οι χρήστες μπορούν να στείλουν τα απαιτούμενα δεδομένα και τις παραμέτρους μέσω ενός **POST αιτήματος (request)** για να τρέξουν την υπηρεσία.

GET /genclass-help/ : Παρέχει βοήθεια για τη χρήση της GenClass

Αυτό το ακραίο σημείο(- endpoint) παρέχει μια αναλυτική περιγραφή και βοήθεια σχετικά με τη χρήση της υπηρεσίας **GenClass**. Οι χρήστες μπορούν να ανακτήσουν οδηγίες για το πώς να αλληλεπιδράσουν με την υπηρεσία και πώς να στείλουν αιτήματα για τη δημιουργία κανόνων ταξινόμησης.



Εικόνα 1: Endpoints του πίσω μέρους της εφαρμογής

Διαχείριση των Αρχείων

POST /upload/: Ανέβασμα αρχείων στη βάση δεδομένων

Οι χρήστες μπορούν να ανεβάσουν αρχεία που είναι απαραίτητα για την εκτέλεση της υπηρεσίας **GenClass**. Αυτά τα ακραία σημεία (- endpoints) δέχονται τα αρχεία και τα

αποθηκεύουν στην βάση δεδομένων. Τα αρχεία αυτά περιέχουν δεδομένα εκπαίδευσης ή άλλες παραμέτρους που απαιτούνται για την ταξινόμηση.

GET /files/: Επιστροφή της λίστας των αρχείων

Εμφανίζει όλα τα αρχεία που έχουν ανέβει στην βάση δεδομένων. Αυτό επιτρέπει στους χρήστες να δουν ποια αρχεία είναι διαθέσιμα για επεξεργασία ή χρήση στην υπηρεσία **GenClass**.

DELETE /files/{filename}: Διαγραφή αρχείου από το σύστημα. Αυτό το ακραίο σημείο(endpoint) χρησιμοποιείται για τη διαγραφή ενός αρχείου από το σύστημα. Οι χρήστες μπορούν να διαγράψουν αρχεία που δεν χρειάζονται πλέον, απελευθερώνοντας έτσι πόρους και αποθηκευτικό χώρο.

main.py : είναι η κύρια είσοδος του backend και χρησιμοποιεί το **FastAPI** framework για να ορίσει το API.

Οι βασικές λειτουργίες του είναι:

Routers : Εισάγει τους routers από τα αρχεία **genclassgenerations.py** και **utils.py** για να επιτρέψει την εκτέλεση του GenClass και τη διαχείριση των αρχείων.

CORS middleware : Επιτρέπει τις συνδέσεις από το frontend, που μπορεί να βρίσκεται σε διαφορετικό origin, ώστε να γίνει η επικοινωνία μεταξύ τους(main).

genclass.py : Ορίζει την κλάση **GenClass**, η οποία περιέχει τα μοντέλα δεδομένων που χρειάζεται το FastAPI για να επικυρώσει τις παραμέτρους του αλγόριθμου. Χρησιμοποιεί το **Pydantic** για να ελέγξει ότι οι παράμετροι που στέλνονται από το frontend είναι σωστές, όπως αριθμοί χρωμοσωμάτων, γενιές, ποσοστά επιλογής, και μετάλλαξης(genclass).

genclassgenerations.py : διαχειρίζεται τις κύριες λειτουργίες του API για την εκτέλεση του αλγόριθμου **GenClass**:

/run-genclass/: Λαμβάνει τις παραμέτρους που στέλνονται από το frontend και καλεί τον αλγόριθμο μέσω subprocess για την εκτέλεσή του.

/genclass-help/: Παρέχει βοήθεια σχετικά με τις παραμέτρους του GenClass, που εμφανίζεται στο frontend μέσω του **GenclassHelp.js** (genclassgenerations) .

utils.py : παρέχει βοηθητικές συναρτήσεις για τη διαχείριση των αρχείων:

- Αποθηκεύει και διαγράφει αρχεία από την βάση δεδομένων, όπου βρίσκονται τα αρχεία εκπαίδευσης και δοκιμής που ανεβάζει ο χρήστης.
- Υποστηρίζει λειτουργίες για τη λίστα των αρχείων που υπάρχουν στον server και την επιστροφή των αντίστοιχων δεδομένων στο frontend(utils).

requirements.txt : περιλαμβάνει τις βιβλιοθήκες που απαιτούνται για την εκτέλεση του backend. Αυτές περιλαμβάνουν το **FastAPI** για την υλοποίηση του API, το **Uvicorn** για την εκτέλεση του server, το httpx για ασύγχρονα HTTP αιτήματα, το motor/pymongo για την υποστηρίξει-σύνδεση στην βάση δεδομένων και άλλες βιβλιοθήκες όπως **Pydantic** για την επικύρωση δεδομένων και **python-multipart** για την υποστήριξη μεταφόρτωσης αρχείων (requirements).

3.1 Γραφικές διεπαφές (εμπρός μέρος/frontend) με React/JAVASCRIPT

Το γραφικό μέρος της εφαρμογής έχει υλοποιηθεί με JavaScript και συγκεκριμένα με την χρήση της React, από μια σειρά με βιβλιοθήκες που παρέχει για την κατασκευή δυναμικών σελίδων/διεπαφών χρήστη. Η React είναι από τα πιο γνωστές βιβλιοθήκες για τον σχεδιασμό διεπαφών χρήστη (User Interfaces – UIs), επιτρέπει στον προγραμματιστή να
βλέπει τις αλλαγές που κάνει ενώ προγραμματίζει σε πραγματικό χρόνο. Αυτό οφείλεται στο ότι είναι μονοσέλιδες εφαρμογές (- Single-Page Applications - SPAs) και επειδή χρησιμοποιούν το Virtual DOM (- Virtual Document Object). Βασίζεται στα components(συστατικά), κομμάτια κώδικα τα οποία επαναχρησιμοποιούνται κι διευκολύνουν την ανάπτυξη και την οργάνωση.

Component-Based Architecture (Αρχιτεκτονική με συνιστωσών - Components)
Το React βασίζεται στην αρχιτεκτονική των συνιστωσών (components), όπου κάθε κομμάτι της διεπαφής αναπαρίσταται ως ένα μικρό, ανεξάρτητο κομμάτι κώδικα, το οποίο μπορεί να επαναχρησιμοποιηθεί σε διάφορα σημεία της εφαρμογής. Κάθε component διαχειρίζεται τη δική του κατάσταση (state) και λογική, καθιστώντας το ευέλικτο και εύκολα επαναχρησιμοποιήσιμο.
Στην εφαρμογή, παραδείγματα τέτοιων components είναι (βλέπε εικόνες στο 3.2):

- **ConfusionMatrixChart.js**: Ένα component που δημιουργεί ένα διάγραμμα πίτας το οποίο απεικονίζει την Confusion Matrix του αλγορίθμου κατηγοριοποίησης, επιτρέποντας στο χρήστη να κατανοήσει καλύτερα τα True Positive (TP), False Negative (FN), False Positive (FP), και True Negative (TN) αποτελέσματα.
- **ErrorChart.js**: Προβάλλει ένα ραβδόγραμμα που παρουσιάζει τα ποσοστά σφάλματος στο σύνολο της εκπαίδευσης και της κατηγοριοποίησης, δίνοντας μια πιο απτή εικόνα της απόδοσης του αλγορίθμου.

Αυτά τα components είναι ανεξάρτητα και μπορούν να χρησιμοποιηθούν ξανά, επιτρέποντας την εύκολη συντήρηση και επέκταση της εφαρμογής.

Διαχείριση Κατάστασης (State Management)

Ένα από τα βασικά πλεονεκτήματα του React είναι η διαχείριση της κατάστασης των components σε πραγματικό χρόνο. Η κατάσταση (state) αναφέρεται στα δεδομένα που μεταβάλλονται καθώς ο χρήστης αλληλεπιδρά με την εφαρμογή. Το React χρησιμοποιεί τα **hooks** για να παρακολουθεί και να διαχειρίζεται αυτές τις μεταβολές χωρίς να απαιτείται εκ νέου φόρτωση της σελίδας. Στην εφαρμογή σου, το **state (κατάσταση)** διαχειρίζεται κυρίως μέσω των hooks όπως το useState και το useEffect, που επιτρέπουν την αποθήκευση και επεξεργασία των δεδομένων σε κάθε συνιστώσα (component). Για παράδειγμα, στο **GenClassForm.js**, η φόρμα διαχειρίζεται την κατάσταση των παραμέτρων που εισάγει ο χρήστης, όπως τον αριθμό των χρωμοσωμάτων, τις γενιές, και τα ποσοστά μετάλλαξης και επιλογής. Κάθε αλλαγή που κάνει ο χρήστης στη φόρμα ενημερώνεται άμεσα στην κατάσταση του component, και τα δεδομένα αποστέλλονται στο πίσω μέρος (backend) όταν ο χρήστης υποβάλει το αίτημά του.

Χρήση Hooks

Τα **hooks** είναι ένας από τους πιο ευέλικτους και σύγχρονους μηχανισμούς της React, που επιτρέπουν την επαναχρησιμοποίηση της λογικής μεταξύ διαφορετικών components. Στην εφαρμογή, έχουν υλοποιηθεί δύο βασικά custom hooks:

- **useFiles.js**: Αυτό το hook είναι υπεύθυνο για τη διαχείριση αρχείων. Παρέχει τη δυνατότητα μεταφόρτωσης, διαγραφής, και ανάκτησης των αρχείων που έχουν αποθηκευτεί στην βάση δεδομένων. Το hook επίσης φροντίζει να εμφανίζει

μηνύματα και ειδοποιήσεις στον χρήστη σχετικά με την κατάσταση της κάθε ενέργειας (π.χ. επιτυχής μεταφόρτωση, σφάλμα διαγραφής).

- **useGenClass.js**: Το useGenClass hook διαχειρίζεται την εκτέλεση του αλγορίθμου GenClass. Αφού ο χρήστης υποβάλει τις παραμέτρους του αλγορίθμου, το hook αναλαμβάνει την αποστολή των δεδομένων στο backend και λαμβάνει τα αποτελέσματα της εκτέλεσης, τα οποία στη συνέχεια προβάλλονται στην οθόνη.

Τα hooks αυτά επιτρέπουν τη μείωση του επαναλαμβανόμενου κώδικα και τη διαχείριση των δεδομένων με αποτελεσματικό τρόπο, διατηρώντας τον κώδικα οργανωμένο και ευέλικτο.

Material-UI (MUI) για Κατασκευή Διεπαφών

Η εφαρμογή χρησιμοποιεί τη βιβλιοθήκη **Material-UI** (MUI) για τη διαχείριση και τον σχεδιασμό των components της διεπαφής. Το MUI προσφέρει μια σειρά από έτοιμα components(συνιστώσες) βασισμένα στο Material Design της Google, που εξασφαλίζουν την κατασκευή μοντέρνων και χρηστικών διεπαφών με λιγότερο κόπο.

Μερικά παραδείγματα στοιχείων που έχουν χρησιμοποιηθεί είναι:

- **TextField**: Χρησιμοποιείται για την είσοδο δεδομένων στη φόρμα παραμετροποίησης του αλγορίθμου GenClass.
- **Button**: Χρησιμοποιείται για διάφορες ενέργειες όπως η εκτέλεση του αλγορίθμου ή η μεταφόρτωση αρχείων.
- **Grid**: Διευκολύνει τη δημιουργία πλεγμάτων και ευθυγραμμίσεων στη διεπαφή, καθιστώντας τη διάταξη των στοιχείων ομαλή και ευανάγνωστη.

Η χρήση του **Material-UI** όχι μόνο μειώνει την πολυπλοκότητα του σχεδιασμού, αλλά εξασφαλίζει επίσης ότι η διεπαφή ακολουθεί τις καλύτερες πρακτικές σε ό,τι αφορά την εμφάνιση και την εμπειρία χρήστη (UI/UX).

Axios για Επικοινωνία με το πίσω μέρος (Backend)

Η βιβλιοθήκη **axios** χρησιμοποιείται για την αποστολή αιτημάτων HTTP στο πίσω-μέρος της εφαρμογής, το οποίο έχει υλοποιηθεί με **FastAPI**. Μέσω της axios, η εφαρμογή μπορεί να στείλει δεδομένα στο backend για την εκτέλεση του αλγορίθμου ή για τη μεταφόρτωση και διαχείριση αρχείων. Τα δεδομένα που επιστρέφονται από το backend παρουσιάζονται στην

οθόνη του χρήστη, όπως τα αποτελέσματα του αλγορίθμου GenClass. Η χρήση του axios καθιστά εύκολη την επικοινωνία μεταξύ του frontend και του backend, εξασφαλίζοντας ότι η διεπαφή παραμένει διαδραστική και τα δεδομένα παρουσιάζονται άμεσα.

Οπτικοποίηση Δεδομένων με Χρήση Γραφικών Διαγραμμάτων

Η εφαρμογή περιλαμβάνει λειτουργίες οπτικοποίησης των δεδομένων που παράγονται από τον αλγόριθμο **GenClass**. Αυτή η οπτικοποίηση γίνεται μέσω των εξειδικευμένων components:

- **ConfusionMatrixChart.js**: Αυτό το component δημιουργεί ένα διάγραμμα πίτας που απεικονίζει το Confusion Matrix του αλγορίθμου. Το Confusion Matrix είναι ένα εργαλείο για την αξιολόγηση της απόδοσης ενός αλγόριθμου κατηγοριοποίησης, δείχνοντας τον αριθμό των σωστών και λανθασμένων προβλέψεων.
- **ErrorChart.js**: Προβάλλει ένα ραβδόγραμμα με τα ποσοστά σφάλματος στο σύνολο εκπαίδευσης (Train Error) και κατηγοριοποίησης (Class Error), δίνοντας στον χρήστη μια πιο απτή αναπαράσταση των σφαλμάτων του αλγόριθμου.

Η χρήση γραφικών διαγραμμάτων καθιστά τα αποτελέσματα πιο κατανοητά και προσφέρει μια οπτική ανατροφοδότηση που διευκολύνει την κατανόηση της απόδοσης του αλγορίθμου.

Διαχείριση Αρχείων

Το **FileManagement.js** component επιτρέπει στον χρήστη να ανεβάζει και να διαχειρίζεται αρχεία που χρησιμοποιούνται για την εκτέλεση του αλγόριθμου. Ο χρήστης μπορεί να δει τη λίστα των αρχείων που έχουν ανεβεί στην βάση δεδομένων, να διαγράψει αρχεία που δεν χρειάζεται πλέον και να ανεβάσει νέα αρχεία με σχετικές πληροφορίες που απαιτούνται για την εκτέλεση του GenClass.

Προσαρμογή Θέματος με Theme.js

Η εφαρμογή χρησιμοποιεί το **Theme.js** αρχείο για τον καθορισμό των χρωμάτων και των στυλ της διεπαφής. Το συγκεκριμένο αρχείο παρέχει μια ενιαία και συνεπή εμφάνιση σε όλα τα components της εφαρμογής, ενώ παράλληλα προσφέρει τη δυνατότητα προσαρμογής του θέματος εύκολα και γρήγορα.

Ανάλυση Αποτελεσμάτων Αλγορίθμου

Το **parseGenClassOutput.js** αρχείο είναι υπεύθυνο για την ανάλυση των δεδομένων που προκύπτουν από την εκτέλεση του αλγόριθμου GenClass. Το αρχείο αυτό λαμβάνει τα

αποτελέσματα που παράγονται από τον αλγόριθμο, τα διαχωρίζει και τα μορφοποιεί σε δομημένα δεδομένα, όπως το Confusion Matrix και τα ποσοστά σφάλματος, τα οποία στη συνέχεια προβάλλονται στον χρήστη μέσω των αντίστοιχων διαγραμμάτων.

3.2 Εργαλεία υλοποίησης

Visual Studio Code

Το **Visual Studio Code (VS Code)**, δημιουργήθηκε από τη **Microsoft**, κυκλοφόρησε για πρώτη φορά το 2015. Στόχος της **Microsoft** ήταν να προσφέρει στους προγραμματιστές ένα εργαλείο που να είναι ταυτόχρονα ελαφρύ, γρήγορο και ευέλικτο, κατάλληλο για πολλές γλώσσες προγραμματισμού. Το Visual Studio Code αποτελεί μια ελαφριά εναλλακτική του βαρύτερου **Visual Studio**, με σκοπό να ικανοποιήσει τις ανάγκες των προγραμματιστών που εργάζονται σε έργα ανοιχτού κώδικα, γλώσσες scripting, καθώς και web development.

Χρήση του Σήμερα: Το VS Code είναι ένα από τα πιο δημοφιλή εργαλεία ανάπτυξης λογισμικού σήμερα. Υποστηρίζει πολλές γλώσσες και προσφέρει μια τεράστια γκάμα επεκτάσεων για γλώσσες προγραμματισμού όπως η **Python**, η **JavaScript**, η **TypeScript** και άλλες. Διαθέτει λειτουργίες όπως **αυτόματη συμπλήρωση κώδικα (IntelliSense)**, **εντοπισμό σφαλμάτων (debugging)**, ενσωματωμένο τερματικό **(integrated terminal)** και εργαλεία **Git** για διαχείριση κώδικα/εκδόσεων. Επίσης, η κοινότητά του αναπτύσσει συνεχώς νέα plugins και επεκτάσεις, κάτι που το καθιστά ιδανικό εργαλείο για επαγγελματίες και ερασιτέχνες προγραμματιστές.

Git/GitHub

Το **Git** δημιουργήθηκε το 2005 από τον **Linus Torvalds**, τον δημιουργό του **Linux Kernel**, με σκοπό να βελτιώσει τον έλεγχο εκδόσεων μεγάλων έργων λογισμικού. Η ανάγκη για ένα κατανεμημένο σύστημα ελέγχου εκδόσεων προέκυψε όταν ο Torvalds χρειαζόταν έναν τρόπο να διαχειριστεί τις διαφορετικές εκδοχές του Linux Kernel που αναπτύσσονταν από χιλιάδες προγραμματιστές παγκοσμίως. Το **GitHub**, το οποίο ξεκίνησε το 2008, είναι μια πλατφόρμα βασισμένη στο Git που επιτρέπει την αποθήκευση και διαχείριση κώδικα μέσω του διαδικτύου.

Χρήση του Σήμερα: Το **Git** είναι το πιο διαδεδομένο εργαλείο για τον έλεγχο εκδόσεων (version control) και χρησιμοποιείται για την παρακολούθηση αλλαγών στον κώδικα. Επιτρέπει τη συνεργασία μεγάλων ομάδων προγραμματιστών, προσφέροντας δυνατότητες όπως η παρακολούθηση της ιστορίας του έργου, η δημιουργία διακλαδώσεων (branches) για πειραματική ανάπτυξη και η συγχώνευση (merge) αλλαγών. Το **GitHub** χρησιμοποιείται από εκατομμύρια προγραμματιστές και οργανισμούς για την αποθήκευση κώδικα, τη συνεργασία και την υποστήριξη ανοιχτού κώδικα έργων. Επίσης, προσφέρει λειτουργίες όπως **pull requests**, που διευκολύνουν τον έλεγχο και την ενσωμάτωση κώδικα από διάφορους προγραμματιστές-εθελοντές.

Python 3.10

Η **Python** ξεκίνησε το 1991 από τον **Guido van Rossum** και σχεδιάστηκε ως μια εύκολη και διαισθητική γλώσσα προγραμματισμού, που προσφέρει ευελιξία και υποστηρίζει πολλαπλά παραδείγματα προγραμματισμού, όπως αντικειμενοστραφή και λειτουργικό προγραμματισμό. Η Python 3 εισήχθη το 2008 για να αντιμετωπίσει διάφορα προβλήματα της Python 2, με σημαντικές αλλαγές στη σύνταξη και την απόδοση. Η έκδοση **Python 3.10** κυκλοφόρησε το 2021, φέρνοντας νέες δυνατότητες όπως **pattern matching**, βελτιώσεις στον **τύπο των δεδομένων (type hinting)**, καθώς και ασύγχρονη επεξεργασία.

Χρήση του Σήμερα: Η **Python** είναι μια από τις πιο δημοφιλείς γλώσσες προγραμματισμού παγκοσμίως και χρησιμοποιείται ευρέως για ανάπτυξη web, επιστήμη δεδομένων, μηχανική μάθηση, αυτοματισμούς και scripting. Η έκδοση **Python 3.10** ενισχύει αυτές τις δυνατότητες με χαρακτηριστικά που βελτιώνουν την παραγωγικότητα των προγραμματιστών και προσφέρουν καλύτερες επιδόσεις. Η Python είναι δημοφιλής για το **εκτεταμένο οικοσύστημα βιβλιοθηκών** της, όπως το **NumPy**, το **Pandas** και το **TensorFlow**, που διευκολύνουν την ανάπτυξη σε τομείς όπως η ανάλυση δεδομένων και η τεχνητή νοημοσύνη.

FastAPI

Το **FastAPI** είναι ένα σύγχρονο, γρήγορο web framework που κυκλοφόρησε το 2018 από τον**Sebastián Ramírez**. Δημιουργήθηκε για να καλύψει την ανάγκη για ένα εργαλείο ανάπτυξης APIs που να είναι ταυτόχρονα απλό στη χρήση και υψηλής απόδοσης, εκμεταλλευόμενο τις δυνατότητες της ασύγχρονης επεξεργασίας της **Python**. Εστιάζει στη χρήση των **type hints** και στη γρήγορη ανάπτυξη RESTful APIs, προσφέροντας βελτιστοποίηση στην απόδοση και την ασφάλεια.

Χρήση του Σήμερα: Το **FastAPI** είναι πλέον ένα από τα πιο δημοφιλή frameworks για την ανάπτυξη web εφαρμογών και APIs στη Python. Η ταχύτητα και η ευκολία με την οποία μπορούν να δημιουργηθούν APIs το καθιστούν ιδανικό για ανάπτυξη εφαρμογών όπως **cloud services**, **microservices** και **machine learning APIs**. Προσφέρει δυνατότητες όπως **αυτόματη δημιουργία τεκμηρίωσης** και **επικύρωση δεδομένων** με τη χρήση της βιβλιοθήκης **Pydantic**.

Pydantic

Η **Pydantic** είναι μια βιβλιοθήκη της Python που κυκλοφόρησε το 2017 και δημιουργήθηκε για την **επικύρωση και τη διαχείριση δεδομένων** με βάση τους τύπους τους. Η Pydantic χρησιμοποιεί το σύστημα **type hints** της Python και διασφαλίζει ότι τα δεδομένα που λαμβάνονται από εξωτερικές πηγές είναι έγκυρα και σωστά διαμορφωμένα.

Χρήση του Σήμερα: Η **Pydantic** χρησιμοποιείται ευρέως μαζί με το **FastAPI** για τη διαχείριση και την επαλήθευση των δεδομένων που λαμβάνονται από APIs. Είναι ιδανική για εφαρμογές που απαιτούν αυστηρό έλεγχο και επαλήθευση δεδομένων, όπως οι τραπεζικές εφαρμογές ή οι εφαρμογές υγειονομικής περίθαλψης, όπου η σωστή διαχείριση δεδομένων είναι κρίσιμη για την ακρίβεια και την ασφάλεια.

Httpx

Το **Httpx** είναι μια βιβλιοθήκη της **Python** για τη διαχείριση HTTP αιτημάτων, η οποία δημιουργήθηκε ως ένας πιο σύγχρονος και ασύγχρονος αντικαταστάτης της γνωστής βιβλιοθήκης **Requests**. Ξεκίνησε ως προσπάθεια να παρέχει μια πιο γρήγορη και ευέλικτη λύση για την επικοινωνία με APIs.

Χρήση του Σήμερα: Το **Httpx** χρησιμοποιείται ευρέως για την επικοινωνία με εξωτερικά APIs, ιδίως σε εφαρμογές που απαιτούν ασύγχρονη διαχείριση αιτημάτων. Ενσωματώνεται άψογα με πλαίσια όπως το **FastAPI** και χρησιμοποιείται για την αποστολή και λήψη δεδομένων μέσω του δικτύου με αποδοτικό τρόπο, διασφαλίζοντας γρήγορες και ασφαλείς συνδέσεις.

Uvicorn

Το **Uvicorn** είναι ένας γρήγορος HTTP server που σχεδιάστηκε για ασύγχρονες εφαρμογές **Python** που χρησιμοποιούν το πρωτόκολλο **ASGI (Asynchronous Server Gateway Interface)**. Δημιουργήθηκε για να υποστηρίξει τη γρήγορη εκτέλεση εφαρμογών που απαιτούν διαχείριση μεγάλου αριθμού αιτημάτων.

Χρήση του Σήμερα: Το **Uvicorn** χρησιμοποιείται ως ο βασικός server σε ασύγχρονες web εφαρμογές που αναπτύσσονται με frameworks όπως το **FastAPI**. Είναι ιδανικό για εφαρμογές που απαιτούν **υψηλή απόδοση και ταχύτητα** στη διαχείριση αιτημάτων και απαντήσεων, ειδικά σε περιβάλλοντα όπου η χαμηλή καθυστέρηση είναι σημαντική.

React

Η **React** δημιουργήθηκε από το **Facebook** το 2013 για να αντιμετωπίσει τις αυξανόμενες ανάγκες της εταιρείας για δυναμικές διεπαφές χρήστη (UIs). Σχεδιάστηκε με στόχο να απλοποιήσει την κατασκευή πολύπλοκων διεπαφών μέσω της χρήσης των **components** (**συνιστωσών)**, προσφέροντας τη δυνατότητα επαναχρησιμοποίησης του κώδικα και την ευκολότερη διαχείριση της κατάστασης δεδομένων.

Χρήση του Σήμερα: Η **React** είναι μία από τις πιο δημοφιλείς βιβλιοθήκες για την ανάπτυξη **Single-Page Applications (SPAs)**. Επιτρέπει στους προγραμματιστές να δημιουργούν διαδραστικές διεπαφές χρήστη που ανανεώνονται σε πραγματικό χρόνο χωρίς την ανάγκη πλήρους φόρτωσης της σελίδας. Η ευκολία χρήσης του και η τεράστια κοινότητα προγραμματιστών το καθιστούν κορυφαίο εργαλείο για την ανάπτυξη εφαρμογών όπως **e-commerce**, **dashboard εφαρμογές** και **cloud-based platforms**.

MongoDB

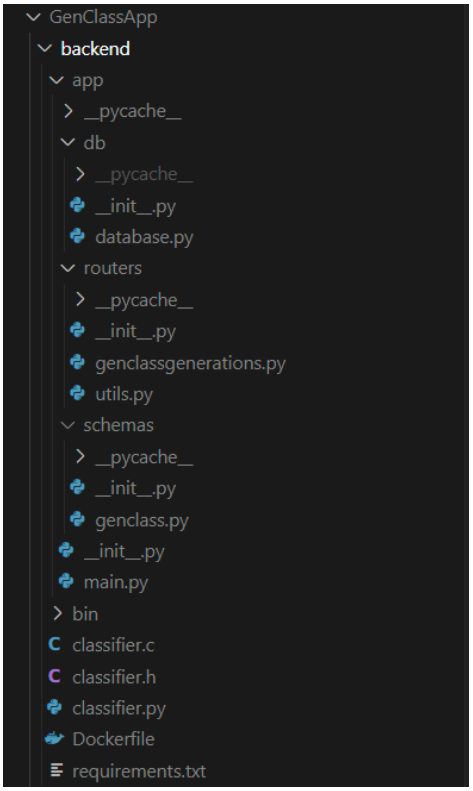
Η MongoDB δημιουργήθηκε το **2007** από την εταιρεία **10gen**, η οποία αργότερα μετονομάστηκε σε **MongoDB Inc.**. Αναπτύχθηκε για να καλύψει την ανάγκη για μια ευέλικτη, επεκτάσιμη βάση δεδομένων που μπορεί να διαχειριστεί μεγάλους όγκους

δεδομένων σε σύγχρονες εφαρμογές. Κυκλοφόρησε ως **ανοιχτού κώδικα** το **2009**, προσφέροντας ένα καινοτόμο μοντέλο αποθήκευσης δεδομένων σε μορφή εγγράφων BSON (Binary JSON), το οποίο επιτρέπει την αποθήκευση διαφορετικών τύπων δεδομένων χωρίς την ανάγκη ενός αυστηρού σχήματος, όπως συμβαίνει στις σχεσιακές βάσεις δεδομένων.

Χρήση του Σήμερα: Η MongoDB έχει καθιερωθεί ως μία από τις πιο δημοφιλείς **NoSQL** βάσεις δεδομένων και χρησιμοποιείται ευρέως σε εφαρμογές που απαιτούν **ευελιξία και κλιμάκωση**. Με δυνατότητες όπως το **sharding** για την κατανομή δεδομένων και τη **replication** για την εξασφάλιση υψηλής διαθεσιμότητας, είναι ιδανική για εφαρμογές **cloud services**, **mobile εφαρμογές**, **real-time analytics**, και **IoT**. Το γεγονός ότι η MongoDB υποστηρίζει σαν **JSON** αποθήκευση και γρήγορη ανάκτηση δεδομένων την καθιστά κορυφαία επιλογή για επιχειρήσεις που χρειάζονται γρήγορη και αποδοτική διαχείριση μεγάλων δεδομένων.

3.3 Τα μέρη της εφαρμογής

3.3.1 Δομή πίσω μέρους (- backend) της εφαρμογής.



Εικόνα 2: Αρχεία και πακέτα του πίσω μέρους

```python
from app.schemas.genclass import GenClass
from fastapi import APIRouter, HTTPException, Depends
from motor.motor_asyncio import AsyncIOMotorDatabase
from app.db.database import get_db
import asyncio
import tempfile
import os
import logging

logging.basicConfig(level=logging.INFO)

genclass_router = APIRouter(tags=["Execution of Genclass"])
    theodorismeko [5 days ago] • added docker-compose and restructured the files      You, 5 days ago • added docker-compose and restruc
@genclass_router.post("/run-genclass/")
async def run_genclass(params: GenClass, db: AsyncIOMotorDatabase = Depends(get_db)):
    try:
        train_file = None
        if params.train_file:
            train_file = await db.train_files.find_one({"filename": params.train_file})
            if not train_file:
                raise HTTPException(status_code=404, detail=f"Train file not found in database: {params.train_file}")

        test_file = None
        if params.train_file:
            test_file = await db.test_files.find_one({"filename": params.test_file})
            if not test_file:
                raise HTTPException(status_code=404, detail=f"Test file not found in database: {params.test_file}")

        grammar_file = None
        if params.grammar:
            grammar_file = await db.grammar_files.find_one({"filename": params.grammar})
            if not grammar_file:
                raise HTTPException(status_code=404, detail=f"Grammar file not found in database: {params.grammar}")

        with tempfile.TemporaryDirectory() as temp_dir:
            # Write files to temporary directory
            train_path = os.path.join(temp_dir, params.train_file)
            test_path = os.path.join(temp_dir, params.test_file)

            with open(train_path, 'wb') as f:
                f.write(train_file['content'])
            with open(test_path, 'wb') as f:
                f.write(test_file['content'])

            # Prepare command
            command = [
                "./bin/genclass",
                "-c", str(params.count),
                "-g", str(params.gens),
                "-d", str(params.threads),
                "-s", f"{params.srate:.2f}",
                "-m", f"{params.mrate:.2f}",
                "-l", str(params.size),
                "-p", train_path,
                "-t", test_path,
                "-o", params.output_method
            ]

            # Add grammar file to command if provided
            if grammar_file:
                grammar_path = os.path.join(temp_dir, params.grammar)
                with open(grammar_path, 'wb') as f:
                    f.write(grammar_file['content'])
                command.extend(["-i", grammar_path])
```

Εικόνα 3: Αρχείο genclassgenerations.py (1)

```
            # Execute command
            process = await asyncio.create_subprocess_exec(
                *command,
                stdout=asyncio.subprocess.PIPE,
                stderr=asyncio.subprocess.PIPE)
            stdout, stderr = await process.communicate()

            if process.returncode != 0:
                raise HTTPException(status_code=400, detail=f"Genclass execution failed: {stderr.decode()}")

            logging.info("Genclass execution completed successfully")
            return {"output": stdout.decode()}

    except HTTPException as he:
        logging.error(f"HTTP Error: {he.detail}")
        raise he
    except Exception as e:
        logging.error(f"An error occurred: {str(e)}")
        raise HTTPException(status_code=500, detail=f"An error occurred: {str(e)}")

@genclass_router.get("/genclass-help/")
async def genclass_help():
    help_options = ["-h", "--help", "-help", "/?"]
    for option in help_options:
        try:
            process = await asyncio.create_subprocess_exec(
                "../bin/genclass",
                option,
                stdout=asyncio.subprocess.PIPE,
                stderr=asyncio.subprocess.PIPE
            )
            stdout, stderr = await process.communicate()
            if process.returncode == 0:
                return {"help": stdout.decode()}
        except Exception:
            continue

    # If no help option worked, provide a custom help message
    custom_help = """
    Usage: genclass [options]

    Options:
    -c count   : Number of chromosomes (default: 500)
    -f count   : Fold count for fold validation (default: 0, no folding)
    -g gens    : Maximum number of generations (default: 200)
    -d count   : Maximum number of threads for OpenMp (default: 16)
    -s srate   : Selection rate (default: 0.10)
    -m mrate   : Mutation rate (default: 0.05)
    -l size    : Size of every chromosome (default: 100)
    -p file    : File containing train data
    -t file    : File containing test data
    -o method  : Output method (simple, csv, or full)
    -r seed    : Seed for random number generator
    -i file    : Input grammar file in BNF format

    For more detailed information, please refer to the program's documentation.
    """
    return {"help": custom_help}
```

Εικόνα 4: Αρχείο genclassgenerations.py (2)

```python
import os
from app.db.database import init_db, close_db
from fastapi import FastAPI, HTTPException, Body
from app.routers.genclassgenerations import genclass_router
from app.routers.utils import utils_router
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI(title="GenClass Backend Service")

origins = [
    "http://localhost:3000",
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"]
)

app.include_router(genclass_router)
app.include_router(utils_router)

@app.on_event("startup")
async def startup_db_client():
    init_db(app)

@app.on_event("shutdown")
async def shutdown_db_client():
    close_db(app)

@app.get("/")
async def root():
    return {"message": "Hello World"}
```

Εικόνα 5: Αρχείο main.py

```python
from fastapi import FastAPI, File, UploadFile, HTTPException, APIRouter, Depends
from fastapi.responses import JSONResponse
import logging
from typing import List, Optional
from bson import ObjectId
from motor.motor_asyncio import AsyncIOMotorDatabase
from app.db.database import get_db

logging.basicConfig(level=logging.INFO)

utils_router = APIRouter(tags=["Utilities"])

async def upload_file(file: UploadFile, db: AsyncIOMotorDatabase, collection_name: str):
    try:
        # Check if file already exists
        existing_file = await db[collection_name].find_one({"filename": file.filename})
        if existing_file:
            raise HTTPException(status_code=400, detail=f"File '{file.filename}' already exists in {collection_name}")

        content = await file.read()
        document = {
            "filename": file.filename,
            "content": content
        }
        result = await db[collection_name].insert_one(document)
        logging.info(f"{collection_name} file uploaded successfully. Document ID: {result.inserted_id}")
        return {"message": f"{collection_name} file uploaded successfully", "document_id": str(result.inserted_id)}
    except HTTPException as he:
        logging.error(f"HTTP Error: {he.detail}")
        raise he
    except Exception as e:
        logging.error(f"Error uploading {collection_name} file: {str(e)}")
        raise HTTPException(status_code=500, detail=f"Error uploading {collection_name} file: {str(e)}")

@utils_router.post("/upload/train")
async def upload_train_file(train_file: UploadFile = File(...), db: AsyncIOMotorDatabase = Depends(get_db)):
    return await upload_file(train_file, db, "train_files")

@utils_router.post("/upload/test")
async def upload_test_file(test_file: UploadFile = File(...), db: AsyncIOMotorDatabase = Depends(get_db)):
    return await upload_file(test_file, db, "test_files")

@utils_router.post("/upload/grammar")
async def upload_grammar_file(grammar: UploadFile = File(...), db: AsyncIOMotorDatabase = Depends(get_db)):
    return await upload_file(grammar, db, "grammar_files")

@utils_router.get("/files/", response_model=List[dict])
async def list_files(db: AsyncIOMotorDatabase = Depends(get_db)):
    try:
        train_files = await db.train_files.find().to_list(length=None)
        test_files = await db.test_files.find().to_list(length=None)
        grammar_files = await db.grammar_files.find().to_list(length=None)    theodorismeko [6 hours ago] • code refactor to

        files = []
        for file_list in [train_files, test_files, grammar_files]:
            files.extend([{
                "id": str(file["_id"]),
                "filename": file["filename"],
                "type": "train" if file in train_files else "test" if file in test_files else "grammar"
            } for file in file_list])

        return files
    except Exception as e:
        logging.error(f"Error listing files: {str(e)}")
        raise HTTPException(status_code=500, detail=f"Error listing files: {str(e)}")
```

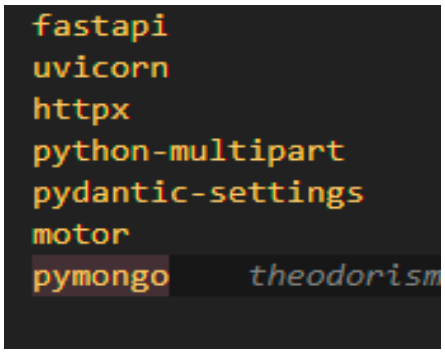
Εικόνα 6: Αρχείο utils.py (1)

```python
@utils_router.delete("/files/{file_type}/{file_id}")
async def delete_file(file_type: str, file_id: str, db: AsyncIOMotorDatabase = Depends(get_db)):
    try:
        if file_type not in ["train", "test", "grammar"]:
            raise HTTPException(status_code=400, detail="Invalid file type")

        collection_name = f"{file_type}_files"
        result = await db[collection_name].delete_one({"_id": ObjectId(file_id)})
        if result.deleted_count == 0:
            raise HTTPException(status_code=404, detail=f"{file_type.capitalize()} file with id '{file_id}' not found.")

        logging.info(f"{file_type.capitalize()} file with id {file_id} deleted successfully.")
        return {"message": f"{file_type.capitalize()} file with id '{file_id}' deleted successfully."}
    except Exception as e:
        logging.error(f"Error deleting file: {str(e)}")
        raise HTTPException(status_code=500, detail=f"Error deleting file: {str(e)}")    theodorismeko [6 hours ago] • code r
```

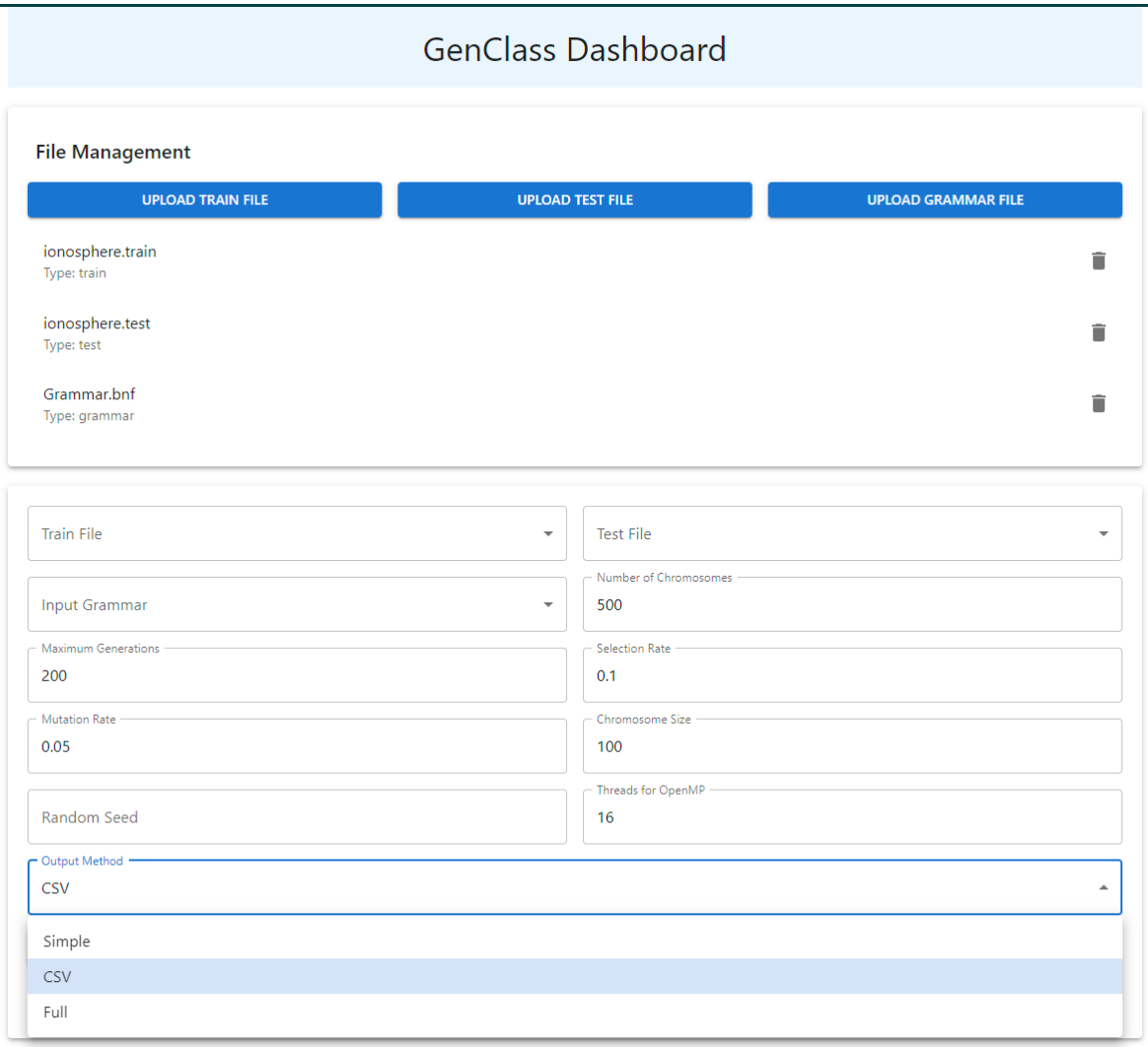
Εικόνα 7: Αρχείο utils.py (2)

```python
from fastapi import FastAPI, HTTPException, Body

from pydantic import BaseModel, Field, ConfigDict

You, 2 days ago | 1 author (You)
class GenClass(BaseModel):
    count: int = Field(default=500, description="Number of chromosomes")
    gens: int = Field(default=200, description="Maximum number of generations")
    threads: int = Field(default=16, description="Maximum number of threads for OpenMp")    theodorismeko [5 days ago]
    srate: float = Field(default=0.10, description="Selection rate")
    mrate: float = Field(default=0.05, description="Mutation rate")
    size: int = Field(default=100, description="Size of every chromosome")
    train_file: str = Field(..., example="ionosphere.train", description="File containing train data")
    test_file: str = Field(..., example="ionosphere.test", description="File containing test data")
    output_method: str = Field(default="simple", example="csv" , description="Output method (simple, csv, or full)")
    grammar: str | None = Field(default=None, example="Grammar.bnf", description="Input Grammar used")

    seed: int | None = Field(default=None, description="Seed for random number generator")


    model_config = ConfigDict(extra = 'forbid',
       json_schema_extra = {
            "example": {
                "count": 500,
                "gens": 200,
                "threads": 16,
                "srate": 0.10,
                "mrate": 0.05,
                "size": 100,
                "train_file": "ionosphere.train",
                "test_file": "ionosphere.test",
                "output_method": "simple",
                "grammar": "Grammar.bnf",
                "seed": 42,
            }
        })
```

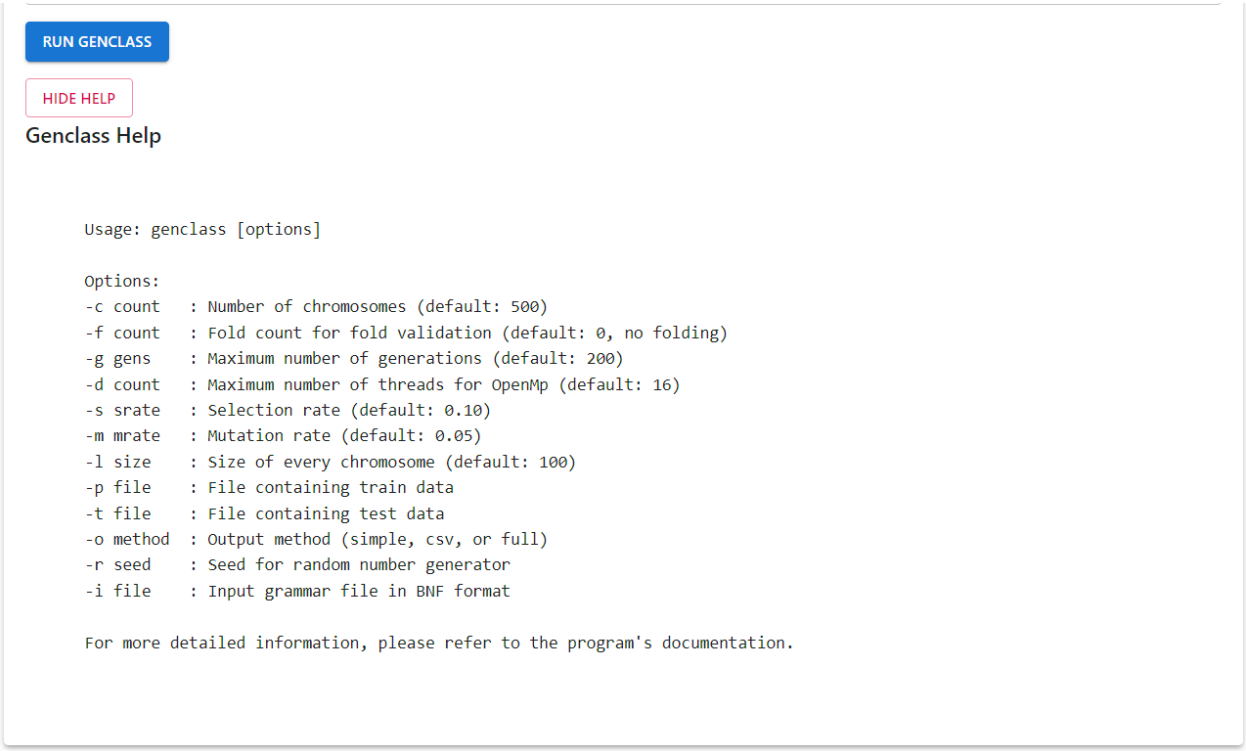
Εικόνα 8: Αρχείο genclass.py



Εικόνα 9: Αρχείο requirements.txt



Εικόνα 10: Dashboard της εφαρμογής



Εικόνα 11: Επιλογές και παράμετροι του genclass

3.3.2 Εμπρός μέρος – Frontend

```jsx
import React from 'react';
import { Typography, Grid, Paper } from '@mui/material';
import { PieChart } from '@mui/x-charts/PieChart';

function ConfusionMatrixChart({ data }) {
  if (!data || data.length !== 4) return null;
  const [tp, fn, fp, tn] = data;

  // Prepare data for the pie chart    theodorismeko [5 days ago] • added docker-compos
  const pieData = [
    { id: 0, value: tp},
    { id: 1, value: fn},
    { id: 2, value: fp},
    { id: 3, value: tn},
  ];

  return (
    <Grid container spacing={2}>
      <Grid item xs={8}>
        <Typography variant="h6" align="center"></Typography>
        <PieChart
          series={[
            { data: pieData,
              highlightScope: { fade: 'global', highlight: 'item' },
              faded: { innerRadius: 30, additionalRadius: -30, color: 'gray' },
            }]}
          width={535}
          height={200}
        />
      </Grid>
      <Grid item xs={12}>
        <Typography variant="h6" align="center"></Typography>
      </Grid>
      <Grid item xs={5}>
        <Paper elevation={4} style={{ padding: '8px', textAlign: 'center' }}>
          <Typography>True Positive: {tp}</Typography>
        </Paper>
      </Grid>
      <Grid item xs={5}>
        <Paper elevation={4} style={{ padding: '8px', textAlign: 'center' }}>
          <Typography>False Negative: {fn}</Typography>
        </Paper>
      </Grid>
      <Grid item xs={5}>
        <Paper elevation={4} style={{ padding: '8px', textAlign: 'center' }}>
          <Typography>False Positive: {fp}</Typography>
        </Paper>
      </Grid>
      <Grid item xs={5}>
        <Paper elevation={4} style={{ padding: '8px', textAlign: 'center' }}>
          <Typography>True Negative: {tn}</Typography>
        </Paper>
      </Grid>
    </Grid>
  );
}

export default ConfusionMatrixChart;
```

Εικόνα 12: Αρχείο ConfussionMatrix.js

```jsx
import React from 'react';    theodorismeko [5 days ago] • added docker-compos
import { BarChart } from '@mui/x-charts';

function ErrorChart({ trainError, classError }) {
  return (
    <BarChart
      xAxis={[{ scaleType: 'band', data: ['Train Error', 'Class Error'] }]}
      series={[{ data: [trainError, classError] }]}
      width={450}
      height={380}
    />
  );
}

export default ErrorChart;
```

Εικόνα 13: Αρχείο ErrorChart.js

```jsx
import React from 'react';
import { Typography, Button, List, ListItem, ListItemText, ListItemSecondaryAction, IconButton, CircularProgress, Grid } from '@mui/material';
import DeleteIcon from '@mui/icons-material/Delete';

function FileManagement({ files, loading, onFileUpload, onDeleteFile }) {
  const fileTypes = [
    { type: 'train', accept: '.train', fieldName: 'train_file' },
    { type: 'test', accept: '.test', fieldName: 'test_file' },
    { type: 'grammar', accept: '.bnf', fieldName: 'grammar' }
  ];

  const handleFileUpload = (event, fieldName) => {
    const file = event.target.files[0];
    if (file) {
      onFileUpload(file, fieldName);
    }
  };

  const renderFileInfo = (file) => {
    const primary = file.filename || 'Unnamed file';
    const secondary = `Type: ${file.type || 'Unknown'}`;
    return { primary, secondary };
  };

  return (
    <>
      <Typography variant="h6" align='justify' p="8px" marginBottom='8px'>File Management</Typography>
      <Grid container spacing={2}>
        {fileTypes.map(({ type, accept, fieldName }) => (
          <Grid item xs={12} sm={4} key={type}>
            <input
              accept={accept}
              style={{ display: 'none' }}
              id={`${type}-file-upload`}
              type="file"
              onChange={(e) => handleFileUpload(e, fieldName)}
              disabled={loading}
            />
            <label htmlFor={`${type}-file-upload`}>
              <Button variant="contained" component="span" disabled={loading} fullWidth>
                Upload {type.charAt(0).toUpperCase() + type.slice(1)} File
              </Button>
            </label>
          </Grid>
        ))}
      </Grid>
      {loading && <CircularProgress size={24} style={{ margin: '15px auto', display: 'block' }} />}
      <List>
        {files.map((file) => {
          const { primary, secondary } = renderFileInfo(file);    theodorismeko [6 hours ago] • code refactor to use a database and deployment using
          return (
            <ListItem key={file.id}>
              <ListItemText primary={primary} secondary={secondary} />
              <ListItemSecondaryAction>
                <IconButton
                  edge="end"
                  aria-label="delete"
                  onClick={() => onDeleteFile(file.type, file.id)}
                  disabled={loading}
                >
                  <DeleteIcon />
                </IconButton>
              </ListItemSecondaryAction>
            </ListItem>
          );
        })}
      </List>
    </>
  );
}

export default FileManagement;
```

Εικόνα 14: Αρχείο FileManagement.js

```jsx
import React from 'react';
import { TextField, Button, Grid, MenuItem } from '@mui/material';

function GenClassForm({ params, onChange, onRun, files }) {
  const getFilesByType = (type) => {
    return files.filter(file => {
      if (typeof file === 'string') {
        return file.endsWith(`.${type}`);
      } else if (file && typeof file === 'object' && file.filename) {
        return file.filename.endsWith(`.${type}`);
      }
      return false;
    });
  };

  const renderFileOption = (file) => {
    const value = typeof file === 'string' ? file : file.filename;
    return (
      <MenuItem key={value} value={value}>
        {value}
      </MenuItem>
    );
  };

  const handleFileChange = (event) => {
    const { name, value } = event.target;
    onChange({
      target: {
        name,
        value: value === 'none' ? '' : value
      }
    });
  };

  const renderFileSelect = (label, name, fileType) => (
    <TextField
      fullWidth
      select
      label={label}
      name={name}
      value={params[name] || ''}
      onChange={handleFileChange}
    >
      <MenuItem value="none">None</MenuItem>    theodorismeko [6 hours ago] • code refactor to use a database and dep
      {getFilesByType(fileType).map(renderFileOption)}
    </TextField>
  );


  return (
    <Grid container spacing={2}>
      <Grid item xs={12} sm={6}>
        {renderFileSelect("Train File", "train_file", "train")}
      </Grid>
      <Grid item xs={12} sm={6}>
        {renderFileSelect("Test File", "test_file", "test")}
      </Grid>
      <Grid item xs={12} sm={6}>
        {renderFileSelect("Input Grammar", "grammar", "bnf")}
      </Grid>
      <Grid item xs={12} sm={6}>
        <TextField
          fullWidth
          label="Number of Chromosomes"
          name="count"
          type="number"
          value={params.count}
          inputProps={{ step: 1, min: 0}}
          onChange={onChange}
        />
      </Grid>
      <Grid item xs={12} sm={6}>
        <TextField
          fullWidth
          label="Maximum Generations"
          name="gens"
          type="number"
          value={params.gens}
          onChange={onChange}
        />
      </Grid>
```

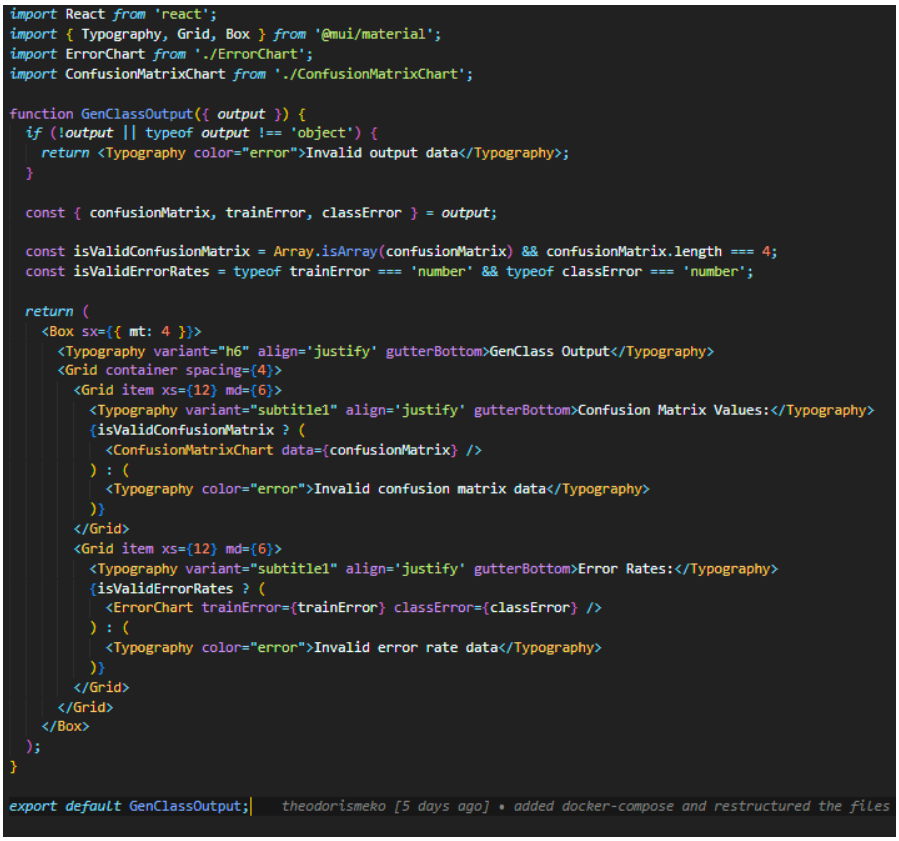
Εικόνα 15: Αρχείο GenClassForm.js (1)

```
function GenClassForm({ params, onChange, onRun, files }) {
    <Grid item xs={12} sm={6}>
      <TextField
        fullWidth
        label="Selection Rate"
        name="srate"
        type="number"
        inputProps={{ step: 0.01, min: 0, max: 1 }}
        value={params.srate}
        onChange={onChange}
      />
    </Grid>
    <Grid item xs={12} sm={6}>
      <TextField
        fullWidth
        label="Mutation Rate"
        name="mrate"
        type="number"
        inputProps={{ step: 0.01, min: 0, max: 1 }}
        value={params.mrate}
        onChange={onChange}
      />
    </Grid>
    <Grid item xs={12} sm={6}>
      <TextField
        fullWidth
        label="Chromosome Size"
        name="size"
        type="number"
        value={params.size}
        inputProps={{ step: 1, min: 0}}
        onChange={onChange}
      />
    </Grid>
    <Grid item xs={12} sm={6}>
      <TextField
        fullWidth
        label="Random Seed"
        name="seed"
        type="number"
        value={params.seed || ''}
        inputProps={{ step: 1, min: 0}}
        onChange={onChange}
      />
    </Grid>
    <Grid item xs={12} sm={6}>
      <TextField
        fullWidth
        label="Threads for OpenMP"
        name="threads"
        type="number"
        value={params.threads || ''}
        inputProps={{ step: 1, min: 0, max: 16 }}
        onChange={onChange}
      />
    </Grid>
    <Grid item xs={12} sm={12}>
      <TextField
        fullWidth
        select
        label="Output Method"
        name="output_method"
        value={params.output_method}
        onChange={onChange}
      >
        <MenuItem value="simple">Simple</MenuItem>
        <MenuItem value="csv">CSV</MenuItem>
        <MenuItem value="full">Full</MenuItem>
      </TextField>
    </Grid>

    <Grid item xs={12}>
      <Button  variant="contained" color="primary" onClick={onRun}>
        Run GenClass
      </Button>
    </Grid>
  </Grid>
  );
};

export default GenClassForm;
```

Εικόνα 16: Αρχείο GenClassForm.js (2)

```jsx
import React, { useState, useEffect } from 'react';
import { Typography, CircularProgress } from '@mui/material';
import { getGenclassHelp } from '../services/api';

const GenclassHelp = () => {
  const [help, setHelp] = useState('');
  const [error, setError] = useState(null);

  useEffect(() => {
    const fetchHelp = async () => {
      try {
        const response = await getGenclassHelp();
        setHelp(response.data.help);
      } catch (err) {
        setError(err.response?.data?.detail || 'Failed to fetch help');
      }
    };

    fetchHelp();
  }, []);

  if (error) return <Typography color="error">Error: {error}</Typography>;
  if (!help) return <CircularProgress />;

  return (
    <div>
      <Typography variant="h6" gutterBottom>Genclass Help</Typography>
      <pre style={{ whiteSpace: 'pre-wrap', wordBreak: 'break-word', padding: '20px', marginBottom: '20px' }}>{help}</pre>
    </div>
  );
};

export default GenclassHelp;
```

Εικόνα 17: Αρχείο GenclassHelp.js



```jsx
import React from 'react';
import { Typography, Grid, Box } from '@mui/material';
import ErrorChart from './ErrorChart';
import ConfusionMatrixChart from './ConfusionMatrixChart';

function GenClassOutput({ output }) {
  if (!output || typeof output !== 'object') {
    return <Typography color="error">Invalid output data</Typography>;
  }

  const { confusionMatrix, trainError, classError } = output;

  const isValidConfusionMatrix = Array.isArray(confusionMatrix) && confusionMatrix.length === 4;
  const isValidErrorRates = typeof trainError === 'number' && typeof classError === 'number';

  return (
    <Box sx={{ mt: 4 }}>
      <Typography variant="h6" align='justify' gutterBottom>GenClass Output</Typography>
      <Grid container spacing={4}>
        <Grid item xs={12} md={6}>
          <Typography variant="subtitle1" align='justify' gutterBottom>Confusion Matrix Values:</Typography>
          {isValidConfusionMatrix ? (
            <ConfusionMatrixChart data={confusionMatrix} />
          ) : (
            <Typography color="error">Invalid confusion matrix data</Typography>
          )}
        </Grid>
        <Grid item xs={12} md={6}>
          <Typography variant="subtitle1" align='justify' gutterBottom>Error Rates:</Typography>
          {isValidErrorRates ? (
            <ErrorChart trainError={trainError} classError={classError} />
          ) : (
            <Typography color="error">Invalid error rate data</Typography>
          )}
        </Grid>
      </Grid>
    </Box>
  );
}

export default GenClassOutput;    theodorismeko [5 days ago] • added docker-compose and restructured the files
```

Εικόνα 18: Αρχείο GenClassOutput.js

```jsx
import { useState, useEffect, useCallback } from 'react';
import { fetchFiles, uploadFile, deleteFile } from '../services/api';  // Ensure this path is correct

function useFiles() {
  const [files, setFiles] = useState([]);
  const [loading, setLoading] = useState(false);
  const [snackbar, setSnackbar] = useState({ open: false, message: '', severity: 'info' });

  const showSnackbar = (message, severity) => {
    setSnackbar({ open: true, message, severity });
  };

  const handleCloseSnackbar = (event, reason) => {
    if (reason === 'clickaway') {
      return;
    }
    setSnackbar({ ...snackbar, open: false });
  };

  const refreshFiles = useCallback(async () => {
    setLoading(true);
    try {
      const response = await fetchFiles();
      setFiles(response.data);
    } catch (error) {
      console.error('Error fetching files:', error);
      showSnackbar('Error fetching files', 'error');
    } finally {
      setLoading(false);
    }
  }, []);

  useEffect(() => {
    refreshFiles();
  }, [refreshFiles]);

  const handleFileUpload = async (file, fileType) => {
    setLoading(true);
    try {
      await uploadFile(file, fileType);
      await refreshFiles();
      showSnackbar('File uploaded successfully', 'success');
    } catch (error) {
      console.error('Error uploading file:', error);
      showSnackbar(error.response?.data?.detail || 'Error uploading file', 'error');
    } finally {
      setLoading(false);
    }
  };

  const handleDeleteFile = async (fileType, fileId) => {
    setLoading(true);
    try {
      await deleteFile(fileType, fileId);
      await refreshFiles();
      showSnackbar('File deleted successfully', 'success');
    } catch (error) {
      console.error('Error deleting file:', error);
      showSnackbar('Error deleting file', 'error');
    } finally {
      setLoading(false);
    }
  };

  return {
    files,
    loading,
    handleFileUpload,
    handleDeleteFile,
    snackbar,    theodorismeko [5 days ago] • added docker-compose and restructured the files      You, 5
    handleCloseSnackbar
  };
}

export default useFiles;
```

Εικόνα 19: Αρχείο useFiles.js

```jsx
import { useState } from 'react';
import { runGenClass } from '../services/api';
import { parseGenClassOutput } from '../utils/parseGenClassOutput';

const useGenClass = () => {
  const [genClassParams, setGenClassParams] = useState({
    count: 500,
    gens: 200,
    threads: 16,
    srate: 0.10,
    mrate: 0.05,
    size: 100,
    train_file: '',
    test_file: '',
    output_method: 'csv',
    grammar: '',
    seed: null
  });    theodorismeko [6 hours ago] • code refactor to use a database and deployment using docker

  const [genClassOutput, setGenClassOutput] = useState(null);

  const handleGenClassParamChange = (event) => {
    const { name, value } = event.target;
    setGenClassParams((prev) => ({
      ...prev,
      [name]: ['gens', 'count', 'threads', 'size', 'seed'].includes(name) ? parseInt(value, 10) :
              ['srate', 'mrate'].includes(name) ? parseFloat(value) : value,
    }));
  };

  const runGenClassAlgorithm = async () => {
    try {
      const response = await runGenClass(genClassParams);
      console.log('API response:', response.data);
      if (response.data.output) {
        const parsedOutput = parseGenClassOutput(response.data.output);

        // Ensure fallback values for error rates
        parsedOutput.trainError = parsedOutput.trainError || 0;
        parsedOutput.classError = parsedOutput.classError || 0;

        setGenClassOutput(parsedOutput);
      } else if (response.data.error) {
        console.error(`Error: ${response.data.error}`);
      }
    } catch (error) {
      console.error('Error running GenClass:', error.response ? error.response.data : error.message);
    }
  };

  return {
    genClassParams,
    genClassOutput,
    handleGenClassParamChange,
    runGenClass: runGenClassAlgorithm
  };

}

export default useGenClass;
```

Εικόνα 20: Αρχείο useGenClass.js

```
import axios from 'axios';

const API_BASE_URL = process.env.REACT_APP_API_BASE_URL ? 'http://backend:80' : 'http://localhost:80';

export const fetchFiles = () => axios.get(`${API_BASE_URL}/files/`);

export const uploadFile = (file, fieldName) => {
  const formData = new FormData();
  formData.append(fieldName, file);
  return axios.post(`${API_BASE_URL}/upload/${fieldName.replace('_file', '')}`, formData, {
    headers: { 'Content-Type': 'multipart/form-data' }
  });
};

export const deleteFile = (fileType, fileId) => axios.delete(`${API_BASE_URL}/files/${fileType}/${fileId}`);

export const runGenClass = (params) => axios.post(`${API_BASE_URL}/run-genclass/`, params);

export const getGenclassHelp = () => axios.get(`${API_BASE_URL}/genclass-help/`);    theodorismeko [5 days ago
```

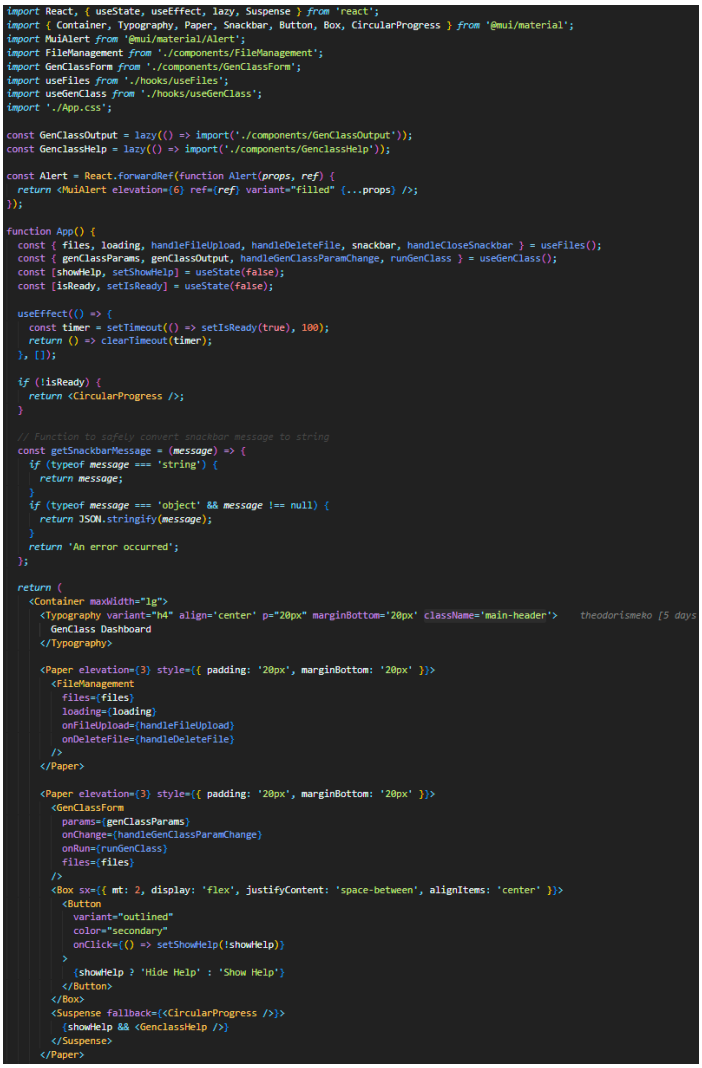
Εικόνα 21: Αρχείο api.js

```
import { createTheme } from '@mui/material/styles';

const theme = createTheme({
  palette: {
    primary: {
      main: '#1976d2',
    },
    secondary: {
      main: '#dc004e',
    },
  },
  typography: {
    fontFamily: [
      '-apple-system',
      'BlinkMacSystemFont',
      '"Segoe UI"',
      'Roboto',
      '"Helvetica Neue"',
      'Arial',
      'sans-serif',
      '"Apple Color Emoji"',
      '"Segoe UI Emoji"',
      '"Segoe UI Symbol"',
    ].join(','),
  },
});

export default theme;    theodorismeko [5 days ago] • added d
```

Εικόνα 22: Αρχείο theme.js

```
export function parseGenClassOutput(output) {
  console.log('Raw output:', output);  // Debug: Log the raw output

  const lines = output.split('\n');
  const classLine = lines.find(line => line.startsWith('CLASS='));
  const trainErrorLine = lines.find(line => line.startsWith('TRAIN ERROR'));
  const classErrorLine = lines.find(line => line.startsWith('CLASS ERROR'));
  const confusionMatrixLines = lines.filter(line => /^\s*\d+\s+\d+\s*$/.test(line));

  const classValue = classLine ? parseFloat(classLine.split('=')[1]) : 0;
  const trainError = trainErrorLine ? parseFloat(trainErrorLine.split('=')[1]) : 0;
  const classError = classErrorLine ? parseFloat(classErrorLine.split('=')[1]) : 0;

  let confusionMatrix = null;
  if (confusionMatrixLines.length === 2) {
    confusionMatrix = confusionMatrixLines
    .map(line => line.trim().split(/\s+/).map(Number))
    .flat();

  } else {
      console.warn('Unexpected confusion matrix format or insufficient lines.');
  }


  return { classValue, trainError, classError, confusionMatrix };
}
```

Εικόνα 23: Αρχείο parseGenClassOutput.js



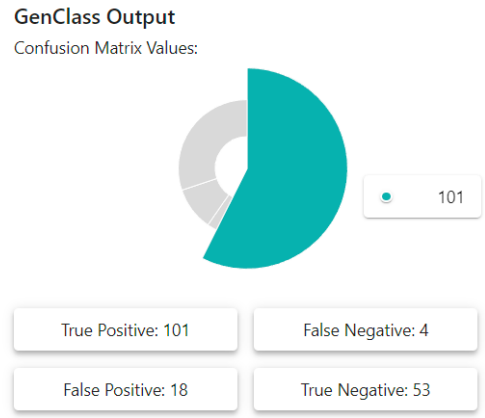
```jsx
import React, { useState, useEffect, lazy, Suspense } from 'react';
import { Container, Typography, Paper, Snackbar, Button, Box, CircularProgress } from '@mui/material';
import MuiAlert from '@mui/material/Alert';
import FileManagement from './components/FileManagement';
import GenClassForm from './components/GenClassForm';
import useFiles from './hooks/useFiles';
import useGenClass from './hooks/useGenClass';
import './App.css';

const GenClassOutput = lazy(() => import('./components/GenClassOutput'));
const GenclassHelp = lazy(() => import('./components/GenclassHelp'));

const Alert = React.forwardRef(function Alert(props, ref) {
  return <MuiAlert elevation={6} ref={ref} variant="filled" {...props} />;
});

function App() {
  const { files, loading, handleFileUpload, handleDeleteFile, snackbar, handleCloseSnackbar } = useFiles();
  const { genClassParams, genClassOutput, handleGenClassParamChange, runGenClass } = useGenClass();
  const [showHelp, setShowHelp] = useState(false);
  const [isReady, setIsReady] = useState(false);

  useEffect(() => {
    const timer = setTimeout(() => setIsReady(true), 100);
    return () => clearTimeout(timer);
  }, []);

  if (!isReady) {
    return <CircularProgress />;
  }

  // Function to safely convert snackbar message to string
  const getSnackbarMessage = (message) => {
    if (typeof message === 'string') {
      return message;
    }
    if (typeof message === 'object' && message !== null) {
      return JSON.stringify(message);
    }
    return 'An error occurred';
  };

  return (
    <Container maxWidth="lg">
      <Typography variant="h4" align='center' p="20px" marginBottom='20px' className='main-header'>    theodorismeko [5 days
        GenClass Dashboard
      </Typography>

      <Paper elevation={3} style={{ padding: '20px', marginBottom: '20px' }}>
        <FileManagement
          files={files}
          loading={loading}
          onFileUpload={handleFileUpload}
          onDeleteFile={handleDeleteFile}
        />
      </Paper>

      <Paper elevation={3} style={{ padding: '20px', marginBottom: '20px' }}>
        <GenClassForm
          params={genClassParams}
          onChange={handleGenClassParamChange}
          onRun={runGenClass}
          files={files}
        />
        <Box sx={{ mt: 2, display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
          <Button
            variant="outlined"
            color="secondary"
            onClick={() => setShowHelp(!showHelp)}
          >
            {showHelp ? 'Hide Help' : 'Show Help'}
          </Button>
        </Box>
        <Suspense fallback={<CircularProgress />}>
          {showHelp && <GenclassHelp />}
        </Suspense>
      </Paper>
```

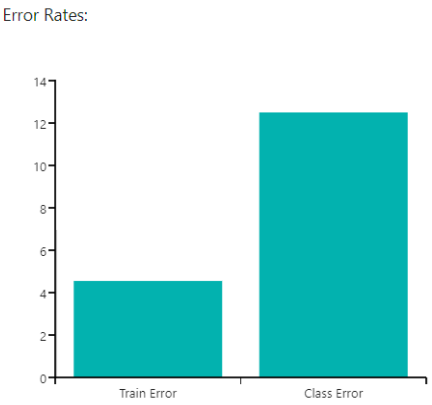
Εικόνα 24: Αρχείο App.js (1)

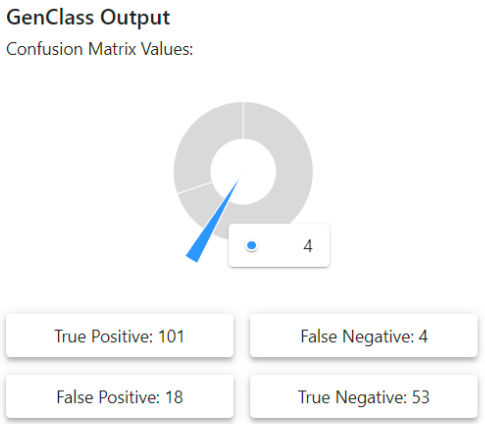
```jsx
      <Paper elevation={0} style={{ padding: '40px', marginBottom: '30px' }}>
        <Suspense fallback={<CircularProgress />}>
          {genClassOutput && <GenClassOutput output={genClassOutput} />}
        </Suspense>
      </Paper>

      <Snackbar open={snackbar.open} autoHideDuration={6000} onClose={handleCloseSnackbar}>
        <Alert onClose={handleCloseSnackbar} severity={snackbar.severity} sx={{ width: '100%' }}>
          {getSnackbarMessage(snackbar.message)}
        </Alert>
      </Snackbar>
    </Container>
  );
}

export default App;
```

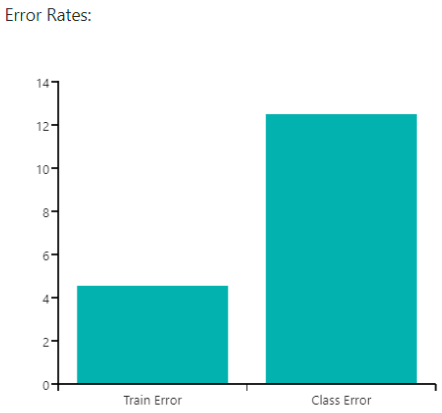
Εικόνα 25: Αρχείο App.js (2)

3.4 Πειραματικά αποτελέσματα



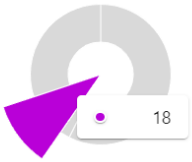






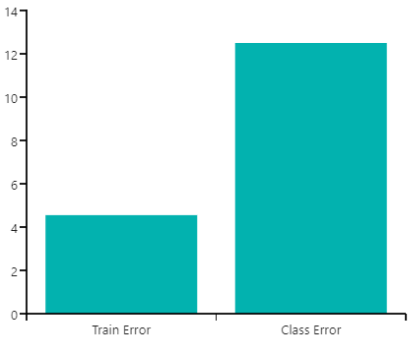
GenClass Output

Confusion Matrix Values:



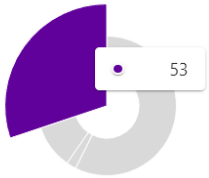
True Positive: 101	False Negative: 4
False Positive: 18	True Negative: 53

Error Rates:



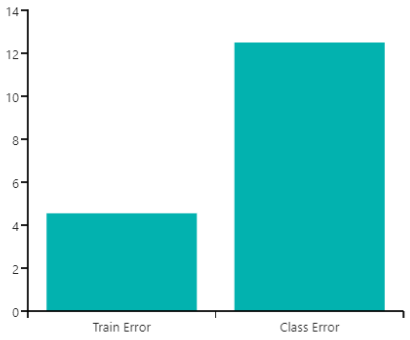
GenClass Output

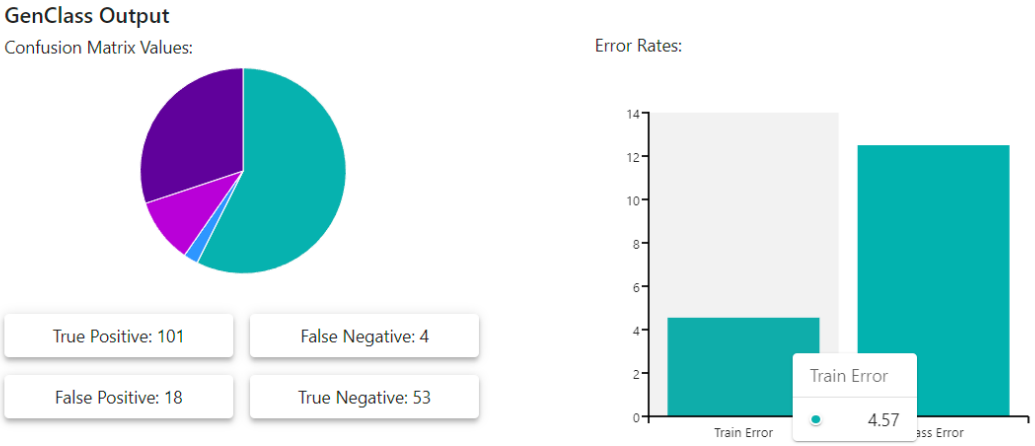
Confusion Matrix Values:

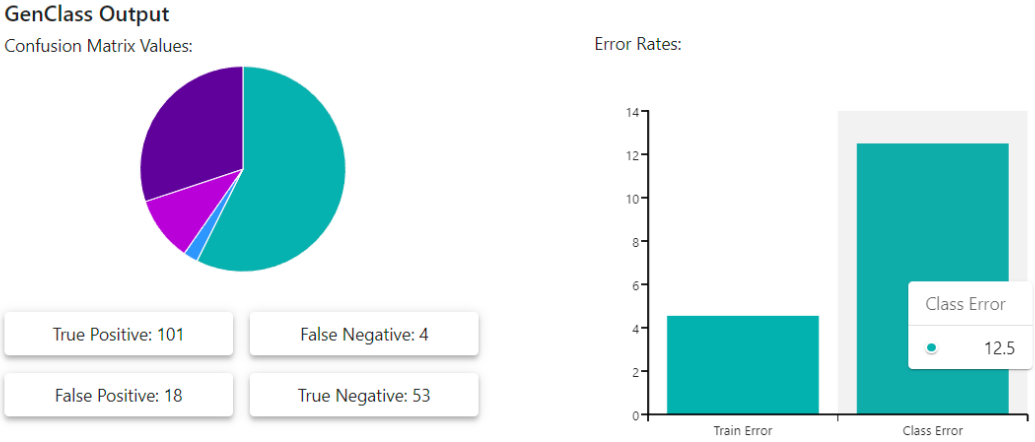


True Positive: 101	False Negative: 4
False Positive: 18	True Negative: 53

Error Rates:







Τα παραπάνω διαγράμματα παρέχουν την δυνατότητα αξιολόγηση της απόδοσης, της ακρίβειας του αλγορίθμου καθώς και τα ποσοστά σφάλματος.

3.5 Πακετάρισμα εφαρμογής με χρήση Docker

Το **Docker** είναι ένα εργαλείο ανοιχτού κώδικα(open-source) που επιτρέπει τη δημιουργία, τη διανομή και την εκτέλεση εφαρμογών μέσα σε κοντέινερ (containers). Είναι μία από τις πιο διαδεδομένες τεχνολογίες για πακετάρισμα εφαρμογών, ειδικά λόγω της ευελιξίας και της

αποδοτικότητάς του, επιτρέποντας την ανάπτυξη και την παράδοση λογισμικού σε διαφορετικά περιβάλλοντα.

3.5.1 Τι είναι το Docker;

Το Docker επιτρέπει στους προγραμματιστές και τους διαχειριστές συστημάτων να πακετάρουν μια εφαρμογή μαζί με όλες τις εξαρτήσεις της (όπως βιβλιοθήκες, εξωτερικά εργαλεία, runtime) σε ένα μικρό και φορητό κοντέινερ. Αυτό το κοντέινερ μπορεί να εκτελεστεί σε οποιοδήποτε περιβάλλον το οποίο έχει λειτουργικό σύστημα, είτε είναι τοπικός υπολογιστής, είτε server είτε σε cloud περιβάλλον, διασφαλίζοντας ότι η εφαρμογή θα λειτουργεί με τον ίδιο τρόπο παντού.

Κύριες Ικανότητες του Docker:

Κοντέινερ (Containers): Τα κοντέινερ είναι ελαφριά, απομονωμένα περιβάλλοντα που περιέχουν όλα όσα χρειάζονται για να εκτελεστεί μια εφαρμογή. Σε αντίθεση με τις εικονικές μηχανές (Virtual Machines), τα κοντέινερ είναι πολύ πιο ελαφριά, καθώς μοιράζονται τον πυρήνα του λειτουργικού συστήματος και τρέχουν μόνο τα απαραίτητα εργαλεία που χρειάζονται.

Μέσα σε ένα κοντέινερ περιλαμβάνεται ο κώδικας της εφαρμογής, οι βιβλιοθήκες, τα runtime, τα εξωτερικά εργαλεία και ό,τι άλλο απαιτείται για να λειτουργήσει η εφαρμογή.

Ελαφριά και Φορητά (Lightweight and Portable): Τα κοντέινερ είναι ελαφρύτερα από τις παραδοσιακές εικονικές μηχανές, καθώς δεν χρειάζονται ένα ολόκληρο λειτουργικό σύστημα μέσα στο κάθε κοντέινερ. Μπορούν να τρέξουν σχεδόν σε οποιοδήποτε περιβάλλον χωρίς να απαιτούνται αλλαγές στον κώδικα της εφαρμογής, εξασφαλίζοντας έτσι τη φορητότητα της εφαρμογής.

Απομόνωση (Isolation): Κάθε κοντέινερ είναι απομονωμένο από τα υπόλοιπα, πράγμα που σημαίνει ότι τρέχει αυτόνομα και δεν επηρεάζεται από άλλες εφαρμογές ή διαδικασίες στον ίδιο server. Αυτό προσφέρει ασφάλεια και σταθερότητα, καθώς προβλήματα σε ένα κοντέινερ δεν επηρεάζουν τα άλλα.

Συνεπές Περιβάλλον (Consistent Environment): Επειδή η εφαρμογή εκτελείται μέσα στο κοντέινερ με όλες τις εξαρτήσεις της, διασφαλίζεται ότι το περιβάλλον στο οποίο τρέχει θα είναι πάντα το ίδιο, ανεξαρτήτως του πού βρίσκεται (τοπικά, σε server, ή σε cloud). Αυτό λύνει προβλήματα όπως "δουλεύει στον υπολογιστή μου αλλά όχι στον server."

Διανομή μέσω Docker Images: Τα **Docker Images** είναι τα "templates" που περιλαμβάνουν όλα τα απαραίτητα αρχεία και ρυθμίσεις για να τρέξει μια εφαρμογή. Αυτά τα images είναι επαναχρησιμοποιήσιμα και μπορούν να διανεμηθούν εύκολα μέσω του **Docker Hub** ή άλλων τρόπον αποθήκευσης. Ένα Docker image μπορεί να δημιουργηθεί μία φορά και να εκτελεστεί σε οποιοδήποτε περιβάλλον χωρίς αλλαγές.

Αποδοτική Χρήση Πόρων: Σε αντίθεση με τις εικονικές μηχανές, τα κοντέινερ χρησιμοποιούν τους πόρους του συστήματος πιο αποδοτικά, καθώς δεν απαιτούν ολόκληρο λειτουργικό σύστημα για κάθε κοντέινερ.

3.5.2 Γιατί είναι χρήσιμο το Docker;

Στους προγραμματιστές:

- Οι προγραμματιστές μπορούν να διασφαλίσουν ότι η εφαρμογή τους θα τρέχει με συνέπεια σε διαφορετικά περιβάλλοντα ανάπτυξης και παραγωγής.
- Εύκολη διαχείριση εξαρτήσεων: δεν χρειάζεται να ανησυχούν για το αν η βιβλιοθήκη ή το εργαλείο που χρησιμοποιούν είναι εγκατεστημένο στο server.

Στους διαχειριστές Συστημάτων:

- Οι διαχειριστές μπορούν να αναπτύξουν εφαρμογές σε οποιοδήποτε σύστημα πιο εύκολα και γρήγορα, χωρίς να ανησυχούν για το αν είναι ρυθμισμένο σωστά το περιβάλλον.
- Επιτρέπει την ταυτόχρονη διαχείριση πολλαπλών εφαρμογών ή υπηρεσιών στον ίδιο server χωρίς να υπάρχει σύγκρουση μεταξύ τους.

Τρόπος Χρήσης του Docker:

Δημιουργία του Docker Image: Οι προγραμματιστές γράφουν ένα **Dockerfile**, το οποίο καθορίζει πώς θα χτιστεί το Docker image, ποιες βιβλιοθήκες και εξαρτήσεις θα εγκατασταθούν, και ποια εντολή θα εκτελεί το κοντέινερ όταν ξεκινήσει. Με την εντολή docker build, το Docker δημιουργεί το στιγμιότυπο όλου του προγράμματος (docker image) που περιλαμβάνει όλα τα απαραίτητα για την εκτέλεση της εφαρμογής.

Εκτέλεση του Docker Container: Αφού δημιουργηθεί το image, η εντολή docker run εκκινεί το κοντέινερ και τρέχει την εφαρμογή μέσα σε αυτό. Το κοντέινερ είναι απομονωμένο και δεν επηρεάζεται από τις υπόλοιπες διεργασίες στο σύστημα.

Docker Compose: Για πιο σύνθετες εφαρμογές που απαιτούν πολλαπλά κοντέινερ (π.χ. μια εφαρμογή με backend, frontend, και βάση δεδομένων), χρησιμοποιούμε το **Docker Compose**. Αυτό το εργαλείο μας επιτρέπει να ορίσουμε όλες τις υπηρεσίες (containers) σε ένα αρχείο docker-compose.yml(-παράδειγμα στον οδηγό εκτέλεσης 3.6) και να τα εκκινήσουμε όλα μαζί.

Πλεονεκτήματα του Docker:

- **Φορητότητα:** Τα Docker containers μπορούν να τρέξουν παντού με συνέπεια.
- **Απομόνωση:** Κάθε container είναι απομονωμένο, εξασφαλίζοντας ασφάλεια και σταθερότητα.
- **Ευελιξία:** Μπορούμε να τρέχουμε πολλαπλές υπηρεσίες στον ίδιο server χωρίς σύγκρουση.
- **Εξοικονόμηση Πόρων:** Είναι πιο ελαφρύ από τις εικονικές μηχανές, καθώς μοιράζεται τον πυρήνα του λειτουργικού συστήματος.
- **Ταχύτητα:** Η δημιουργία και η εκτέλεση containers είναι γρήγορη, κάνοντας την ανάπτυξη και τη δοκιμή εφαρμογών πιο αποδοτική.

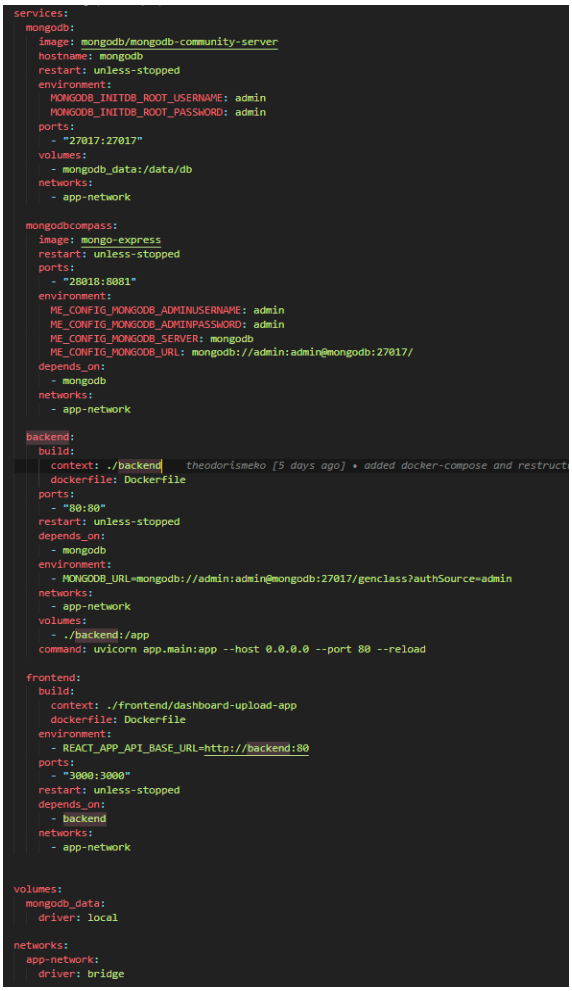
3.6 Εξήγησή και οδηγίες εκτέλεσης εφαρμογής με χρήση Docker εντολών

Το αρχείο Dockerfile για το εμπρός μέρος (frontend) που χρησιμοποιεί το παρακάτω docker-compose.yml.

```
# Build stage
FROM node:20 as build

WORKDIR /app

# Copy package files
COPY package*.json ./

# Install dependencies
RUN npm ci

# Copy the app source code
COPY . .

# Build the app
RUN npm run build

# Production stage
FROM node:20-slim

WORKDIR /app

# Install serve
RUN npm install -g serve

# Copy built assets from build stage    theodorismeko [6 hours ago]
COPY --from=build /app/build ./build

# Expose port
EXPOSE 3000

# Start the app
CMD ["serve", "-s", "build", "-l", "3000"]
```

Το αρχείο Dockerfile για το πίσω μέρος (backend) που χρησιμοποιεί το παρακάτω docker-compose.yml

```
# Use official python:3.10 as the base image
FROM python:3.10

# Set the working directory
WORKDIR /backend

# Copy requirements file
COPY requirements.txt .

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copy the app source code
COPY . .

# Expose port
EXPOSE 80
```



```yaml
services:
  mongodb:
    image: mongodb/mongodb-community-server
    hostname: mongodb
    restart: unless-stopped
    environment:
      MONGODB_INITDB_ROOT_USERNAME: admin
      MONGODB_INITDB_ROOT_PASSWORD: admin
    ports:
      - "27017:27017"
    volumes:
      - mongodb_data:/data/db
    networks:
      - app-network

  mongodbcompass:
    image: mongo-express
    restart: unless-stopped
    ports:
      - "28018:8081"
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: admin
      ME_CONFIG_MONGODB_ADMINPASSWORD: admin
      ME_CONFIG_MONGODB_SERVER: mongodb
      ME_CONFIG_MONGODB_URL: mongodb://admin:admin@mongodb:27017/
    depends_on:
      - mongodb
    networks:
      - app-network

  backend:
    build:
      context: ./backend   theodorismeko [5 days ago] • added docker-compose and restructu
      dockerfile: Dockerfile
    ports:
      - "80:80"
    restart: unless-stopped
    depends_on:
      - mongodb
    environment:
      - MONGODB_URL=mongodb://admin:admin@mongodb:27017/genclass?authSource=admin
    networks:
      - app-network
    volumes:
      - ./backend:/app
    command: uvicorn app.main:app --host 0.0.0.0 --port 80 --reload

  frontend:
    build:
      context: ./frontend/dashboard-upload-app
      dockerfile: Dockerfile
    environment:
      - REACT_APP_API_BASE_URL=http://backend:80
    ports:
      - "3000:3000"
    restart: unless-stopped
    depends_on:
      - backend
    networks:
      - app-network


volumes:
  mongodb_data:
    driver: local

networks:
  app-network:
    driver: bridge
```

Αρχείο docker-compose.yml

```yaml
mongodb:
  image: mongodb/mongodb-community-server
  hostname: mongodb
  restart: unless-stopped
  environment:
    MONGODB_INITDB_ROOT_USERNAME: admin
    MONGODB_INITDB_ROOT_PASSWORD: admin
  ports:
    - "27017:27017"
  volumes:
    - mongodb_data:/data/db
  networks:
    - app-network
```

Η υπηρεσία της MongoDB

Εικόνα (Image): Δηλώνει την εικόνα που θα τραβήξει από το docker hub

Όνομα εξυπηρετητή (Hostname): Είναι το όνομα του κοντέινερ, ώστε οι άλλες υπηρεσίες να συνδεθούν σε αυτό το όνομα

Επανεκκίνηση (Restart): Ξανά ξεκινάει, εκτός αν σταματήσει χειροκίνητά

Περιβάλλον (Environment): Ορισμός των μεταβλητών για την αρχικοποίηση του περιβάλλοντος της βάσης δεδομένων

Πόρτες/Θύρες (Ports): Ορισμός της πόρτας που τρέχει(27017) το κοντέινερ τοπικά και(:) η πόρτα που ακούει(27017)

Χωρητικότητα (Volumes): Χρήση του μόνιμου χώρου για την αποθήκευση δεδομένων στην βάση δεδομένων(MongoDB), ώστε να διατηρούνται τα δεδομένα ακόμη και αν έχει σταματήσει ή αφαιρεθεί το κοντέινερ

Δίκτυο (Network): Ορισμός δικτύου στο οποίο βρίσκεται η υπηρεσία.

Το αντίστοιχο γίνεται και για την υπηρεσία Mongo Express

```yaml
mongodbcompass:
  image: mongo-express
  restart: unless-stopped
  ports:
    - "8081:8081"
  environment:
    ME_CONFIG_MONGODB_ADMINUSERNAME: admin
    ME_CONFIG_MONGODB_ADMINPASSWORD: admin
    ME_CONFIG_MONGODB_SERVER: mongodb
    ME_CONFIG_MONGODB_URL: mongodb://admin:admin@mongodb:27017/
  depends_on:
    - mongodb
  networks:
    - app-network
```

Η υπηρεσία του πίσω-μέρους (Backend)

```yaml
backend:
  build:
    context: ./backend
    dockerfile: Dockerfile
  ports:
    - "80:80"
  restart: unless-stopped
  depends_on:
    - mongodb
  environment:
    - MONGODB_URL=mongodb://admin:admin@mongodb:27017/genclass?authSource=admin
  networks:
    - app-network
  volumes:
    - ./backend:/app
  command: uvicorn app.main:app --host 0.0.0.0 --port 80 --reload
```

Κατασκευή (Build): Δηλώνει το πλαίσιο για την υπηρεσία backend και το Dockerfile που θα πάρει για να δημιουργηθεί

Πόρτες/θύρες (Ports): Ορισμός της πόρτας που τρέχει(80) το κοντέινερ τοπικά και(:) η πόρτα που ακούει(80)

Επανεκκίνηση (Restart): Η επανεκκίνηση που αντιστοιχεί και στις προηγούμενες υπηρεσίες

Εξαρτάται από (Depends on): Εξασφαλίζει ό,τι η υπηρεσία mongodb ξεκινάει πριν από την υπηρεσία backend

Περιβάλλον (Environment): Ορίζεται ο σύνδεσμος ο οποίος χρησιμοποιείται από την εφαρμογή backend, δίνοντας το όνομα χρήστη και ο κωδικός χρήστη

Δίκτυο (Network): Το δίκτυο στο οποίο βρίσκονται όλες οι υπηρεσίες για την επικοινωνία

Χωρητικότητα (Volumes): Αντιστοιχεί τον τοπικό φάκελο ./backend με το φάκελο /app μέσα στο κοντέινερ, επιτρέποντας ενημερώσεις στον κώδικα σε πραγματικό χρόνο

Εντολή (Command): Εκτελεί την εφαρμογή backend χρησιμοποιώντας τον server Uvicorn, δίνοντας πρόσβαση στην διεύθυνση και στην πόρτα.

Η υπηρεσία του εμπρός-μέρους(frontend)

```yaml
frontend:
  build:
    context: ./frontend/dashboard-upload-app
    dockerfile: Dockerfile
  environment:
    - REACT_APP_API_BASE_URL=http://backend:80
  ports:
    - "3000:3000"
  restart: unless-stopped
  depends_on:
    - backend
  networks:
    - app-network
```

Κατασκευή (Build): Όπως και στην υπηρεσία backend, δηλώνει το πλαίσιο στο οποίο κατασκευαστεί η υπηρεσία

Περιβάλλον (Environment): Ορίζει μεταβλητή περιβάλλοντος, η οποία είναι υπεύθυνη για την επικοινωνία με την υπηρεσία πίσω-μέρους(backend)

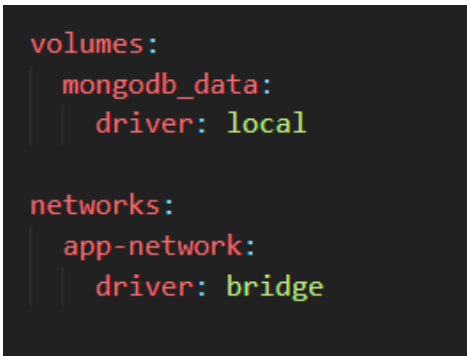
Πόρτες/θύρες (Ports): Ορισμός της πόρτας που τρέχει(3000) το κοντέινερ τοπικά και(:) η πόρτα που ακούει(3000)

Επανεκκίνηση (Restart): Η επανεκκίνηση που αντιστοιχεί και στις προηγούμενες υπηρεσίες

Εξαρτάται από (Depends on): Εξασφαλίζει ό,τι η υπηρεσία backend ξεκινάει πριν από την υπηρεσία backend

Δίκτυο (Network): Το δίκτυο στο οποίο βρίσκονται όλες οι υπηρεσίες για την επικοινωνία

Όγκος των δεδομένων αποθήκευσης & Δίκτυα (Volumes & Networks)



Volumes: Δημιουργεί το μόνιμο χώρο για την αποθήκευση δεδομένων στην βάση δεδομένων(MongoDB), ώστε να διατηρούνται τα δεδομένα.

Networks: Δημιουργεί ένα δίκτυο γέφυρα (bridge), παρέχοντας την δυνατότητα να επικοινωνούν μεταξύ τους οι υπηρεσίες τους χρησιμοποιώντας τα ονόματά των υπηρεσιών.

4. Παρατηρήσεις σχετικά με την εφαρμογή

Η εφαρμογή που αναπτύχθηκε συνδυάζει την ευελιξία της **Γραμματικής Εξέλιξης** με την ισχύ των τεχνολογιών του **FastAPI** και του **React** για την κατασκευή ενός συστήματος που μπορεί να προσφέρει την αποτελεσματική και επεξηγηματική ταξινόμηση δεδομένων. Οι βασικές παρατηρήσεις που μπορούν να γίνουν από την υλοποίηση και τη χρήση της εφαρμογής είναι οι εξής:

Επεξηγηματικότητα των Μοντέλων: Η χρήση της **Γραμματικής Εξέλιξης** για την κατασκευή των κανόνων ταξινόμησης προσφέρει έναν μεγάλο βαθμό επεξηγηματικότητας. Σε αντίθεση με άλλες μεθόδους της μηχανικής μάθησης, όπως τα νευρωνικά δίκτυα, η δημιουργία κανόνων με βάση τη γραμματική επιτρέπει στους χρήστες να κατανοούν τους

αλγορίθμους και τις αποφάσεις που λαμβάνονται από το σύστημα. Αυτό είναι ιδιαίτερα σημαντικό σε τομείς όπου η διαφάνεια στη λήψη αποφάσεων είναι απαραίτητη.

Αποδοτικότητα: Οι κανόνες που κατασκευάζονται μέσω της εφαρμογής έχουν αποδειχθεί αποδοτικοί, ενώ η βελτιστοποίηση μέσω των εξελικτικών διαδικασιών αυξάνει την ακρίβεια και την απόδοση των μοντέλων ταξινόμησης. Η προσέγγιση αυτή διασφαλίζει ότι οι λύσεις που παράγονται είναι όσο το δυνατόν πιο προσαρμοσμένες στις ανάγκες των δεδομένων.

Επεκτασιμότητα: Η υλοποίηση της πλατφόρμας με το **FastAPI** επιτρέπει την εύκολη ενσωμάτωση νέων λειτουργιών και την κλιμάκωση του συστήματος, κάτι που είναι σημαντικό για εφαρμογές που απαιτούν μεγάλης κλίμακας επεξεργασία δεδομένων. Η υποστήριξη για την ασύγχρονη εκτέλεση αιτημάτων διασφαλίζουν ότι η εφαρμογή μπορεί να εξυπηρετεί πολλαπλούς χρήστες ταυτόχρονα χωρίς καθυστερήσεις.

Ευκολία Χρήσης: Η γραφική διεπαφή με την **React** προσφέρει μια φιλική εμπειρία στον χρήστη, επιτρέποντας την εύκολη διαχείριση των δεδομένων και των κανόνων ταξινόμησης. Το περιβάλλον είναι σχεδιασμένο έτσι ώστε να προσφέρει μια σαφή και απλή ροή εργασιών, ενώ η ενσωμάτωση της τεκμηρίωσης του API μέσω του OpenAPI παρέχει στους χρήστες όλες τις πληροφορίες που χρειάζονται για τη χρήση του συστήματος.

Αξιοπιστία και Ακρίβεια: Το σύστημα αξιολόγησης που έχει ενσωματωθεί επιτρέπει στους χρήστες να αξιολογούν την απόδοση των μοντέλων ταξινόμησης με βάση πολλαπλές μετρικές, όπως η ακρίβεια και η ανάκληση. Αυτό τους επιτρέπει να επιλέγουν τα βέλτιστα μοντέλα και να έχουν μεγαλύτερη εμπιστοσύνη στις αποφάσεις που προκύπτουν από τη χρήση της εφαρμογής.

4.1 Συμπεράσματα

Η εφαρμογή που αναπτύχθηκε για τη δημιουργία κανόνων ταξινόμησης μέσω της **Γραμματικής Εξέλιξης** συνδυάζει τις τελευταίες τεχνολογίες στον τομέα των web (ιστού) εφαρμογών με τις εξελικτικές μεθόδους της μηχανικής μάθησης. Η χρήση του **FastAPI**, της **React** και τις MongoDB προσφέρουν ένα αξιόπιστο και αποδοτικό σύστημα που μπορεί να επεκταθεί και να προσαρμοστεί ανάλογα με τις ανάγκες των χρηστών.

Η εφαρμογή αυτή αποτελεί ένα εργαλείο για την ανάλυση δεδομένων, παρέχοντας στους χρήστες τη δυνατότητα να μπορούν να δημιουργούν επεξηγηματικά και αποτελεσματικά μοντέλα ταξινόμησης.

4.2 Μελλοντικές επεκτάσεις

Η παρούσα εφαρμογή μπορεί να βελτιωθεί περαιτέρω και να επεκταθεί σε πολλαπλές κατευθύνσεις, οι οποίες μπορούν να προσφέρουν ακόμα μεγαλύτερη λειτουργικότητα και απόδοση στους χρήστες. Ορισμένες πιθανές επεκτάσεις είναι οι εξής:

1. **Υποστήριξη Περισσότερων Αλγορίθμων Μηχανικής Μάθησης**: Ενσωμάτωση επιπρόσθετων αλγορίθμων, όπως βαθιά νευρωνικά δίκτυα (deep learning) και άλλες εξελικτικές τεχνικές, για την παροχή ακόμα περισσότερων επιλογών στους χρήστες όσον αφορά την ταξινόμηση και την επεξεργασία δεδομένων.
2. **Αυτόματη Προσαρμογή Γραμματικής**: Η προσθήκη ενός μηχανισμού που επιτρέπει την αυτόματη προσαρμογή της γραμματικής, με βάση το εκάστοτε σύνολο δεδομένων, θα μπορούσε να αυξήσει την αποδοτικότητα της Γραμματικής Εξέλιξης και να εξασφαλίσει ότι τα μοντέλα που δημιουργούνται είναι όσο το δυνατόν πιο προσαρμοσμένα στις ανάγκες του χρήστη.
3. **Ενσωμάτωση με Cloud Υπηρεσίες**: Η προσαρμογή της εφαρμογής για χρήση σε περιβάλλοντα **Cloud** θα επέτρεπε την κλιμάκωση της εφαρμογής, δίνοντας στους χρήστες τη δυνατότητα να επεξεργάζονται μεγαλύτερα σύνολα δεδομένων με μεγαλύτερη απόδοση και ταχύτητα, εκμεταλλευόμενοι τα πλεονεκτήματα του υπολογιστικού νέφους.

Τα εργαλεία και οι βιβλιοθήκες που χρησιμοποιήθηκαν για την υλοποίηση:

Visual Studio Code: Visual Studio Code - Code Editing. Redefined

- Visual Studio Code - Wikipedia

Git/GitHub:

- **Git - Wikipedia**
- **History of Git - GeeksforGeeks**
- **GitHub - Wikipedia**
- **What is a GitHub Wiki and How Do You Use it? (freecodecamp.org)**

Docker: Docker: Accelerated Container Application Development

React: Getting Started – React (reactjs.org)

FastAPI: FastAPI (tiangolo.com)

Material UI Components: Components - MUI

Python Subprocess Module: https://docs.python.org/3/library/subprocess.html

Pydantic για FastAPI: Welcome to Pydantic - Pydantic

MongoDB: MongoDB: The Developer Data Platform | MongoDB

- Our Story | MongoDB

Πηγές και Βιβλιογραφία

1. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*.
2. Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*.
3. Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.
4. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer.
5. O'Neill, M., & Ryan, C. (2003). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers.
6. O'Neill, M., & Ryan, C. (2001). *Grammatical Evolution*. IEEE Trans. Evol. Comput., 5, 349–358.
7. Tsoulos, I.G. (2020). *Creating classification rules using grammatical evolution*. International Journal of Computational Intelligence Studies, 9, 161-171.

8. Ortega, A., Sánchez, R., & Alfonseca Moreno, M. (2002). *Automatic composition of music by means of grammatical evolution*. APL '02 Proceedings of the 2002 conference on APL, 148 - 155.

9. O'Neill, M., Brabazon, A., Ryan, C., & Collins, J.J. (2003). *Evolving Market Index Trading Rules Using Grammatical Evolution*. Lecture Notes in Computer Science, 2037, 343-352.

10. Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.

11. Koza, J.R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

12. Banzhaf, W., Nordin, P., Keller, R.E., & Francone, F.D. (1998). *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann.

13. Poli, R., Langdon, W.B., McPhee, N.F., & Koza, J.R. (2008). *A Field Guide to Genetic Programming*. Lulu.com.

14. Διαφάνεια και επεξηγησιμότητα στα συστήματα τεχνητής νοημοσύνης υπό το πρίσμα του γενικού κανονισμού προστασίας δεδομένων (2021-2022 Πανεπιστήμιο Πειραιά(UniPi) - Μεταπτυχιακή εργασία Ι. Μαρσέλλου)