Υλοποίηση Μοντέλου KNN

Ραφαήλ Ασλανίδης

Τελικό Project ΠΛΗΠΡΟ – Ελληνικό Ανοιχτό Πανεπιστήμιο – 2025

Εισαγωγή

Ο αλγόριθμος KNN (K-Nearest Neighbors) είναι ένας αλγόριθμος μηχανικής μάθησης που χρησιμοποιείται τόσο για ταξινόμηση (classification) όσο και για παλινδρόμηση (regression). Βασίζεται στην υπόθεση ότι δεδομένα με παρόμοια χαρακτηριστικά βρίσκονται γεωμετρικά κοντά το ένα στο άλλο σε έναν πολυδιάστατο χώρο χαρακτηριστικών και η απαόδοση του μοντέλου εξαρτάται από την κατάλληλη επιλογή του της παραμέτρου K, όπου K είναι ο αριθμός των γειτόνων που θα ληφθούν υπόψη για την πρόβλεψη μετά την ταξινόμηση των δεδομένων με βάση την απόσταση τους από το δεδομένο όπου θέλουμε να κάνουμε πρόβλεψη.

Στην παρούσα εργασία υλοποιείται ένας αλγόριθμος KNN με χρήση της τη βιβλιοθήκη scikit-learn και χρησιμοποιεί την μέθοδο k-fold_cross-validation με διαφορετικούς συνδυασμούς τιμών των (K, k) => (γείτονες, πλήθους fold). Στόχος είναι να εντοπιστεί η τιμή του K που εμφανίζεται συχνότερα ως η βέλτιστη κατά τη διάρκεια των επαναληπτικών πειραμάτων, ώστε να επιλεγεί ως η πλέον κατάλληλη για το τελικό μοντέλο. Θα υπολογίζονται επίσης και οι μετρικές απόδοσης του μοντέλου, όπως accuracy, precision, class-specific accuracy, και class-specific precision.

Ο παραπάνω αλγόριθμος είναι το βασικό μέρος μιας GUI εφαρμογής που αναπτύσσεται στο πλαίσιο του ομαδικού project του μαθήματος ΠΛΗΠΡΟ στο Ελληνικό Ανοιχτό Πανεπιστήμιο για την πρόβλεψη της ανταπόκρισης πελατών σε καμπάνιες marketing με χρήση μηχανικής μάθησης.

Μεθοδολογία και Συνεισφορά

Ποιό συγκεκριμένα, ο αλγόριθμος KNN που υλοποίησα, χρημοποιεί την μέθοδο της ταξινόμησης (classification), δηλαδή, αφού έχει υπολογίσει της αποστάσεις με χρήση της Euclidean απόστασης μεταξύ των δεδομένων, ταξιννομέι τα δεδομένα και επιλέγει τους K πλησιέστερους γείτονες, και στην συνέχεια, κάνει την πρόβλεωη για το δεδομένο που θέλουμε να ταξινομήσουμε, με βάση την πλειοψηφία των κατηγοριών αυτών των γειτόνων.

Η σωστή επιλογή του παραμέτρου k είναι κρίσιμη για την απόδοση του μοντέλου K-NN. Μικρό K οδηγεί σε υπερπροσαρμογή (overfitting), όπου το μοντέλο απομνημονεύει τα δεδομένα αντί να γενικεύει. Μεγάλο K προκαλεί υποπροσαρμογή (underfitting), κάνοντας το μοντέλο πολύ απλό και ανίκανο να αντικρίσει μοτίβα. Η ιδανική τιμή του K ισορροπεί μεταξύ ακρίβειας και γενίκευσης.

Για αυτόν τον λόγο, χρησιμοποιείται η μέθοδος GridSearchCV για να βρεθεί η βέλτιστη τιμή του αριθμού γειτόνων K. Η μέθοδος αυτή εφαρμόζει διασταυρωμένη επικύρωση (cross-validation), χωρίζοντας το σύνολο δεδομένων σε k υποσύνολα (folds). Σε κάθε επανάληψη, το μοντέλο εκπαιδεύεται σε k - 1 folds και επικυρώνεται στο εναπομείναν fold. Αυτή η διαδικασία επαναλαμβάνεται k φορές (μία για κάθε fold ως σύνολο επικύρωσης) και για κάθε υποψήφια τιμή του K.

Ο συνδυασμός διαφορετικών τιμών για τον αριθμό γειτόνων K και για τον αριθμό των folds k μπορεί να υλοποιηθεί με χρήση της `GridSearchCV`, περνώντας:

- Ένα εύρος τιμών για τον αριθμό των γειτόνων μέσω του παραμέτρου `param_grid`, π.χ. `{"classifier__n_neighbors": [1, 3, 5, 7, 9]}`,
- Και διαφορετικές διασταυρωμένες επικυρώσεις μέσω της παραμέτρου `cv`, χρησιμοποιώντας το `StratifiedKFold(n_splits=k)` για κάθε k. Επίσης η μέθοδος αυτή εξασφαλίζει ότι τα splits θα είναι ισορροπημένα (balanced) ως προς τις κλάσεις του target.

Για να εκτελεστεί η διαδικασία για διαφορετικά πλήθη folds, απαιτείται να γίνουν επαναλαμβανόμενες εκτελέσεις του `GridSearchCV`, μία για κάθε τιμή του k (αριθμός splits στο `StratifiedKFold`).

```
for k in fold_range:
    grid_search = GridSearchCV(
        {classifier__n_neighbors: neighbor_range},
        cv=StratifiedKFold(n_splits=c),
        # ...
    )
```

Και στο τέλος, για να βρεθεί η βέλτιστη τιμή του K, μπορούμε να χρησιμοποιήσουμε την μέθοδο `mode()` της βιβλιοθήκης pandas για να αναλύσουμε τα αποτελέσματα που έχουμε συλλέξει (`results_df`) και να επιλέξουμε την τιμή του K που εμφανίζεται συχνότερα.

```
results_df = pd.DataFrame(self.results)
self.best_n_neighbors = results_df["neighbors"].mode().iloc[0]
```

Παρακάτω παρετίθονται κάποια screenshots για την ανάλυση των αποτελεσμάτων και την επιλογή της βέλτιστης τιμής του K:

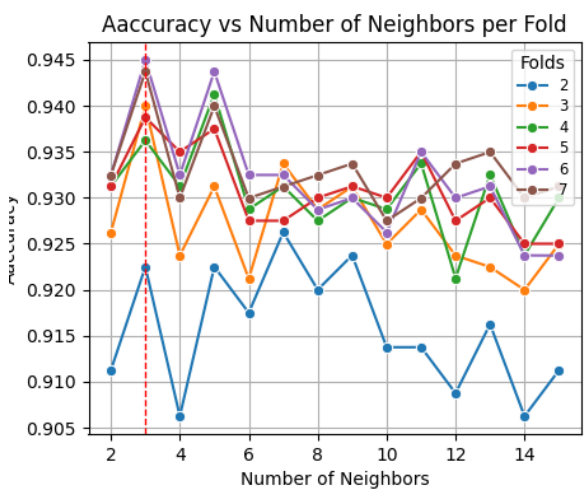

Figure 1: "Απόδοση του μοντέλου για διαφορετικές τιμές K ανά πλήθος fold"

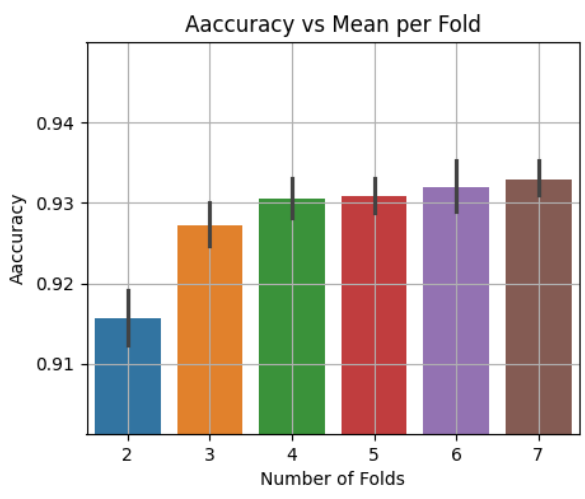

Figure 2: "Μέση απόδοση του μοντέλου ανά fold"

Μέθοδοι της Κλάσης KNN

- `__init__()` Αρχικοποιεί τα attributes της κλάσης KNN.
- `find_best_neighbors()` Για κάθε διαφορετικό πλήθος fold (π.χ. 5 και 10) τρέχουμε τον αλγόριθμο KNN με διαφορετικές τιμές του K και καταγράφουμε την απόδοση του μοντέλου. Στο τέλος, μετατρέπουμε τα αποτελέσματα σε dataframes και επιλέγουμε την τιμή του K που εμφανίζεται συχνότερα ως η βέλτιστη πρώτα ανά run (δηλαδή για κάθε πλήθος fold) και μετά σε όλα τα runs (δηλαδή για όλα τα πλήθη fold) και την αποθηκεύουμε στο attribute `best_n_neighbors`. με τη βοήθεια της μεθόδου mode() της βιβλιοθήκης pandas.
- `feed_data()` Φορτώνει τα δεδομένα και τα μετατρέπει σε μορφή κατάλληλη για KNN. Τα κατηγορικά γίνονται δυαδικά με `OneHotEncoder()`, και όλα τα χαρακτηριστικά κανονικοποιούνται με `StandardScaler()`. Αυτό γίνεται γιατί ο KNN βασίζεται σε αποστάσεις: όλα τα δεδομένα πρέπει να είναι αριθμητικά και στην ίδια κλίμακα για σωστό υπολογισμό.
- `fit()` Εκπαιδεύει το μοντέλο KNN με τα δεδομένα εκπαίδευσης.
- `predict()` Κάνει πρόβλεψη για νέα δεδομένα και αποθηκεύει τα αποτελέσματα σε αρχείο εάν δοθεί το `output_path`.
- `gen_metrics()` Ξανά εκπεραιδεύει το μοντέλο KNN, με το K οπού επιλέχθηκε (είτε αυτόματα είτε χειροκίνητα), με τα δεδομένα εκπαίδευσης και υπολογίζει τις μετρικές απόδοσης του μοντέλου, όπως accuracy, precision, class-specific accuracy, και class-specific precision και με τα δεδομένα επικύρωσης και τις μεθόδους της βιβλιοθήκης scikit-learn (`classification_report()`, `confusinn_matrix()`), τέλος, τα αποθηκεύει σε dicts αλλά και σε ένα string για εύκολη εμφάνιση στο GUI. Επιπλέον, αν χρησιμοποιηθεί η μέθοδος `find_best_neighbors()`, αποθηκεύει αναλυτικα τα αποτελέσματα σε ένα dict και δημιουργεί τα plots με την βοήθεια της κλάσης `Plotter` αποθηκευοντάς τα στον φάκελο `plots/`.
  - Όπου:
    - `accuracy`: $\frac{TP}{TP+TN+FP+FN}$
      Το ποσοστό των σωστών προβλέψεων σε σχέση με το σύνολο των προβλέψεων.
    - `precision`: $\frac{TP}{TP+FP}$
      Το ποσοστό των σωστών θετικών προβλέψεων σε σχέση με το σύνολο των θετικών προβλέψεων.
    - `class_specific_accuracy`: $\frac{TP_i}{TP_i+FN_i}$
      Η ακρίβεια για κάθε κλάση ξεχωριστά (Yes/No) δηλαδή το ποσοστό των σωστών προβλέψεων για κάθε κλάση σε σχέση με το σύνολο των προβλέψεων για αυτή την κλάση.
    - `class_specific_precision`: $\frac{TP_i}{TP_i+FP_i}$
      Η ακρίβεια για κάθε κλάση ξεχωριστά (Yes/No), δηλαδή το ποσοστό των σωστών θετικών προβλέψεων για κάθε κλάση σε σχέση με το σύνολο των θετικών προβλέψεων για αυτή την κλάση.

Επιπλέον, δημιούργησα και μια κλάση `Plotter` για την οπτικοποίηση της απόδοσης του μοντέλου ανά συνδυασμό τιμών K και folds σε περίπτωση που ο χρήστης επιλέξει να χρησιμοποιήσει την μέθοδο `find_best_neighbors()`, ώστε να χρησιμοποιηθούν σε αύτο το paper.

Μέθοδοι της Κλάσης Plotter

Η παρακάτω κλάση δημιουργήθηκε για να διευκολύνει την οπτικοποίηση των αποτελεσμάτων στο ομαδικό paper και περιλαμβάνει τις ακολούθες μεθόδους:

- `__init__()` Αρχικοποιεί τα attributes της κλάσης Plotter.
- `plot_neighbors_vs_metric_per_fold()` Δημιουργεί ένα graph που απεικονίζει την απόδοση του μοντέλου ανά συνδυασμό τιμών K και folds για μια συγκεκριμένη μετρική (π.χ. accuracy).
- `plot_mean_metric_per_fold()` Δημιουργεί ένα graph που απεικονίζει την μέση απόδοση του μοντέλου ανά fold για μια συγκεκριμένη μετρική (π.χ. accuracy).

Επίσης, βοήθησα στη δημιουργία keybind για την έξοδο του GUI της εφαρμογής στον κώδικα του Κώνσταντινου Πιτσαρή, καθώς και με κάποια προβλήματα που προέκυψαν κατά το setup του project στον υπολογιστή της Αντωνίας Κρανίτσας. Ακόμη, ως συντονιστής της ομάδας, βοήθησα στην οργάνωση των εργασιών και στην δημιουργία ενός GitHub repository για την αποθήκευση του κώδικα και των αρχείων της εργασίας.

Ώρες Εργασίας

- Εισαγωγική μελέτη του αλγορίθμου KNN και των τεχνικών cross-validation και grid search. (≈ **1** ώρα)
- Υλοποίηση της κλάσης `KNN` με χρήση της βιβλιοθήκης scikit-learn. (≈ **6** ώρες)
- Δημιουργία της κλάσης `Plotter` για την οπτικοποίηση των αποτελεσμάτων. (≈ **2** ώρες)
- Τεκμηρίωση του κώδικα και των μεθόδων με σχόλια και docstrings. (≈ **1** ώρα)
- Συγγραφή του ατομικού paper. (≈ **5** ώρες)
- Συνεισφορά στη συγγραφή του ομαδικού paper. (≈ **3** ώρες)

Βιβλιογραφία

- Scikit-learn
  - API
  - Pipeline
  - ColumnTransformer
  - KNeighborsClassifier
  - Preprocessing
  - OneHotEncoder
  - StandardScaler
  - TrainTestSplit
  - GridSearchCV
  - StratifiedKFold
  - AccuracyScore
  - PrecisionScore
  - ConfusionMatrix
  - ClassificationReport
- **Pandas**
  - DataFrame
- **Matplotlib**
  - Pyplot
- **Seaborn**
  - Lineplot
  - Barplot
- **LLMs**
  - Χρησιμοποιήθηκε το ChatGPT της OpenAI ως βοηθητικό εργαλείο μάθησης για την κατανόηση εννοιών που δεν μου ήταν ξεκάθαρες κατά την μελέτη του documentation των βιβλιοθηκών. Δεν υπήρξε αντιγραφή κώδικα.
- **Βίντεο**
  - [YouTube] K Nearest Neighbors | Intuitive explained | Machine Learning Basics
  - [YouTube] StatQuest: K-nearest neighbors, Clearly Explained
  - [YouTube] Machine Learning Fundamentals: Cross Validation
  - [YouTube] Hyperparameters Tuning: Grid Search vs Random Search
- **Extra**
  - Typst Documentation
- **Bonus**
  - Terry A. Davis