



Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης  
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών  
Υπολογιστών

---

Ανάπτυξη Συστήματος Προτάσεων για  
την Εξαγωγή Τμημάτων Κώδικα από  
Διαδικτυακές Πηγές

---

Καραγιαννόπουλος Γεώργιος (ΑΕΜ: 7805)

Ανδρέας Λ. Συμεωνίδης

Θεμιστοκλής Διαμαντόπουλος

Τομέας Ηλεκτρονικής και Υπολογιστών

Εργαστήριο Επεξεργασίας Πληροφορίας και Υπολογιστών  
Θεσσαλονίκη 2017



# Περίληψη

Η εξάπλωση του Διαδικτύου οδήγησε στην ευκολότερη εύρεση χρήσιμου κώδικα από αποθήκες λογισμικού, αλλάζοντας ριζικά τον τρόπο με τον οποίο το λογισμικό αναπτύσσεται και συντηρείται. Ωστόσο, παρά την πληθώρα διαθέσιμων επιλογών, ο προγραμματιστής χρειάζεται συχνά να εγκαταλείψει το προγραμματιστικό του περιβάλλον και να καταφύγει σε μηχανές αναζήτησης για την εύρεση χρήσιμου κώδικα. Ως συνέπεια αυτού, μειώνεται η παραγωγικότητα και η συγκέντρωση του. Για την αντιμετώπιση αυτών των δυσκολιών, έχει αναπτυχθεί τελευταία ο τομέας έρευνας των Συστημάτων Προτάσεων στην Τεχνολογία Λογισμικού. Τα συστήματα αυτά δέχονται ερωτήματα από τον προγραμματιστή και, μέσα από διαδικασίες εξόρυξης δεδομένων, επιστρέφουν έτοιμες λύσεις, όπως τμήματα κώδικα.

Στη σύγχρονη βιβλιογραφία, υπάρχουν αρκετά συστήματα που λαμβάνουν ερωτήματα σε κάποια μορφή και επιστρέφουν έτοιμα προς χρήση τμήματα κώδικα. Ωστόσο, σε πολλά από αυτά τα συστήματα χρησιμοποιείται κάποια γλώσσα ερωτημάτων, απαιτώντας έτσι αρκετή προσπάθεια για την σωστή κατασκευή ενός ερωτήματος. Επιπλέον, ο τρόπος παρουσίασης των αποτελεσμάτων είναι συχνά περιοριστικός, καθώς δίνεται στον προγραμματιστή μόνο μια λίστα από τμήματα κώδικα, χωρίς αυτά να ομαδοποιούνται και χωρίς να δίνεται περαιτέρω πληροφορία ως προς την ποιότητα των αποτελεσμάτων.

Στην παρούσα εργασία, σχεδιάζουμε και αναπτύσσουμε ένα νέο σύστημα προτάσεων για την αντιμετώπιση των παραπάνω προκλήσεων. Το σύστημα μας δέχεται ερωτήματα σε φυσική γλώσσα και αναζητεί χρήσιμα τμήματα κώδικα από πληθώρα διαδικτυακών πηγών. Στη συνέχεια, εφαρμόζονται τεχνικές εξόρυξης δεδομένων και μηχανικής μάθησης, με σκοπό την αξιολόγηση και την ομαδοποίηση των τμημάτων κώδικα. Η αξιολόγηση των αποτελεσμάτων γίνεται τόσο ως προς την χρησιμότητα όσο και ως προς την ποιότητα (αναγνωσιμότητα), ενώ η ο τρόπος παρουσίασης επι-

---

τρέπει στον προγραμματιστή να διαχρίνει εύκολα την υλοποίηση που επιθυμεί να χρησιμοποιήσει. Τέλος, αξιολογούμε το σύστημά μας σε ένα σύνολο ερωτημάτων για να επιβεβαιώσουμε την καλή του λειτουργία.

**Λέξεις κλειδιά:** Επαναχρησιμοποίηση Κώδικα, Εξόρυξη Τμημάτων Κώδικα, Συστήματα Προτάσεων στην Τεχνολογία Λογισμικού

# Abstract

## Design and Development of a Recommendation System for Extracting Source Code Snippets from Online Sources

The outspread of the Internet has facilitated the search for useful code from online software repositories, therefore fundamentally changing the way software is developed and maintained. Software engineers focus their effort on combining the best examples and interfaces in order to achieve the optimal solutions. Nevertheless, even with a huge variety of available choices, the developer is often forced to leave his programming environment and resort to search engines in order to find useful code and examples. In the aftermath, his productivity and concentration are reduced. Lately, the research area of *Recommendation Systems in Software Engineering* has been developed in an attempt to confront these challenges. These are systems that receive queries from the developer, and through data mining techniques, aspire to provide ready-to-use solutions, such as reusable code.

In current literature, there are several systems that receive some form of query and return ready-to-use code snippets. Nevertheless, most of these systems use complex query languages, thus requiring significant effort for properly constructing a query. Furthermore, the presentation of the results is often limited, as the developer is only given a list of snippets, without any grouping and without any further information regarding their quality.

In this work, we design and develop a new recommendation system in order to confront the aforementioned challenges. Our system receives queries in natural language and searches for useful snippets in multiple online sources. After that, data mining and machine learning techniques are employed in order to assess and cluster the snippets. The results are assessed both for their usefulness and their quality (readability), while their presentation allows the developer to easily distinguish among

---

the implementation that is most desired. Finally, we evaluate our system in a set of queries to confirm its proper functionality.

*Karagiannopoulos Giorgos*

*gfkaragiannopoulos@gmail.com*

*Thessaloniki, 2017*

**Keywords:** Code Reuse, Snippet Mining, Recommendation Systems in Software Engineering

# Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να εκφράσω την ευγνωμοσύνη μου στα άτομα που συνείσφεραν στο έργο μου αυτό είτε με άμεσο είτε με έμμεσο τρόπο.

Αρχικά, στον κ. Ανδρέα Συμεωνίδη που με εμπιστεύτηκε και μου ανέθεσε αυτό το αντικείμενο της διπλωματικής εργασίας, καθώς επίσης και για την άψογη συνεργασία που είχαμε.

Στη συνέχεια, οφείλω ένα μεγάλο ευχαριστώ στον υποψήφιο Διδάκτορα του τμήματος, Θεμιστοκλή Διαμαντόπουλο, που με την καθοδήγηση του και με τις ιδέες του μου έδινε διαρκώς κίνητρο να αναζητήσω νέα πράγματα έως ότου επιτευχθεί ένα ικανοποιητικό αποτέλεσμα.

Τέλος, θα ήθελα να ευχαριστήσω εκ βάθους καρδίας του γονείς μου, Φίλιππο και Θεοδοσία, που χωρίς τη στήριξη και τη βοήθεια τους δε θα είχα φτάσει ποτέ ως εδώ.

# Δήλωση Πνευματικών Δικαιωμάτων

Copyright (C) Καραγιαννόπουλος Γεώργιος & Εργαστήριο Πληροφορίας και Υπολογιστών, Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης, 2017

Υπογραφή Φοιτητή

# Περιεχόμενα

<b>1 Εισαγωγή</b>	<b>16</b>
1.1 Γενικά . . . . .	16
1.2 Ορισμός του Προβλήματος . . . . .	18
1.3 Σκοπός της Διπλωματικής Εργασίας . . . . .	19
1.4 Δομή του Κειμένου . . . . .	20
<b>2 Εισαγωγή στη Βιβλιογραφία</b>	<b>21</b>
2.1 Γενικά . . . . .	21
2.2 Επαναχρησιμοποίηση Κώδικα . . . . .	21
2.2.1 Τύποι Επαναχρησιμοποίησης Κώδικα . . . . .	22
2.2.2 Παραδείγματα Επαναχρησιμοποίησης Κώδικα . . . . .	22
2.3 Προγραμματισμός Επικεντρωμένος στα Παραδείγματα . . . . .	24
2.3.1 Εφαρμογές Επικεντρωμένου στα Παραδείγματα Προγραμματισμού . . . . .	24
2.4 Συστήματα Προτάσεων στην Τεχνολογία Λογισμικού . . . . .	27
2.5 Σχεδίαση με Βάση Συντακτικά Δένδρα . . . . .	30
2.5.1 Bing Code Search . . . . .	31
2.5.2 Blueprint . . . . .	33
2.5.3 SnipMatch . . . . .	34
2.5.4 DECKARD . . . . .	36
2.5.5 MAPO . . . . .	37
2.5.6 PARSEWeb . . . . .	38
2.5.7 PRIME . . . . .	39
2.5.8 XSnippet . . . . .	41
2.6 Σχεδίαση με Βάση το Σημασιολογικό Περιεχόμενο . . . . .	43

2.6.1	Strathcona . . . . .	43
2.6.2	FrUiT . . . . .	44
2.6.3	CodeBroker . . . . .	45
2.7	Σχεδίαση με Βάση Στατιστικών Μοντέλων . . . . .	47
2.7.1	SLANG . . . . .	47
2.7.2	CodeHint . . . . .	48
2.7.3	Codex . . . . .	49
2.8	Σύνοψη Κεφαλαίου . . . . .	52
<b>3</b>	<b>Περιγραφή του Προβλήματος</b>	<b>53</b>
3.1	Γενικά . . . . .	53
3.2	Ορίζοντας το πρόβλημα . . . . .	54
3.3	Παρόμοια Συστήματα - Τεχνική Ανάλυση . . . . .	55
3.3.1	Bing Code Search . . . . .	55
3.3.2	PARSEWeb . . . . .	58
3.3.3	MAPO . . . . .	59
3.4	Σύνοψη Κεφαλαίου . . . . .	63
<b>4</b>	<b>Codecatch: Ένα Σύστημα Προτάσεων για Τμήματα Κώδικα</b>	<b>65</b>
4.1	Γενικά . . . . .	65
4.2	Δομή του Συστήματος . . . . .	65
4.3	Query Augmenter . . . . .	67
4.4	Downloader . . . . .	68
4.5	Parser . . . . .	70
4.5.1	Τι είναι ο Parser; . . . . .	71
4.5.2	Αναφορά σε Δημοφιλείς Parsers για τη γλώσσα Java . . . . .	71
4.5.3	Εφαρμογή του Parser στο σύστημα μας . . . . .	72
4.6	API Miner . . . . .	74
4.6.1	Η αποθήκη Ανοιχτού Λογισμικού GitHub . . . . .	75
4.6.2	Κατασκευή ενός Λεξιλογίου από APIs . . . . .	76
4.7	Readability Evaluator . . . . .	77
4.7.1	Κατανοώντας τη μετρική της Αναγνωσιμότητας Κώδικα (Code Readability) . . . . .	77

---

4.7.2	Έρευνα για τους παράγοντες που επηρεάζουν την Αναγνωστικότητα . . . . .	78
4.7.3	Το μοντέλο της Αναγνωσιμότητας Κώδικα . . . . .	79
4.7.4	Εκπαίδευση του Μοντέλου . . . . .	80
4.8	Clusterer . . . . .	94
4.8.1	Προεπεξεργασία Δεδομένων . . . . .	95
4.8.2	Επιλογή του Πλήθους των Ομάδων (Clusters) . . . . .	99
4.8.3	Ομαδοποίηση (Clustering) . . . . .	108
4.9	Presenter . . . . .	109
4.9.1	Κατάταξη των Αποτελεσμάτων . . . . .	110
4.9.2	Παρουσίαση των Αποτελεσμάτων . . . . .	111
4.10	Σύνοψη Κεφαλαίου . . . . .	115
<b>5</b>	<b>Αξιολόγηση και Πειράματα</b>	<b>116</b>
5.1	Γενικά . . . . .	116
5.2	Συγκέντρωση Συνόλου Ερωτημάτων . . . . .	117
5.3	Πείραμα 1ο - Ακρίβεια Αποτελεσμάτων . . . . .	118
5.4	Πείραμα 2ο - Μέσο Μήκος Αναζήτησης . . . . .	123
5.5	Πείραμα 3ο - Ποιότητα Ομαδοποίησης . . . . .	125
5.6	Σύνοψη Κεφαλαίου . . . . .	127
<b>6</b>	<b>Συμπεράσματα και Μελλοντική Εργασία</b>	<b>129</b>
6.1	Συμπεράσματα . . . . .	129
6.2	Μελλοντική Εργασία . . . . .	130

# Κατάλογος σχημάτων

2.1	Η διεπαφή πρότασης κώδικα του συστήματος Bing Code Search. . . . .	31
2.2	Αρχιτεκτονική του συστήματος Blueprint. . . . .	33
2.3	To SnipMatch plugin για το IDE του Eclipse. . . . .	35
2.4	Γενική όψη του τρόπου αναζήτησης τμημάτων κώδικα με το SnipMatch.	36
2.5	Αρχιτεκτονική του συστήματος DECKARD. . . . .	37
2.6	Εποπτική παρουσίαση του τρόπου λειτουργίας του PARSEWeb. . . . .	39
2.7	Partial Temporal Specifications αποκτημένες από τέσσερα διαφορετικά code snippets που χρησιμοποιούν ένα FTP client. . . . .	40
2.8	Συνενώνοντας τις προδιαγραφές. . . . .	41
2.9	Αρχιτεκτονική του συστήματος XSnippet. . . . .	42
2.10	Στιγμιότυπο από την οθόνη λειτουργίας του συστήματος FrUiT. . . . .	45
2.11	Αρχιτεκτονική του συστήματος SLANG. . . . .	48
2.12	To Codex εξετάζει projects ανοιχτού κώδικα για να δημιουργήσει διεπαφές που ενσωματώνουν αναδυόμενες προγραμματιστικές τεχνικές.	51
2.13	To Codex επιδεικνύει μη συνηθισμένο κώδικα σε ένα snippet (κίτρινη υπογράμμιση) και εμφανίζει προειδοποιητικό μήνυμα στο κάτω μέρος.	51
3.1	Σχηματική αναπαράσταση της αρχιτεκτονικής στο σύστημα MAPO. .	60
4.1	Η δομή του συστήματος μας (Codecatch). . . . .	66
4.2	To scraping αποτελεί μία από τις πιο συνηθισμένες τεχνικές συλλογής δεδομένων στα συστήματα ανάκτησης πληροφορίας. . . . .	69
4.3	Η δομή ενός τυπικού parser. . . . .	71
4.4	Η αποθήκη ανοικτού λογισμικού GitHub, ίσως η μεγαλύτερη και δημοφιλέστερη αποθήκη λογισμικού αυτή τη στιγμή στο Διαδίκτυο. . . .	75

---

4.5 Κατανομή των μέσων βαθμολογιών ως προς την αναγνωσιμότητα για όλα τα τμημάτων κώδικα. . . . .	79
4.6 Η βιβλιοθήκη Scikit-Learn της Python αποτελεί μία από τις πιο ολο- κληρωμένες λύσεις στον τομέα της μηχανικής εκμάθησης. . . . .	81
4.7 Γραφική αναπαράσταση overfitting. . . . .	82
4.8 Εκπαίδευση μοντέλου με k-fold cross-validation και $k=4$ [3]. . . . .	83
4.9 Αποτελέσματα μοντέλου αναγνωσιμότητας που χρησιμοποιεί τον αλ- γόριθμο KNN. . . . .	88
4.10 Αποτελέσματα μοντέλου αναγνωσιμότητας που χρησιμοποιεί τον αλ- γόριθμο Random Forests. . . . .	91
4.11 Αποτελέσματα μοντέλου αναγνωσιμότητας που χρησιμοποιεί τον αλ- γόριθμο AdaBoost. . . . .	94
4.12 Δενδρόγραμμα ιεραρχικής (agglomerative) ταξινόμησης. . . . .	101
4.13 Η μέθοδος του γονάτου (elbow method). . . . .	102
4.14 Ανάλυση διαγράμματος σιλουέτας. . . . .	105
4.15 Η μέθοδος Affinity Propagation. . . . .	107
5.1 Αποτελέσματα Average Precision. . . . .	123
5.2 Αποτελέσματα Mean Average Precision. . . . .	123
5.3 Αποτελέσματα μέσου μήκους αναζήτησης για $n = 5, 10$ . . . . .	126
5.4 Αποτελέσματα μέσου μήκους αναζήτησης για $N = 1, \dots, 10$ . . . . .	126

# Κατάλογος αλγορίθμων

1	Ψευδοκώδικας KNN classification . . . . .	85
2	Ψευδοκώδικας Random Forests classification . . . . .	89
3	Απλοποιημένος ψευδοκώδικας του αλγορίθμου AdaBoost . . . . .	92
4	Ψευδοκώδικας Hierarchical (Agglomerative) Clustering . . . . .	100
5	Ψευδοκώδικας αλγορίθμου K-Means . . . . .	109

# Κατάλογος πινάκων

3.1 Μετρικές του BCS για την ταξινόμηση των αποτελεσμάτων. . . . .	56
4.1 Λέξεις-κλειδιά που χρησιμοποιεί ο Augmenter στην προσαύξηση του ερωτήματος του χρήστη. . . . .	68
4.2 Το σύνολο των χαρακτηριστικών που λαμβάνονται υπόψιν για τον υπολογισμό της αναγνωσιμότητας. . . . .	80
4.3 Μορφή πίνακα σύγχυσης (confusion matrix) σε δυαδική ταξινόμηση (binary classification) . . . . .	84
4.4 Αποτελέσματα μετρικών ταξινόμησης για το KNN μοντέλο. . . . .	87
4.5 Αποτελέσματα μετρικών ταξινόμησης για το Random Forests μοντέλο. . . . .	92
4.6 Αποτελέσματα μετρικών ταξινόμησης για το AdaBoost μοντέλο. . . . .	93
4.7 Αναπαράσταση των τμημάτων κώδικα με την προσέγγιση Bag of Words. . . . .	97
5.1 Το σύνολο των ερωτημάτων που χρησιμοποιείται στην αξιολόγηση του συστήματος. . . . .	118
5.2 Τα δέκα πρώτα αποτελέσματα ενός ερωτήματος. . . . .	119
5.3 Παράδειγμα εύρεσης μήκους αναζήτησης. . . . .	125
5.4 Αποτελέσματα των μετρικών Silhouette Coefficient και CoPhenetic Correlation Coefficient. . . . .	127

# Λίστα ακρωνύμιων

- API** Application Programming Interface. 31, 40–43, 47, 49, 50, 57, 60–66, 68, 80, 123–125, 127, 133
- AST** Abstract Syntax Tree. 39, 42, 53, 61, 63
- BCS** Bing Code Search. 34–36, 58
- CPCC** CoPhenetic Correlation Coefficient. 129, 130
- CSV** Comma-Separated Values. 123
- DAG** Directed Acyclic Graph. 42, 44
- I/O** Input/Output. 51
- IDE** Intergrated Development Environment. 29, 30, 34, 39, 41, 51, 57
- JSON** JavaScript Object Notation. 72, 76
- LSA** Latent Semantic Analysis. 49
- MIS** Method Invocation Sequences. 41, 42
- MRR** Mean Reciprocal error Rank. 35
- MSL** Mean Search Length. 13, 127–129
- NDCG** Normalized Discounted Cumulative Gain. 35
- PTS** Partial Temporal Specification. 43, 44

**REPL** Read-Evaluation-Print Loop. 28

**RSSE** Recommendation Systems in Software Engineering. 21, 22, 30–33, 36, 41, 44, 49, 55, 56, 58, 62, 66, 132

**SL** Search Length. 126, 128

**SLM** Statistical Language Model. 50, 51

**SSE** Sum of Squared Errors. 111

**TF-IDF** Term Frequency - Inverse Document Frequency. 49

**UI** User Interface. 28



# Κεφάλαιο 1

## Εισαγωγή

### 1.1 Γενικά

Η ιστορία της επιστήμης των υπολογιστών, της τεχνολογίας λογισμικού, και του Διαδικτύου είναι πλούσια, συναρπαστική, και κρύβει ιδιαίτερες εκπλήξεις για κάποιον που δεν έχει ασχοληθεί με το αντικείμενο. Ξεκινάει σε μία εποχή όταν τα προγράμματα υπολογιστών ήταν κυριολεκτικά εντολές για τον χειρισμό φυσικών συσκευών και εμπεριέχει αρκετά σημεία καμπής που οδήγησαν πρώτα στην εμπορικοποίηση και τελικά στον καταναλωτισμό της τεχνολογίας των υπολογιστών.

Πίσω στα τέλη της δεκαετίας του '50 και στις αρχές του '60, οι προγραμματιστές δεν αλληλεπιδρούσαν απευθείας με τις υπολογιστικές συσκευές. Παρέδιδαν τα προγράμματα τους σε τεχνικούς και λάμβαναν τα αποτελέσματα ώρες ή ακόμη και μέρες αργότερα όταν οι τελευταίοι είχαν ολοκληρώσει την εκτέλεση τους. Για το λόγο αυτό, αρχικά η χρήση των υπολογιστών προσανατολίζονταν προς μαθηματικούς υπολογισμούς, οι οποίοι δεν απαιτούσαν άμεση αλληλεπίδραση.

Η πρώτη ευρέως χρησιμοποιούμενη γλώσσα προγραμματισμού, η Fortran<sup>1</sup> αναπτύχθηκε το 1957 από την εταιρία IBM για μαθηματικούς και επιστημονικούς υπολογισμούς. Άλλη μία, η Cobol<sup>2</sup>, κυκλοφόρησε από τον στρατό των Ηνωμένων Πολιτειών το 1962 για εφαρμογές στον τομέα των επιχειρήσεων.

Αμέσως έγινε φανερό ότι ο σχεδιασμός περίπλοκων συστημάτων λογισμικού θα απαιτούσε καλύτερα εργαλεία και προσεγγίσεις, για αυτό οι επιστήμονες του τομέα

<sup>1</sup><https://en.wikipedia.org/wiki/Fortran>

<sup>2</sup><https://en.wikipedia.org/wiki/COBOL>

οργάνωσαν το 1969 ένα συνέδριο ώστε να βρεθεί μία λύση. Το σημείο αυτό είναι όπου μπήκαν οι ρίζες του όρου *"Software Engineering"*. Μέσα στις επόμενες δεκαετίες, υπήρξε τριβή στον τομέα του προγραμματισμού ανάμεσα στην ακαδημαϊκή κοινότητα, η οποία αναζητούσε ιδανικές λύσεις σε προβλήματα μηχανικής, και στη βιομηχανία, που έψαχνε πιο πρακτικές λύσεις για οικονομικούς κυρίως λόγους. Στις αρχές του '70 οι μηχανικοί λογισμικού άρχισαν να υιοθετούν τη φιλοσοφία του διαχωρισμού τεράστιων έργων σε άλλα, μικρότερα κομμάτια, τα οποία επικοινωνούσαν μέσω διεπαφών.

Άλλη μία τεράστια καμπή συνέβη στις αρχές του 1980 με την υιοθέτηση του αντικειμενοστραφή προγραμματισμού<sup>3</sup> (object-oriented programming). Συγκεκριμένα, η νέα αυτή πρακτική επέτρεψε επέτρεψε στους προγραμματιστές να δημιουργούν και να αλληλεπιδρούν με εικονικά αντικείμενα όπως θα το έκαναν με ένα αντικείμενο που έχει φυσική υπόσταση. Έτσι, προέκυψαν γραφικές διεπαφές<sup>4</sup> (Graphical User Interfaces - GUIs), όπως μενού, εικονίδια, και παράθυρα.

Τα χρόνια που ακολούθησαν σημαδεύτηκαν με εκπληκτική αύξηση στην υπολογιστική ισχύ των υπολογιστικών συστημάτων υπακούοντας με ακρίβεια στο Νόμου του Moore<sup>5</sup> (Moore's Law). Αυτή η πρωτοφανής υπολογιστική ισχύς δεν ήταν αποκλειστικά επικερδής στον τομέα του λογισμικού. Όπου πριν οι μηχανικοί λογισμικού έπρεπε να είναι ιδιαίτερα προσεκτικοί στο σχεδιασμό αποδοτικών προγραμμάτων, τα οποία θα μπορούσαν να λειτουργούν με τους περιορισμένους υλικούς πόρους, η τεράστια υπολογιστική ισχύς οδήγησε πολλές φορές σε αρνητικά βήματα όσον αφορά την ποιότητα του κώδικα.

Το 1989 το Διαδίκτυο γεννιέται όταν ο μηχανικός υπολογιστών Tim Berners-Lee, στο CERN της Ελβετίας, έγραψε μία επιστημονική αναφορά για τη σύνδεση εγγράφων με υπερκείμενο (hypertext).

Μία άλλη σημαντική εξέλιξη στη δεκαετία του '90 ήταν η εξέλιξη της ιδέας του ανοιχτού λογισμικού (open-source software). Το γεγονός αυτό είναι ένας από τους κύριους λόγους της έκρηξης στην παραγωγικότητα της τεχνολογίας λογισμικού. Ένας σημαντικός αριθμός από γλώσσες προγραμματισμού, εργαλεία, και συστήματα είναι διαθέσιμα στο ευρύ κοινό επειδή οι δημιουργοί τους αποφάσισαν να

<sup>3</sup>[https://en.wikipedia.org/wiki/Object-oriented\\_programming](https://en.wikipedia.org/wiki/Object-oriented_programming)

<sup>4</sup>[https://en.wikipedia.org/wiki/Graphical\\_user\\_interface](https://en.wikipedia.org/wiki/Graphical_user_interface)

<sup>5</sup>[https://en.wikipedia.org/wiki/Moore's\\_law](https://en.wikipedia.org/wiki/Moore's_law)

υιοθετήσουν τη φιλοσοφία του open-source.

Η πιο πρόσφατη ιστορία είναι γνωστή στους περισσότερους. Για παράδειγμα, πως η άνοδος των εμπορικών υπολογιστών οδήγησε στην ανάπτυξη της υπολογιστικής νέφους<sup>6</sup> (cloud computing), επιτρέποντας σε εφαρμογές να ανανεώνονται και να παρέχουν πρόσβαση σε πραγματικό χρόνο από πολλαπλές συσκευές. Τέλος, κομβικό σημείο ήταν και η εμφάνιση των smartphones και των tablets που οδήγησε την ανάπτυξη mobile λογισμικού σε τεράστια ύψη.

## 1.2 Ορισμός του Προβλήματος

Αν και όπως είδαμε με την εξάπλωση του Διαδικτύου και του ανοιχτού λογισμικού δημιουργήθηκε πληθώρα νέων γλωσσών προγραμματισμού και συστημάτων, η κύρια δυσκολία του μηχανικού λογισμικού παραμένει σχεδόν σταθερή. Η δυσκολία αυτή είναι στη χρήση των σωστών διαδικασιών και εργαλείων για την επίλυση προβλημάτων.

Παρόλο που υπάρχει εύκολη πρόσβαση στα περισσότερα εργαλεία και σε μεγάλες αποθήκες λογισμικού οι προγραμματιστές αφιερώνουν τεράστια ποσά χρόνου στην αναζήτηση των σωστών μεθόδων που πρέπει να χρησιμοποιήσουν. Το μεγαλύτερο ποσοστό χρόνου στη διαδικασία ανάπτυξης λογισμικού αφιερώνεται στη χρήση μηχανών αναζήτησης μέσω των οποίων οι μηχανικοί ευελπιστούν να βρουν λύσεις στα προβλήματα τους. Ωστόσο, αυτές οι μηχανές αναζήτησης είναι γενικού σκοπού με αποτέλεσμα να αδυνατούν να εξυπηρετήσουν στο έπακρο το πολύ εξειδικευμένο αντικείμενο της ανάπτυξης λογισμικού.

Για τους λόγους αυτούς, την τελευταία δεκαετία έχουν αναπτυχθεί τα λεγόμενα *Συστήματα Προτάσεων στην Τεχνολογία Λογισμικού* (*Recommendation Systems in Software Engineering (RSSE)*). Τα συστήματα αυτά αποσκοπούν να λύσουν το πρόβλημα της επιλογής των σωστών διαδικασιών για την επίλυση ενός προγραμματιστικού στόχου. Αυτό το πετυχαίνουν αυτοματοποιώντας τη διαδικασία αναζήτησης και προτείνοντας στο χρήστη τα κατάλληλα παραδείγματα. Έτσι, βοηθούν τον προγραμματιστή να μειώσει το χρόνο και τον κόπο που θα χρειαζόταν να δαπανήσει σε ένα φυλλομετρητή (browser) αναζητώντας παραδείγματα μέσω μίας μηχανής

---

<sup>6</sup>[https://en.wikipedia.org/wiki/Cloud\\_computing](https://en.wikipedia.org/wiki/Cloud_computing)

αναζήτησης.

Ωστόσο, αν και έχουν αναπτυχθεί πολλά από τα εν λόγω συστήματα, τα περισσότερα από αυτά πλέον δεν συντηρούνται ενώ τα υπόλοιπα που λειτουργούν μέχρι και σήμερα παρουσιάζουν αδυναμίες. Ως επί το πλείστον, τα περισσότερα συστήματα απαιτούν την κατανόηση κάποιας γλώσσας ερωτημάτων για τη χρήση τους. Επίσης, πολλά από αυτά αν δεν λάβουν ως είσοδο μία ικανοποιητική ποσότητα πληροφορίας (τύπους μεταβλητών, όνομα διεπαφής που χρησιμοποιείται, το γενικότερο πλαίσιο κώδικα του χρήστη κλπ.) δεν μπορούν να παράγουν χρήσιμα αποτελέσματα. Τα παραπάνω σε συνδυασμό με το μη κατατοπιστικό τρόπο παρουσίασης των αποτελεσμάτων που συναντάται σε πολλά από αυτά, προκαλεί τους προγραμματιστές να μην τα συμπεριλαμβάνουν στην εργαλειοθήκη τους και να προτιμούν τις κλασικές μεθόδους αναζήτησης παραδειγμάτων.

### 1.3 Σκοπός της Διπλωματικής Εργασίας

Σκοπός της παρούσας διπλωματικής εργασίας είναι η συνεισφορά αφενός γενικά στον τομέα της *Τεχνολογίας Λογισμικού*, και αφετέρου στη τεχνολογία των *Συστημάτων Προτάσεων RSSEs*. Για το λόγο αυτό, θα μελετήσουμε τα συστήματα που υπάρχουν στη βιβλιογραφία και στη συνέχεια θα υλοποιήσουμε ένα δικό μας ώστε να επιλύσουμε το πρόβλημα από τη δική μας οπτική γωνία.

Ειδικότερα, θα προσπαθήσουμε να σχεδιάσουμε ένα σύστημα το οποίο θα αυτοματοποιεί ολόκληρη τη διαδικασία που θα ακολουθούσε ένας μηχανικός λογισμικού στην προσπάθεια του να βρει παραδείγματα για τη λύση ενός προβλήματος. Επίσης, πέρα από την αυτοματοποίηση της διαδικασίας θα πρέπει να εγγυηθούμε στο χρήστη παραδείγματα υψηλής ποιότητας μέσω του συστήματος μας. Έτσι, το σύστημα που θα σχεδιάσουμε πρέπει να μπορεί να αναλύσει την πληροφορία που θα βρίσκει στο Διαδίκτυο και να κρίνει αν αυτή είναι κατάλληλη ώστε να προβληθεί στο χρήστη. Ακόμη, η πληροφορία πρέπει να ομαδοποιηθεί κατάλληλα ώστε να είναι εύκολα διαχωρίσιμη, και τέλος η παρουσίαση της να γίνει με τέτοιο τρόπο ώστε ο χρήστης να μπορεί με την ελάχιστη δυνατή προσπάθεια να εντοπίζει την επιθυμητή λύση.

Τέλος, θα πρέπει να αποδείξουμε ότι μέσω του συστήματος μας ο χρήστης θα

μπορέσει να εξοικονομήσει περισσότερους πόρους και χρόνο σε σύγκριση με παραδοσιακούς τρόπους αναζήτησης, όπως είναι οι εμπορικές μηχανές αναζήτησης. Για το λόγο αυτό αφού σχεδιάσουμε και υλοποιήσουμε το σύστημα μας θα το φέρουμε σε σύγκριση με εναλλακτικές λύσεις.

## 1.4 Δομή του Κειμένου

Στο παρόν κεφάλαιο έγινε μία σύντομη εισαγωγή του αναγνώστη στο αντικείμενο της διπλωματικής εργασίας. Η δομή του κειμένου οργανώνεται στα παρακάτω κεφάλαια.

Στο *Κεφάλαιο 2* θα γίνει κατηγοριοποίηση των συστημάτων RSSE και αναφορά στη βιβλιογραφία τους. Επίσης, θα αποσαφηνιστούν οι έννοιες της επαναχρησιμοποίησης λογισμικού και του επικεντρωμένου στα παραδείγματα προγραμματισμού.

Στο *Κεφάλαιο 3* θα περιγράψουμε το πρόβλημα που προσπαθούν να αντιμετωπίσουν τα συστήματα RSSE και θα περάσουμε σε τεχνικές λεπτομέρειες συστημάτων που παρουσιάζουν ομοιότητες με το σύστημα που θα αναπτύξουμε.

Στο *Κεφάλαιο 4* θα περιγράψουμε αναλυτικά την υλοποίηση του συστήματος μας. Μέσα από την περιγραφή θα γίνουν φανερές οι δυσκολίες που μπορεί να συναντήσει κανείς στην κατασκευή ενός τέτοιου συστήματος και πιθανοί τρόποι αντιμετώπισης τους.

Στο *Κεφάλαιο 5* θα αξιολογήσουμε το σύστημα μας συγκρίνοντας το με εναλλακτικούς τρόπους εύρεσης παραδειγμάτων στην ανάπτυξη λογισμικού. Από τη σύγκριση θα παρουσιαστούν αριθμητικά αποτελέσματα και γραφικές παραστάσεις.

Στο *Κεφάλαιο 6* θα εξάγουμε κάποια συμπεράσματα που προκύπτουν από την έρευνα μας πάνω στα συστήματα προτάσεων. Τέλος, θα προτείνουμε πιθανές μελλοντικές επεκτάσεις του συστήματος μας που μπορούν να πραγματοποιηθούν.

# Κεφάλαιο 2

## Εισαγωγή στη Βιβλιογραφία

### 2.1 Γενικά

Στο κεφάλαιο αυτό θα έρθουμε σε πρώτη επαφή με τους γενικούς όρους που αφορούν την παρούσα διπλωματική εργασία και θα αναλύσουμε εν συντομίᾳ τα συστήματα που εκφράζουν την παρούσα τεχνολογική κατάσταση στο χώρο όπου θα δραστηριοποιηθούμε. Έτσι, αρχικά γίνεται αναφορά στο ζήτημα της επαναχρησιμοποίησης κώδικα (code reuse) καθώς και σε πρακτικές εφαρμογές όπου αυτή χρησιμεύει. Στη συνέχεια αποσαφηνίζεται η έννοια του επικεντρωμένου στα παραδείγματα προγραμματισμού (example-centric programming), και παρουσιάζονται οι βασικότεροι τομείς που αυτός βρίσκει εφαρμογή. Τέλος, διευκρινίζεται η έννοια των συστημάτων προτάσεων στην τεχνολογία λογισμικού (Recommendation Systems in Software Engineering – RSSE), και μελετώνται μερικά συστήματα που εκφράζουν την κατάσταση της τεχνολογίας στον τομέα αυτό.

### 2.2 Επαναχρησιμοποίηση Κώδικα

Η επαναχρησιμοποίηση κώδικα έχει εφαρμοστεί από τους πρώτους καιρούς του προγραμματισμού. Οι προγραμματιστές πάντα χρησιμοποιούσαν τμήματα κώδικα, πρότυπα, συναρτήσεις, και διαδικασίες που βρίσκαν από άλλες πηγές. Η επαναχρησιμοποίηση λογισμικού ως αναγνωρισμένος τομέας μελέτης στη τεχνολογία λογισμικού, χρονολογείται πίσω στο 1968 όταν ο Douglas McIlroy, της εταιρίας Bell

Labs, πρότεινε η βιομηχανία λογισμικού να βασιστεί σε επαναχρησιμοποιήσιμες συνιστώσες-εξαρτήματα.

Ο βασικός στόχος της επαναχρησιμοποίησης κώδικα είναι εξοικονόμηση χρόνου και πόρων εκμεταλλευόμενη στοιχεία που έχουν ήδη δημιουργηθεί στη διαδικασία ανάπτυξης λογισμικού [4]. Η βασική ιδέα είναι ότι μέρη ενός προγράμματος, τα οποία γράφτηκαν κάποια στιγμή, μπορούν ή πρέπει να χρησιμοποιηθούν στην κατασκευή άλλων νέων προγραμμάτων.

### 2.2.1 Τύποι Επαναχρησιμοποίησης Κώδικα

Όσον αφορά τα κίνητρα και τους κατευθυντικούς παράγοντες, η επαναχρησιμοποίηση μπορεί να είναι:

- *Καιροσκοπική* – Όταν μία ομάδα ετοιμάζεται να ξεκινήσει ένα project λογισμικού, συνειδητοποιεί ότι υπάρχουν έτοιμα εργαλεία που μπορούν να επαναχρησιμοποιηθούν.
- *Προγραμματισμένη* – Μία ομάδα στρατηγικά σχεδιάζει τα εργαλεία της ώστε να τα επαναχρησιμοποιήσει σε μελλοντικά έργα.

Η επαναχρησιμοποίηση μπορεί κατηγοριοποιηθεί επιπλέον ως:

- *Εσωτερική* – Μία ομάδα επαναχρησιμοποιεί τα δικά της εργαλεία. Αυτό μπορεί να αποτελεί πολιτική της επιχείρησης, καθώς η ομάδα μπορεί να θέλει να έχει τον απόλυτο έλεγχο για ένα εργαλείο κρίσιμο για το έργο.
- *Εξωτερική* – Μία ομάδα επιλέγει να επαναχρησιμοποιήσει αδειοδοτημένο λογισμικό από τρίτους. Η εξαγορά της άδειας συνήθως κοστίζει στην ομάδα 1 έως 20 % από αυτό που θα κόστιζε η εξολοκλήρου ανάπτυξη όλων των εργαλείων από την αρχή [5]. Η ομάδα επίσης πρέπει να λάβει υπόψη της και το χρόνο που χρειάζεται για να βρει, να μάθει, και να ενσωματώσει ένα εργαλείο.

### 2.2.2 Παραδείγματα Επαναχρησιμοποίησης Κώδικα

Τα παραδείγματα στα οποία ενδείκνυται η επαναχρησιμοποίηση κώδικα είναι πολλά και καλύπτουν μεγάλο εύρος εφαρμογών. Παρακάτω αναλύουμε εν συντομίᾳ μερικές περιπτώσεις.

*Βιβλιοθήκες Λογισμικού (Software Libraries).* Ένα πολύ συνηθισμένο παράδειγμα επαναχρησιμοποίησης κώδικα είναι η τεχνική χρησιμοποίησης μιας βιβλιοθήκης λογισμικού. Πολλές συνηθισμένες λειτουργίες, όπως η μετατροπή πληροφορίας αναμεταξύ γνωστών μορφών, η πρόσβαση σε εξωτερικό αποθηκευτικό χώρο, η διασύνδεση με εξωτερικά προγράμματα, ή ο χειρισμός πληροφορίας (αριθμοί, λέξεις, ονόματα, τοποθεσίες, ημερομηνίες, κλπ.) με συνήθεις τρόπους, χρειάζονται σε πολλά διαφορετικά προγράμματα. Οι συγγραφείς νέων προγραμμάτων μπορούν να χρησιμοποιούν τον κώδικα που υπάρχει σε βιβλιοθήκες λογισμικού για να πραγματοποιήσουν αυτές τις λειτουργίες, αντί να πρέπει να «ανακαλύψουν ξανά τον τροχό», γράφοντας καινούργιο κώδικα απευθείας μέσα στο πρόγραμμα. Οι βιβλιοθήκες λογισμικού συχνά έχουν το προνόμιο ότι είναι καλά υλοποιημένες και δοκιμασμένες. Τα μειονεκτήματα τους περιλαμβάνουν την αδυναμία τροποποιήσης λεπτομερειών, οι οποίες μπορούν να επηρεάσουν την απόδοση ή την επιθυμητή έξοδο, καθώς και ο χρόνος και το κόστος που απαιτείται για την απόκτηση, εκμάθηση, και τροποποίηση τους [6].

*Σχεδιαστικά Motίβα (Design Patterns).* Ένα design pattern είναι μία γενική λύση σε ένα επαναλαμβανόμενο πρόβλημα. Τα σχεδιαστικά μοτίβα είναι περισσότερο εννοιολογικά παρά απτά, και μπορούν να τροποποιηθούν για να εξυπηρετήσουν την ακριβή ανάγκη που παρουσιάζεται. Ωστόσο, αφηρημένες κλάσεις (abstract classes) και διεπαφές (interfaces) μπορούν να επαναχρησιμοποιηθούν για να υλοποιήσουν συγκεκριμένα μοτίβα.

*Συστήματα (Frameworks).* Οι προγραμματιστές γενικά επαναχρησιμοποιούν μεγάλα κομμάτια λογισμικού μέσω τρίτων εφαρμογών ή άλλων συστημάτων. Τα συστήματα αυτά είναι συνήθως εξειδικευμένα σε συγκεκριμένους τομείς και εφαρμόσιμα μόνο σε οικογένειες εφαρμογών.

*Ασφάλεια Υπολογιστών (Computer Security).* Στην ασφάλεια υπολογιστών η επαναχρησιμοποίηση κώδικα χρησιμοποιείται για την εκμετάλλευση αδυναμιών του λογισμικού [7]. Όταν ένας κακόβουλος χρήστης δεν μπορεί να εισάγει κώδικα (code injection) σε ένα σύστημα με σκοπό την τροποποίηση της ροής ελέγχου του, μπορεί να ανακατευθύνει τη ροή ελέγχου σε ακολουθίες κώδικα που υπάρχουν ήδη στη μνήμη. Παραδείγματα τέτοιων επιθέσεων αποτελούν τα *return-to-libc attacks*<sup>1</sup>, *return-*

<sup>1</sup>[https://en.wikipedia.org/wiki/Return-to-libc\\_attack](https://en.wikipedia.org/wiki/Return-to-libc_attack)

*oriented programming*<sup>2</sup>, και *jump-oriented programming* [8].

*Κομμάτια (Components)*. Ένα component, σε μία αντικειμενοστραφή προσέγγιση, αντιπροσωπεύει ένα σύνολο από συνεργαζόμενες κλάσεις (ή μία μόνο κλάση) και τις διεπαφές (interfaces) του συνόλου αυτού. Οι διεπαφές είναι υπεύθυνες για να καθιστούν εφικτή την επανατοποθέτηση των διάφορων κομματιών.

## 2.3 Προγραμματισμός Επικεντρωμένος στα Παραδείγματα

Όπως αναφέρει και ο Jonathan Edwards του πανεπιστημίου MIT, πολλές από τις παθολογίες του λογισμικού οφείλονται στα υψηλά επίπεδα αφαιρετικότητας (abstractness), σε σύγχριση με άλλους τομείς της σχεδίασης και της μηχανικής γενικότερα [9]. Η αφαιρετική σκέψη είναι δύσκολη, απαιτεί ταλέντο, εξάσκηση, και μεγάλη προσοχή, τα οποία είναι σπάνιοι πόροι. Είναι ευρεία διαπιστωμένο ότι σε πολλούς τομείς ο καλύτερος τρόπος για να μάθει και να καταλάβει κανείς τον αφαιρετικό τρόπο σκέψης αποτελούν τα παραδείγματα. Αναλόγως κατά τη διάρκεια της εξάσκησης, έχουμε την τάση να προσπαθούμε να καταλάβουμε τον κώδικα δουλεύοντας παραδείγματα στο μυαλό μας, τακτική η οποία δεν αποτελεί και πολύ καλή στρατηγική. Εγκεφαλικά τρέχουμε ένα μεταγλωττιστή, ενώ έχουμε έναν υπολογιστή να βρίσκεται μπροστά μας!

### 2.3.1 Εφαρμογές Επικεντρωμένου στα Παραδείγματα Προγραμματισμού

Η επικέντρωση στα παραδείγματα μπορεί να βρει εφαρμογές για ολόκληρο το φάσμα προγραμματιστικών δραστηριοτήτων, και για τους αρχάριους αλλά και για τους πιο έμπειρους προγραμματιστές. Αυτές εξερευνώνται στις ακόλουθες υποενότητες.

<sup>2</sup>[https://en.wikipedia.org/wiki/Return-oriented\\_programming](https://en.wikipedia.org/wiki/Return-oriented_programming)

### 2.3.1.1 Διδασκαλία με Παραδείγματα

Φαίνεται λογικό να περιμένει κανείς ότι ένα example centric περιβάλλον θα ήταν χρήσιμο ως ένα εργαλείο διδασκαλίας. Τα παραδείγματα είναι το επίκεντρο σε πολλές προσεγγίσεις διδασκαλίας, και ως συνέπεια παρουσιάζονται στα βιβλία προγραμματισμού. Το Amazon<sup>3</sup> απαριθμεί δεκάδες τίτλους βιβλίων της μορφής ”*<Language> Programming by Examples*”. Τα προγραμματιστικά περιβάλλοντα που είναι ειδικά φτιαγμένα για διδασκαλία, προσπαθούν να κάνουν την εκτέλεση των παραδειγμάτων όσο το δυνατόν πιο άμεση και εποπτική γίνεται. Το Example Centric Programming εξυπηρετεί άμεσα προς όλες αυτές τις τάσεις.

### 2.3.1.2 Επικεντρωμένη στα Παραδείγματα Αποσφαλμάτωση

Στο Example Centric Programming, η αποσφαλμάτωση (debugging) μεταπίπτει σε ένα ζήτημα επιθεώρησης, τουλάχιστον για τους ελέγχους και οποιοδήποτε πρόβλημα μπορεί να αναπαραχθεί αυτόματα. Αυτό που είναι καινούργιο εδώ είναι η εφαρμογή αυτής της τεχνολογίας σε έναν πρωτοποριακό debugger.

Συνήθως, ένας κοινότυπος debugger χρησιμοποιείται σε διαφορετική κατάσταση από τον editor. Πρώτα συγγράφουμε τον κώδικα στον editor, μετά μεταβαίνουμε στον debugger και χειροκίνητα τρέχουμε τον κώδικα με κάποιες εισόδους. Ο debugger παρουσιάζει ένα ολοκληρωτικά διαφορετικό User Interface (UI) από τον editor. Ο στόχος εδώ είναι να εξαλειφθεί αυτή η ανάγκη για μετάβαση από τον editor στον debugger και αντίστροφα, ενοποιώντας τον debugger και τον editor σε ένα σταθερό UI.

Πέρα από τον παραμερισμό του debugger, η προσέγγιση αυτή παραγκωνίζει και την ανάγκη για ένα βρόχο Ανάγνωσης-Αξιολόγησης-Εκτύπωσης (Read-Evaluation-Print Loop (REPL)). Οι εκφράσεις που πληκτρολογούνται σε ένα REPL είναι τώρα απλώς κομμάτια κώδικα (*code snippets*) σε ένα αρχείο πηγαίου κώδικα, με τα αποτελέσματα τους να εμφανίζονται κατευθείαν σε ένα βοηθητικό παράθυρο και να μην χρειάζεται η εισαγωγή τους στον επίσημο κώδικα. Τα αποτελέσματα ανανεώνονται αυτόματα οποτεδήποτε αλλάζει ο κώδικας.

Η ενοποίηση του debugger με τον editor απλοποιεί τη διαδικασία συγγραφής-ελέγχου, ενώ παράλληλα βοηθά στην καλύτερη συγκέντρωση του προγραμματιστή.

<sup>3</sup><https://www.amazon.co.uk/>

αφού δεν χρειάζεται να αλλάξει από μία κατάσταση στην άλλη ανά τακτά χρονικά διαστήματα.

### 2.3.1.3 Επικεντρωμένος στα Παραδείγματα Έλεγχος

Αν και ο έλεγχος λογισμικού δεν αποτελεί σχετικό ζήτημα για την παρούσα διπλωματική εργασία, αναφέρεται απλώς ότι ο χρήστης μπορεί να αποκτήσει μέσω παραδειγμάτων άμεσο feedback για τον κώδικα του. Ακόμη πιο σημαντικά, τα παραδείγματα σε αυτή την περίπτωση εξαφανίζουν την ανάγκη για γράψιμο ελέγχων εναν-έναν από τον χρήστη, γεγονός πολύ κουραστικό και χρονοβόρο.

### 2.3.1.4 Ανάπτυξη Λογισμικού Οδηγούμενη από Παραδείγματα

Μέχρι αυτό το σημείο η συζήτηση ήταν περί το πώς τα παραδείγματα μπορούν να βοηθήσουν στην ανάγνωση και τον έλεγχο του κώδικα. Είναι ακόμη πιο σημαντικό να εξετάσουμε πως μπορούν να βοηθήσουν στο γράψιμο και στην τροποποίηση του τελευταίου. Το Example-Driven Development ξεκινά με ένα σύνολο καλών επιλεγμένων παραδειγμάτων (τα οποία καθόλου συμπτωματικά αποτελούν επίσης ένα καλό τρόπο να προσδιοριστεί ένα πρόγραμμα, και συχνά καλούνται σενάρια χρήσης). Ο κώδικας στη συνέχεια μπορεί να γραφεί χρησιμοποιώντας αυτά τα παραδείγματα ως οδηγό.

Γενικά, το Example-Driven Development αντιμετωπίζει τη δυσκολία της ανάπτυξης λογισμικού με τη στρατηγική «διαιρεί και βασίλευε» (devide and conquer). Αυτό είναι στο ίδιο πνεύμα με το Test-Driven Development, όπου ο προγραμματισμός οδηγείται από ολοένα και πιο απαιτητικούς ελέγχους. Το Example-Driven Development παρέχει υποστήριξη στο Intergrated Development Environment (IDE) για αυτή τη διαδικασία, βοηθώντας και ενισχύοντας την. Τα παραδείγματα αποτελούν ένα είδος «σκαλωσιάς» για το υπό κατασκευή πρόγραμμα. Οι επικαλύψεις (overrides) είναι ένας τρόπος να «σκηνοθετηθεί» προσωρινά η συμπεριφορά μέσα στα παραδείγματα, χωρίς να πρέπει να παρθούν σχεδιαστικές αποφάσεις για το γενικότερο κώδικα και τη ροή-ελέγχου του προγράμματος. Οι εξομοιώσεις αυτές, βοηθούν το πρόγραμμα να εξελιχθεί διαιρώντας μεγάλα λογικά άλματα αφηρημένης σκέψης, σε μία απλή ακολουθία βοηθούμενων βημάτων.

## 2.4 Συστήματα Προτάσεων στην Τεχνολογία Λογισμικού

Η τεχνολογία λογισμικού αποτελεί μία γνωστικά έντονη δραστηριότητα, η οποία παρουσιάζει πολλές προκλήσεις ως προς την εξερεύνηση της διαθέσιμης πληροφορίας. Η πληροφορία αυτή, βρίσκεται διανεμημένη στον πηγαίο κώδικα και στο ιστορικό αλλαγών του λογισμικού, σε λίστες συζητήσεων και σε forums, στο IDE, και γενικότερα βρίσκεται διάσπαρτη στο Διαδίκτυο. Η τεχνική φύση, το μέγεθος, και η δυναμικότητα των παραπάνω περιοχών πληροφορίας παροτρύνουν την ανάπτυξη μιας ειδικής κατηγορίας εφαρμογών για να υποστηρίξουν τους προγραμματιστές, τα συστήματα προτάσεων στην τεχνολογία λογισμικού (Recommendation Systems in Software Engineering – RSSEs). Μπορεί να δοθεί ο παρακάτω ορισμός για ένα RSSE:

...μία εφαρμογή που παρέχει πληροφορία εκτιμώμενη ως χρήσιμη για ένα έργο ανάπτυξης λογισμικού σε ένα δεδομένο πλαίσιο.

Η αναφορά στην εκτίμηση ξεχωρίζει τα RSSEs από τα κλασσικά εργαλεία αναζήτησης ή τα εργαλεία παραπομπής που συναντώνται στα σύγχρονα προγραμματιστικά περιβάλλοντα. Την ίδια στιγμή, η εκτίμηση δεν ισοδυναμεί απαραίτητα με πρόβλεψη. Τα RSSEs δεν βασίζονται σε ακριβή πρόβλεψη της συμπεριφοράς του προγραμματιστή ή του συστήματος. Η έννοια της χρησιμότητας αναφέρεται σε δύο πτυχές ταυτόχρονα: 1) καινοτομία, επειδή τα RSSEs υποστηρίζουν την ανακάλυψη νέας πληροφορίας, και 2) οικειότητα και ενίσχυση, επειδή τα RSSEs υποστηρίζουν την επιβεβαίωση της υπάρχουσας γνώσης. Τέλος, η αναφορά σε ένα συγκεκριμένο έργο και πλαίσιο διακρίνουν τα RSSEs από τα γενικής χρήσης εργαλεία αναζήτησης.

Ο παραπάνω ορισμός των RSSEs είναι, ωστόσο, αρκετά ευρύς και επιτρέπει μια μεγάλη ποικιλία προτάσεων για τους προγραμματιστές. Συγκεκριμένα, ένας μεγάλος αριθμός αντικειμένων πληροφορίας μπορούν να προταθούν, όπως:

- *Πηγαίος κώδικας μέσα σε ένα project.* Τα συστήματα προτάσεων μπορούν να βοηθήσουν τους προγραμματιστές να πλοηγηθούν στον κώδικα που έχουν ήδη γράψει ώστε να γίνει επανεξέταση και διόρθωση του.
- *Επαναχρησιμοποίηση πηγαίου κώδικα.* Άλλα συστήματα προσπαθούν να βοη-

θήσουν τους χρήστες να ανακαλύψουν τα στοιχεία ενός Application Programming Interface (API), όπως κλάσεις, συναρτήσεις, και scripts που μπορούν να βοηθήσουν στην ολοκλήρωση ενός έργου.

- **Παραδείγματα κώδικα.** Σε μερικές περιπτώσεις, ένας προγραμματιστής μπορεί να γνωρίζει τον πηγαίο κώδικα ή ποια στοιχεία ενός API απαιτούνται για την ολοκλήρωση ενός έργου, αλλά να αγνοεί πως χρησιμοποιούνται κατάλληλα. Ως βοηθητικό συμπλήρωμα στην ανάγνωση του documentation, τα συστήματα προτάσεων μπορούν να παρέχουν παραδείγματα κώδικα που διευκρινίζουν τη χρήση των στοιχείων ενδιαφέροντος.
- **Αναφορές προβλημάτων.** Πολλή γνώση για ένα project λογισμικού μπορεί να εδρεύει στη βάση δεδομένων με αναφερθέντα προβλήματα. Όταν δουλεύεται ένα κομμάτι κώδικα ή γίνονται προσπάθειες επίλυσης ενός προβλήματος, τα συστήματα προτάσεων μπορούν να ανακαλύψουν σχετικές αναφορές προβλημάτων.
- **Εργαλεία, εντολές, και λειτουργίες.** Τα μεγάλα προγραμματιστικά περιβάλλοντα γίνονται ολοένα και πιο σύνθετα, και ο αριθμός των εργαλείων ανοιχτού-κώδικα και των προσθέτων (*plugins*) απεριόριστος. Τα συστήματα προτάσεων μπορούν να βοηθήσουν τους προγραμματιστές και άλλους μηχανικούς λογισμικού προτείνοντας εργαλεία, εντολές, ή άλλες δράσεις που θα μπορούσαν να λύσουν το πρόβλημα ή να αυξήσουν την αποδοτικότητα τους.
- **Άλλα άτομα.** Σε ορισμένες περιπτώσεις τα συστήματα προτάσεων μπορούν επίσης να βοηθήσουν βρίσκοντας το κατάλληλο άτομο για την ανάθεση ενός έργου, ή έναν ειδικό προς επικουνωνία για την επίλυση της ερώτησης.

Παρόλο που δεκάδες RSSEs έχουν αναπτυχθεί για να παρέχουν κάποια από τη λειτουργικότητα που αναλύθηκε παραπάνω, δεν έχει προκύψει κάποια αναφορική αρχιτεκτονική σχεδιασμού τέτοιων συστημάτων μέχρι και σήμερα. Η μεγάλη ποικιλομορφία αρχιτεκτονικών στα RSSEs είναι πιθανόν συνέπεια του γεγονότος ότι τα περισσότερα RSSEs λειτουργούν με μία κυρίαρχη πηγή δεδομένων, και κατά συνέπεια είναι σχεδιασμένα με τέτοιο τρόπο ώστε να ενσωματώνουν στενά την τελευταία. Μολαταύτα, τα κύρια σχεδιαστικά ζητήματα για τα συστήματα προτάσεων

σε γενικές γραμμές συναντώνται και στον τομέα της ανάπτυξης λογισμικού, το καθένα από τα οποία έχει τις δικές του προκλήσεις όπως αναλύεται στα παρακάτω σημεία.

- *Προ επεξεργασία δεδομένων.* Στην τεχνολογία λογισμικού, πολλή προσπάθεια απαιτείται στην προ επεξεργασία των ακατέργαστων δεδομένων για τη μετατροπή τους σε μία επαρκώς ερμηνεύσιμη μορφή. Για παράδειγμα, ο πηγαίος κώδικας πρέπει να αναλυθεί (*parsing*), τα σχόλια του κώδικα να συγκεντρωθούν, το λογισμικό να απεικονιστεί σε εξαρτημένους γράφους κλπ. Αυτή η προσπάθεια συνήθως χρειάζεται επιπρόσθετες εργασίες προεπεξεργασίας όπως ο εντοπισμός ακραίων τιμών (outliers detection) και η αντικατάσταση των τιμών που λείπουν (*missing values*).
- *Συλλογή πληροφοριών του πλαισίου ενός έργου.* Ενώ σε άλλους τομείς, όπως το ηλεκτρονικό εμπόριο, οι προτάσεις είναι ισχυρά εξαρτώμενες από το προφίλ του χρήστη, στην τεχνολογία λογισμικού, είναι συνήθως το έργο το οποίο είναι η κεντρική έννοια σχετιζόμενη με τις προτάσεις. Το γενικό πλαίσιο του έργου είναι η αναπαράσταση όλων των πληροφοριών, σχετικά φυσικά με το έργο, που έχει πρόσβαση το RSSE ώστε να παράγει προτάσεις. Σε πολλές περιπτώσεις, το πλαίσιο του έργου αποτελείται από μία μερική όψη της λύσης του έργου. Για παράδειγμα, ένα τμήμα κώδικα που έχει γράψει ο προγραμματιστής, ένα στοιχείο στον κώδικα που ο χρήστης έχει επιλέξει, ή μια αναφορά προβλήματος που διαβάζει ο χρήστης. Το πλαίσιο μπορεί επίσης να προσδιοριστεί ρητά, όπου στην περίπτωση αυτή ο ορισμός του πλαισίου συγχωνεύεται με αυτόν ενός ερωτήματος (*query*) και αποστέλλεται σε ένα παραδοσιακό σύστημα ανάκτησης πληροφορίας (π.χ. μία μηχανή αναζήτησης). Σε κάθε περίπτωση, η συλλογή πληροφοριών για το πλαίσιο ενός έργου για την παραγωγή συστάσεων περιέχει μιας μορφής παράδοξου. Όσο πιο ακριβείς είναι οι πληροφορίες σχετικά με το έργο που θέλουμε να κάνουμε, τόσο πιο ακριβείς μπορούν να γίνουν και οι προτάσεις, αλλά τόσο πιο απίθανο ο χρήστης να χρειαστεί βοήθεια στην περίπτωση αυτή. Με άλλα λόγια, ένας χρήστης που βρίσκεται σε μεγάλη ανάγκη για καθοδήγηση, μπορεί να μην είναι σε θέση να παρέχει αρκετές πληροφορίες στο σύστημα ώστε να λάβει χρήσιμα αποτελέσματα. Για το λόγο αυτό, τα συστήματα προτάσεων πρέπει να λάβουν υπόψη ότι το πλαίσιο

εργασίας του χρήστη θα είναι γενικά ατελές και θορυβώδες.

- **Παραγωγή προτάσεων.** Άπαξ και γίνει η προ επεξεργασία των δεδομένων και υπάρχουν επαρκείς πληροφορίες για το έργο, οι αλγόριθμοι συστάσεων μπορούν να εκτελεστούν. Η ποικιλία των στρατηγικών στο σημείο αυτό, περιορίζεται μόνο από το χώρο του προβλήματος για το οποίο θέλουμε να βρούμε αποτελέσματα, και τη δημιουργικότητα του σχεδιαστή του συστήματος.
- **Παρουσίαση των αποτελεσμάτων.** Στην απλούστερη της μορφή, η παρουσίαση αποτελεσμάτων ισοδυναμεί με την καταγραφή των αντικειμένων πιθανού ενδιαφέροντος (συναρτήσεις, κλάσεις, παραδείγματα κώδικα, αναφορές γνωστών προβλημάτων κλπ.). Κοντά στο ζήτημα της παρουσίασης, ωστόσο, βρίσκεται και αυτό της εξερεύνησης. Γιατί ένα αντικείμενο προτάθηκε; Η απάντηση στην ερώτηση αυτή είναι συνήθως μία σύνοψη της στρατηγικής συστάσεων («μέση βαθμολογία», «καταναλωτές που αγόρασαν αυτό το αντικείμενο αγόρασαν επίσης...», κλπ.). Στην τεχνολογία λογισμικού, η εννοιολογική απόσταση μεταξύ ενός αλγορίθμου συστάσεων και της περιοχής γνώσεων οικίας προς το χρήστη, είναι πολύ μεγαλύτερη από ότι σε άλλους τομείς. Για παράδειγμα, αν ένα παράδειγμα κώδικα προταθεί στο χρήστη επειδή ταίριάζει μερικά με τον κώδικα πάνω στον οποίο δουλεύει ο χρήστης εκείνη τη στιγμή, πώς μπορεί αυτό το ταίριασμα να συνοψιστεί; Η απουσία μίας γενικής ιδέας, όπως η βαθμολόγηση, σημαίνει ότι για κάθε νέο τύπο συστάσεων, η ερώτηση για το πώς αυτή ερμηνεύεται πρέπει να αναθεωρηθεί.

## 2.5 Σχεδίαση με Βάση Συντακτικά Δένδρα

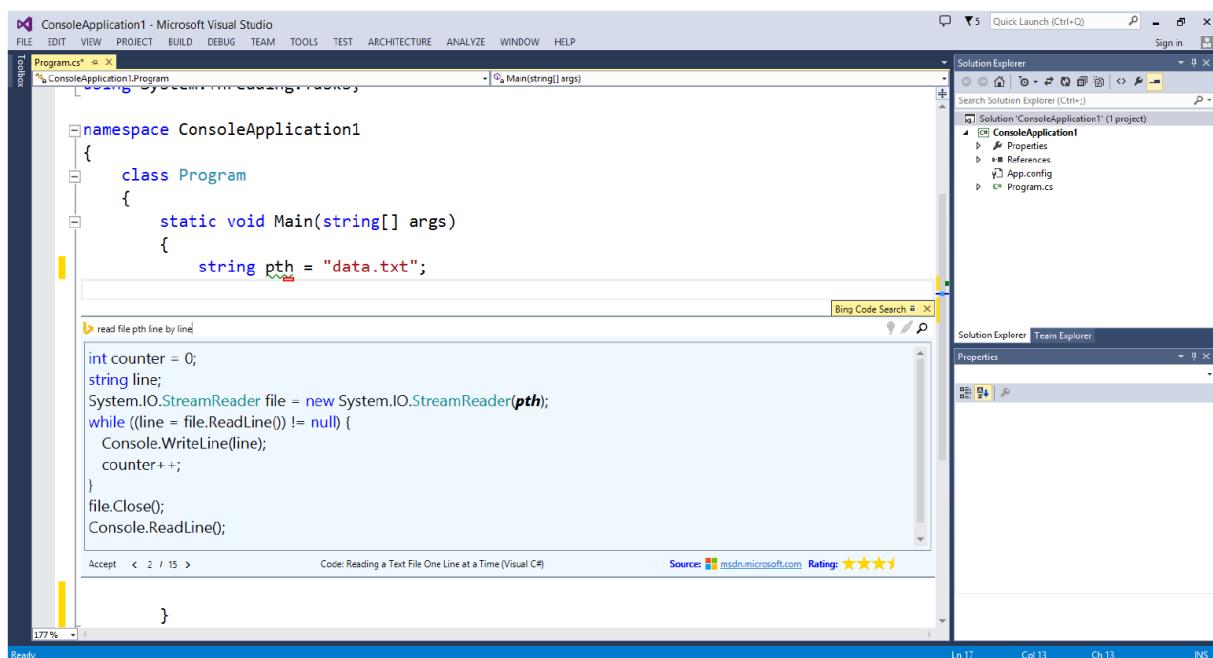
Τα συστήματα που θα παρουσιαστούν σε αυτήν την ενότητα έχουν ως σημαντική ομοιότητα τον τρόπο που εξετάζουν των πηγαίο κώδικα. Συγκεκριμένα, ως βασικό στόχο έχουν την ανάλυση της δομής του σε συντακτικά δένδρα<sup>4</sup>. Μέσω της αποδόμησης αυτής αντλούν πληροφορίες για τις εξαρτήσεις που εμφανίζονται ανάμεσα στις διάφορες κλάσεις, συναρτήσεις, και μεθόδους του κώδικα. Στη συνέχεια, αυτή η πληροφορία μπορεί να χρησιμοποιηθεί για να εντοπιστούν μοτίβα και αλληλεξαρτήσεις. Παρακάτω περιγράφονται μερικά από τα πιο διαδομένα RSSEs αυτής της

<sup>4</sup>[https://en.wikipedia.org/wiki/Abstract\\_syntax\\_tree](https://en.wikipedia.org/wiki/Abstract_syntax_tree)

σχεδίασης.

### 2.5.1 Bing Code Search

To Bing Code Search (BCS) [10], αναπτύχθηκε από μία ερευνητική ομάδα της Microsoft<sup>5</sup> το 2015. Πρόκειται για ένα εργαλείο το οποίο επιτρέπει στον εκάστοτε προγραμματιστή να διατυπώνει ερωτήματα σε φυσική γλώσσα, και να λαμβάνει ως απάντηση τμήματα κώδικα (code snippets) σε γλώσσα C#<sup>6</sup>. Το κύριο χαρακτηριστικό του εργαλείου αυτού είναι η προσαρμογή του επιστρεφόμενου τμήματος κώδικα στον υπόλοιπο κώδικα, δηλαδή πραγματοποιείται και η απαραίτητη μετονομασία μεταβλητών που είναι αναγκαία. Επίσης, οι μελλοντικές προτάσεις του συστήματος βελτιώνονται ανάλογα με τις αλληλεπιδράσεις του χρήστη. Τέλος, πρόκειται για ένα plugin στο IDE VisualStudio<sup>7</sup> της Microsoft.



Σχήμα 2.1: Η διεπαφή πρότασης κώδικα του συστήματος Bing Code Search.

Η αποτελεσματικότητα του BCS, βασίζεται σε ένα σετ από ανεξάρτητα ως προς τη γλώσσα προγραμματισμού χαρακτηριστικά, τα οποία κατατάσσουν τα επιστρεφόμενα τμήματα κώδικα με πιο αποτελεσματικό τρόπο από ότι η μηχανή αναζήτησης Bing<sup>8</sup>. Ακόμη, οφείλεται στον αλγόριθμο για προσαρμογή του επιστρεφόμενου

<sup>5</sup><https://www.microsoft.com>

<sup>6</sup>[https://en.wikipedia.org/wiki/C\\_Sharp\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/C_Sharp_(programming_language))

<sup>7</sup><https://www.visualstudio.com>

<sup>8</sup><https://www.bing.com/>

κώδικα σε αυτόν του προγραμματιστή. Ένα βασικό σενάριο χρήσης του BCS περιγράφεται από τα ακόλουθα βήματα:

1. Ο χρήστης ενεργοποιεί τη λειτουργία του IntelliSense του VisualStudio και διαλέγει την επιλογή “*How do I...*”, η οποία ενεργοποιεί τη γραφική διεπαφή του BCS.
2. Ο χρήστης πληκτρολογεί το ερώτημα του στο παράθυρο αναζήτησης. Ένα παράδειγμα ερωτήματος θα ήταν το “*read file pth line by line*”, όπως φαίνεται και στο Σχήμα 2.1.
3. Αφού πατήσει enter ο χρήστης βρίσκεται ενώπιον μίας καταταγμένης λίστας από τμήματα κώδικα.
4. Το επιλεγμένο τμήμα κώδικα προσαρμόζεται στον κώδικα του χρήστη μέσω μετονομασίας μεταβλητών. Τέλος, η επιλογή του χρήστη καταγράφεται από το σύστημα με στόχο την βελτίωση των μελλοντικών προτάσεων του.

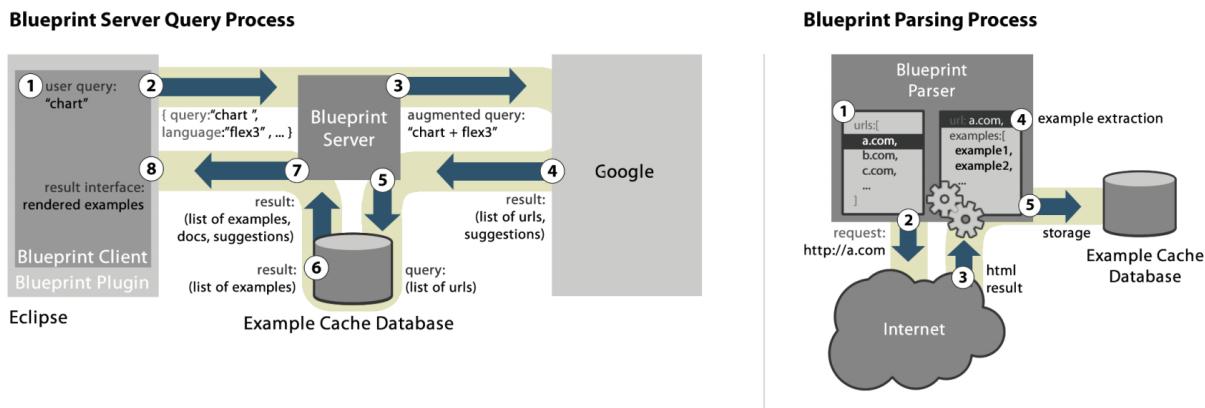
Η επιστρεφόμενη πληροφορία του BCS αξιολογείται με βάση τις ακόλουθες δύο μετρικές:

- *Mean Reciprocal error Rank (MRR)*, η οποία αποτελεί ένδειξη του κατά πόσο το πρώτο στην επιστρεφόμενη λίστα snippet είναι η επιθυμητή λύση στο πρόβλημα του χρήστη. Λαμβάνει τιμές στο κλειστό διάστημα [0, 1], όπου το 0 σημαίνει ότι ποτέ ο χρήστης δεν λαμβάνει την λύση ως πρώτο αποτέλεσμα της λίστας, ενώ το 1 ότι πάντα το πρώτο επιστρεφόμενο snippet είναι και το επιθυμητό από τον χρήστη.
- *Normalized Discounted Cumulative Gain (NDCG)*, η οποία αποτελεί μια αξιολόγηση της επιστρεφόμενης λίστας αποτελεσμάτων εξετάζοντας την ως ακολουθία και κάθε επιστρεφόμενο snippet ως στοιχείο αυτής της ακολουθίας. Έτσι, κάθε στοιχείο αξιολογείται για παράδειγμα σε ένα κλειστό διάστημα [0, 10], με το 0 να συμβολίζει ότι το συγκεκριμένο στοιχείο-τμήμα κώδικα είναι εντελώς άσχετο με το ερώτημα του χρήστη, και με το 10 ότι είναι απολύτως σχετικό. Η τελική κανονικοποίηση πραγματοποιείται διαιρώντας τις παραπάνω μετρήσεις με τις ιδανικές τιμές, δηλαδή τη βέλτιστη δυνατή ακολουθία βαθμολογημένων αποτελεσμάτων.

Τέλος, το BCS, στηρίζεται σε αρχιτεκτονική τύπου *client-server*, με το plugin του VisualStudio να είναι ο client. Το plugin στέλνει ερωτήματα (queries) στον server, ο οποίος στη συνέχεια καλεί τη μηχανή αναζήτησης του Bing, από την οποία εξάγει και αξιολογεί τα τυμάτα κώδικα, εφαρμόζει την αλλαγή μεταβλητών και επιστρέφει το αποτέλεσμα στο plugin.

## 2.5.2 Blueprint

To Blueprint [11] αποτελεί ένα RSSE σύστημα το οποίο προέκυψε από συνεργασία του πανεπιστημίου Stanford<sup>9</sup> και της Adobe<sup>10</sup>, το 2010. Πρόκειται για μία διαδικτυακή διεπαφή αναζήτησης που είναι ενσωματωμένη στο περιβάλλον ανάπτυξης Adobe Flex Builder<sup>11</sup>, το οποίο με τη σειρά του αποτελεί plugin του IDE Eclipse<sup>12</sup>. To Blueprint, το οποίο είναι δομημένο σε μία αρχιτεκτονική client-server, έχει την ικανότητα να δημιουργεί αυτόματα ερωτήματα (queries) όταν ο χρήστης επιλέγει συγκεκριμένα σημεία του κώδικα του. Στη συνέχεια, παρουσιάζει τα αποτελέσματα της αναζήτησης στον χρήστη, ενσωματώνοντάς τα στο περιβάλλον ανάπτυξης (editor), και διατηρώντας το σύνδεσμο από τον οποίο προήλθε ο κώδικας κάθε αποτελέσματος (link).



Σχήμα 2.2: Αρχιτεκτονική του συστήματος Blueprint. Στα αριστερά φαίνεται η διαδικασία εξυπηρέτησης ενός ερωτήματος του χρήστη. Στα δεξιά φαίνεται η διαδικασία της ανάλυσης ιστοσελίδων για την εξαγωγή παραδειγμάτων.

To Blueprint εμπεριέχει στην ουσία ένα plugin (client), το οποίο παρέχει στο χρήστη τη διεπαφή για αναζήτηση και περιήγηση μεταξύ των αποτελεσμάτων, και

<sup>9</sup><https://www.stanford.edu/>

<sup>10</sup><http://www.adobe.com/>

<sup>11</sup><http://www.adobe.com/products/flex.html>

<sup>12</sup><http://www.eclipse.org/downloads/eclipse-packages/>

τον Blueprint server, ο οποίος εκτελεί αναζητήσεις για παραδείγματα κώδικα. Στο Σχήμα 2.2 φαίνεται μία εικονική αναπαράσταση που περιγράφει το σύστημα. Όπως βλέπουμε και στην εικόνα, η επικοινωνία μεταξύ client-server επιτυγχάνεται μέσω του πρωτοκόλλου επικοινωνίας HTTP<sup>13</sup> χρησιμοποιώντας μορφή αναπαράστασης δεδομένων JSON<sup>14</sup>. Η διαδικασία εξαγωγής παραδειγμάτων περιλαμβάνει τον διαχωρισμό του κώδικα από τα υπόλοιπα τμήματα του κάθε αποτελέσματος της αναζήτησης (π.χ. σχόλια σε κώδικα, αρίθμηση γραμμών κλπ.), την εξαίρεση ελαττωματικών παραδειγμάτων (buggy examples), την εξαγωγή κειμένου και την διατήρηση ενός συνδέσμου με την πηγή προέλευσης έτσι ώστε τα παραδείγματα να μπορούν να ανανεωθούν και στον κώδικα του χοήστη σε περίπτωση που γίνει κάποια αλλαγή τους στον ιστότοπο από τον οποίο προήλθαν, όπως για παράδειγμα κάποια διόρθωση.

### 2.5.3 SnipMatch

Το 2012 από μία άλλη συνεργασία, πάλι της Adobe, αλλά τώρα με πανεπιστημιακή ομάδα του Queen's University στον Καναδά, προέκυψε το SnipMatch[12]. Πρόκειται για άλλο ένα plugin στο IDE του Eclipse και κύριος στόχος του είναι να βελτιώσει την ποιότητα των επιστρεφόμενων τμημάτων κώδικα, λαμβάνοντας υπόψιν το ευρύτερο προγραμματιστικό περιεχόμενο του έργου πάνω στο οποίο εργάζεται ο προγραμματιστής. Η δημιουργία του SnipMatch, στηρίχθηκε στην επίγνωση δύο κυρίων αξόνων. Πρώτον, κάνοντας χοήση των κατάλληλων μεταδεδομένων από την ενσωμάτωση ενός τμήματος κώδικα στον ήδη υπάρχον πηγαίο κώδικα ενός project, μπορούν να δημιουργηθούν πολύ πιο ισχυρά εργαλεία για την ανάκτηση και προσαρμογή νέων τμημάτων κώδικα. Δεύτερον, όταν κάποιος δημιουργεί ένα τμήμα κώδικα από το μηδέν, είναι και πρόθυμος να καταβάλει αξιόλογη προσπάθεια ώστε να είναι αυτό κατάλληλα παραμετροποιήσιμο και να γίνεται με πλέον εύκολο τρόπο η ενσωμάτωση του. Οι κύριες συνεισφορές του εργαλείου SnipMatch είναι:

- Ένας αλγόριθμος αναζήτησης για επιμελημένα τμήματα κώδικα που αξιοποιεί το γενικότερο πλαίσιο του υπάρχοντος κώδικα.

<sup>13</sup>[https://en.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)

<sup>14</sup><https://en.wikipedia.org/wiki/JSON>

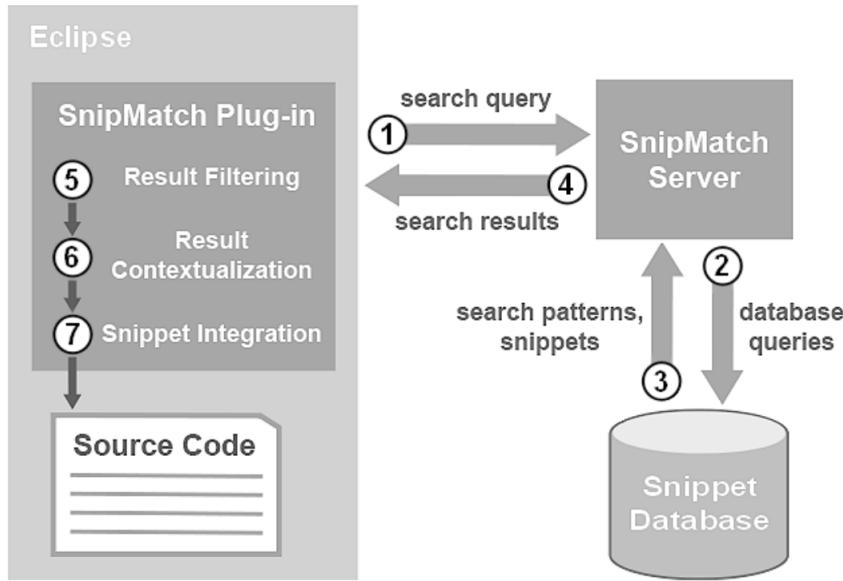
- Μία markup<sup>15</sup> γλώσσα για τον προσδιορισμό των οδηγιών ενσωμάτωσης των τμημάτων κώδικα.
- Επίγνωση σχετικά με το πώς χρησιμοποιούνται τα εργαλεία εύρεσης έτοιμου κώδικα, η οποία προκύπτει από την υλοποίηση και αξιολόγηση του SnipMatch.



Σχήμα 2.3: To SnipMatch plugin για το IDE του Eclipse. Μία συντόμευση από το πληκτρολόγιο ανοίγει το παράθυρο αναζήτησης (1) στη θέση όπου ο προγραμματιστείς έχει τοποθετήσει το δρομέα. Τα αποτελέσματα της αναζήτησης (2) ανανεώνονται καθώς πληκτρολογείται το ερώτημα από τον προγραμματιστή. Το ερώτημα αναζήτησης επίσης επηρεάζει την ενσωμάτωση: η τοπική μεταβλητή playerScores συμπεριλαμβάνεται στο τμήμα κώδικα (2). Ο χρήστης βλέπει μία προεπισκόπηση της ενσωμάτωσης του τμήματος κώδικα στον υπάρχων κώδικα.

Όπως αναφέρθηκε και παραπάνω ένα από τα κύρια χαρακτηριστικά και ταυτόχρονα και πλεονεκτήματα του SnipMatch είναι το γεγονός ότι στις αναζητήσεις των τμημάτων κώδικα, συνυπολογίζεται και ο υπάρχων κώδικας που είναι γραμμένος από τον προγραμματιστή. Ο αλγόριθμος αναζήτησης, ταιριάζει τα ερωτήματα αναζήτησης σε μοτίβα αναζήτησης. Ένα μοτίβο αναζήτησης είναι μια περιγραφή σε μορφή κειμένου του αποτελέσματος ενός τμήματος κώδικα, όπου υπάρχουν διάσπαρτα διάφορα placeholders για τις επιθυμητές παραμέτρους. Για παράδειγμα, όπως φαίνεται στην Εικόνα 2.3, η περιγραφή του συγκεκριμένου μοτίβου είναι “sort <array> in ascending order”, όπου το placeholder <array> αναφέρεται σε μία παράμετρο του snippet.

<sup>15</sup>[https://en.wikipedia.org/wiki/Markup\\_language](https://en.wikipedia.org/wiki/Markup_language)



Σχήμα 2.4: Γενική όψη του τρόπου αναζήτησης τμημάτων κώδικα με το SnipMatch.

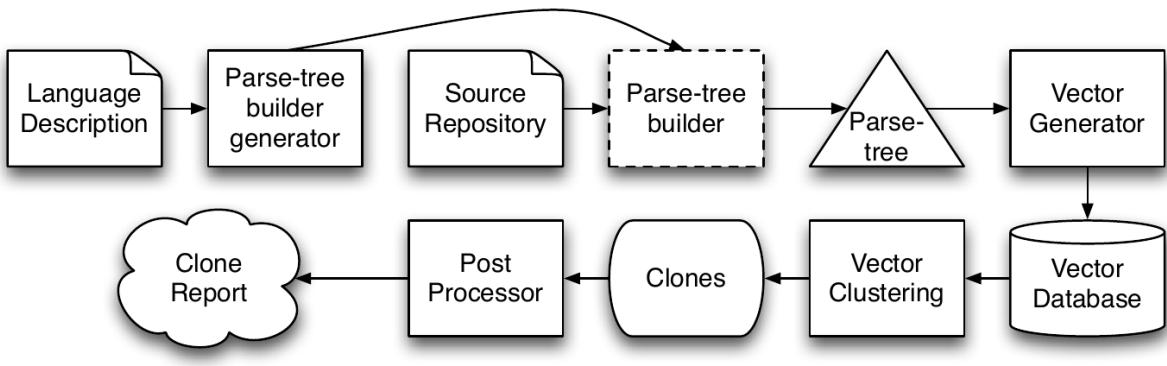
Όταν ένα νέο τμήμα κώδικα καταχωρείται στο SnipMatch, μία απλοϊκή markup γλώσσα μπορεί να χρησιμοποιηθεί ώστε να προσδιορισθούν οι διάφορες θέσεις των παραμέτρων στο μοτίβο αναζήτησης. Η ίδια markup, μπορεί επίσης να χρησιμοποιηθεί για την εισαγωγή εξαρτήσεων και προϋποθέσεων που απαιτούνται, για να εμφανιστεί ένα τμήμα κώδικα στα αποτελέσματα της αναζήτησης. Μόλις τα αποτελέσματα επιστραφούν από τον κεντρικό server, το plugin φιλτράρει και προσαρμόζει τα τμήματα κώδικα με βάση τη markup και το γενικότερο πλαίσιο του κώδικα στο IDE του Eclipse.

#### 2.5.4 DECKARD

Μία ακόμη προσέγγιση στην κατηγορία εργαλείων που λειτουργούν με Abstract Syntax Trees (ASTs) αποτελεί το DECKARD[13], το οποίο δημιουργήθηκε το 2007 από το πανεπιστήμιο της Καλιφόρνια, στο Davis. Ο αλγόριθμος που κρύβεται πίσω από αυτό το εργαλείο στηρίζεται σε ένα καινοτόμο χαρακτηρισμό των υποδένδρων με αριθμητικά διανύσματα στον Ευκλείδειο χώρο, και την αποτελεσματικότητα της ομαδοποίησης (clustering) τα διανυσμάτων αυτών με κριτήριο την Ευκλείδεια απόσταση τους.

Στο σχήμα 2.5 φαίνεται η αρχιτεκτονική υψηλού επιπέδου του DECKARD:

1. Ένας αναλυτής κειμένου (parser) δημιουργείται αυτόματα από μία επίσημη συντακτική γραμματική.



Σχήμα 2.5: Αρχιτεκτονική του συστήματος DECKARD.

2. Ο parser μεταφράζει τα αρχεία πηγαίου κώδικα σε προσπελάσιμα δένδρα (parse trees).
3. Τα parse trees υπόκεινται επεξεργασία ώστε να προκύψει ένα σύνολο από διανύσματα σταθερού μήκους, διατηρώντας έτσι τη συντακτική πληροφορία του κώδικα.
4. Τα διανύσματα ομαδοποιούνται (vector clustering) με κριτήριο τις Ευκλείδειες αποστάσεις τους.
5. Πραγματοποιείται μετα-επεξεργασία για να σχηματιστούν αναφορές κλώνων κώδικα.

### 2.5.5 MAPO

Περί το 2006, όπου οι μηχανές αναζήτησης έχουν γίνει αρκετά ισχυρές και πλέον έχουν εμφανιστεί εξειδικευμένες μηχανές για την αναζήτηση κώδικα, αναπτύχθηκε το MAPO[14], από τους Xie και Pei. Οι κύριοι στόχοι του εργαλείου αυτού με βάση τους οποίους σχεδιάστηκε είναι οι εξής:

- Το εργαλείο θα πρέπει να είναι ικανό να εξάγει πληροφορία για τη χρήση ενός API από ένα αρχείο πηγαίου κώδικα το οποίο μπορεί να μην είναι ικανό να γίνει compile από έναν μεταγλωττιστή. Ο λόγος είναι ότι μία μηχανή αναζήτησης μπορεί να μην επιστρέψει όλα τα αρχεία από τα οποία εξαρτάται το συγκεκριμένο αρχείο.

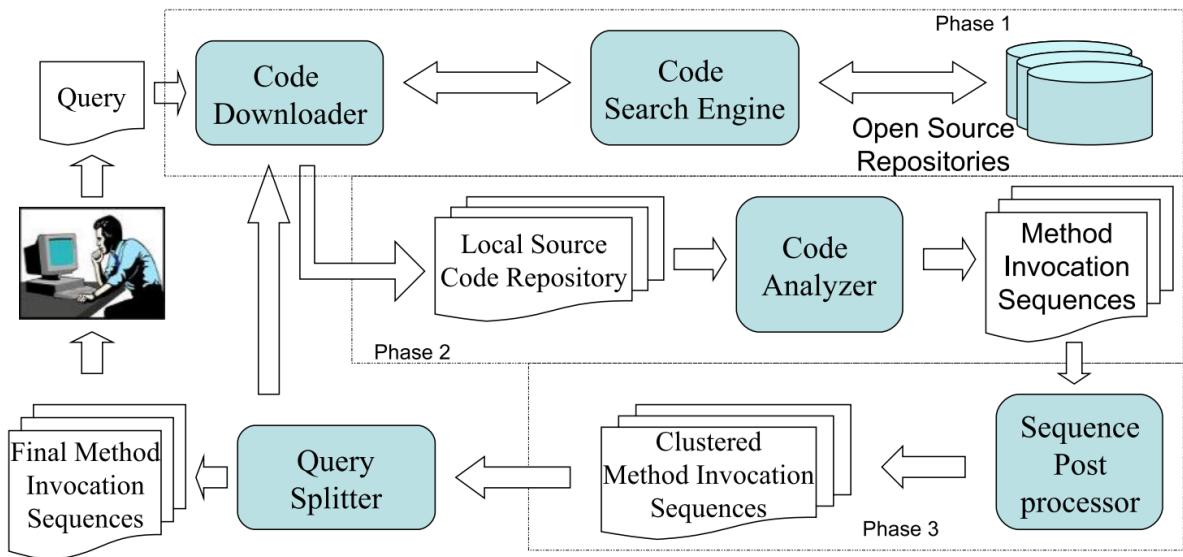
- Το εργαλείο θα πρέπει να μπορεί να εξάγει συχνές χρήσεις ενός API, που περιλαμβάνουν μια αλληλουχία κλήσεων μεθόδων.
- Το εργαλείο θα πρέπει να είναι ικανό να εξορύξει συχνές χρήσεις ενός API, που περιλαμβάνουν κλήσεις μεθόδων από πολλές διαφορετικές κλάσεις, διότι η ρεαλιστική χρήση ενός API συνήθως περιλαμβάνει μεθόδους από πολλαπλές κλάσεις.
- Το εργαλείο θα πρέπει να παράγει μία σύντομη λίστα (μοτίβα) από συχνές χρήσεις ενός API για επιθεώρηση.

Το MAPO αποτελεί ένα πολύ καλό και διαδομένο εργαλείο RSSE που χρησιμοποιείται ακόμη και σήμερα, ωστόσο παρουσιάζει αρκετά μειονεκτήματα. Αρχικά, δεν καταφέρνει να χειριστεί καλά τα αυθαίρετα τμήματα κώδικα, όπως συνήθως συναντώνται στο Ίντερνετ. Επίσης, δεν υπάρχουν τεχνικές ενοποίησης των ξεχωριστών μεθόδων ώστε να αντληθεί μια πληρέστερη εικόνα για τον τρόπο χρήσης του API. Επιπρόσθετα, η ανάλυση του MAPO αγνοεί τη σχέση μεταξύ των ακολουθιών κλήσης μεθόδων και των επιμέρους αντικειμένων, καταλήγοντας σε θόρυβο που αντανακλά ανάμεικτα πρότυπα χρήσης από πολλαπλά αντικείμενα.

## 2.5.6 PARSEWeb

Ένα από τα RSSEs συστήματα που αποτελούν ορόσημα αναπτύχθηκε το 2007 από τους Thummalapenta και Xie. Το σύστημα αυτό έχει πολλές ομοιότητες με το MAPO, και μπορεί πρακτικά να θεωρηθεί ως βελτιωμένη έκδοση του. Ο σκοπός του PARSEWeb [15] είναι να εντοπίσει την πιθανή ακολουθία κλήσης μεθόδων (Method Invocation Sequences (MIS)) που πρέπει να χρησιμοποιήσει ο προγραμματιστής ώστε να πάρει ως αποτέλεσμα το επιθυμητό αντικείμενο στον κώδικα του. Πρόκειται για ένα plugin του Eclipse IDE, και προσφέρεται για την επαναχρησιμοποίηση κώδικα στη γλώσσα Java.

Ο τρόπος λειτουργίας του είναι σχετικά απλός. Αρχικά ο χρήστης πρέπει να σχηματίσει ένα ερώτημα της μορφής “*Source object type* —→ *Destination object type*”. Το PARSEWeb, χρησιμοποιεί στη συνέχεια μια μηχανή αναζήτησης κώδικα (την Google Code Search Engine [16]) για να βρει παραδείγματα κώδικα που ικανοποιούν το παραπάνω ερώτημα, και τα αποθηκεύει σε μία τοπική αποθήκη λογισμι-



Σχήμα 2.6: Εποπτική παρουσίαση του τρόπου λειτουργίας του PARSEWeb.

κού. Στη συνέχεια κάθε παράδειγμα αναλύεται κάνοντας χρήση των ASTs [17] και δημιουργείται ένα κατευθυνόμενος άκυκλος γράφος (Directed Acyclic Graph (DAG)) [18]. Ο γράφος αυτός αναπαριστά τη ροή ελέγχου μεταξύ των μεθόδων. Έπειτα, το PARSEWeb διασχίζει το DAG για να εξάγει το MIS το οποίο παίρνει σαν είσοδο το “*Source object*” και καταλήγει στο “*Destination object*”. Το επιστρεφόμενο στο χρήστη αποτέλεσμα περιλαμβάνει και κάποιες επιπλέον λεπτομέρειες όπως το όνομα του αρχείου, το όνομα της μεθόδου, την κατάταξη του παραδείγματος σε σχέση με τα υπόλοιπα, και τον αριθμό των εμφανίσεων μίας ακολουθίας.

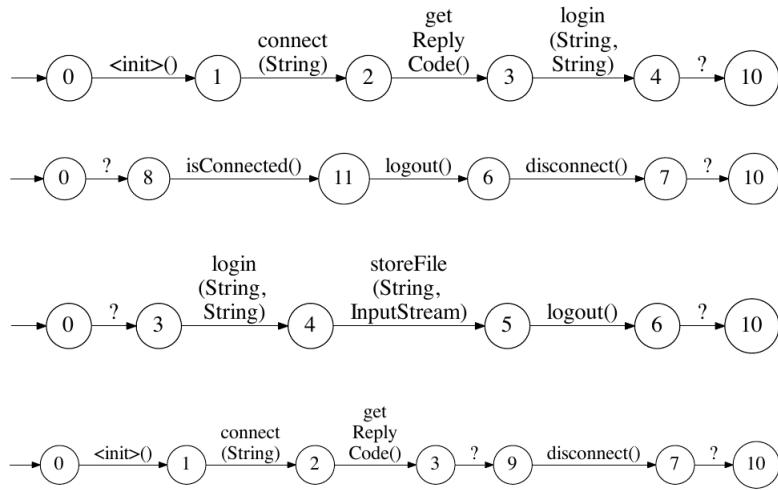
Ένας από τους λόγους που το PARSEWeb υπερτερεί του MAPO, είναι ότι το τελευταίο δεν μπορεί να αντιμετωπίσει ερωτήματα της μορφής “*Source → Destination*”. Έτσι ο προγραμματιστής για να χρησιμοποιήσει το MAPO πρέπει να γνωρίζει ποιο API πρέπει να χρησιμοποιηθεί εκ των προτέρων.

## 2.5.7 PRIME

Το PRIME [19] αποτελεί εργαλείο το οποίο αναπτύχθηκε το 2012 συνεργατικά από τους Mishne, Shoham και Yahav. Για την κατασκευή ενός ευρετηρίου αναζήτησης για ένα συγκεκριμένο API, η ομάδα στοχεύει στη χρήση όλων των διαθέσιμων τιμημάτων κώδικα που μπορούν να βρεθούν σε μηχανές αναζήτησης, ιστότοπους με αναγνωρισμένο υψηλής ποιότητας κώδικα, και άλλες πηγές. Επομένως, σε αντίθεση με τις προσεγγίσεις που χρησιμοποιούν άλλα εργαλεία, δεν γίνεται η υπόθεση ότι

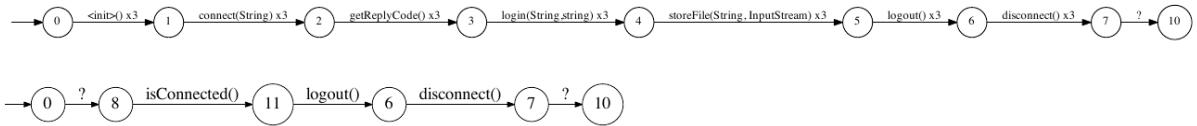
ολόκληρος ο κώδικας από τα projects μέσα στα οποία γίνεται η αναζήτηση της λύσης υπάρχει εξ αρχής, αλλά το PRIME στοχεύει στη σύνδεση ενός μεγάλου αριθμού από snippets, τα οποία προέρχονται από διάφορες πηγές. Μάλιστα όλα αυτά χωρίς να υπάρχει η απαίτηση γίνει μεταγλώττιση και εκτέλεση ολόκληρων projects. Αυτός ο στόχος παρουσιάζει δύο κύριες προκλήσεις: 1) ανάλυση των τμημάτων κώδικα, 2) ενοποίηση της διάσπαρτης πληροφορίας η οποία αποκτήθηκε από τα ξεχωριστά τμήματα κώδικα.

Η αντιμετώπιση αυτών των προκλήσεων εξαρτάται από το επίπεδο της πληροφορίας που χρησιμοποιείται ως βάση για την αναζήτηση. Για το λόγο αυτό, χρησιμοποιείται μία ελαφρώς γενικευμένη έννοια ιδιοτήτων *typestate*, η οποία θεωρείται πως είναι η χρυσή τομή. Σε αντίθεση με άλλες υπάρχουσες προσεγγίσεις, οι οποίες απλώς παρακολουθούν τις ακολουθίες κλήσεων των μεθόδων από την άποψη του τύπου του αποτελέσματος, η ανίχνευση ιδιοτήτων *typestate*, παρέχει μία πιο οικοιβής περιγραφή της χρήσης του API.



Σχήμα 2.7: Partial Temporal Specifications αποκτημένες από τέσσερα διαφορετικά code snippets που χρησιμοποιούν ένα FTP client.

Για την αναπαράσταση της σημασιολογικής πληροφορίας η οποία εξάγεται από τα τμήματα κώδικα, χρησιμοποιείται ο όρος Partial Temporal Specification (PTS). Διαισθητικά, κάθε PTS μπορεί να θεωρηθεί ως ένα κομμάτι ενός παζλ, και ο συνδυασμός αυτών η τοποθέτηση όλων των κομματιών μαζί για απόκτηση μιας πληρέστερης εικόνας. Από τεχνική άποψη, το PRIME αναλύει κάθε snippet για να παράγει ένα αυτόματο, και συνενώνει τα ξεχωριστά αυτόματα μαζί για να δημιουργήσει δύο συνόψεις χρήσης, όπως φαίνεται στο Σχήμα 2.7.



Σχήμα 2.8: Συνενώνοντας τις προδιαγραφές.

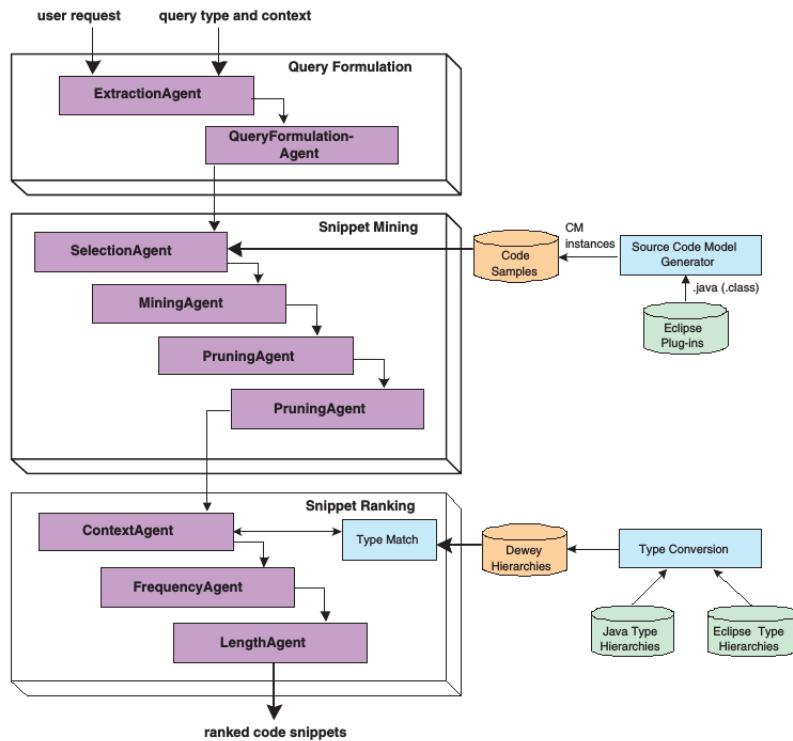
Όπως φαίνεται και στο Σχήμα 2.8, ιδιαίτερο ενδιαφέρον παρουσιάζει η συγχώνευση πολλών PTSs σε ένα. Για το σκοπό αυτό, έχει αναπτυχθεί μία τεχνική που ορίζεται ως “*unknow elimination*”, η οποία με επαναληπτικό τρόπο προσπαθεί να ενώσει όσα PTSs ταιριάζουν. Με τον τρόπο αυτό, οι άγνωστοι κόμβοι σε ένα PTS, αντικαθίστανται με μονοπάτια που βρίσκονται σε άλλα PTSs. Αυτή η διαδικασία επαναλαμβάνεται μέχρις ότου δεν μπορούν να γίνουν περαιτέρω εξαλείψεις άγνωστων κόμβων (ειδική φροντίδα χρειάζεται ώστε να είναι σίγουρος ο τερματισμός της διαδικασίας). Συνεπώς, το PRIME επιτρέπει να γίνεται εκμετάλλευση παραδειγμάτων κώδικα από το Ίντερνετ χωρίς αυτά να μεταγλωτιστούν.

### 2.5.8 XSnippet

Ένα ακόμη εργαλείο RSSE σχεδιάστηκε και αναπτύχθηκε από τους Sahavechaphan και Claypool, το 2006, δίνοντας του το όνομα XSnippet [20]. Είναι και αυτό ένα plugin στο Eclipse, το οποίο επιτρέπει στους προγραμματιστές να στέλνουν ερωτήματα σε μία αποθήκη λογισμικού και να παίρνουν ως αποτέλεσμα προτεινόμενα snippets. Η αρχιτεκτονική του XSnippet (2.9) μπορεί να συνοψιστεί στα παρακάτω στάδια:

1. Αρχικά, έχουμε το σχηματισμό ερωτημάτων, τα οποία χωρίζονται σε εξειδικευμένα και γενικευμένα. Υπάρχει, δηλαδή, η δυνατότητα να δοθεί ως είσοδος είτε ένας μονάχα τύπος αντικειμένου, είτε ολόκληρο τμήμα κώδικα.
2. Ακολουθεί η εξόρυξη κώδικα. Χρησιμοποιούνται διάφορες μέθοδοι και αλγόριθμοι, όπως οι DAG [15], τα συντακτικά δένδρα [14] και μία επεκτεταμένη έκδοση του αλγορίθμου BFSMINE [21] (Breadth-First Search Mining). Στη συνέχεια, φιλτράρονται τα αποτελέσματα απομακρύνοντας τις διπλοεγγραφές από τις σχετικές εγγραφές και μετασχηματίζονται τα μονοπάτια που δημιουργήθηκαν σε κώδικα της γλώσσας Java.
3. Τέλος, βαθμολογούνται τα αποτελέσματα, λαμβάνοντας υπόψη τη μετρική

*context match* (που σχετίζεται όπως δηλώνει και το όνομα της με το βαθμό ομοιότητας του περιεχομένου του κώδικα με το ερώτημα), τη συχνότητα εμφάνισης του αποτελέσματος, καθώς και το μήκος του, χρησιμοποιώντας ευριστική μέθοδο παρόμοια με το γνωστό ελάχιστο μονοπάτι (shortest path).



Σχήμα 2.9: Αρχιτεκτονική του συστήματος XSnippet.

Η διαδικασία μπορεί να συνοψιστεί στο ακόλουθο σενάριο. Ο χρήστης αιτείται τη δημιουργία ενός ερωτήματος, με κατάλληλη επιλογή από το μενού του προγράμματος επεξεργασίας (editor). Τα ερωτήματα, που μπορεί να είναι γενικότερα ή πιο εξειδικευμένα, μεταφέρονται στο τμήμα εξόρυξης κώδικα (snippet mining). Το τμήμα αυτό πραγματοποιεί ανάλυση των γράφων και είναι υπεύθυνο για την επιλογή τμημάτων κώδικα που τηρούν ορισμένες προϋποθέσεις, με βάση το ερώτημα του χρήστη. Τα τμήματα κώδικα (snippets) που προκύπτουν, μεταβιβάζονται στο επόμενο και τελικό στάδιο, όπου θα γίνει η βαθμολόγηση του κώδικα (snippet ranking). Τελικά, τα αποτελέσματα επιστρέφονται στο χρήστη.

## 2.6 Σχεδίαση με Βάση το Σημασιολογικό Περιεχόμενο

Στην κατηγορία αυτή ανήκουν συστήματα, που αντίθετα με τα προηγούμενα που εστίαζαν τη λειτουργία στο δομικό περιεχόμενο του κώδικα, τα οποία προσπαθούν να αντλήσουν πληροφορία από το ευρύτερο σημασιολογικό περιεχόμενο του κώδικα. Αυτή η ιδέα βασίζεται στη λογική ότι πολλά από τα στοιχεία του κώδικα (π.χ. σχόλια, ονόματα κλάσεων, μεθόδων, μεταβλητών) κρύβουν επιπλέον πληροφορία που μπορεί να χρησιμοποιηθεί στα συστήματα προτάσεων.

### 2.6.1 Strathcona

To Strathcona [22] είναι ένα εργαλείο πρότασης κώδικα που λειτουργεί με βάση το σημασιολογικό περιεχόμενο. Δημιουργήθηκε από τη συνεργασία των πανεπιστημίων Calgary και British Columbia, του Καναδά και πρόκειται για ακόμη ένα plugin στο ολοκληρωμένο περιβάλλον του Eclipse και θεωρείται ιδιαίτερα αξιόλογο ακόμη και σήμερα αν και χρονολογείται πίσω στο 2005.

Το εργαλείο Strathcona βοηθάει τους προγραμματιστές να εντοπίσουν παραδείγματα κώδικα που είναι σχετικά με το δικό τους project. Για το σκοπό αυτόν, αναπτύχθηκε μία σειρά από ευριστικές τεχνικές ώστε να εντοπιστούν συγκεκριμένα σημεία-κλειδιά που χρησιμοποιούν οι προγραμματιστές όταν αναζητούν για παραδείγματα. Η διαδικασία περιλαμβάνει τα εξής στάδια:

1. Ένα ερώτημα (query) σχηματίζεται με βάση το γενικότερο πλαίσιο του κώδικα που είναι γραμμένος από το χρήστη.
2. Το ερώτημα στέλνεται σε μία αποθήκη παραδειγμάτων κώδικα και συγκρίνεται με μία συλλογή από υπάρχοντα projects.
3. Τα παραδείγματα που είναι περισσότερο σχετικά ως προς τη περιεχόμενο τους επιστρέφονται στο χρήστη.
4. Τα επιστρέφόμενα αυτά παραδείγματα μπορούν να εξερευνηθούν από τον χρήστη, είτε γραφικά είτε μέσω κειμένου, ώστε να αποφασίσει ποιο είναι το κατάλληλο για τη δική του ανάγκη.

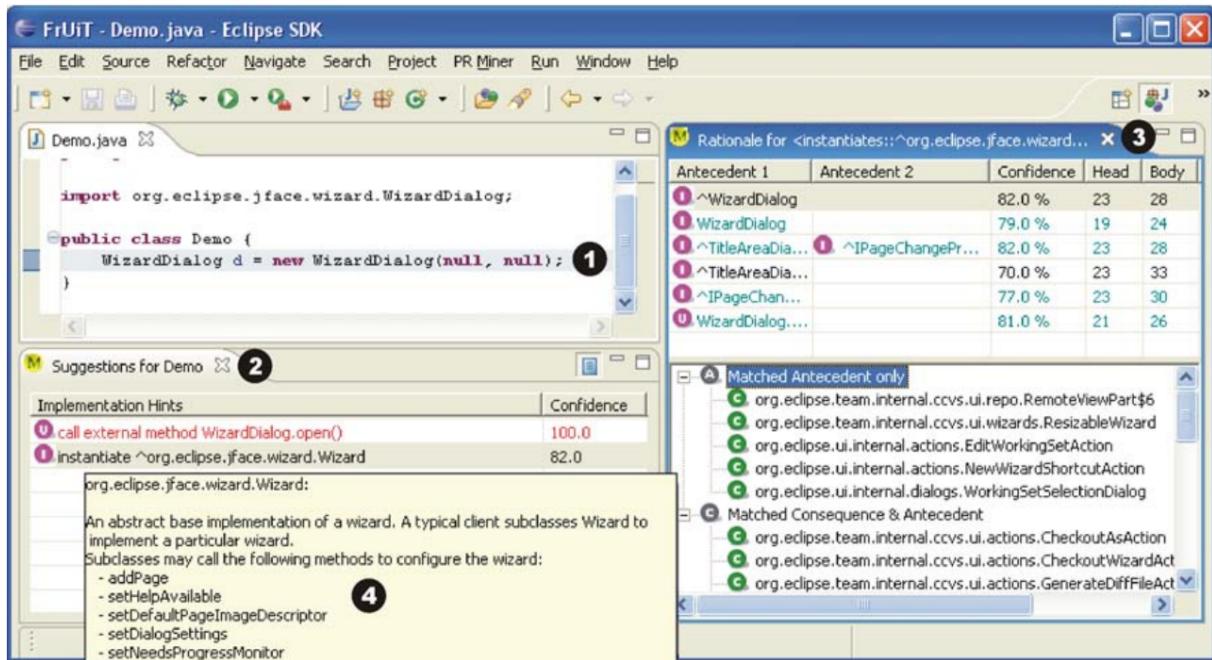
Το μεγάλο θετικό του εργαλείου αυτού είναι ο αυτόματος σχηματισμός του ερωτήματος από τα συμφραζόμενα του κώδικα του χρήστη, καθώς επίσης και η υποστήριξη οποιουδήποτε framework. Μάλιστα, φροντίζει ώστε να λαμβάνεται υπόψη ο πλέον πρόσφατος τρόπος χρήσης ενός framework και όχι οι πιθανοί απαρχαιωμένοι του. Ωστόσο, η προσέγγιση αυτή έχει και τα μειονεκτήματα της. Το κυρίως πρόβλημα είναι ότι ο προγραμματιστής πρέπει να έχει κάποια γνώση από πού και πώς να ξεκινήσει να γράφει τον κώδικα του ώστε το Strathcona να μπορεί να αρχίσει το σχηματισμό ερωτημάτων. Ακόμη, η προσέγγιση του Strathcona είναι βασισμένη σε ευριστικές τεχνικές οι οποίες κατά περίπτωση μπορούν να γίνουν αρκετά γενικές και όχι εναρμονισμένες με το τρέχον πρόβλημα. Αυτός ο περιορισμός οδηγεί συχνά σε μη σχετικά επιστρεφόμενα παραδείγματα. Τέλος, η αναζήτηση βασίζεται στο σημασιολογικό περιεχόμενο, το οποίο περιλαμβάνει λεπτομέρειες για την τρέχουσα μέθοδο που γράφεται, την περιεχόμενη κλάση της, και τις προηγούμενες μεθόδους που καλέστηκαν. Συνεπώς, προσωρινή πληροφορία όπως η σειρά με την οποία καλέστηκαν οι μέθοδοι δεν λαμβάνεται υπόψη.

## 2.6.2 FrUiT

Το εργαλείο FrUiT [23] αποτελεί ένα plugin του Eclipse το οποίο αναπτύχθηκε για να υποστηρίξει τη χρήση διάφορων frameworks. Το FrUiT δείχνει τις συσχετίσεις μεταξύ του πηγαίου κώδικα του framework που τείνουν να υπάρχουν συχνά σε παρόμοια projects με αυτό που χρησιμοποιεί εκείνη τη στιγμή ο προγραμματιστής. Στο Σχήμα 2.10 φαίνεται το σύστημα FrUiT κατά τη διάρκεια χρήσης του από τον τελικό χρήστη.

Τα παρεχόμενα παραδείγματα του FrUiT, υπολογίζονται σε τρεις φάσεις. Πρώτα, το FrUiT εξάγει όλες τις συσχετίσεις από ένα σύνολο ήδη υλοποιημένων εφαρμογών, δεδομένου ενός framework ή API. Τέτοιες συσχετίσεις περιλαμβάνουν *extends* (class A extends class B), *implements* (class A implements an interface B), *overrides* (class A overrides an inherited method m), *calls* (any of the methods of class A calls a method m), και *instantiates* (a constructor of class B is invoked from the implementation of class A).

Δεύτερον, το FrUiT χρησιμοποιεί ένα κανόνα σύνδεσης για να αναγνωρίσει συσχετίσεις που εμφανίζονται συχνά όταν ένα συγκεκριμένο framework ή API χορ-



Σχήμα 2.10: Στιγμιότυπο από την οθόνη λειτουργίας του συστήματος FrUIT. (1) Ο πιγγαίος κώδικας στον editor προσδιορίζει το πλαίσιο προγραμματισμού του χρήστη. (2) Τα προτεινόμενα βοηθήματα του FrUIT για το τρέχον αρχείο φαίνονται σε ένα παράθυρο μέσα στο IDE. (3) Παρουσιάζεται η λογική εκτέλεσης πίσω από κάθε προτεινόμενο βοήθημα. (4) Για κάθε παράδειγμα ο χρήστης μπορεί να διαβάσει το αντίστοιχο JavaDoc της προτεινόμενης κλάσης ή μεθόδου

σιμοποιείται. Οι συνδέσεις αυτές είναι της μορφής, *antecedent* → *consequent*, με ένα ποσοστό που συμβολίζει το *confidence* της συγκεκριμένης σύνδεσης. Για παράδειγμα, ένας τέτοιος κανόνας θα μπορούσε να γραφεί ως, *call: m* ↔ *instantiate : B*, *confidence* 80%. Ο κανόνας αυτός σημαίνει ότι όταν μία εφαρμογή που χρησιμοποιεί το framework καλεί τη μέθοδο *m*, επίσης τείνει (στο 80% των περιπτώσεων) να καλεί και έναν *constructor* της κλάσης *B*.

Τρίτον, λόγω του παραπάνω κανόνα σύνδεσης το FrUIT παράγει ένα τεράστιο αριθμό αποτελεσμάτων. Έτσι, είναι απαραίτητο τα αποτελέσματα να υποστούν ένα είδος φιλτραρίσματος πριν επιστραφούν στο χρήστη. Το φιλτράρισμα αυτό γίνεται μέσω κάποιων κατωφλιών, για παράδειγμα ελάχιστο *confidence*.

### 2.6.3 CodeBroker

Το CodeBroker [24] είναι ένα εργαλείο το οποίο αναπτύχθηκε το 2005 από την συνεργασία μεταξύ της εταιρίας SRA Inc<sup>16</sup>, που εδρεύει στο Tokyo και του τμήματος επιστήμης των υπολογιστών του πανεπιστημίου του Colorado, των Η.Π.Α.

<sup>16</sup><https://www.sra.co.jp/index-en.html>

To CodeBroker, προτείνει την αλήση μεθόδων, βασιζόμενο στα σχόλια και στα ορίσματα αυτών, στην τρέχουσα θέση του δρομέα. Ο στόχος του συστήματος είναι να υποστηρίξει την ανάπτυξη μίας νέας μεθόδου, βρίσκοντας υπάρχοντες μεθόδους οι οποίες (μερικώς) υλοποιούν την συνολική λειτουργικότητα για την οποία στοχεύει ο προγραμματιστής.

Το CodeBroker είναι ένα εργαλείο RSSE ενσωματωμένο στον editor emacs<sup>17</sup>. Το κύριο χαρακτηριστικό του εργαλείου αυτού είναι ότι αναγνωρίζει ενεργά την ανάγκη για νέες προτάσεις ελέγχοντας τον τρόπο αλήσης των μεθόδων που γράφει ο προγραμματιστής. Αυτό σημαίνει ότι κάθε φορά που ο δρομέας αλλάζει θέση, αυτόματα στέλνονται ερωτήματα σε μία αποθήκη λογισμικού. Το ερώτημα περιγράφει τη μέθοδο που αναπτύσσεται εκείνη τη στιγμή λαμβάνοντας υπόψη τις λέξεις που χρησιμοποιούνται στα σχόλια, καθώς και τις λέξεις και τους τύπους μεταβλητών στον ορισμό της. Ο στόχος είναι να βρεθούν μέθοδοι με παρόμοια σχόλια ή ορίσματα. Στην ουσία, πρόκειται για μία τεχνική που χρησιμοποιεί την τεχνική αντιστοίχισης κειμένου, ονόματι Latent Semantic Analysis (LSA) [25], η οποία αναπαριστά κάθε μέθοδο ως ένα διάνυσμα τύπου Boolean το οποίο υποδεικνύει σε ποιες λέξεις γίνεται αναφορά από τον ορισμό της μεθόδου, και ένα διάνυσμα βαρών που υποδεικνύει ποιες λέξεις είναι σημαντικές με βάση τη συχνότητα Term Frequency - Inverse Document Frequency (TF-IDF) [26], ώστε να βγει συμπέρασμα για την ομοιότητα των δύο μεθόδων. Τα αποτελέσματα της αναζήτησης στη συνέχεια φιλτράρονται ανάλογα με την ομοιότητα μεταξύ των τύπων των μεταβλητών και παρουσιάζονται στο χρήστη σε ένα παράθυρο του editor.

Το σύστημα είναι χωρισμένο στο *front-end* και στο *back-end* τμήμα του. Το *front-end* παρακολουθεί το δρομέα και στέλνει ερωτήματα στο *back-end* με σχετικές πληροφορίες από τον κώδικα, και παρουσιάζει τα αποτελέσματα. Το *back-end* διαιρείται με τη σειρά του σε δύο τμήματα: 1) αυτό που αποθηκεύει και ανανεώνει τα σχόλια και τα ορίσματα των μεθόδων από ένα σετ βιβλιοθηκών, APIs, και frameworks για επαναχρησιμοποίηση, και 2) αυτό που αφαιρεί μεθόδους, αλάσεις, ή πακέτα από την προτεινόμενη λίστα, ενώ παράλληλα αποφασίζει αν η αφαίρεση πρέπει να γίνει μόνο για την τρέχουσα συνεδρία (εφόσον η πρόταση δεν έχει καμία σχέση με τη λειτουργία στο συγκεκριμένο κομμάτι κώδικα) ή για όλες τις συνεδρίες

<sup>17</sup><https://www.gnu.org/software/emacs/>

(εφόσον η πρόταση δεν έχει γενικώς κάποια χρησιμότητα ως προς το χρήστη π.χ. όταν αυτός γνωρίζει πώς να την κατασκευάζει από μόνος του).

## 2.7 Σχεδίαση με Βάση Στατιστικών Μοντέλων

Στην ενότητα αυτή θα μελετήσουμε συστήματα, τα οποία ως κύριο γνώμονα τους για την παραγωγή αποτελεσμάτων, έχουν στατιστικά και στοχαστικά μοντέλα. Τα συστήματα αυτά χρησιμοποιούν πιθανότητες για να προτείνουν στον τελικό χρήστη παραδείγματα, πιθανές βελτιώσεις στον ήδη γραμμένο κώδικα, ή να τον αποτρέψουν από πιθανά λάθη στον κώδικα που γράφει εκείνη τη στιγμή.

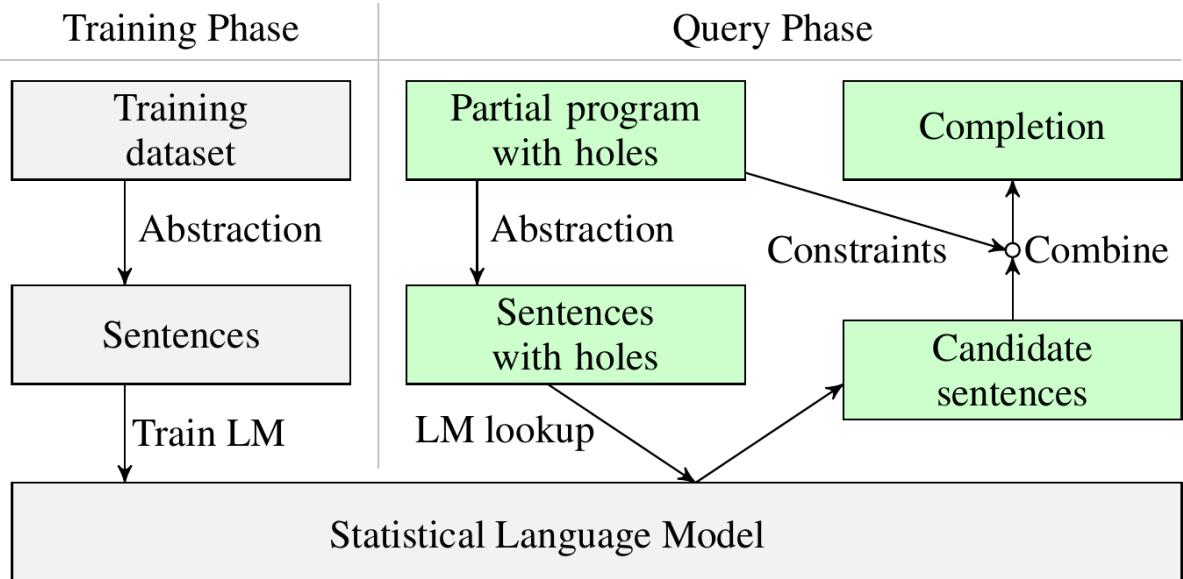
### 2.7.1 SLANG

Το SLANG [27] αποτελεί σχετικά ένα πρόσφατο εργαλείο πρότασης κώδικα το οποίο αναπτύχθηκε το 2014 από μία ομάδα διδακτορικών φοιτητών στο πανεπιστήμιο ETH, της Ζυρίχης. Το εργαλείο αυτό δεν επιστρέφει τμήματα κώδικα στον προγραμματιστή, αλλά δεδομένου ενός προγράμματος που λείπουν ορισμένα κομμάτια του, αναλαμβάνει τη συμπλήρωση των κενών αυτών με τις πιο πιθανές ακολουθίες κλήσεων μεθόδων. Ο τρόπος που πετυχαίνει το τελευταίο είναι με τη χρήση Statistical Language Models (SLMs) [28].

Ο στόχος του SLM είναι να δημιουργηθεί ένα μοντέλο το οποίο θα είναι ικανό να εκτιμήσει την κατανομή πιθανότητας φυσικής γλώσσας όσο το δυνατόν με μεγαλύτερη ακρίβεια. Ένα SLM είναι η κατανομή πιθανότητας  $P(S)$  μίας συμβολοσειράς  $S$ , η οποία αντικατοπτρίζει πόσο συχνά η συμβολοσειρά αυτή εμφανίζεται σε μία πρόταση. Με τον συμβολισμό ποικίλων πρωτοφανών φαινομένων μίας γλώσσας με όρους απλών παραμέτρων σε ένα SLM, μπορούμε να έχουμε έναν απλό τρόπο αντιμετώπισης της σύνθετης φυσικής γλώσσας σε έναν υπολογιστή.

Η εποπτική ροή λειτουργίας του SLANG περιλαμβάνει τις παρακάτω φάσεις:

- Κατά τη διάρκεια της εκπαίδευσης του συστήματος, πραγματοποιείται ανάλυση προγράμματος για την εξαγωγή ακολουθιών κλήσεων σε API από απόθήκες λογισμικού. Στη συνέχεια, ένα SLM εκπαιδεύεται από τα εξαγόμενα δεδομένα. Για τη δουλειά αυτή, χρησιμοποιείται το μοντέλο N-gram [29], επαναλαμβανόμενα νευρωνικά δίκτυα (Recurrent Neural Networks) [30] και ένας



Σχήμα 2.11: Αρχιτεκτονική του συστήματος SLANG.

συγκερασμός των δύο. Το αποτέλεσμα της φάσης εκπαίδευσης είναι μία πιθανότητα που αντιστοιχίζεται με κάθε εξαγόμενη ακολουθία μεθόδων.

- Για να αλληλεπιδράσει ο χρήστης με το SLANG, παρέχει ένα τμήμα κώδικα το οποίο έχει κενά. Μέσω ανάλυσης προγράμματος, εξάγονται οι ακολουθίες από αυτό το μερικό πρόγραμμα, και χρησιμοποιείται το SLM για να υπολογιστεί ένα σετ υποψήφιων ακολουθιών για την ολοκλήρωση του κώδικα. Το τελικό ολοκληρωμένο αποτέλεσμα, επιλέγεται με βάση την υψηλότερη πιθανότητα και την προϋπόθεση ότι αυτό ικανοποιεί τους περιορισμούς που θέτει κάθε κενό.

## 2.7.2 CodeHint

Το εργαλείο CodeHint [31], που και αυτό στοχεύει στην συμπλήρωση κενών στον κυρίως σκελετό του κώδικα, δημιουργήθηκε το 2014, από μία πανεπιστημιακή ομάδα στην Καλιφόρνια, των Η.Π.Α. Πρόκειται για άλλο ένα plugin στο Eclipse IDE, το οποίο παράγει και αξιολογεί τον κώδικα σε συνθήκες *runtime* και ως εκ τούτου μπορεί να χρησιμοποιηθεί για τη σύνθεση κώδικα Java που περιλαμβάνει Input/Output (I/O), κώδικα που μπορεί να ελέγχει τον υπόλοιπο κώδικα του συστήματος ή να αυτοελέγχεται (*reflection*<sup>18</sup>), *native calls*, και άλλα προχωρημένα χαρακτηριστικά της γλώσσας Java. Για την υλοποίηση του βασικού αλγορίθμου που βρίσκεται

<sup>18</sup><https://stackoverflow.com/questions/37628/what-is-reflection-and-why-is-it-useful>

πίσω από το CodeHint, δημιουργήθηκε και εκπαιδεύτηκε ένα πιθανοτικό μοντέλο ταξινόμησης, για την πιθανότητα εμφάνισης τύπων αντικειμένων, εκφράσεων κ.α.

Στη γενική του εικόνα, το CodeHint, αποτελεί ένα πολύ δυναμικό σύστημα, εύκολο στη χρήση και αρκετά διαδραστικό. Η δυναμικότητα του, δίνει τη δυνατότητα για ευκολότερη εύρεση και φιλτράρισμα υποψήφιου κώδικα, σημείο όπου οι στατικές τεχνικές υστερούν. Έτσι ο χρήστης μπορεί να βρει πολύ συγκεκριμένα παραδείγματα για το επιθυμητό αποτέλεσμα. Το CodeHint υποστηρίζει μία μεγάλη γκάμα από προσδιορίσμες προδιαγραφές που βοηθούν τον χρήστη να βρει τον κώδικα που φάχνει, ακόμη και αν αυτός έχει λίγη γνώση για το αντικείμενο. Επιπρόσθετα, οι χρήστες μπορούν να χρησιμοποιούν δυναμικές μεταβλητές όταν ορίζουν προδιαγραφές για τον κώδικα που αναζητούν και να βλέπουν τα αποτελέσματα της εκτέλεσης των υποψήφιων λύσεων, το οποίο μπορεί να είναι βοηθητικό στο να διαλέξουν την πιο σχετική. Ο βασικός στόχος της ομάδας ανάπτυξης, ήταν η εξασφάλιση ότι ο χρήστης θα βρει αποτελέσματα ακόμη και αν έχει ελάχιστες πληροφορίες να παρέχει στο μηχανισμό αναζήτησης. Ακόμη, οι χρήστες μπορούν να προσδιορίσουν καλύτερα το χώρο αναζήτησης δίνοντας ένα προσχέδιο (code skeleton) του επιθυμητού κώδικα με κενά τα άγνωστα κομμάτια.

### 2.7.3 Codex

ο 2014 μία συνεργασία του πανεπιστημίου του Stanford με την ομάδα έρευνας και ανάπτυξης της Adobe, οδήγησαν στην ανάπτυξη του εργαλείου Codex [32]. Πρόκειται για ένα βοήθημα στη γλώσσα προγραμματισμού Ruby<sup>19</sup>, το οποίο ακολουθεί μία οδηγούμενη από τα δεδομένα (data-driven) προσέγγιση, αναλύοντας αναγνωρισμένο κώδικα υψηλής ποιότητας. Αρχικά με μεθόδους στατικής ανάλυσης κώδικα βρίσκει συνηθισμένες τεχνικές που χρησιμοποιούνται στη γλώσσα Ruby (idioms), και στη συνέχεια επιτρέπει στις εφαρμογές να στέλνουν ερωτήματα στη βάση δεδομένων του για υποστήριξη νέων προγραμματιστικών διεπαφών. Οι κύριες εφαρμογές του Codex είναι τρεις και αναλύονται περιγραφικά παρακάτω:

1. *Statistical Linting*. Οι προγραμματιστές έχουν συνήθως την τάση να γράφουν κακής ποιότητας κώδικα. Αυτός ο κακής ποιότητας κώδικας προκαλεί μεγάλη «ζημιά» στα projects λογισμικού. Το Codex εκμεταλλεύεται το γεγονός ότι ο

<sup>19</sup><https://www.ruby-lang.org/en/>

κακός κώδικας κατά το πλείστον διαφέρει συντακτικά από τον καλό ποιοτικά κώδικα. Έτσι πραγματοποιείται έλεγχος στα τμήματα κώδικα για την επισύμανση κακογραμμένου κώδικα. Ο έλεγχος αυτός γίνεται (i) ελέγχοντας τη σύνδεση των συναρτήσεων και τη δόμηση αυτών αναλύοντας το επιστρεφόμενο μπλοκ τιμών, (ii) αναλύοντας τον τύπο κάθε συνάρτησης, και (iii) αναλύοντας τις μεταβλητές.

2. *Pattern Annotation*. Πολλές φορές, χρήσιμες προγραμματιστικές τεχνικές (idioms) δε συλλέγονται και καταγράφονται από το επίσημο documentation. Το Codex γεφυρώνει το κενό αυτό εντοπίζοντας κοινότυπα idioms καθώς επεξεργάζεται κώδικα και τα στέλνει σε ένα πλήθος ανθρώπινου δυναμικού ειδικών για να τα σχολιάσει. Οι σχολιασμοί εμφανίζονται οποτεδήποτε ο προγραμματιστής ανοίγει ένα αρχείο που περιέχει το idiom. Οι χρήστες επωφελούνται από τον σχολιασμένο αυτό κώδικα κάτω από πολλά σενάρια, όπως στην χρήση κώδικα που σαρώθηκε από κάποιο βοήθημα στο διαδίκτυο, στο άνοιγμα ενός μη οικείου τύπου αρχείου που έδωσε ένας συνεργάτης, ή στην επανεξέταση ενός τμήματος αντιγραμμένου κώδικα.

3. *Library Generation*. Ο χαρακτήρας ανασύνθεσης που παρουσιάζουν τα τμήματα κώδικα που έχει στη βάση του το Codex υποδεικνύει ότι οι προγραμματιστές μπορούν να επωφεληθούν από την ενσωμάτωση τους σε μία νέα βιβλιοθήκη, η οποία αντικατοπτρίζει τη λειτουργικότητα που λείπει από τις υπάρχουσες βιβλιοθήκες στη γλώσσα Ruby. Συνεπώς, δημιουργώντας μία νέα βιβλιοθήκη προσπαθεί να εξαλείψει το παραπάνω κενό.

Ο αλγόριθμος του Codex περιέχει δύο σημαντικά τμήματα, ένα για τον εντοπισμό συχνά χρησιμοποιούμενων Ruby idioms, και ένα για τον υπολογισμό των συχνοτήτων των κόμβων του AST και την πιθανότητα αυτοί να συνδυάζονται (π.χ. ακολουθίες κλήσεων μεθόδων).

*Pattern Finding Module*. Εντοπίζει ένα σύνολο από Ruby idioms και τμήματα κώδικα υψηλού επιπέδου που χρησιμοποιούνται συχνά από τους προγραμματιστές. Αυτό επιτυγχάνεται φιλτράροντας τα τμήματα κώδικα που οι προγραμματιστές είναι λιγότερο πιθανό να βρουν ενδιαφέροντα και χρήσιμα. Η γενική μορφή ερωτημάτων για την εύρεση μοτίβων μέσω του Codex λαμβάνει υπόψη της πέντε παραμέ-

The screenshot shows a dark-themed IDE interface. On the left, a code editor displays Ruby code related to creating a nested hash:

```

1 # Annotations in the Codex IDE
2
3 # Creating a nested hash
4
5
6 choices = Hash.new { |h,k| h[k] = {} }
7
8
9
10 # Using a nested hash
11
12 choices[:is][:happy] = true
13 choices[:is][:sad] = false
14
15
16

```

On the right, there's a panel titled "Creating a Nested Hash" with the following details:

- Total Count 75 Project Count 17
- Code snippet: `new do |var0, var1|
 var0[var1] = {}
end`
- Projects: cassandra, fakeweb, fog, jruby, MacRuby, maglev, mechanize, rails, recommendify, redcar, reek, replicate, rubinius, ruby, rubygems, state\_machine, more...
- Description: Assigns an empty hash as the default key value

Σχήμα 2.12: Το Codex εξετάζει projects ανοιχτού κώδικα για να δημιουργήσει διεπαφές που ενσωματώνουν αναδυόμενες προγραμματιστικές τεχνικές. Εδώ, το Codex κάνει σχολιασμό δημοφιλών idioms (pattern annotation) που εμφανίζονται στον κώδικα του χρήστη.

τρους, (i) τον αριθμό των μοναδικών projects στα οποία έχει εμφανιστεί ένα snippet (Project Count), (ii) τον συνολικό αριθμό εμφανίσεων ενός snippet (Total Count), (iii) τον αριθμό των ξεχωριστών αρχείων στα οποία εμφανίζεται σε ένα τμήμα κώδικα (File Count), (iv) τον αριθμό μοναδικών μεταβλητών, κλήσεων συναρτήσεων, και άλλων θεμελιακών στοιχείων που εμφανίζονται σε ένα snippet (Token Count), και (v) τον αριθμό μοναδικών κλήσεων συναρτήσεων σε ένα snippet (Function Count).

The screenshot shows a code editor with the following code:

```

5
6
7 # CodexLint Example:
8 "a string".split("\n").to_s
9
10

```

Below the code, a message states: "Function to\_s has appeared 12 times and split has appeared 29 times, and they've appeared 0 times together."

Σχήμα 2.13: Το Codex επιδεικνύει μη συνηθισμένο κώδικα σε ένα snippet (κίτρινη υπογράμμιση) και εμφανίζει προειδοποιητικό μήνυμα στο κάτω μέρος.

*Statistical Analysis Module.* Το τμήμα αυτό του Codex προειδοποιεί τους χρήστες όταν ο κώδικας που γράφουν δεν είναι συνηθισμένος με βάση πιθανοτήτων (Σχήμα 2.13). Ο τρόπος που εντοπίζεται ο ασυνήθιστος κώδικα είναι ο εξής:

1. Ανάλυση του τρόπου κλήσεων συναρτήσεων (Function Call Analysis): Αν μία αρκετά συνηθισμένη συνάρτηση καλείται με παραμέτρους που είναι σπάνια παρατηρούμενοι από το Codex, είναι πιθανόν ο κώδικας να είναι προβληματικός.
2. Ανάλυση του τρόπου σύνδεσης των συναρτήσεων (Function Chaining Analysis):

Εδώ ελέγχεται πόσες φορές μία συνάρτηση συνδέεται με μία άλλη. Δύο συχνά χρησιμοποιούμενες συναρτήσεις που δεν έχει παρατηρηθεί να συνδέονται ποτέ, αποτελούν ένδειξη ασυνήθιστου κώδικα.

3. Ανάλυση του μπλοκ επιστρεφόμενης τιμής (Block Return Value Analysis): Η ανάλυση αυτή ελέγχει πόσες φορές ένα συγκεκριμένο μπλοκ έχει επιστρέψει ένα συγκεκριμένο είδος τιμής.
4. Ανάλυση αναγνωριστικών (Identifier Analysis): Η ανάλυση αυτή ελέγχει πόσες φορές ένα αναγνωριστικό μίας μεταβλητής έχει ανατεθεί σε ένα συγκεκριμένο τύπο θεμελιακού στοιχείου.

## 2.8 Σύνοψη Κεφαλαίου

Στο κεφάλαιο αυτό ήρθαμε σε μία επαφή με τη σχετική βιβλιογραφία που είναι απαραίτητη για την εξοικείωση του αναγνώστη με τους όρους και τις συνήθεις τεχνικές και μεθόδους που χρησιμοποιούνται στα συστήματα προτάσεων στην τεχνολογία λογισμικού. Συγκεκριμένα, διακρίναμε τους διάφορους τύπους επαναχρησιμοποίησης κώδικας που μπορεί να συναντήσει κανείς, και δώσαμε παραδείγματα εφαρμογών. Μέσω των παραδειγμάτων έγινε κατανοητή η σημασία της επαναχρησιμοποίησης κώδικα στην διαδικασία ανάπτυξης λογισμικού.

Έπειτα, στρέψαμε το ενδιαφέρον μας στον ειδικότερο θέμα του επικεντρωμένου στα παραδείγματα προγραμματισμού (example-centric programming) και είδαμε πρακτικές εφαρμογές αυτού. Τελευταία και πιο σημαντικά, ορίσαμε τι αποτελεί ένα σύστημα προτάσεων RSSE και κάναμε μία προσπάθεια διαχωρισμού των διαφόρων τέτοιων συστημάτων που έχουν αναπτυχθεί στη βιβλιογραφία. Αν και με μια πρώτη θεώρηση είναι λογικό κανείς να συμπεράνει ότι πρόκειται για ένα χαοτικό πεδίο έρευνας, με λίγη προσπάθεια μπορεί να διαπιστώσει πως τα συστήματα αυτά κρύβουν πολλές ομοιότητες μεταξύ τους. Για το λόγο αυτό έγινε ένας διαχωρισμός τους ανάλογα με την κύρια φιλοσοφία λειτουργίας του καθενός. Στο επόμενο κεφάλαιο, θα περιγράψουμε ορισμένα συστήματα αναλυτικότερα περνώντας σε τεχνικές λεπτομέρειες αυτών. Έτσι, θα ξεκαθαρίσει περισσότερο το τοπίο ως προς τον τρόπο λειτουργίας των RSSEs.

# Κεφάλαιο 3

## Περιγραφή του Προβλήματος

### 3.1 Γενικά

Όπως έγινε φανερό στο προηγούμενο κεφάλαιο η επαναχρησιμοποίηση ακόδικα αποτελεί ζωτικής σημασίας ανάγκη για την ανάπτυξη λογισμικού στη σημερινή εποχή. Για το λόγο αυτό κατά καιρούς αναπτύχθηκαν διάφορα συστήματα RSSE για τη διευκόλυνση των μηχανικών στο σκοπό αυτό. Το καθένα από αυτά ακολουθεί τη δική του προσέγγιση και στοχεύει στη δική του ερμηνεία του προβλήματος. Ωστόσο, το πρόβλημα αυτό, παρουσιάζει παρουσιάζει παρόμοια όψη από όπου και αν το εξετάσουμε.

Πολλές φορές εξετάζοντας ένα έργο λογισμικού εξωτερικά ως προς τη συνολική εικόνα είναι εύκολο κανείς να αγνοήσει το βαθμό πολυπλοκότητας που μπορεί αυτό να κρύβει στο εσωτερικό του. Επίσης, συνήθως είναι δύσκολο να ερμηνεύσουμε τον τρόπο λειτουργίας, ενός συστήματος, εξετάζοντας το σαν ένα ενιαίο σύνολο. Για να μπορέσουμε να κατανοήσουμε το «εσωτερικό» ενός έργου λογισμικού πρέπει να διαχωρίσουμε τα διάφορα κομμάτια του, και να σκεφτούμε πώς αυτά συνδέονται μεταξύ τους. Αυτός, κατά το πλείστον, είναι και ο τρόπος που δημιουργούνται τα μεγάλα έργα λογισμικού, συνθέτοντας ολοένα και μεγαλύτερα τμήματα, το καθένα από τα οποία επιτελεί και το δικό τους ανεξάρτητο έργο.

Διαπιστώνει εύκολα κανείς, ότι για τη δημιουργία ενός μεγάλου και σύνθετου έργου λογισμικού, ο αριθμός των ανεξάρτητων τμημάτων που απαιτούνται αυξάνει με πολύ γρήγορο ρυθμό. Έτσι, με στόχο να επιτευχθεί το επιθυμητό ολοκληρωμένο

αποτέλεσμα, γεννιέται η ανάγκη της επαναχρησιμοποίησης λογισμικού. Ωστόσο, αν και μπορεί η χρήση έτοιμου κώδικα να φαντάζει εύκολη υπόθεση, πολλές φορές δεν είναι έτσι. Λόγω της εξάπλωσης του Διαδικτύου, μπορούμε να βρούμε τεράστιο αριθμό ανοιχτών βιβλιοθηκών λογισμικού και συστημάτων. Ακόμη, για την υλοποίηση ενός στόχου υπάρχουν πολλές διαφορετικές προτεινόμενες λύσεις. Έτσι, σύντομα γεννιούνται στον προγραμματιστή ερωτήματα όπως: "Ποιο λογισμικό πρέπει να επιλέξω;", "Ποια λύση είναι πιο αποτελεσματική;", "Πώς θα ενσωματώσω την εκάστοτε λύση στο δικό μου έργο;" κ.α.

## 3.2 Ορίζοντας το πρόβλημα

Ένας μηχανικός λογισμικού επιθυμεί συχνά να χρησιμοποιήσει APIs από εξωτερικά συστήματα και βιβλιοθήκες. Ωστόσο, οι λεπτομέρειες χρήσης πολλών APIs δεν είναι ξεκάθαρες μιας και η τεκμηρίωση (documentation) αυτών είναι συνήθως ελλιπής ή μη πρόσφατα ενημερωμένη. Μια εναλλακτική πηγή πληροφορίας για το πως χρησιμοποιούνται συγκεκριμένα APIs, αποτελούν άλλα ολοκληρωμένα έργα λογισμικού. Όμως, η αναζήτηση σε έργα χιλιάδων γραμμών κώδικα είναι ανιαρή και πολύ χρονοβόρα.

Εναλλακτικά, ο προγραμματιστής στρέφεται στις παραδοσιακές μηχανές αναζήτησης στο Internet με σκοπό να βρει βοηθήματα, παραδείγματα χρήσης των APIs, αποσπάσματα κώδικα, ώστε να εκπληρώσει τις δικές του ανάγκες. Η παραπάνω διαδικασία συνοψίζεται συνήθως στα βήματα αναζήτηση-αντιγραφή-επικόλληση-προσαρμογή. Η ροή αυτή έχει μειονεκτήματα όπως:

- Ο προγραμματιστής μειώνει την παραγωγικότητα του: αφού πρέπει προσωρινά να εγκαταλείψει το IDE του και να μεταφερθεί στον browser. Κατά τη διάρκεια αναζήτησης, πρέπει να μείνει συγκεντρωμένος και να μην αποσπαστεί η προσοχή του από διαφημίσεις και άλλο άσχετο υλικό. Στη συνέχεια, πρέπει να βρει το κατάλληλο σημείο και να το μεταφέρει στον editor του IDE.
- Οι παραδοσιακές μηχανές αναζήτησης δεν είναι οι κατάλληλες: οι μηχανές αναζήτησης (π.χ. Google<sup>1</sup>) είναι σχεδιασμένες για αναζήτηση υλικού γενικού

<sup>1</sup><https://www.google.com>

περιεχομένου. Αυτό σημαίνει ότι κατά τη διάρκεια μιας αναζήτησης ο προγραμματιστής βρίσκεται αντιμέτωπος με αρκετά μη σχετικά αποτελέσματα. Έτσι, συνήθως για να εντοπίσει τη λύση που επιθυμεί αναγκάζεται να επισκεφτεί αρκετούς διαφορετικούς ιστότοπους, με αποτέλεσμα να χρονοτριβεί, να κινδυνεύει να αποσπαστεί η προσοχή του, αλλά πολλές φορές και να εκνευρίζεται.

Λόγω των παραπάνω προβλημάτων, καθίσταται αναγκαία η ανάπτυξη εργαλείων που θα είναι σε θέση να βοηθήσουν τον προγραμματιστή να κατανοήσει τον τρόπο χρήσης λογισμικού από τρίτους, και μάλιστα με τρόπο σύντομο, κατανοητό και εύχρηστο. Τέτοιου είδους συστήματα προτάσεων, όπως είδαμε και στο προηγούμενο κεφάλαιο, αποτελούν τα RSSEs.

### 3.3 Παρόμοια Συστήματα - Τεχνική Ανάλυση

Στο Κεφάλαιο 2 διαχωρίσαμε τα συστήματα προτάσεων σε κατηγορίες και αναλύσαμε περιληπτικά τα πιο δημοφιλή από αυτά. Παρακάτω θα εξετάσουμε μερικά συστήματα πιο εξονυχιστικά εξετάζοντας τεχνικές λεπτομέρειες αυτών. Ο λόγος είναι ότι διάφορα χαρακτηριστικά των συστημάτων αυτών μας ενέπνευσαν στη δημιουργία του δικού μας συστήματος.

#### 3.3.1 Bing Code Search

Ένα από τα υπάρχοντα συστήματα προτάσεων που μας επηρέασε αρκετά και στη δική μας σχεδίαση είναι το BCS που αναφέραμε και στο κεφάλαιο της βιβλιογραφίας. Αν και το σύστημα αυτό στοχεύει σε διαφορετική γλώσσα προγραμματισμού σε σχέση με το δικό μας, θελήσαμε να υιοθετήσουμε αρκετά χαρακτηριστικά του. Επίσης, ένας λόγος είναι ότι πρόκειται από τα πλέον πρόσφατα συστήματα μιας και κυκλοφόρησε το 2015 και χρησιμοποιείται ακόμη και σήμερα, με συνέπεια να εκφράζει τις ανάγκες της εποχής.

Ένα χαρακτηριστικό που έχει το BCS, το οποίο απουσιάζει από τα περισσότερα συστήματα είναι ότι επιτρέπει ερωτήματα σε φυσική γλώσσα. Αυτό θεωρήσαμε πως είναι κάτι αρκετά σημαντικό στην ανάπτυξη ενός συστήματος προτάσεων το

οποίο θα μπορούν να χρησιμοποιούν με ευκολία χρήστες διαφορετικών γνωστικών επιπέδων.

Πέρα από αυτό ιδιαίτερο ενδιαφέρον παρουσιάζει και η διαδικασία αξιολόγησης των αποτελεσμάτων στο Bing Code Search. Η διαδικασία αξιολόγησης λαμβάνει υπόψιν της τα αποτελέσματα της μηχανής αναζήτησης Bing, ευνοώντας τα τμήματα κώδικα που έχουν μεγάλη συσχέτιση με το ερώτημα του χρήστη, και τέλος έχουν υψηλή ποιότητα κώδικα. Στον Πίνακα 3.1 παρουσιάζονται συνοπτικά τα χαρακτηριστικά, στα οποία στηρίζεται ο αλγόριθμος ταξινόμησης του BCS. Τα χαρακτηριστικά χωρίζονται σε τρεις κατηγορίες.

Πίνακας 3.1: Μετρικές του BCS για την ταξινόμηση των αποτελεσμάτων.

Όνομα Μετρικής	Τύπος	Περιγραφή
Url position	Real	η κατάταξη της ιστοσελίδας από όπου εξαγάγει το snippet
First/Second/Third Url	Binary	1 αν το snippet είναι από την ιστοσελίδα με κατάταξη 1/2/3, αλλιώς 0
In-page order	Real	κατάταξη του snippet εσωτερικά της ιστοσελίδας
First in-page	Binary	1 αν το snippet εμφανίζεται πρώτο στην ιστοσελίδα, αλλιώς 0
Clickthrough	Real	τιμή ένδειξης σχετικότητας των APIs με το query
Tf-idf	Real	τιμή ένδειξης ομοιότητας των APIs ανάμεσα στα snippets
Snippet length	Real	το σύνολο των γραμμών κώδικα του snippet
API calls	Real	ο αριθμός των κλήσεων API στο snippet
Unknown API calls	Real	ο αριθμός των κλήσεων API στο snippet που είναι άγνωστα
Unknown types	Real	ο αριθμός των άγνωστων τύπων αντικειμένων που αναφέρονται στο snippet

Η πρώτη κατηγορία χαρακτηριστικών (*URL position* έως *First in-page*) συλλέγουν πληροφορία από την κατάταξη της μηχανής αναζήτησης. Το *URL position* έχει να κάνει με τη θέση της ιστοσελίδας στα αποτελέσματα της μηχανής αναζήτησης από την οποία προέρχεται το εξαγόμενο κομμάτι κώδικα. Το *First/Second/Third url* χαρακτηριστικό επισημαίνει τα code snippets που προέρχονται από τις πρώτες θέσεις στη μηχανή αναζήτηση. Το *First in-page* δηλώνει αν το code snippet είναι το πρώτο που εμφανίζεται στην ιστοσελίδα, ενώ το *In-page order* δείχνει την ακριβή θέση του μέσα σε αυτήν.

Η δεύτερη κατηγορία έχει να κάνει με τη σχετικότητα του κώδικα. Το χαρακτηριστικό *Clickthrough* μοντελοποιεί την αντιστοίχιση από τα ερωτήματα του χρήστη σε λέξεις-κλειδιά της γλώσσας C#. Το Bing Code Search αντιμετωπίζει το κάθε snippet ως μία ακολουθία λέξεων (tokens) και υπολογίζει, για κάθε token  $t$ , την πιθανότητα να συμβεί το  $t$  για ένα δεδομένο query  $Q$ ,  $P(t|Q)$ . Έστω  $Q = q_1, q_2, \dots, q_n$  είναι το σύνολο των λέξεων για ένα query. Τότε η παραπάνω πιθανότητα θα ισούται με:

$$P(t|Q) = P(t|q_1, q_2, \dots, q_n) = \sum_{i=1}^n P(t|q_i) \cdot P(q_i|Q)$$

όπου το  $P(q|Q)$  είναι η πιθανότητα ο όρος  $q$  να συμβεί στο query  $Q$ . Ο παραπάνω υπολογισμός ποσοτικοποιεί πόσο πιθανό είναι ο όρος  $q$  να συμβεί στα queries:

$$P(q|Q) = \frac{\alpha_q}{\sum_{q' \in Q} \alpha_{q'}}$$

όπου το  $\alpha_q$  είναι η συχνότητα εμφάνισης του  $q$  σε όλα τα δυνατά queries. Για τον υπολογισμό του  $P(t|q)$  για κάθε token  $t$  σε κάθε snippet και για κάθε όρο  $q$ , η ομάδα σχεδίασης χρησιμοποίησε ένα μοντέλο στατιστικής ευθυγράμμισης λέξεων [33]. Τα δεδομένα που χρησιμοποίησαν για την εκπαίδευση του μοντέλου προέρχονταν από το αρχείο της μηχανής αναζήτησης Bing. Για την εκτίμηση του  $\alpha_q$ , αξιοποιούνται πάλι τα δεδομένα της μηχανής αναζήτησης:

$$\alpha_q = \frac{\text{\#times } q \text{ occurs in query log}}{\text{total term count in query log}}$$

Έχοντας υπολογίσει την πιθανότητα  $P(t|Q)$ , η συνολική πιθανότητα ενός snippet  $S$  να συμβεί δεδομένου ενός query  $Q$  είναι

$$P(S|Q) = P(t_1, t_2, \dots, t_m|Q) = \prod_{j=1}^m P(t_j|Q)$$

όπου το snippet  $S$  αποτελείται από τα tokens  $t_1, \dots, t_m$  τα οποία για απλότητα θεωρούνται ανεξάρτητα. Το χαρακτηριστικό *Clickthrough* ισούται με την τιμή της πιθανότητας αυτής.

Το *tf-idf* χαρακτηριστικό στηρίζεται στο γεγονός ότι για ένα query συνήθως υπάρχει ένα σύνολο από APIs που χρησιμοποιούνται στη λύση. Ένα μέρος από τα APIs αυτά πιθανώς αναφέρονται και σε άλλα snippets για αυτό το query. Το χαρακτηριστικό αυτό δίνει μεγάλες τιμές σε tokens που εμφανίζονται συχνά σε ένα snippet, ενώ παράλληλα δίνει μικρές τιμές σε tokens που εμφανίζονται συχνά σε όλα τα snippets. Η τελική τιμή που δίνεται στο χαρακτηριστικό αυτό είναι ο μέσος όρος από όλα τα tokens που εμφανίζονται στο snippet. Μία μεγαλύτερη τιμή υποδεικνύει ότι το snippet μπορεί να είναι η επιθυμητή λύση.

Τέλος, η τρίτη κατηγορία χαρακτηριστικών (από *Snippet length* έως *Compilation errors*) μοντελοποιεί την ποιότητα κώδικα από συντακτική και σημασιολογική άποψη.

To *snippet length* ισούται με τις γραμμές κώδικα του snippet. To *API calls* είναι ο

αριθμός των αλήσεων διαφόρων APIs σε ένα snippet. Αυτά τα δύο χαρακτηριστικά έχουν να κάνουν με τη σύνταξη του κώδικα και οι τιμές τους υπολογίζονται από το AST.

Τα χαρακτηριστικά *Unknown API calls*, *Unknown types* και *Compilation errors* έχουν να κάνουν με το σημασιολογικό χαρακτήρα των snippets, και για να υπολογιστούν πρέπει να γίνει η μεταγλώττιση των snippets. Κάτι τέτοιο τις περισσότερες φορές δεν είναι εφικτό. Ωστόσο, μέσω του Parser Roslyn<sup>2</sup> της C# εξάγουν χρήσιμες πληροφορίες για το snippet μέσω των μηνυμάτων σφαλμάτων για τα χαρακτηριστικά της κατηγορίας αυτής.

### 3.3.2 PARSEWeb

Ακόμη ένα σύστημα στο οποίο αναφερθήκαμε στο δεύτερο κεφάλαιο είναι το PARSEWeb. Στο σημείο αυτό θα εξετάσουμε λίγο λεπτομερέστερα το κομμάτι του συστήματος που κάνει ομαδοποίηση των ακολουθιών και βαθμολόγηση αυτών.

Όσον αφορά την ομαδοποίηση, η προσέγγιση της ομάδας ήταν να αγνοήσει την σειρά των APIs με την οποία εμφανίζονται σε μία ακολουθία. Για παράδειγμα, οι ακολουθίες ”2,3,4,5” και ”2,4,3,5”, όπου κάθε νούμερο αντιπροσωπεύει ένα συγκεκριμένο API, θεωρούνται όμοιες επειδή διαφορετικοί προγραμματιστές μπορούν να γράψουν τα ενδιάμεσα βήματα για την επίτευξη μίας λύσης με διαφορετική σειρά. Στη συνέχεια μέσω μίας μετρικής που ονομάζουν ως *cluster precision*, μετρούν το πλήθος των APIs κατά το οποίο διαφέρουν δύο ακολουθίες. Έστερα θέτοντας ένα κατώφλι στη μετρική αυτή, σχηματίζουν ομάδες ακολουθιών που έχουν *cluster precision* μικρότερο από αυτό το όριο. Π.χ. ας θεωρήσουμε τις ακολουθίες ”8,9,6,7” και ”8,6,10,7”. Αυτές οι δύο ακολουθίες έχουν τρία κοινά APIs (8,6,7) και διαφέρουν ως προς ένα. Η προσέγγιση του PARSEWeb θεωρεί τις δύο αυτές ακολουθίες όμοιες για κατώφλι ίσο με ένα και υψηλότερο. Αυτή η ευριστική μέθοδος στηρίζεται στο γεγονός ότι για την επίτευξη ενός αντικειμένου χρησιμοποιούνται παρόμοια APIs.

Σχετικά με τη βαθμολόγηση των αποτελεσμάτων, λαμβάνουν υπόψιν τους δύο ευρυστικές με βάση τις οποίες ταξινομούν τις ακολουθίες ώστε να μπορούν οι χρήστες να τις εντοπίσουν ευκολότερα. Η πρώτη, βαθμολογεί καλύτερα τις ακολουθίες

---

<sup>2</sup><https://github.com/dotnet/roslyn>

όπου συναντώνται APIs με μεγαλύτερη συχνότητα εμφάνισης. Αυτό συνδέεται με την υπόθεση ότι APIs που εμφανίζονται συχνότερα είναι πιο πιθανό να είναι χρησιμότερα και να χρησιμοποιηθούν στο τελικό αποτέλεσμα. Η δεύτερη, στηρίζεται στη διαπίστωση ότι οι χρήστες ενδιαφέρονται περισσότερο για τις μικρότερες σε μήκος ακολουθίες παρά για τις μεγαλύτερες. Έτσι, προωθούν σε υψηλότερες θέσεις τις συντομότερες ακολουθίες.

### 3.3.3 MAPO

Το επόμενο σύστημα που μας επηρέασε αρκετά στο σχεδιασμό του δικού μας συστήματος είναι το MAPO (Mining API usage Patterns from Open source repositories) [14], το οποίο συναντήσαμε και στο κεφάλαιο 2. Όπως είδαμε πρόκειται για ένα σύστημα πρωτοπόρο στον τομέα των RSSEs. Στο σημείο αυτό θα αναλύσουμε λεπτομερέστερα τα δομικά του στοιχεία που φαίνονται στο Σχήμα 3.1. Η δομή του συστήματος συνοψίζεται στα εξής τμήματα:

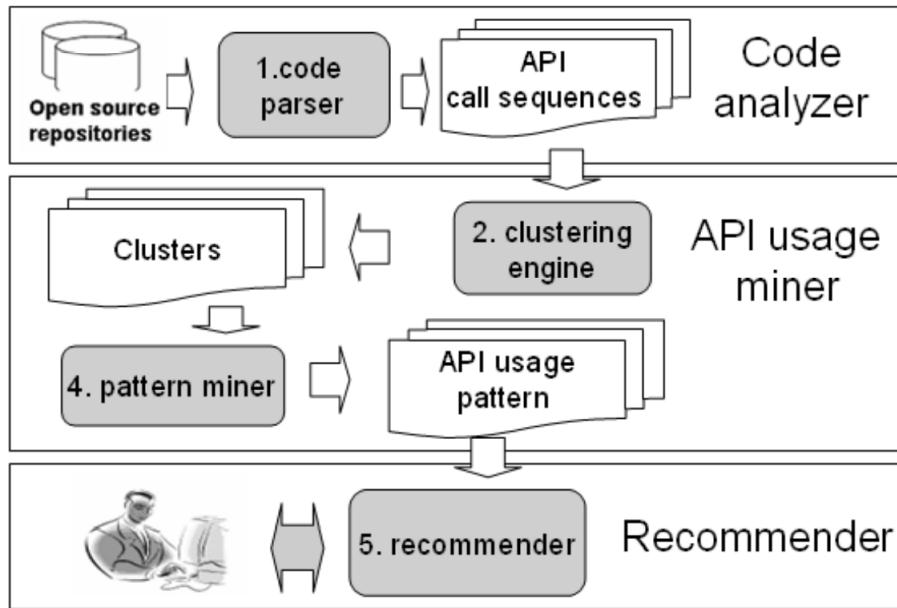
- Στον αναλυτή πηγαίου κώδικα (Code analyzer), ο οποίος συλλέγει τις πληροφορίες χρήσης των APIs και τις οργανώνει με βάση τις μεθόδους που καλούνται.
- Στο κομμάτι που κάνει εξόρυξη χρήσης των APIs (API Usage miner), όπου αφού περάσουν τα δεδομένα από τον αναλυτή πηγαίου κώδικα, οργανώνονται σε ομάδες (clusters) και επιδέχονται sequential pattern mining ανά cluster. Έπειτα τα αποτελέσματα αποθηκεύονται σε XML<sup>3</sup> μορφή για το επόμενο στάδιο.
- Τέλος στο υπεύθυνο για τις προτάσεις υποσύστημα (recommender), το οποίο είναι ενσωματωμένο στο περιβάλλον συγγραφής κώδικα Eclipse και είναι αυτό που παρουσιάζει τα τμήματα κώδικα στο χρήστη.

#### 3.3.3.1 Αναλυτής Πηγαίου Κώδικα

Ο Eclipse JDT Compiler<sup>4</sup> βρίσκεται στην καρδιά του Source Code Analyzer. Κάθε μέθοδος στον κώδικα που δίνεται ως είσοδος στο σύστημα από μηχανές αναζήτησης

<sup>3</sup><https://en.wikipedia.org/wiki/XML>

<sup>4</sup><https://www.eclipse.org/jdt/core/>



Σχήμα 3.1: Σχηματική αναπαράσταση της αρχιτεκτονικής στο σύστημα MAPO.

κώδικα υπάγεται σε μετατροπή σε ακολουθία βάσει των παρακάτω παραδοξών σχετικά με το τι ορίζεται από τους συγγραφείς ως κλήση API:

- Μία κλήση ενός constructor μιας γονικής κλάσης (super class) όταν η super class βρίσκεται σε μια εξωτερική (third-party) βιβλιοθήκη λογισμικού.
- Μία μετατροπή τύπου πεδίου (typecasting) όταν οποιαδήποτε σχετιζόμενη με την προς αλλαγή κλάση προέρχεται από third-party βιβλιοθήκες.
- Μία κλήση μεθόδου όταν η κλάση στην οποία βρίσκεται προέρχεται από third-party βιβλιοθήκη.
- Η αρχικοποίηση μιας κλάσης όταν αυτή προέρχεται από μία third-party βιβλιοθήκη.

Αφού έχουμε ορίσει τι είναι μία κλήση API θα εξετάσουμε πως ο πηγαίος κώδικας μετασχηματίζεται σε ακολουθιακή μορφή. Για την επίτευξη αυτού του στόχου το MAPO χρησιμοποιεί το AST ως μία δενδρική αναπαράσταση του κώδικα στη μέθοδο υπό επεξεργασία. Στη συνέχεια το εν λόγω δένδρο υπόκειται μεταδιατεταγμένη διάσχιση (post-order traversal)<sup>5</sup> για τη δημιουργία μίας ακολουθίας. Ένα παράδειγμα δημιουργίας ακολουθίας για μια εμφωλευμένη κλήση μεθόδου φαίνεται παρακάτω.

<sup>5</sup>[https://en.wikipedia.org/wiki/Tree\\_traversal](https://en.wikipedia.org/wiki/Tree_traversal)

```

1 getGraphicalViewer().setRootEditPart(new ScalableRootEditPart())
2
3 @new org.eclipse.gef.editparts.ScalableRootEditPart
4 @org.eclipse.gef.EditPartViewer#setRootEditPart
5 @org.eclipse.gef.ui.parts.GraphicalEditor#getGraphicalViewer

```

To `@` είναι σύμβολο αρχής της κλήσης μεθόδου, ενώ το `new` δηλώνει αρχικοποίηση κλάσης. Τέλος `#` χωρίζει μία μέθοδο (δεξιά) από την κλάση η οποία την καλεί (αριστερά).

Μετά την ολοκλήρωση της συγκέντρωσης όλων των πιθανών ακολουθιών επιλέγεται ένα υποσύνολο αυτών με κριτήριο να περιέχουν την εμφάνιση όσων περισσότερων μεθόδων γίνεται. Αυτό συμβαίνει γιατί η παραγωγή όλων των πιθανών ακολουθιών σε if/for loops μπορεί να δώσει μεγαλύτερο βάρος σε μία μέθοδο στο στάδιο εξόρυξης. Επιπλέον, ένα επιχείρημα που δικαιολογεί την κατασκευή του υποσυνόλου αυτού είναι πως όλες οι πιθανές ακολουθίες θα έχουν κοινές υπακολουθίες επειδή κλήθηκαν από το ίδιο μονοπάτι (path) οπότε το μονοπάτι αυτό θα παρουσιαζόταν από τον αλγόριθμο εξόρυξης ως συχνό πρότυπο (frequent pattern).

Τέλος, όσον αφορά την ανάλυση κώδικα, ακολουθείται μια αναδρομική λογική. Ακόμη και αν μια μέθοδος δεν προέρχεται από third-party βιβλιοθήκη αλλά συναντάται σε κάποια κλήση API τότε εξετάζεται και αυτή για την περίπτωση που περιέχει η ίδια κλήσεις σε κάποιο API. Η λογική πίσω από αυτό κατά τους συγγραφείς είναι πως οι προγραμματιστές τείνουν να διασκορπίζουν την πλήρη υλοποίηση τους σε διαφορετικές μεθόδους.

### 3.3.3.2 API Usage Pattern Miner

Με τα δεδομένα σε ακολουθιακή μορφή ακολουθεί το στάδιο της εξόρυξης αφού προηγηθεί ομαδοποίηση των ακολουθιών με σκοπό το διαχωρισμό τους σε ομάδες με διαφορετική χρήση του API η καθεμία. Η ομαδοποίηση γίνεται με τα εξής κριτήρια:

1. Το όνομα της μεθόδου.
2. Το όνομα της κλάσης που περιέχει τη μέθοδο.
3. Τα APIs που καλούνται από τη μέθοδο.

Για το πρώτο και το δεύτερο υποστηρίζεται πως παρόμοια ονόματα μεθόδων και των κλάσεων που τις περιέχουν συνήθως δείχνουν σε παρόμοια προγραμματιστική συμπεριφορά. Το τελευταίο σαν επέκταση των παραπάνω προέρχεται από την πεποίθηση πως αν κάποιος θέλει να πετύχει κάτι συγκεκριμένο, θα το κάνει μέσω των ίδιων APIs.

Για την ομοιότητα μεταξύ ονομάτων στην ομαδοποίηση χρησιμοποιείται η βιβλιοθήκη Simmetrics<sup>6</sup> που υπολογίζει την ομοιότητα μεταξύ λέξεων. Οι πλήρεις υπογραφές χωρίζονται σε λέξεις και οι ομοιότητά τους ανάγεται στον μέσο όρο της ομοιότητας των επιμέρους λέξεών τους. Για δύο ακολουθίες ( $s_1$  και  $s_2$ ), η μετρική ομοιότητας ορίζεται ως:

$$sim(s_1, s_2) = \frac{\# \text{ of API calls in } I_1 \cap I_2}{\# \text{ of API calls in } I_1 \cup I_2}$$

όπου τα  $I_1$  και  $I_2$  είναι τα αντίστοιχα σύνολα από APIs που εμφανίζονται στις δύο ακολουθίες. Η τελική ομαδοποίηση γίνεται με χρήση ενός ιεραρχικού αλγορίθμου.

Ο Miner χρησιμοποιεί τον αλγόριθμο *BIDE* [34] για την εξαγωγή συχνών ακολουθιών για το δεδομένο ελάχιστο support που δίνει ο χρήστης. Σε ένα cluster  $C$ , το support μίας ακολουθίας κλήσεων API ορίζεται ως εξής:

$$support(s) = \frac{\# \text{ of API call sequences with } s}{\# \text{ of API call sequences in } C}$$

Δηλαδή το πηλίκο του αριθμού των ακολουθιών στις οποίες περιέχεται η  $s$  προς το πλήθος της ομάδας ακολουθιών που βρίσκεται η  $s$ . Αφού παραχθούν τα αποτελέσματα του αλγορίθμου μετατρέπονται πάλι σε ακολουθιακή αναπαράσταση για να προωθηθούν στον recommender.

### 3.3.3.3 API Usage Pattern Recommender

Τέλος, τα προτεινόμενα snippets παρουσιάζονται στο χρήστη μέσω του API Usage Pattern Recommender, που είναι στην ουσία ένα plugin στο προγραμματιστικό περιβάλλον του Eclipse. Αντί να απαιτείται από τους χρήστες να εξετάσουν ένα προς ένα τα αποτελέσματα, δημιουργείται μία προεπισκόπηση όπου φαίνεται ο βαθμός ομοιότητας, αν το ζητούμενο API εμπεριέχεται στο snippet, το όνομα της

<sup>6</sup><https://github.com/Simmetrics/simmetrics>

κλάσης κλπ. Έτσι ο χρήστης μπορεί να περιηγηθεί γρήγορα στα αποτελέσματα και το σημαντικότερο χωρίς να χρειάζεται να εγκαταλείψει το προγραμματιστικό του περιβάλλον για να καταφύγει σε κάποιο φυλλομετρητή. Στη συνέχεια, όποιο πρότυπο επιθυμεί ο χρήστης μπορεί να το εξερευνήσει περαιτέρω. Τα αποτελέσματα που επιστρέφει το MAPO είναι λιγότερα από αυτά που θα επέστρεψε μία μηχανή αναζήτησης έτσι ο χρήστης χρειάζεται λιγότερο χρόνο για να τα εξερευνήσει.

## 3.4 Σύνοψη Κεφαλαίου

Στο κεφάλαιο αυτό ορίσαμε το πρόβλημα που έχουμε να αντιμετωπίσουμε και θα προσπαθήσουμε να επιλύσουμε στο επόμενο κεφάλαιο. Μέσα από αυτό είδαμε γιατί είναι σημαντικά τα συστήματα προτάσεων στην τεχνολογία λογισμικού και τι εναλλακτικές δυνατότητες αυτά προσφέρουν

Παρόλο που τα παραπάνω συστήματα αποτελούν αξιόλογα συστήματα στον τομέα των RSSEs, παρουσιάζουν κάποιες αδυναμίες τις οποίες θα προσπαθήσουμε να αντιμετωπίσουμε σχεδιάζοντας ένα νέο δικό μας σύστημα.

Μέσω της ανάπτυξης ενός δικού μας συστήματος στοχεύουμε στην αντιμετώπιση των αδυναμιών και των ελλείψεων που συναντώνται στα υπόλοιπα όπως:

- Πολλά από τα συστήματα δεν επιστρέφουν έτοιμα προς χρήση (ready-to-use) παραδείγματα. Τα περισσότερα από αυτά επιστρέφουν ακολουθίες από API calls ή συμπληρώνουν κενά στον ήδη υπάρχοντα κώδικα του χρήστη. Η χρήση τέτοιων συστημάτων προϋποθέτει ο προγραμματιστής να έχει αρκετή γνώση του αντικειμένου που επιθυμεί να υλοποιήσει.
- Η πλειοφηφία των συστημάτων δεν υποστηρίζει ερωτήματα σε φυσική γλώσσα. Θεωρούμε πως η κατανόηση μιας γλώσσας ερωτημάτων για τη χρήση ενός βοηθητικού συστήματος είναι αποθαρρυντική για το χρήστη.
- Συνήθως δεν αξιολογείται η ποιότητα των αποτελεσμάτων. Έτσι προβάλλεται στο χρήστη ένα μεγάλο πλήθος αποτελεσμάτων που μέσα σε αυτά υπάρχει μεγάλη ποσότητα άχρηστης πληροφορίας.
- Τέλος, λίγα είναι τα συστήματα που υιοθετούν τεχνικές ομαδοποίησης των αποτελεσμάτων. Κάτι τέτοιο το θεωρούμε πολύ σημαντικό μιας και βοηθάει

το μηχανικό να εντοπίσει ευκολότερα την επιθυμητή λύση.

Έτσι, στο επόμενο κεφάλαιο θα περιγράψουμε αναλυτικά τον τρόπο σχεδίασης και ανάπτυξης του συστήματος μας. Μέσα από τη μελέτη αυτή ο αναγνώστης θα κατανοήσει βαθύτερα τα προβλήματα και τις δυσκολίες που απαιτεί το έργο αυτό. Κυρίως όμως θα καταλάβει τις δυνατότητες και τη χρησιμότητα ενός τέτοιου συστήματος.

# Κεφάλαιο 4

## Codecatch: Ένα Σύστημα Προτάσεων για Τμήματα Κώδικα

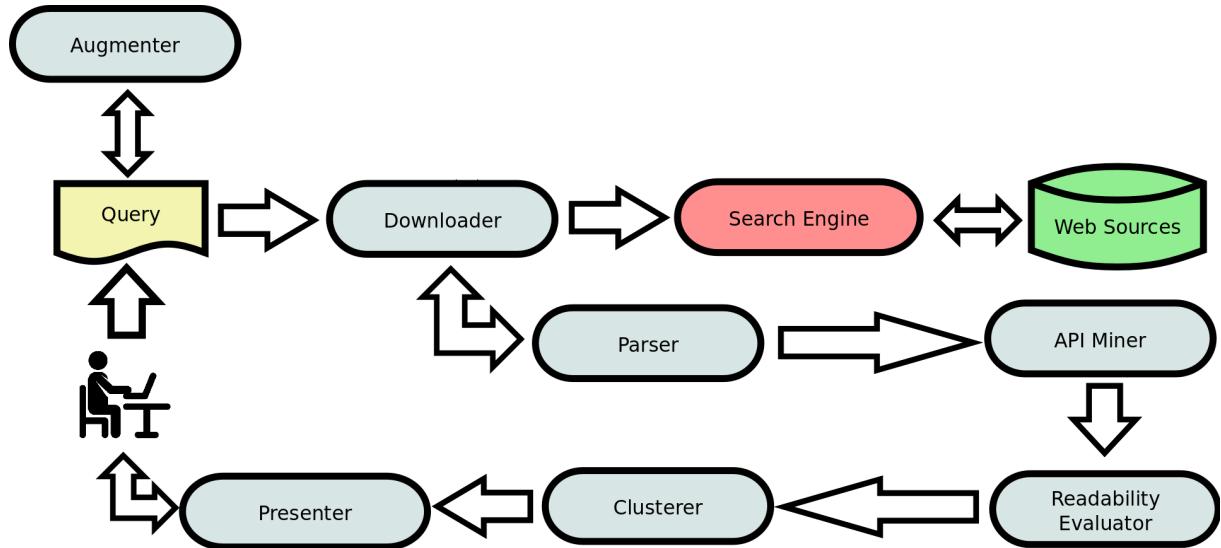
### 4.1 Γενικά

Έχοντας σκιαγραφήσει το πρόβλημα, στο κεφάλαιο αυτό θα περιγράψουμε αναλυτικά τον τρόπο κατασκευής του συστήματος RSSE Codecatch, καθώς και τις σχεδιαστικές αποφάσεις που πάρθηκαν κατά τη διάρκεια της διαδικασίας. Η δομή του συστήματος, θα περιγραφεί στις επόμενες ενότητες, καθεμία περιγράφοντας ένα από τα κύρια τμήματα του συστήματος. Η διαδικασία παρουσίασης εναρμονίζεται με τη σειρά που το σύστημα εκτελεί τα διάφορα στάδια, επιτρέποντας στον αναγνώστη την εύκολη παρακολούθηση της.

### 4.2 Δομή του Συστήματος

Η δομή του συστήματος χωρίζεται στα τμήματα του 1) *Προσαυξητή Ερωτήματος* (*Query Augmenter*), του 2) υπεύθυνου για το κατέβασμα αρχείων (*Downloader*), του 3) του αναλυτή (*Parser*), του 4) του εξορύκτη API (*API Miner*), του 5) *Αξιολογητή Αναγνωσιμότητας* (*Readability Evaluator*), του 6) *Ομαδοποιητή* (*Clusterer*), και τέλος του 7) *Παρουσιαστή Αποτελεσμάτων* (*Results Presenter*). Μία εποπτική άποψη της δομής του συστήματος μας απεικονίζεται στο Σχήμα 4.1.

Πριν περάσουμε στην ανάλυση του εκάστοτε τμήματος καλό θα ήταν να περι-



Σχήμα 4.1: Η δομή του συστήματος μας (Codecatch).

γράφουμε εν συντομίᾳ τα διάφορα στάδια εκτέλεσης του συστήματος. Ο τρόπος λειτουργίας, λοιπόν, του συστήματος συνοψίζεται στα εξής σημεία:

1. Ο χρήστης εκκινεί το σύστημα, συγγράφει ένα ερώτημα (query), πυροδοτεί τη διαδικασία αναζήτησης.
2. Το ερώτημα του χρήστη, μέσω του *Augmenter*, προσαυξάνεται με λέξεις-κλειδιά της γλώσσας Java, ώστε τα αποτελέσματα από τη μηχανή αναζήτησης να είναι πιο στοχευμένα.
3. Ο *Downloader* στέλνει το προσαυξημένο ερώτημα του χρήστη στη μηχανή αναζήτησης του Google<sup>1</sup>, και κατεβάζει πληροφορία από τους διάφορους ιστότοπους που αυτή επιστρέφει.
4. Ο *Parser* σαρώνει τα δεδομένα που έχει κατεβάσει ο *Downloader* και ξεκαθαρίζει ποια από αυτά αποτελούν τμήματα κώδικα σε γλώσσα Java (code snippets).
5. Στη συνέχεια, ο *API Miner*, σαρώνει τα τμήματα κώδικα για να βρει πιθανά APIs που χρησιμοποιούνται μέσα σε αυτά.
6. Ο *Readability Evaluator*, αξιολογεί ποια από τα τμήματα κώδικα είναι ευανάγνωστα και ποια όχι.

<sup>1</sup><https://www.google.com>

7. Ο *Clusterer*, φτιάχνει ομάδες με παρόμοια ως προς το περιεχόμενο τους code snippets.
8. Τέλος, ο *Presenter*, είναι υπεύθυνος για να παρουσιάσει τα αποτελέσματα του συστήματος στο χρήστη, καθώς επίσης και να τον βοηθήσει να περιηγηθεί σε αυτά ώστε να βρει την επιθυμητή λύση.

## 4.3 Query Augmenter

Ως πρώτο κομμάτι του συστήματος, και ίσως το πιο απλό, θα σχολιάσουμε τον Query Augmenter. Η φιλοσοφία αυτού του τμήματος είναι να μπορέσουμε να πάρουμε όσο πιο ειδικά αποτελέσματα από μία μηχανή αναζήτησης μπορούμε. Συνήθως, ένας προγραμματιστής που θέλει να βρει παραδείγματα στο Internet μέσω μίας μηχανής αναζήτησης, γράφει το ερώτημα του και προσθέτει ακόμη τη γλώσσα προγραμματισμού για την οποία ενδιαφέρεται να βρει λύσεις (π.χ. "*How to open a file in java*"). Τις περισσότερες φορές κάτι τέτοιο είναι αρκετό. Ωστόσο, υπάρχουν περιπτώσεις που ένα τέτοιο ερώτημα από μόνο του, μπορεί να «μπερδέψει» τη μηχανή αναζήτησης, και να επιστρέψει μαζί και πολλά μη σχετικά αποτελέσματα. Ο λόγος που συμβαίνει αυτό είναι επειδή οι μηχανές αναζήτησης, όπως το Google, είναι από τη φύση τους γενικού περιεχομένου, με αποτέλεσμα να μην επικεντρώνονται σε κάθικα όπως θα ήθελε ο προγραμματιστής. Έτσι, μαζί με παραδείγματα κάθικα που θα επέστρεψε το παραπάνω ερώτημα, είναι πιθανό, να συνυπάρχουν και αποτελέσματα σχετικά με τον καφέ (καθώς η Java πέρα από γλώσσα προγραμματισμού αποτελεί και είδος καφέ), σχετικά με το νησί Java της Ινδονησίας, ή ίσως σχετικά με μια γλώσσα προγραμματισμού με παρόμοιο όνομα, όπως η Javascript, αλλά εντελώς διαφορετική από αυτό που φάχνουμε.

Μία λύση ώστε να περιορίσουμε τα αποτελέσματα μας, κατά ένα μεγάλο ποσοστό, σε αυτά που αφορούν κάθικα είναι να προσαυξήσουμε το ερώτημα του χρήστη με λέξεις-κλειδιά που χρησιμοποιούνται στη συγκεκριμένη γλώσσα προγραμματισμού, στη δική μας περίπτωση της Java (Πίνακας 4.1). Αποσαφηνίζεται πως δεν απαιτούμε να υπάρχουν ταυτόχρονα όλες οι λέξεις-κλειδιά στα αποτελέσματα αλλά έστω μία ή περισσότερες από αυτές. Οπότε η σύνδεση τους μπορεί να γίνει με το λογικό συνθετικό "OR".

Πίνακας 4.1: Λέξεις-κλειδιά που χρησιμοποιεί ο Augmenter στην προσαύξηση του ερωτήματος του χρήστη.

java	import	class	interface	public
protected	abstract	final	static	if
for	void	int	long	double

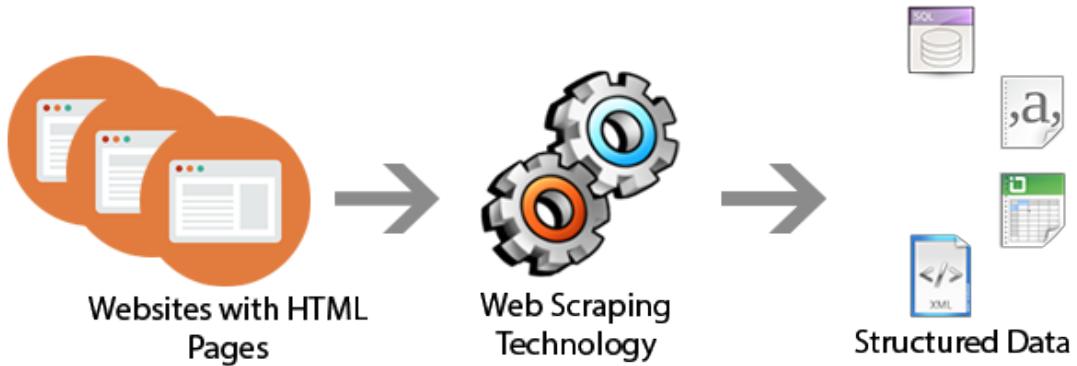
Η προσαύξηση αυτή, αν και απλή διαδικασία, αποφέρει αρκετά πιο στοχευμένα αποτελέσματα από τη μηχανή αναζήτησης. Επίσης, αν και φυσικά ο χρήστης θα μπορούσε να κάνει από μόνος του μια τέτοια τροποποίηση, θα ήταν πολύ χρονοβόρο και κουραστικό να το κάνει για κάθε ερώτημα μόνος του.

## 4.4 Downloader

Εφόσον το ερώτημα του χρήστη έχει τροποποιηθεί κατάλληλα, αποστέλλεται στη μηχανή αναζήτησης ώστε να επιστραφούν σχετικά αποτελέσματα με αυτό. Ως προεπιλεγμένη μηχανή αναζήτησης έχει οριστεί αυτή της Google, μιας και αποτελεί την πιο ευρέως χρησιμοποιούμενη και χωρίς αμφιβολία πολύ αξιόπιστη στο είδος της.

Στο σημείο αυτό, αξίζουν να σημειωθούν τα πλεονεκτήματα από την αξιοποίηση μίας υπάρχουσας εμπορικής μηχανής αναζήτησης, όπως της Google, έναντι στην κατασκευή μίας καινούργιας (π.χ. [35, 36]). Πρώτα, είναι ουσιαστικά πιο αποδοτικό από πλευράς πόρων, καθώς η διατήρηση μίας ενημερωμένης βάσης δεδομένων είναι δύσκολη και μη συμφέρουσα. Δεύτερον, η παραγωγή υψηλής ποιότητας αποτελεσμάτων από ερωτήματα σε φυσική γλώσσα είναι δύσκολη υπόθεση. Ως συνέπεια, μία εμπορική μηχανή αναζήτησης, η οποία μάλιστα έχει υποστεί τεχνικές βελτιστοποίησης, είναι πιθανότερο να παράγει καλύτερα αποτελέσματα από μία πρωτότυπη μηχανή αναζήτησης με περιορισμένο πεδίο πληροφορίας. Τέλος, μια γενικού σκοπού μηχανή αναζήτησης αντλεί αποτελέσματα από ιστότοπους με βοηθητικό περιεχόμενο, blogs, και από τις σελίδες όπου βρίσκεται η τεκμηρίωση (documentation) των APIs. Τα παραδείγματα που βρίσκονται από τέτοιες πηγές έχουν πιο διδακτικό χαρακτήρα από ολόκληρα τμήματα κώδικα που βρίσκονται σε μεγάλες αποθήκες λογισμικού.

Τα αποτελέσματα που επιστρέφονται από την αναζήτηση γίνονται *scraping* μέσω



Σχήμα 4.2: Το scraping αποτελεί μία από τις πιο συνηθισμένες τεχνικές συλλογής δεδομένων στα συστήματα ανάκτησης πληροφορίας.

του framework Scrapy<sup>2</sup> της γλώσσας *Python*. Ο τρόπος με τον οποίο εντοπίζουμε τα σημεία όπου μπορεί να βρίσκεται κώδικας σε ένα website έχει να κάνει με τη δομή του και συγκεκριμένα τα HTML tags<sup>3</sup>. Με μερικές αναζητήσεις, εύκολα μπορεί να διαπιστώσει κανείς ότι τα τμήματα κώδικα συνήθως εμπεριέχονται σε μικρό αριθμό διαφορετικών HTML tags, όπως `<pre>`, `<crayon-pre>`, και `<code>`. Κάνοντας scraping το περιεχόμενο αυτών των tags μαζεύουμε το μεγαλύτερο μέρος του κώδικα. Μάλιστα με εποπτική διαπίστωση και δοκιμές το ποσοστό αυτό ξεπερνά το 92%. Το ποσοστό αυτό προέκυψε στέλνοντας 5 διαφορετικά ερωτήματα και εξετάζοντας για το καθένα τα 20 πρώτα αποτελέσματα της μηχανής αναζήτησης. Φυσικά, μαζί με τη χρήσιμη πληροφορία του κώδικα, ένα μεγάλο ποσοστό των δεδομένων που μαζεύουμε αποτελεί για εμάς θόρυβο (π.χ. κώδικας από άλλες γλώσσες προγραμματισμού, outputs προγραμμάτων, σχόλια κ.α.). Σε πρώτη φάση αυτό δεν αποτελεί σημαντικό πρόβλημα. Αυτό που μας ενδιαφέρει προς στιγμήν είναι να μαζέψουμε όσο το δυνατόν περισσότερο κώδικα μπορούμε, πέρα από άχρηστα δεδομένα. Αφού, έχουμε συλλέξει την πληροφορία μέσω του scraper, κάνουμε αρχικά ένα HTML Parsing, ώστε να «καθαρίσουμε» τα δεδομένα από τα περιττά HTML στοιχεία που υπάρχουν σε αυτά. Στη συνέχεια, αποθηκεύουμε τα δεδομένα σε μορφή JavaScript Object Notation (JSON)<sup>4</sup> ώστε να μπορούμε εύκολα να τα επεξεργαστούμε στα επόμενα βήματα.

Παρακάτω παρουσιάζεται ένα τμήμα των δεδομένων που συλλέγουμε σε μορφή JSON. Όπως φαίνεται, συλλέγονται πληροφορίες για την ιστοσελίδα από όπου προ-

<sup>2</sup><https://scrapy.org/>

<sup>3</sup>[https://www.w3schools.com/TAGs/tag\\_html.asp](https://www.w3schools.com/TAGs/tag_html.asp)

<sup>4</sup>[https://www.w3schools.com/js/js\\_json\\_intro.asp](https://www.w3schools.com/js/js_json_intro.asp)

έρχεται ο κώδικας (url, url position), καθώς επίσης τα τμήματα κώδικα (code) και η σχετική θέση αυτών μέσα στην ιστοσελίδα (in page order). Τέλος, παρατηρούμε ότι σε αυτό το σημείο συλλέγεται πληροφορία ανεξάρτητα αν είναι χρήσιμη ή όχι. Για παράδειγμα, με μία γρήγορη ματιά το πρώτο τμήμα κώδικα φαίνεται να αναπαριστά χρήσιμη πληροφορία σε αντίθεση με το δεύτερο που δε δείχνει να έχει κάτι να προσφέρει.

```

1   {
2     "url": "https://www.tutorialspoint.com/javaexamples/applet_sound.htm"
3     "url position": 18,
4     "query": "how to play audio file",
5     "segments": [
6       {
7         "code": "\nimport java.applet.*;\nimport java.awt.*;\n\nimport java.awt.event.*;\n\n\npublic class PlaySoundApplet extends Applet implements ActionListener {\n\n    Button play,stop;\n    AudioClip audioClip;\n\n    public void init() {\n        play = new Button(\" Play in Loop \");\n        add(play);\n        play.addActionListener(this);\n\n        stop = new Button(\" Stop \");\n        add(stop);\n\n        stop.addActionListener(this);\n\n        audioClip = getAudioClip(getCodeBase(), \"Sound.wav\");\n    }\n\n    public void actionPerformed(ActionEvent ae) {\n\n        Button source = (Button)ae.getSource();\n\n        if (source.getLabel() == \" Play in Loop \") {\n            audioClip.play();\n        } else if(source.getLabel() == \" Stop \"){\n            audioClip.stop();\n        }\n    }\n\n    \"in_page_order\": 1\n},\n{\n    \"code\": \"\nView in Browser.\n\", \n    \"in_page_order\": 2\n}\n]
}

```

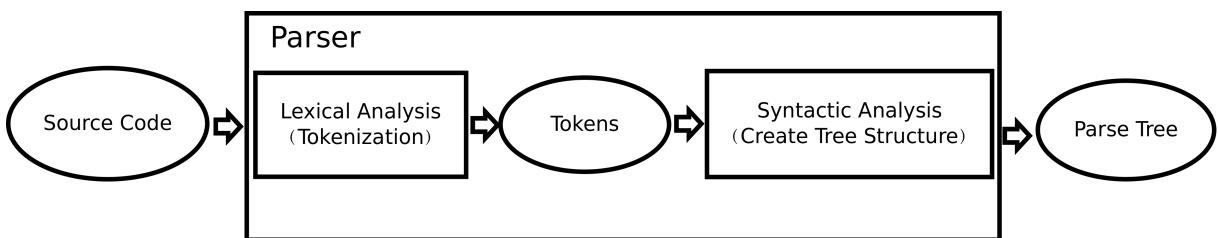
## 4.5 Parser

Σε αυτό το σημείο, έχουμε κατεβάσει τα δεδομένα μας από το Internet με βάση το ερώτημα του χρήστη, και τα έχουμε οργανώσει σε έναν αρχείο JSON. Στη συνέχεια, πρέπει να αφαιρέσουμε από τα δεδομένα αυτά που αποτελούν θόρυβο, και

έτσι δεν έχουν κάποια χρησιμότητα για το σύστημα μας. Για να το πετύχουμε αυτό ένας τρόπος είναι να χρησιμοποιήσουμε έναν Parser.

#### 4.5.1 Τι είναι ο Parser;

Ο Parser είναι ένα τμήμα λογισμικού το οποίο δέχεται ως είσοδο δεδομένα (συνήθως κείμενο) και παράγει ως έξοδο μια δομή δεδομένων (συνήθως μία ιεραρχική δομή όπως ένα αφηρημένο συντακτικό δένδρο) με βάση μία γραμματική ή ένα σύνολο κανόνων. Πριν από τον parser συνήθως υπάρχει ένας λεξικολογικός αναλυτής (lexical analyser), ο οποίος κατακερματοποιεί (tokenization) την είσοδο ώστε να μπορεί να επεξεργαστεί από τον parser (Σχήμα 4.3).



Σχήμα 4.3: Η δομή ενός τυπικού parser.

Στην περίπτωσή μας, θα πρέπει να χρησιμοποιηθεί ως γραμματική αυτή της γλώσσας για την οποία ενδιαφερόμαστε, δηλαδή της Java. Με τη βοήθεια του parser θα προσπαθήσουμε να εξάγουμε ορισμένες πληροφορίες που σχετίζονται με τη δομή και το περιεχόμενο του κώδικα.

#### 4.5.2 Αναφορά σε Δημοφιλείς Parsers για τη γλώσσα Java

Για την εργασία του parsing της γλώσσας προγραμματισμού Java, έχουν αναπτυχθεί αρκετοί parsers οι οποίοι έχουν ολοκληρωμένες γραμματικές και κανόνες. Συνοπτικά μερικοί από αυτούς είναι:

- *ANTLR* [37]. Πρόκειται ίσως για το πιο ανεπτυγμένο και σύνθετο εργαλείο στον τομέα αυτό. Μπορεί να χρησιμοποιηθεί για την ανάγνωση, επεξεργασία, εκτέλεση, ή μετάφραση δομημένου κειμένου. Το εργαλείο αυτό είναι ευρέως χρησιμοποιούμενο στην ακαδημαϊκή κοινότητα, και στη βιομηχανία για την ανάπτυξη εργαλείων συμβατών με πληθώρα γλωσσών προγραμματισμού και συστημάτων.

- *SrcML*<sup>5</sup>. Ένα ιδιαίτερα εύχρηστο εργαλείο, που έχει το πλεονέκτημα ότι δεν απαιτεί ο κώδικας να είναι μεταγλωττίσιμος, ενώ «συγχωρεί», θα λέγαμε, συντακτικά λάθη, όπως η παράλειψη αγκυλών ή παρενθέσεων. Δίνει τη δυνατότητα εξαγωγής σε *xml*<sup>6</sup>, ενώ είναι πιθανός ο ανασχηματισμός του αρχικού κώδικα από τη μορφή *xml* (διατηρεί δηλαδή όλα τα στοιχεία του κώδικα). Το μειονέκτημά του είναι ότι υποστηρίζει μέχρι και την έκδοση 1.5 της Java και συνεπώς εμφανίζονται προβλήματα στις πιο νέες εκδόσεις.
- *PMD*<sup>7</sup>. Πρόκειται για έναν αναλυτή πηγαίου κώδικα (source code analyzer). Είναι αρκετά γνωστός στη βιβλιογραφία και χρησιμοποιείται για διάφορες εργασίες (όπως το Code Clone Detection, ή η παραβίαση ορισμένων κανόνων σχεδίασης). Μία από τις υπηρεσίες του είναι και η εξαγωγή του AST ενός τμήματος κώδικα. Ωστόσο, η μορφή που εξάγεται είναι ιδιαίτερα πολύπλοκη και κατ' επέκταση, δύσκολα διαχειρίσιμη. Τέλος, δεν δίνει τη δυνατότητα προσπέλασης μη μεταγλωττίσιμων αρχείων.
- *Javalang*<sup>8</sup>. Αυτός είναι και ο parser που επιλέξαμε να ενσωματώσουμε στο σύστημα μας. Πρόκειται για μία βιβλιοθήκη, γραμμένη σε γλώσσα Python, η οποία παρέχει εργαλεία για πηγαίο κώδικα της Java 8. Πρόκειται για μία πολύ γρήγορη και αποτελεσματική υλοποίηση η οποία τηρεί όλες τις προδιαγραφές της γλώσσας Java όπως έχουν οριστεί από την Oracle [38]. Το μεγάλα πλεονεκτήματα της βιβλιοθήκης αυτής είναι η ευκολία και η απλότητα στη χρήση της, καθώς επίσης και οι εναλλακτικές δυνατότητες που προσφέρει στη προσπέλαση μη μεταγλωττίσιμων αρχείων.

### 4.5.3 Εφαρμογή του Parser στο σύστημα μας

Εδώ έχουμε να αντιμετωπίσουμε το ακόλουθο πρόβλημα. Σε αντίθεση με το μεγαλύτερο ποσοστό των συστημάτων RSSE, τα οποία συλλέγουν κώδικα από αποθήκες ανοιχτού λογισμικού και συνεπώς ολοκληρωμένα αρχεία κώδικα, το δικό μας σύστημα συλλέγει τμήματα κώδικα (*snippets*). Ως συνέπεια, το μεγαλύτερο ποσοστό

<sup>5</sup><http://www.srcml.org/>

<sup>6</sup><https://en.wikipedia.org/wiki/XML>

<sup>7</sup><http://pmd.sourceforge.net/>

<sup>8</sup><https://github.com/c2nes/javalang>

των παραδειγμάτων που συγκεντρώνομε δεν είναι μεταγλωττίσιμα. Από ένα μεταγλωττίσιμο αρχείο είναι εύκολο να παραχθεί το αφηρημένο συντακτικό του δένδρο (Abstract Syntax Tree - AST) και μέσω αυτού να αντληθεί πολύ χρήσιμη πληροφορία. Δυστυχώς, στην δική μας περίπτωση δεν έχουμε αυτή την «πολυτέλεια».

Για να αντιμετωπίσουμε το παραπάνω πρόβλημα χρειάζεται να διασπάσουμε την πληροφορία μας σε μικρότερα τμήματα και μέσω ενός Parser της γλώσσας Java να επεξεργαστούμε αυτά. Είναι προφανές ότι η διαδικασία αυτή θα μας βοηθήσει να απομονώσουμε και τα χρήσιμα δεδομένα από το θόρυβο μιας και αυτά που δεν αποτελούν κώδικα σε Java δεν μπορούν να περάσουν επιτυχώς από τον Parser.

```

1 // Write to CSV file
2 String csv = "C:\\output.csv";
3 CSVWriter writer = new CSVWriter(new FileWriter(csv));
4 String [] country = "India#China#United States".split("#");
5 writer.writeNext(country);
6 writer.close();

```

Για παράδειγμα ας υποθέσουμε ότι έχουμε το παραπάνω τμήμα κώδικα. Με μία γρήγορα ματιά παρατηρούμε ότι το συγκεκριμένο snippet δεν είναι μεταγλωττίσιμο. Ωστόσο, εξετάζοντάς το ανά γραμμή κώδικα, και με τη βοήθεια του Parser, μπορούμε να συγκεντρώσουμε χρήσιμη πληροφορία. Συγκεκριμένα, σε αυτό το snippet μπορούμε να κρατήσουμε ότι υπάρχουν τρεις κλήσεις μεθόδων (API calls), των split(), writeNext(), και close(). Με αντίστοιχο τρόπο, μπορούμε να τροφοδοτήσουμε όλα τα τμήματα κώδικα που έχουμε κατεβάσει και να εμπλουτίσουμε το JSON αρχείο μας με πληροφορίες που μας δίνει ο Parser μας.

Για παράδειγμα, το τμήμα από το αρχείο JSON που είδαμε στον Downloader στην περίπτωση αυτή θα ανανεωνόταν όπως δείχνεται παρακάτω. Παρατηρούμε ότι για κάθε τμήμα κώδικα έχουν προστεθεί οι κλήσεις μεθόδων που εμφανίζονται σε αυτό (API calls).

```

1 {
2     "url": "https://www.tutorialspoint.com/javaexamples/applet_sound.htm"
3     "url position": 18,
4     "query": "how to play audio file",
5     "segments": [

```

```

6      {
7          "code": "\nimport java.applet.*;\nimport java.awt.*;\n\nimport java.awt.event.*;\n\npublic class PlaySoundApplet extends Applet implements ActionListener {\n    \n    Button play,stop;\n    AudioClip audioClip;\n\n    public void init() {\n        play = new Button(\" Play in Loop \");\n        add(play);\n        play.addActionListener(this);\n\n        stop = new Button(\" Stop \");\n        add(stop);\n        stop.addActionListener(this);\n\n        audioClip = getAudioClip(getCodeBase(), \"Sound.wav\");\n    }\n\n    public void actionPerformed(ActionEvent ae) {\n        \n        Button source = (Button)ae.getSource();\n\n        if (source.getLabel() == \" Play in Loop \") {\n            \n            audioClip.play();\n        } else if(source.getLabel() == \" Stop \"){\n            \n            audioClip.stop();\n        }\n    }\n\n    \"in_page_order\": 1,\n    \"API calls\": {\n        \"add\": true,\n        \"addActionListener\": true,\n        \"getAudioClip\": false,\n        \"play\": true,\n        \"stop\": true\n    },\n},\n{\n    \"code\": \"\nView in Browser.\n\", \n    \"in_page_order\": 2,\n    \"API calls\": {}\n}\n]\n}

```

## 4.6 API Miner

Στο σημείο αυτό έχουμε καταφέρει να διαχωρίσουμε ποια από τα δεδομένα μας αποτελούν πηγαίο κώδικα, και επίσης, να συγκεντρώσουμε πληροφορίες για τα API calls που υπάρχουν μέσα σε αυτά.

Σαν επόμενο στάδιο, θα προσπαθήσουμε να απομονώσουμε αρχεία που εμπεριέχουν ασυνήθιστο κώδικα. Για να το πετύχουμε αυτό θα κάνουμε τις ακόλουθες υπόθεσεις.

**Τυπόθεση 1:** Σημαντικά έργα λογισμικού κάνουν χρήση ευρέως αποδεκτών και υψηλής ποιότητας APIs.

Ως σημαντικό έργο λογισμικού θεωρείται κάποιο που έχει υψηλή επισκεψιμότητα σε μία αναγνωρισμένη αποθήκη λογισμικού. Συνήθως, δημοφιλή έργα λογισμικού έχουν καλή ποιότητα κώδικα [39].

**Τυπόθεση 2:** Ένα τμήμα κώδικα που χρησιμοποιεί API calls όμοια με σημαντικά έργα λογισμικού, εμπεριέχει και το ίδιο καλής ποιότητας API calls.

Με αυτήν την υπόθεση θέλουμε να απομονώσουμε τμήματα κώδικα που χρησιμοποιούν API calls τα οποία δεν εμφανίζονται συχνά σε μεγάλα και αναγνωρισμένα έργα λογισμικού. Συνήθως, ο προγραμματιστής που αναζητά πως θα υλοποιήσει μία εργασία, ενδιαφέρεται για τις πλέον αποτελεσματικές και γενικές λύσεις και όχι για μία λύση που υλοποιήθηκε πρόχειρα από κάποιον ώστε να υλοποιεί κάτι πολύ συγκεκριμένο. Άλλωστε, η συγγραφή κώδικα από καλά δομικά στοιχεία οδηγεί σε πιο ευανάγνωστα και εύκολα συντηρήσιμα έργα λογισμικού.

#### 4.6.1 Η αποθήκη Ανοιχτού Λογισμικού GitHub

Στις μέρες μας, ολοένα και περισσότεροι μηχανικοί και εταιρίες υιοθετούν την ιδέα του ανοιχτού λογισμικού<sup>9</sup> (open-source software). Ως λογικό επακόλουθο, έχουν προκύψει πολλές υπηρεσίες που προσφέρουν διαδικτυακή στέγη για το open-source λογισμικό. Μία από τις μεγαλύτερες, αν όχι η μεγαλύτερη, είναι το GitHub<sup>10</sup>, το οποίο αριθμεί εκατομμύρια έργα λογισμικού σε όλες τις γλώσσες προγραμματισμού.



Σχήμα 4.4: Η αποθήκη ανοικτού λογισμικού GitHub, ίσως η μεγαλύτερη και δημοφιλέστερη αποθήκη λογισμικού αυτή τη στιγμή στο Διαδίκτυο.

To GitHub, πέρα από τη γραφική διεπαφή του, παρέχει ένα πολύ σύγχρονο και εύχρηστο API για τους developers<sup>11</sup>. Βέβαια, το συγκεκριμένο API, με σκοπό να μπορεί να εξυπηρετεί όλους τους πελάτες με μία αξιόλογη ποιότητα υπηρεσιών, θέτει αρκετούς περιορισμούς. Οι πιο σημαντικοί από αυτούς είναι το σχετικά χαμηλό

<sup>9</sup>[https://en.wikipedia.org/wiki/Open-source\\_software](https://en.wikipedia.org/wiki/Open-source_software)

<sup>10</sup><https://github.com/>

<sup>11</sup><https://developer.github.com/v3/>

όριο των αιτημάτων που επιτρέπει στον κάθε χρήστη ανά λεπτό (αυτή τη στιγμή το Όριο έχει τεθεί στα 30 αιτήματα ανά λεπτό), και το γεγονός ότι δεν επιτρέπει την αναζήτηση κώδικα απευθείας σε όλα τα αποθετήρια (repositories) λογισμικού. Συγκεκριμένα, η αναζήτηση κώδικα περιορίζεται μόνο στο συγκεκριμένο αποθετήριο που θα ορίσει ο χρήστης, ή στον κώδικα που ανήκει σε κάποιον συγκεκριμένο χρήστη του GitHub. Καταλαβαίνει κανείς ότι αν επιλέγαμε να χρησιμοποιήσουμε το GitHub ως μηχανή αναζήτησης κώδικα, αυτό θα αποτελούσε σημαντική τροχοπέδη για το σύστημα μας.

Εντούτοις, το GitHub εξακολουθεί να αποτελεί σημαντική πηγή πληροφορίας για το σύστημα μας. Ειδικότερα, το GitHub στεγάζει μερικά από τα δημοφιλέστερα έργα λογισμικού στη γλώσσα Java, και ο στόχος μας είναι να μελετήσουμε αυτά ώστε να βρούμε πληροφορίες για τα APIs που χρησιμοποιούν. Έτσι, μέσω του GitHub API κατεβάζουμε τα χίλια πιο δημοφιλή έργα λογισμικού στη γλώσσα Java (με βάση το πλήθος αστεριών-stars που αυτά έχουν) και τα αποθηκεύουμε τοπικά. Η διαδικασία του κατεβάσματος των αποθετηρίων είναι εύκολο να επιτευχθεί μέσω αιτημάτων HTTP, τα οποία περιγράφονται αναλυτικά στο API του GitHub<sup>12</sup>. Για την υλοποίηση στο σύστημα μας, επιλέχθηκε για την εργασία αυτή, η βιβλιοθήκη Requests<sup>13</sup> της Python.

Εφόσον έχουμε συγκεντρώσει τα έργα λογισμικά τοπικά, μπορούμε να εξάγουμε από αυτά τα αρχεία πηγαίου κώδικα και να τα επεξεργαστούμε. Να σημειωθεί ότι μόνο τα αρχεία πηγαίου κώδικα φτάνουν σε μέγεθος περίπου 4.2 GB. Είναι φανερό λοιπόν, ότι η επεξεργασία αυτή δεν αποτελεί διαδικασία που μπορεί να επαναλαμβάνεται κάθε φορά που εκκινεί το σύστημα μας. Ωστόσο, αυτό δεν αποτελεί ιδιαίτερο πρόβλημα μιας και έχοντας επεξεργαστεί τόσο μεγάλο όγκο πληροφορίας στο τέλος της διαδικασίας, θα έχουμε ήδη καλύψει το μεγαλύτερο ποσοστό των περιπτώσεων.

## 4.6.2 Κατασκευή ενός Λεξιλογίου από APIs

Ως επόμενο βήμα πρέπει να επεξεργαστούμε την πληροφορία που ανακτήσαμε από τα αποθετήρια του GitHub. Για το σκοπό αυτό, θα επιστρατεύσουμε και πάλι

<sup>12</sup><https://developer.github.com/v3/repos/contents/#get-archive-link>

<sup>13</sup><http://docs.python-requests.org/en/master/>

τον parser μας. Στην τρέχουσα περίπτωση τα πράγματα είναι πιο απλά από πριν μιας και τώρα έχουμε να κάνουμε με ολοκληρωμένα αρχεία πηγαίου κώδικα. Αυτό σημαίνει ότι τώρα μπορούμε απευθείας, με τη βοήθεια του parser, να παράγουμε το AST, και μέσω αυτού να συλλέξουμε την απαραίτητη πληροφορία.

Αφού συγκεντρώσουμε όλα τα API calls, πρέπει να τα οργανώσουμε σε έναν ολοκληρωμένο κατάλογο ώστε να μπορούμε στη συνέχεια να τα συγχρίνουμε με τα API calls που μας απασχολούν για το τρέχον ερώτημα. Το τελικό λεξιλόγιο που έχουμε σχηματίσει το αποθηκεύουμε έτσι ώστε να μην απαιτείται κάθε φορά η παραπάνω διαδικασία όταν πρέπει να συγχρίνουμε τα API calls των τμημάτων κώδικα μας με αυτά του λεξιλογίου.

Γενικά, με την παραπάνω σύγκριση μπορούμε να υπολογίσουμε τι ποσοστό των APIs ενός τμήματος κώδικα συναντάται συχνά. Ως προεπιλεγμένο κατώφλι, έχει οριστεί το ποσοστό του 75%. Δηλαδή, ένα τμήμα κώδικα που λιγότερο από το 75% των συνολικών API calls συναντώνται επίσης και σε υψηλής ποιότητας έργων της Java, είναι ανεπιθύμητο. Τέλος, σημειώνεται ότι ο σχηματισμός του λεξιλογίου, η επεξεργασία του και η διαχείριση του γίνεται με τη βοήθεια της βιβλιοθήκης *gensim*<sup>14</sup> [40], η οποία παρέχει αποτελεσματικές και γρήγορες λύσεις σε προβλήματα που έχουν να κάνουν με το Text Mining<sup>15</sup>.

## 4.7 Readability Evaluator

Στην ενότητα αυτή θα ορίσουμε την έννοια της αναγνωσιμότητας στον κώδικα, και στη συνέχεια θα περιγράψουμε πως μέσω τεχνικών μηχανικής μάθησης<sup>16</sup> (machine learning), μπορούμε να εκπαιδεύσουμε ένα μοντέλο ώστε να την προβλέπει.

### 4.7.1 Κατανοώντας τη μετρική της Αναγνωσιμότητας Κώδικα (Code Readability)

Ορίζουμε την αναγνωσιμότητα (*readability*) ως την ανθρώπινη κρίση για το κατά πόσο είναι εύκολη ή μη η κατανόηση ενός κειμένου [1]. Η αναγνωσιμότητα ενός

<sup>14</sup><https://radimrehurek.com/gensim/index.html>

<sup>15</sup>[https://en.wikipedia.org/wiki/Text\\_mining](https://en.wikipedia.org/wiki/Text_mining)

<sup>16</sup>[https://en.wikipedia.org/wiki/Machine\\_learning](https://en.wikipedia.org/wiki/Machine_learning)

προγράμματος σχετίζεται με τη διατηρισιμότητα του, και συνεπώς αποτελεί κύριο παράγοντα για την καθολική ποιότητα του λογισμικού. Τυπικά, η συντήρηση του λογισμικού στοιχίζει το 70% του συνολικού του κόστους σε όλο τον κύκλο ζωής του [41]. Άλλοι ερευνητές έχουν επισημάνει πως το έργο της αναγνωσης κώδικα είναι το πιο χρονοβόρο κομμάτι από όλες τις δραστηριότητες συντήρησης [42], [43], [44]. Η αναγνωσιμότητα είναι τόσο σημαντική, που στην πραγματικότητα, πολλές σοβαρές εταιρίες συμπεριλαμβάνουν στάδια κατά την ανάπτυξη λογισμικού, όπου αναλαμβάνουν να τροποποιήσουν τον ήδη λειτουργικό κώδικα σε πιο αναγνώσιμο (refactoring).

Υποθέτουμε ότι ακόμη και ένας αρχάριος προγραμματιστής έχει ενστικτωδώς μία αντίληψη εννοιών όπως η στοίχιση κώδικα, η επιλογή ονομάτων μεταβλητών, η ύπαρξη σχολίων, και γνωρίζει ότι τέτοιες έννοιες είναι πιθανό να παιξουν ρόλο στην αναγνωσιμότητα. Ωστόσο, είναι σημαντικό να επισημάνουμε ότι η αναγνωσιμότητα δεν είναι το ίδιο με την πολυπλοκότητα. Ενώ οι μετρικές για την πολυπλοκότητα λαμβάνουν υπόψιν το μέγεθος των κλάσεων και των μεθόδων, και τις αλληλεπιδράσεις μεταξύ αυτών, η αναγνωσιμότητα βασίζεται κυρίως σε τοπικούς, γραμμή προς γραμμή, παράγοντες.

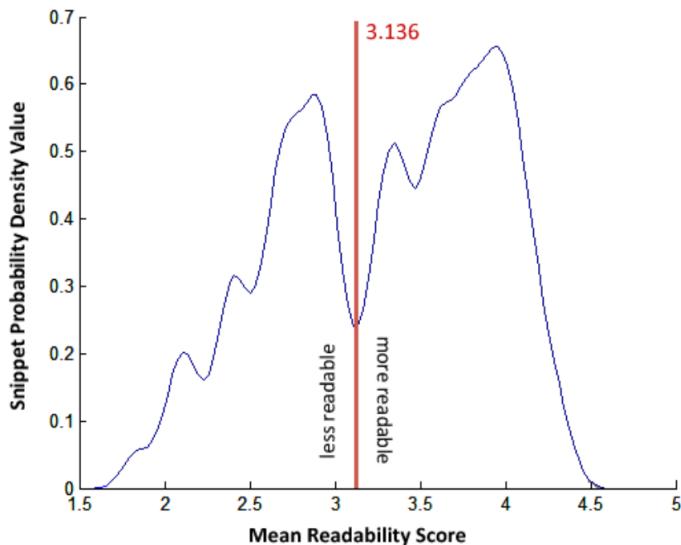
#### 4.7.2 Έρευνα για τους παράγοντες που επηρεάζουν την αναγνωσιμότητα

Ο Raymond Buse, με την επίβλεψη του Westley Weimer, στα πλαίσια της διδακτορικής του διατριβής ([45], [1]) διεξήγαγε την ακόλουθη έρευνα, με σκοπό τον προσδιορισμό των παραγόντων που επηρεάζουν την αναγνωσιμότητα του κώδικα.

Αρχικά, συγκέντρωσε 100 διαφορετικά τμήματα κώδικα από πέντε έργα ανοιχτού λογισμικού της γλώσσας Java. Έπειτα, ζήτησε από 120 συμμετέχοντας να βαθμολογήσουν τα παραπάνω τμημάτων κώδικα από το 1 έως το 5. Επίσης, τους δόθηκαν οι οδηγίες ότι βαθμολογίες κοντά στο 5 πρέπει να χρησιμοποιηθούν για τα "περισσότερο αναγνώσιμα" τμήματα κώδικα, ενώ βαθμολογίες κοντά στο 1 για τα "λιγότερο αναγνώσιμα". Μια βαθμολογία ίση με 3 συμβολίζει "ουδετερότητα".

Η ανάλυση που προέκυψε από την παραπάνω έρευνα επιβεβαιώνει την ευρέως γνωστή αντίληψη ότι οι προγραμματιστές συμφωνούν σε μεγάλο βαθμό στο πως μοιάζει ο εύκολα αναγνώσιμος κώδικας. Τα αποτελέσματα της έρευνας καθώς και

το dataset είναι διαθέσιμα για το κοινό στο Διαδίκτυο<sup>17</sup>.



Σχήμα 4.5: Κατανομή των μέσων βαθμολογιών ως προς την αναγνωσιμότητα για όλα τα τμημάτων κώδικα [1]. Η μορφή της κατανομής υποδεικνύει ένα σημείο αποκοπής, το οποίο μας βοηθά να εκπαιδεύσουμε έναν δυαδικό ταξινομητή. Η καμπύλη αναπαριστά την πυκνότητα πιθανότητας της κατανομής.

### 4.7.3 Το μοντέλο της Αναγνωσιμότητας Κώδικα

Ενώ οι συμμετέχοντες στην έρευνα του R. Buse έδειξαν να συμφωνούν ως προς την αναγνωσιμότητα των τμημάτων κώδικα που τους δόθηκαν να αξιολογήσουν, τα χαρακτηριστικά που κάνουν τον κώδικα εύκολο αναγνώσιμο δεν φαίνονται ξεκάθαρα. Για το λόγο αυτό, συνθέτουμε ένα σύνολο από χαρακτηριστικά τα οποία μπορούν να εντοπιστούν στατικά από ένα τμήμα κώδικα. Επιλέγουμε χαρακτηριστικά τα οποία είναι σχετικά απλά, και διαισθητικά φαίνεται να έχουν κάποια επίδραση στην αναγνωσιμότητα. Είναι παράγοντες που σχετίζονται με τη δομή, την πυκνότητα, την λογική πολυπλοκότητα, το σχολιασμό, κ.α. Επίσης επισημαίνεται ότι κάθε χαρακτηριστικό είναι ανεξάρτητο του μεγέθους του τμήματος κώδικα, κάτι που είναι σημαντικό για τη θεωρία της αναγνωσιμότητας.

Κάθε χαρακτηριστικό μπορεί να εφαρμοστεί σε ένα αυθαίρετου μεγέθους τμήματος κώδικα, και συμβολίζει είτε μία μέση τιμή για κάθε γραμμή, είτε μία μέγιστη τιμή για όλες τις γραμμές. Για παράδειγμα, υπάρχει ένα χαρακτηριστικό το οποίο αντιπροσωπεύει το μέσο μήκος γραμμής στο τμήμα κώδικα, και ένα δεύτερο που

<sup>17</sup><http://www.arrestedcomputing.com/readability>

αντιπροσωπεύει το μέγιστο μήκος γραμμής για τον ίδιο κώδικα. Συνολικά, υπάρχουν 25 χαρακτηριστικά (Πίνακας 4.2), τα οποία δημιουργούν μία απεικόνιση από τα τμήματα κώδικα σε διανύσματα πραγματικών αριθμών, τα οποία είναι κατάλληλα για χρήση σε έναν αλγόριθμο μηχανικής μάθησης.

Πίνακας 4.2: Το σύνολο των χαρακτηριστικών που λαμβάνονται υπόψιν για τον υπολογισμό της αναγνωσιμότητας (readability) [45]. Το σύμβολο "#" διαβάζεται ως "το πλήθος των..."

Avg.	Max.	Feature Name
✓	✓	line length (# characters)
✓	✓	# identifiers
✓	✓	identifier length
✓	✓	indentation (preceding whitespace)
✓	✓	# keywords
✓	✓	# numbers
✓		# comments
✓		# periods
✓		# commas
✓		# spaces
✓		# parenthesis
✓		# arithmetic operators
✓		# comparison operators
✓		# assignments (=)
✓		# branches (if)
✓		# loops (for, while)
✓		# blank lines
	✓	# occurrences of any single character
	✓	# occurrences of any single identifier

#### 4.7.4 Εκπαίδευση του Μοντέλου

Για την εκπαίδευση του μοντέλου μας, αρχικά πρέπει να εξάγουμε από τα αξιολογημένα τμήματα κώδικα στην έρευνα του R. Buse, το σύνολο των χαρακτηριστικών του Πίνακα 4.2. Εφόσον το κάνουμε, καταλήγουμε σε ένα σύνολο 100 δειγμάτων (instances) επί 25 διαστάσεις (features). Στη συνέχεια, και με βάση το σημείο αποκοπής που εμφανίζεται στο Σχήμα 4.5, χωρίζουμε τα τμήματα κώδικα σε "*more readable*" και "*less readable*". Στην πρώτη κατηγορία αντιστοιχίζονται τα τμήματα κώδικα που έλαβαν μέση βαθμολογία μεγαλύτερη από 3.14, ενώ στη δεύτερη κατάτάσσονται τα υπόλοιπα. Στο παραπάνω έργο της οργάνωσης των δεδομένων χρησιμοποιήθηκε η βιβλιοθήκη pandas<sup>18</sup> της γλώσσας Python, η οποία αποτελεί ένα από

<sup>18</sup><http://pandas.pydata.org/talks.html>

τα πλέον εξελιγμένα εργαλεία στον τομέα της ανάλυσης δεδομένων [46], [47].



Σχήμα 4.6: Η βιβλιοθήκη Scikit-Learn της Python αποτελεί μία από τις πιο ολοκληρωμένες λύσεις στον τομέα της μηχανικής εκμάθησης.

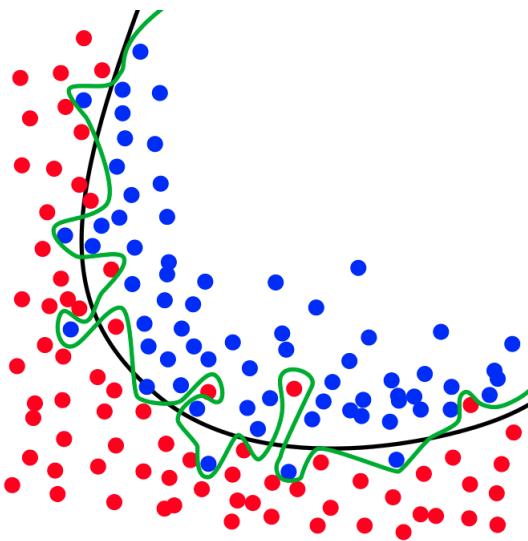
Επίσης, για την εκπαίδευση του μοντέλου, η οποία στην ουσία ισοδυναμεί με την εκπαίδευση ενός δυαδικού ταξινομητή (binary classifier), επιλέχθηκε η βιβλιοθήκη Scikit-Learn<sup>19</sup>, η οποία περιέχει μία πολύ πλούσια γκάμα αλγορίθμων ταξινόμησης, ενώ παράλληλα δεν υστερεί σε θέματα προεπεξεργασίας δεδομένων, τεχνικών βελτιστοποίησης κ.α. [48].

Στα πλαίσια της υλοποίησης του συστήματος μας, εκπαιδεύτηκαν τρεις διαφορετικοί ταξινομητές, οι οποίοι αποτελούν ένα συμβιβασμό ανάμεσα σε ταχύτητα και ακρίβεια αποτελεσμάτων. Σημειώνεται ότι ο τελικός χρήστης έχει την δυνατότητα να επιλέγει ποιος ταξινομητής θα χρησιμοποιηθεί στην εκάστοτε περίπτωση. Ωστόσο, πριν δούμε τους ταξινομητές έναν προς έναν, αξίζει να δώσουμε προσοχή σε μερικά λεπτά σημεία που αποτελούν κίνδυνο για την αξιοπιστία των αποτελεσμάτων μας, καθώς επίσης και σε κάποιες χρήσιμες μετρικές αξιολόγησης.

#### 4.7.4.1 Overfitting

Κατά τη διαδικασία εκπαίδευσης του μοντέλου, ιδιαίτερη προσοχή χρειάζεται για να αποφευχθούν κάποια σφάλματα. Ένα από αυτά τα σφάλματα είναι η υπερ-εκπαίδευση *overfitting* του μοντέλου, που συμβαίνει όταν μοντέλο γίνεται εξαιρετικά σύνθετο κατά τη διαδικασία της εκπαίδευσης (δηλ. έχει υπερβολικό αριθμό παραμέτρων σχετικά με τον πλήθος των παρατηρήσεων). Η συνέπεια σε αυτή την περίπτωση είναι το μοντέλο να προσαρμόζεται στις λεπτομέρειες και στο θόρυβο των δειγμάτων (Σχήμα 4.7) σε βαθμό που αυτό να επιδρά αρνητικά στην απόδοση του πάνω σε νέα, άγνωστα δεδομένα.

<sup>19</sup><http://scikit-learn.org/stable/index.html>



Σχήμα 4.7: Η πράσινη γραμμή αντιπροσωπεύει ένα μοντέλο το οποίο έχει υποστεί overfitting, ενώ η μαύρη γραμμή ένα σωστά εκπαίδευμένο μοντέλο. Ενώ η πράσινη γραμμή ακολουθεί πολύ καλά τα δεδομένα εκπαίδευσης, είναι υπερβολικά εξαρτημένη από αυτά και έτσι οι προβλέψεις του μοντέλου αυτού να είναι χειρότερες σε άγνωστα δεδομένα από αυτές της μοντέλου της μαύρης γραμμής [2].

Για να αποφύγουμε το παραπάνω σφάλμα, μία κοινή τακτική είναι κατά την εποπτευόμενη μάθηση<sup>20</sup> (supervised learning) είναι ο διαχωρισμός των δεδομένων σε δύο υποσύνολα, εκπαίδευσης (train set) και ελέγχου (test set). Η εκπαίδευση του μοντέλου γίνεται με το σύνολο εκπαίδευσης και στη συνέχεια δοκιμάζεται η απόδοση του πάνω στα δεδομένα του συνόλου ελέγχου τα οποία δεν χρησιμοποιήθηκαν κατά τη διαδικασία εκπαίδευσης.

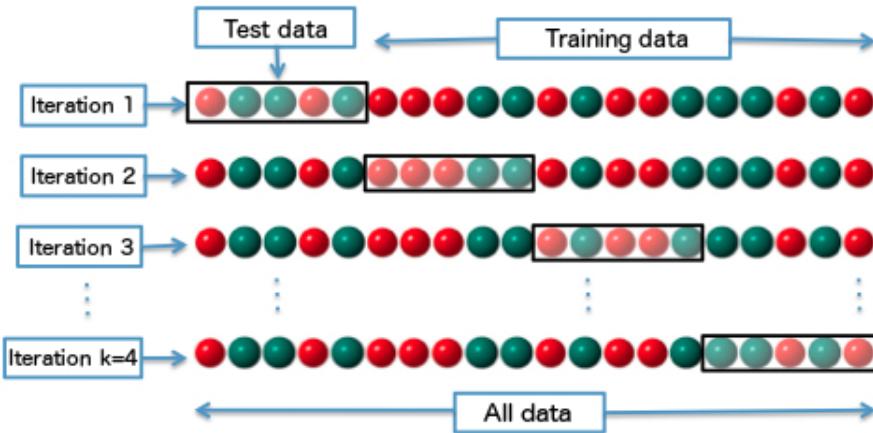
Οστόσο σε περιπτώσεις που δεν υπάρχουν αρκετά διαθέσιμα δεδομένα για την εκπαίδευση του μοντέλου, όπως και στη δική μας, η διατήρηση ενός μέρους από αυτά μόνο για την αξιολόγηση του μοντέλου μπορεί να είναι ασύμφορη. Η λύση για το πρόβλημα αυτό είναι μία διαδικασία, η οποία ονομάζεται *cross-validation*<sup>21</sup> [49, 50]. Κατά τη προσέγγιση αυτή (Σχήμα 4.8), το σετ εκπαίδευσης χωρίζεται σε  $k$  μικρότερα σετ ( $k$ -folds). Η ακόλουθη διαδικασία ακολουθείται για καθένα από τα  $k$  folds.

- Το μοντέλο εκπαιδεύεται χρησιμοποιώντας τα  $k - 1$  folds των δεδομένων εκπαίδευσης.
- Το μοντέλο αξιολογείται αξιολογείται στα εναπομείναντα δεδομένα (δηλ. το κομμάτι των δεδομένων που δεν συμπεριλήφθηκε στην εκπαίδευση χρησιμο-

<sup>20</sup>[https://en.wikipedia.org/wiki/Supervised\\_learning](https://en.wikipedia.org/wiki/Supervised_learning)

<sup>21</sup>[https://en.wikipedia.org/wiki/Cross-validation\\_\(statistics\)](https://en.wikipedia.org/wiki/Cross-validation_(statistics))

ποιείται για τον υπολογισμό μετρικών απόδοσης).



Σχήμα 4.8: Εκπαίδευση μοντέλου με  $k$ -fold cross-validation και  $k=4$  [3].

Η απόδοση του τελικού μοντέλου προκύπτει από τις μέσες τιμές που προέκυψαν από τις  $k$  επαναλήψεις τις παραπάνω διαδικασίας. Η προσέγγιση αυτή μπορεί να είναι υπολογιστική «ακριβή», αλλά δεν σπαταλάει δεδομένα, το οποίο είναι ένα τεράστιο πλεονέκτημα όταν το πλήθος των διαθέσιμων δειγμάτων είναι μικρό. Για το λόγο αυτό, η ρύθμιση των παραμέτρων στην εκπαίδευση των μοντέλων που θα δούμε στη συνέχεια έγινε με cross-validation και 10 folds.

#### 4.7.4.2 Μετρικές Αξιολόγησης

Για να μπορέσουμε να δημιουργήσουμε μια εικόνα για την απόδοση ενός μοντέλου ταξινόμησης χρειαζόμαστε μετρικές αξιολόγησης. Αν και υπάρχουν πολλές και διάφορες τέτοιου είδους μετρικές, εμείς θα ασχοληθούμε με αυτές που αξιολογούν δυαδικά προβλήματα ταξινόμησης. Ένα πολύ χρήσιμο εργαλείο για την ανάγκη αυτή είναι ο πίνακας σύγχυσης (*confusion matrix*). Η τυπική μορφή του φαίνεται στον Πίνακα 4.3. Στον πίνακα αυτόν εμφανίζονται 4 διαφορετικά είδη κελιών. Οι σημασία αυτών είναι εξής:

- *True Negative (TN)*. Ως TN χαρακτηρίζεται μία πρόβλεψη όταν το δείγμα ανήκει πραγματικά στην κλάση 0 (Negative) και ο ταξινομητής το έχει κατατάξει στην κλάση 0 (Negative). Άρα έχουμε σωστή πρόβλεψη.
- *True Positive (TP)*. Ως TP χαρακτηρίζεται μία πρόβλεψη όταν το δείγμα ανήκει πραγματικά στην κλάση 1 (Positive) και ο ταξινομητής το έχει κατατάξει στην κλάση 1 (Positive). Άρα έχουμε σωστή πρόβλεψη.

- *False Negative (FN)*. Ως FN χαρακτηρίζεται μία πρόβλεψη όταν το δείγμα ανήκει πραγματικά στην κλάση 1 (Positive) και ο ταξινομητής το έχει κατατάξει στην κλάση 0 (Negative). Άρα έχουμε λάθος πρόβλεψη.
- *False Positive (FP)*. Ως FP χαρακτηρίζεται μία πρόβλεψη όταν το δείγμα ανήκει πραγματικά στην κλάση 0 (Negative) και ο ταξινομητής το έχει κατατάξει στην κλάση 1 (Positive). Άρα έχουμε λάθος πρόβλεψη.

Πίνακας 4.3: Μορφή πίνακα σύγχυσης (confusion matrix) σε δυαδική ταξινόμηση (binary classification)

	Predicted: 0	Predicted: 1
Actual: 0	TN	FP
Actual: 1	FN	TP

Ο πίνακας σύγχυσης «κρύβει» αρκετή ενδιαφέρουσα πληροφορία και μετρικές που μπορούν εξαχθούν. Οι πιο σημαντικές από αυτές είναι:

- *Accuracy*. Η μετρική αυτή συμβολίζει τη συχνότητα των σωστών προβλέψεων του classifier.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- *Recall*. Είναι γνωστή και ως "Sensitivity" ή "True Positive Rate". Εκφράζει το ποσοστό των σωστά θετικών προβλέψεων (κλάση "1") προς το πλήθος των δειγμάτων που ανήκουν όντως στην κλάση "1".

$$\text{Recall} = \frac{TP}{TP + FN}$$

- *Precision*. Δηλώνει το ποσοστό των προβλέψεων που σωστά κατατάχθηκαν στη θετική κλάση "1".

$$\text{Precision} = \frac{TP}{TP + FP}$$

- *F1-Score*. Αποτελεί τον αρμονικό μέσο<sup>22</sup> των μετρικών precision και recall.

$$F1 = 2 \cdot \frac{1}{\frac{1}{precision} + \frac{1}{recall}} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

<sup>22</sup>[https://en.wikipedia.org/wiki/Harmonic\\_mean#Harmonic\\_mean\\_of\\_two\\_numbers](https://en.wikipedia.org/wiki/Harmonic_mean#Harmonic_mean_of_two_numbers)

#### 4.7.4.3 K-Nearest Neighbors

Η αρχή πίσω από τη μέθοδο των K-Nearest Neighbors (KNN) [51] είναι να βρεθεί ένας προκαθορισμένος αριθμός δειγμάτων τα οποία βρίσκονται σε κοντινότερη απόσταση από το υπό πρόβλεψη σημείο (είναι δηλαδή «γειτονες»), και να προβλεφθεί η κλάση του από αυτά. Ο αριθμός των δειγμάτων-γειτόνων που απαιτούνται είναι μία σταθερά  $K$ , την οποία την καθορίζει ο χρήστης. Γενικά, μπορεί να χρησιμοποιηθεί οποιαδήποτε μετρική απόστασης. Η πιο συνηθισμένη επιλογή είναι η Ευκλείδεια απόσταση. Παρά την απλότητα της, η μέθοδος αυτή είναι επιτυχής σε μεγάλο αριθμό προβλημάτων ταξινόμησης.

---

#### Αλγόριθμος 1: Ψευδοκώδικας KNN classification

---

**Data:**  $\mathbf{X}$ : training data,  $\mathbf{Y}$ : class labels of  $\mathbf{X}$ ,  $x$ :unknown sample

```

1 Classify( $\mathbf{X}$ ,  $\mathbf{Y}$ ,  $x$ )
2 for  $i \leftarrow 0$  to  $m$  do
3   | Compute distance  $d(X_i, x)$ ;
4 end
5 Compute set  $I$  containing indices for the  $k$  smallest distances  $d(X_i, x)$ ;
6 return majority label for  $Y_i$  where  $i \in I$ 
```

---

Ο ψευδοκώδικας του ταξινομητή φαίνεται στον Αλγόριθμο 6, ενώ στη συνέχεια παραθέτουμε τις επιλογές που δίνονται όσον αφορά τις παραμέτρους:

- *K Neighbors*. Ο αριθμός των πλησιέστερων γειτόνων (δειγμάτων) που απαιτούνται για να γίνει η πρόβλεψη.
- *Weight Function*. Η συνάρτηση βάρους η οποία θα χρησιμοποιηθεί στην πρόβλεψη. Μπορεί να είναι ομοιόμορφη, εάν επιθυμούμε όλα τα δείγματα να επηρεάζουν το ίδιο, ή αντιστρόφως ανάλογη της απόστασης, εάν επιθυμούμε τα πλησιέστερα δείγματα να επηρεάζουν περισσότερο την πρόβλεψη από αυτά που βρίσκονται πιο μακριά.
- *Distance Metric*. Η μετρική της απόστασης η οποία θα χρησιμοποιηθεί. Για δύο σημεία  $\mathbf{p} = (p_1, p_2, \dots, p_n)$  και  $\mathbf{q} = (q_1, q_2, \dots, q_n)$  σε καρτεσιανές συντεταγμένες, η μεταξύ τους απόσταση για διάφορες μετρικές, δίνεται από τους παρακάτω μαθηματικούς τύπους:

– Euclidean:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (q_i - p_i)^2}$$

– Squared Euclidean:

$$d^2(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^n (p_i - q_i)^2$$

– Manhattan:

$$d(\mathbf{p}, \mathbf{q}) = |p_1 - q_1| + |p_2 - q_2| + \dots + |p_n - q_n| = \sum_{i=1}^n (|p_i - q_i|)$$

– Chebyshev:

$$d(\mathbf{p}, \mathbf{q}) = \max(|p_1 - q_1|, |p_2 - q_2|, \dots, |p_n - q_n|)$$

– Minkowski:

$$d(\mathbf{p}, \mathbf{q}) = \left( \sum_{i=1}^n |p_i - q_i|^m \right)^{\frac{1}{m}}, m \geq 1$$

Γενικά, βλέπουμε ότι ακόμα και μια απλή μέθοδος όπως αυτή, έχει αρκετές παραμέτρους που μπορούν να ρυθμιστούν ώστε να βελτιωθούν οι προβλέψεις του μοντέλου μας. Καταλαβαίνει κανείς ότι ο χειροκίνητος έλεγχος όλων των συνδυασμών των παραμέτρων, αν όχι αδύνατος, είναι μια πολύ κουραστική και χρονοβόρα διαδικασία. Ευτυχώς η βιβλιοθήκη Scikit-Learn, για το σκοπό αυτό, παρέχει ευέλικτες λύσεις μέσω πακέτων της<sup>23</sup>.

Στη δική μας περίπτωση, έγιναν δοκιμές για διάφορες μετρικές απόστασης, καθώς και για διαφορετικές συναρτήσεις βάρους. Επειδή η παρουσίαση όλων των δυνατών συνδυασμών και των αποτελεσμάτων αυτών θα απαιτούσε μεγάλη έκταση, και συνεπώς θα καταντούσε κουραστική για τον αναγνώστη, παρουσιάζονται μόνο αποτελέσματα από συνδυασμούς παραμέτρων που τα αποτελέσματα τους ήταν καλύτερα σε σχέση με άλλους. Έτσι, στο Σχήμα 4.9, παρουσιάζεται η απόδοση του μοντέλου με τις μετρικές *F1*, *Recall*, *Precision*, και *Accuracy*. Επίσης η συνάρτηση βάρους είναι με βάση την απόσταση και όχι ομοιόμορφη, μιας και διαπιστώθηκε ότι

<sup>23</sup>[http://scikit-learn.org/stable/modules/grid\\_search.html#grid-search](http://scikit-learn.org/stable/modules/grid_search.html#grid-search)

Πίνακας 4.4: Αποτελέσματα μετρικών ταξινόμησης για το KNN μοντέλο. Για την κάθε μετρική εμφανίζεται ο καλύτερος συνδυασμός παραμέτρων, καθώς και το αποτέλεσμα (score) που αυτός αποδίδει.

	Weight Function	K neighbors	Distance Metric	Score
F1	distance	11	manhattan	<b>0.79</b>
Recall	uniform	27	euclidean	<b>0.87</b>
Precision	uniform	6	manhattan	<b>0.85</b>
Accuracy	distance	11	manhattan	<b>0.75</b>

αυτή δίνει καλύτερα αποτελέσματα εν αρμονίᾳ με τις σχεδιαστικές μας παραδοχές.

Γενικά, παρατηρούμε ότι δεν υπάρχει ένας και μοναδικός συνδυασμός παραμέτρων που δίνει το «τέλειο» μοντέλο. Έτσι, πρέπει να γίνουν κάποιοι συμβιβασμοί ανάλογα με τις θέλουμε να πετύχουμε. Στο δικό μας σύστημα, δόθηκε περισσότερο βάρος στη μετρική *F1*, η οποία αποτελεί ένα συμψηφισμό των μεταβλητών Recall και Precision. Ο λόγος που παίρνουμε αυτή τη σχεδιαστική απόφαση είναι ότι θέλουμε να ανακτούμε μεγάλο ποσοστό από το συνολικό πλήθος των τμημάτων κώδικα που ανήκουν στα *more readable* (υψηλό recall), ενώ παράλληλα η πρόβλεψη αυτή να είναι και σωστή (υψηλό precision).

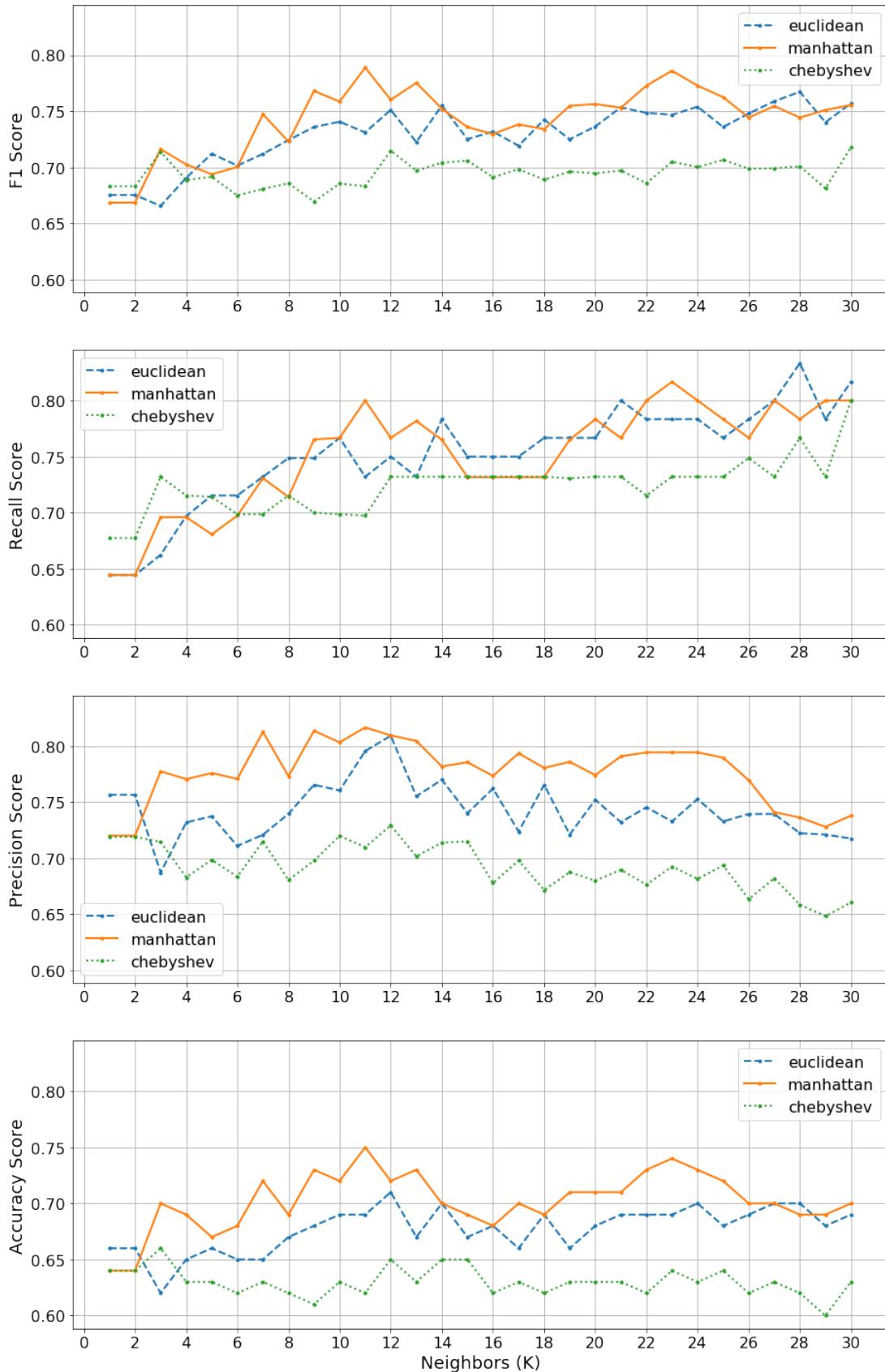
Τέλος, στον Πίνακα 4.4 φαίνονται οι συνδυασμοί παραμέτρων που επιτυγχάνουν τα καλύτερα αποτελέσματα για κάθε μετρική ταξινόμησης. Οπότε, δίνοντας βάρος όπως είπαμε στη μετρική *F1*, το τελικό μοντέλο ταξινομητή με αλγόριθμο *KNN* έχει 11 Neighbors, distance Weight Function, Manhattan Distance Metric ως παραμέτρους.

#### 4.7.4.4 Random Forests

Ο εκτιμητής που θα δούμε σε αυτήν την υποενότητα ονομάζεται *Random Forests*, και ανήκει σε μία ευρύτερη οικογένεια μεθόδων, αυτή των *ensemble methods*. Ο στόχος των μεθόδων αυτών είναι ο συνδυασμός προβλέψεων από διάφορους ανεξάρτητους εκτιμητές, έτσι ώστε να προκύψει ένας νέος εκτιμητής με βελτιωμένη γενικευσιμότητα και ευρωστία. Τα ensemble methods συνήθως διακρίνονται σε αυτά που συλλέγουν προβλέψεις από διάφορους ανεξάρτητους εκτιμητές και τις συνδυάζουν κατά μέσο όρο ώστε να προκύψει ένας νέος εκτιμητής με μικρότερες διακυμάνσεις (bagging), και σε αυτά που φτιάχνουν πολλούς εκτιμητές τον έναν διαδοχικά με βάση τα αποτελέσματα του προηγούμενου ώστε να μειωθεί η πόλωση (boosting).

Συγκεκριμένα ο εκτιμητής *Random Forests*, που λειτουργεί με λογική bagging ensemble, χρησιμοποιεί τις εκτιμήσεις από ένα πλήθος δένδρων απόφασης<sup>24</sup> (decision

<sup>24</sup><http://scikit-learn.org/stable/modules/tree.html>



Σχήμα 4.9: Στις παραπάνω γραφικές παραστάσεις παρουσιάζονται αποτελέσματα του μοντέλου που κάνει προβλέψεις χρησιμοποιώντας τον αλγόριθμο KNN, και διάφορες μετρικές απόστασης. Με τη σειρά από πάνω προς τα κάτω, εμφανίζονται αποτελέσματα για τις μετρικές F1, Recall, Precision, και Accuracy. Η συνάρτηση βάρους είναι με βάση την απόσταση (distance).

---

**Αλγόριθμος 2:** Ψευδοκώδικας Random Forests classification

---

**Data:** Training set  $S := (x_1, y_1), \dots, (x_n, y_n)$ , features  $F$ , trees in forest  $B$

```

1 Function RandomForest( $S, F$ )
2    $H \leftarrow 0$ 
3   for  $i \in 1, \dots, B$  do
4      $S^i \leftarrow$  A bootstrap sample from  $S$ 
5      $h_i \leftarrow$  RandomizedTreeLearn( $S^i, F$ )
6      $H \leftarrow H \cup \{h_i\}$ 
7   end
8   return  $H$ 
9 Function RandomizedTreeLearn( $S, F$ )
10  At each node:
11     $f \leftarrow$  very small subset of  $F$ 
12    Split on best feature in  $f$ 
13  return The learned tree

```

---

trees), ώστε να βελτιώσει την ακρίβεια στις προβλέψεις του και να περιορίσει το over-fitting [52]. Μερικές από τις παραμέτρους που έχει τη δυνατότητα να ρυθμίσει ο χρήστης για να πετύχει καλύτερα αποτελέσματα με αυτόν τον classifier περιγράφονται στα παρακάτω σημεία:

- *Number of Estimators.* Το πλήθος των δένδρων απόφασης που θα χρησιμοποιηθούν στη δημιουργία του τελικού εκτιμητή.
- *Criterion.* Η συνάρτηση με την οποία θα εκτιμηθεί η ποιότητα των προβλέψεων που προσφέρουν οι ανεξάρτητοι εκτιμητές (τα δένδρα απόφασης). Οι επιλογές είναι:
  - Gini impurity. Η μετρική αυτή αποτελεί δείγμα της λανθασμένης ταξινόμησης των δειγμάτων. Για να υπολογιστεί το gini impurity για ένα σύνολο δειγμάτων  $J$  κλάσεων, έστω  $i \in 1, 2, \dots, J$ , και ότι  $p_i$  είναι το ποσοστό των δειγμάτων που ανήκουν στην κλάση  $i$ .

$$I_G(p) = \sum_{i=1}^J p_i(1 - p_i) = \sum_{i=1}^J (p_i - p_i^2) = \sum_{i=1}^J p_i - \sum_{i=1}^J p_i^2 = 1 - \sum_{i=1}^J p_i^2$$

- Entropy. Η επιλογή αυτή αξιολογεί των διαχωρισμό των δειγμάτων που γίνεται, με βάση το κέρδος πληροφορίας (information gain). Αν  $p_1, p_2, \dots$  είναι τα κλάσματα που το άθροισμα τους είναι ίσο με 1, και αντιπροσωπεύουν το ποσοστό κάθε κλάσης σε κάθε διαχωρισμό των δένδρων

απόφρασης, τότε η εντροπία ισούται με:

$$H(T) = I_E(p_1, p_2, \dots, p_J) = - \sum_{i=1}^J p_i \log_2 p_i$$

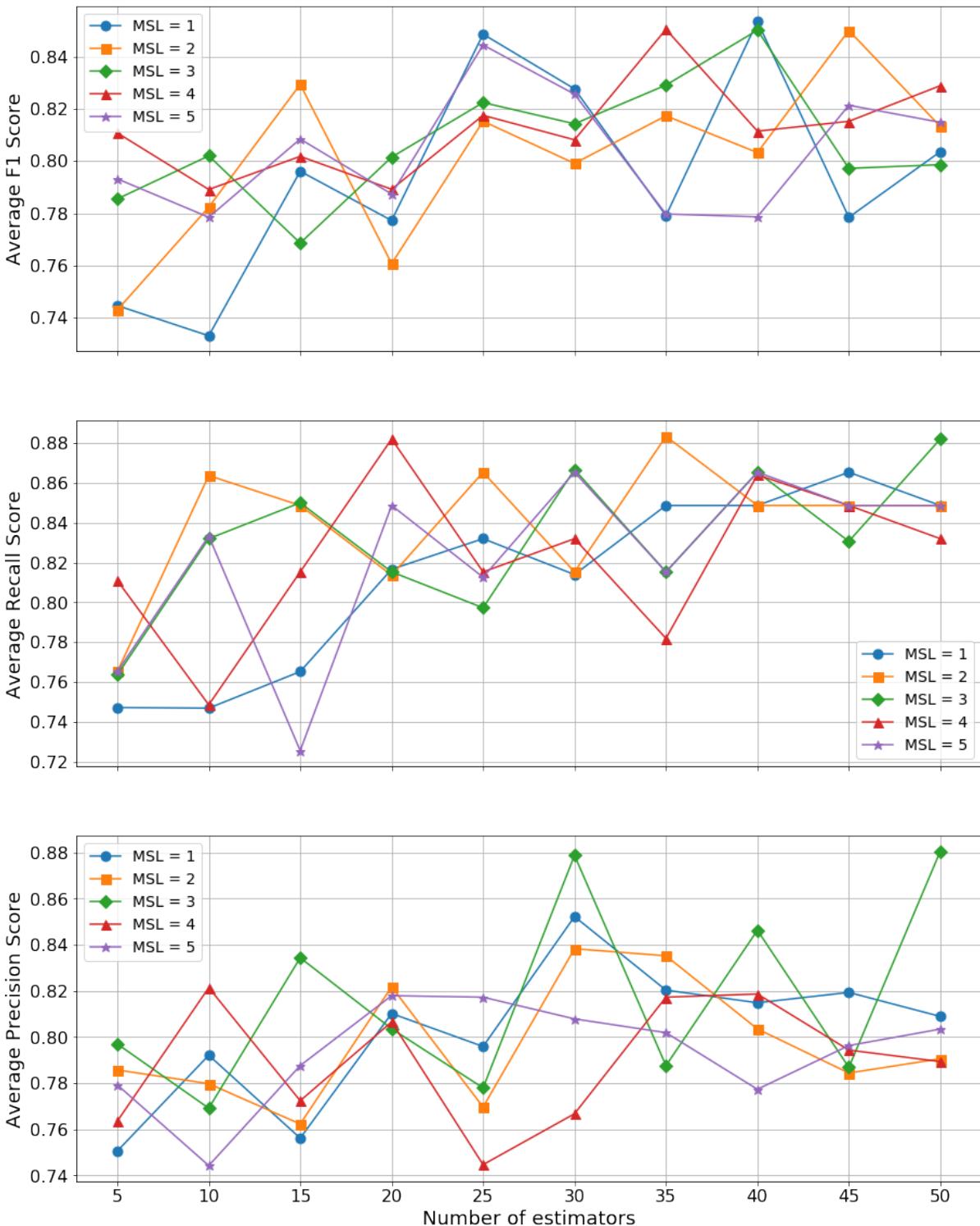
- *Minimum samples in leaf node.* Ο ελάχιστος αριθμός δειγμάτων που απαιτούνται ώστε ένας κόμβος να είναι τερματικός.
- *Minimum samples split.* Ο ελάχιστος αριθμός δειγμάτων που απαιτούνται ώστε ένας κόμβος να μπορεί να διαχωριστεί σε άλλος υποκόμβους.
- *Maximum Features.* Ο αριθμός των δειγμάτων που λαμβάνονται υπόψιν όταν εξετάζεται κάποιος διαχωρισμός.
- *Maximum Depth.* Το μέγιστο βάθος των δένδρων.

Οι παραπάνω παράμετροι είναι μερικοί από τους πιο σημαντικούς, ενώ υπάρχουν και άλλοι. Ωστόσο, ο βέλτιστος προσδιορισμός όλων των παραμέτρων πέρα από το ότι δεν βελτιώνει εξαιρετικά το μοντέλο, απαιτεί πολύ μεγάλο χρονικό διάστημα και υπολογιστική ισχύ για να επιτευχθεί. Για παράδειγμα, δοκιμές με συνδυασμούς τεσσάρων διαφορετικών παραμέτρων χρειάζονται αρκετές ώρες επεξεργασίας σε μία σύγχρονη *CPU*, ακόμη και με παραλληλοποίηση των εργασιών. Ακόμη, κάτι τέτοιο είναι επικίνδυνο και για την αξιοπιστία των αποτελεσμάτων καθώς μπορεί να υπερεκπαιδεύσει το μοντέλο μας (*over-fitting*).

Μετά από δοκιμές, καταλήξαμε σε μερικές παραμέτρους που επηρεάζουν θετικά το μοντέλο. Αποτελέσματα της διαδικασίας ρύθμισης των παραμέτρων απεικονίζονται στο Σχήμα 4.10. Οι συνδυασμοί των μεταβλητών για την επίτευξη των καλύτερων μετρικών ταξινόμησης συνοφίζονται στον Πίνακα 4.5. Δίνοντας προτεραιότητα στη μετρική *F1*, επιλέγουμε ως παραμέτρους για το τελικό μοντέλο, *Criterion Gini*, *Minimum Samples Leaf 4*, και *Number of Estimators 40*.

#### 4.7.4.5 AdaBoost

Στην υποενότητα αυτή έχουμε να κάνουμε πάλι με έναν εκτιμητή της οικογένειας ensemble, τον AdaBoost [53]. Ωστόσο, η κύρια αρχή πίσω από τον αλγόριθμο αυτόν διαφέρει από αυτή που είδαμε στον Random Forests. Εδώ ο στόχος είναι να εκπαιδεύσουμε διαδοχικά «αδύναμους» εκτιμητές (δηλ. μοντέλα που είναι ελαφρώς



Σχήμα 4.10: Στις παραπάνω γραφικές παραστάσεις παρουσιάζονται αποτελέσματα του μοντέλου που κάνει προβλέψεις χρησιμοποιώντας τον classifier RandomForests, για διάφορα πλήθη estimators (οριζόντιος άξονας), και διάφορα minimum samples leaf (MSL). Με τη σειρά από πάνω προς τα κάτω, εμφανίζονται αποτελέσματα για τις μετρικές F1, Recall, Precision. Η συνάρτηση εκτίμησης (criterion) είναι η συνάρτηση *gini impurity*.

Πίνακας 4.5: Αποτελέσματα μετρικών ταξινόμησης για το Random Forests μοντέλο. Για την κάθε μετρική εμφανίζεται ο καλύτερος συνδυασμός παραμέτρων, καθώς και το αποτέλεσμα (score) που αυτός αποδίδει.

	Criterion	Minimum samples leaf	Estimators	Score
F1	Gini	4	40	0.86
Recall	Gini	1	35	0.90
Precision	Gini	1	10	0.87
Accuracy	Entropy	2	35	0.84

καλύτερα από το να προβλέπουν εντελώς τυχαία, όπως μικρά δένδρα απόφασης) σε συνεχώς τροποποιημένες εκδόσεις των δεδομένων μας. Οι προβλέψεις από όλους συγκεντρώνονται και προστίθενται με κατάλληλα βάρη ώστε να προκύψει η τελική απόφαση. Οι τροποποιήσεις των δεδομένων σε κάθε επανάληψη (boosting) απαρτίζονται από την επιβολή βαρών  $w_1, w_2, \dots, w_N$  σε καθένα από τα δείγματα. Για κάθε επιτυχημένη επανάληψη, τα βάρη των δειγμάτων τροποποιούνται ατομικά, και ο αλγόριθμος εκμάθησης επανεφαρμόζεται στα νέα δείγματα. Σε κάθε βήμα, τα δείγματα που προβλέφθηκαν λανθασμένα από το βελτιωμένο (boosted) μοντέλο, αποκτούν αυξημένα βάρη, ενώ παράλληλα μειώνονται τα βάρη σε αυτά που έγινε σωστή πρόβλεψη. Όσο προχωρούν οι επαναλήψεις, τα δείγματα που είναι δύσκολο να προβλεφθούν έχουν όλο και μεγαλύτερη επιρροή. Συνεπώς, ο κάθε ακόλουθος αδύναμος εκτιμητής αναγκάζεται να επικεντρωθεί σε αυτά τα δείγματα.

---

### Αλγόριθμος 3: Απλοποιημένος φευδοκώδικας του αλγορίθμου AdaBoost

---

- 1 Set uniform example weights.
  - 2 **for** Each base learner **do**
  - 3     Train base learner with weighted sample.
  - 4     Test base learner on all data
  - 5     Set learner weight with weighted error
  - 6     Set example weights based on ensemble predictions.
  - 7 **end**
- 

Και εδώ ο χρήστης έχει επιλογές ως προς το πως θα ρυθμίσει τον AdaBoost classifier. Στα παρακάτω σημεία αναφέρονται εν συντομίᾳ οι επιλογές αυτές:

- *Base Estimator*. Ο εκτιμητής βάσης (π.χ. δένδρα απόφασης) πάνω στον οποίο θα χτιστεί ο βελτιωμένος (boosted) εκτιμητής.
- *Number of Estimators*. Το μέγιστο πλήθος των εκτιμητών στο οποίο σταματάνε οι επαναλήψεις του αλγορίθμου (boosting).

Πίνακας 4.6: Αποτελέσματα μετρικών ταξινόμησης για το AdaBoost μοντέλο. Για την κάθε μετρική εμφανίζεται ο καλύτερος συνδυασμός παραμέτρων, καθώς και το αποτέλεσμα (score) που αυτός αποδίδει.

	Estimators	Learning Rate	score
F1	160	0.6	<b>0.85</b>
Recall	160	0.6	<b>0.87</b>
Precision	160	0.5	<b>0.87</b>
Accuracy	160	0.6	<b>0.82</b>

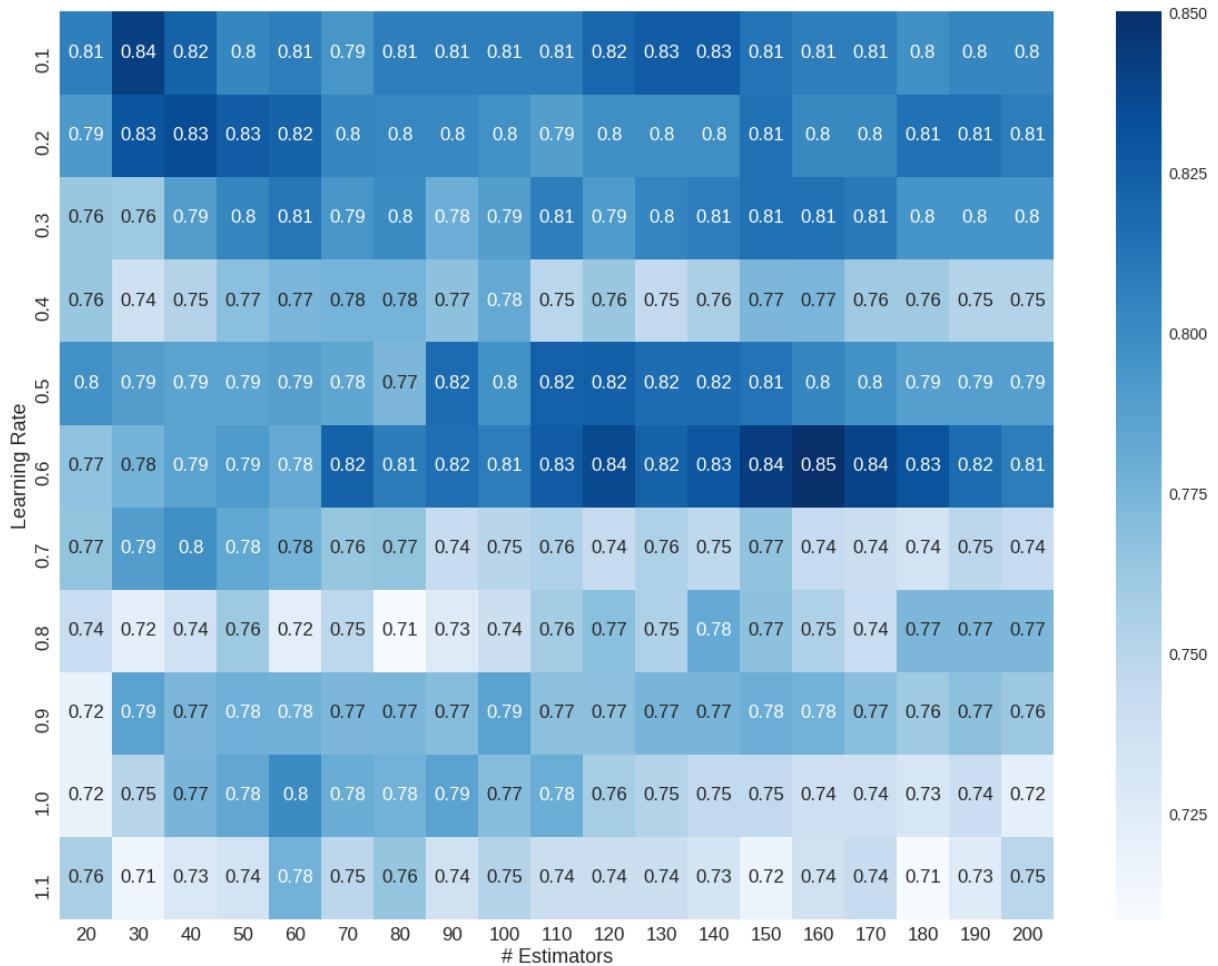
- *Learning Rate*. Η παράμετρος αυτή συρρικνώνει τη συνεισφορά κάθε εκτιμητή κατά την τιμή της. Υπάρχει ένας συμβιβασμός ανάμεσα στην τιμή αυτή και στην προηγούμενη παράμετρο (number of estimators).

Μετά από δοκιμές, καταλήξαμε ότι ιδιαίτερα αποδοτική επιλογή είναι να χρησιμοποιήσουμε δένδρα απόφασης ως εκτιμητή βάσης (base estimator). Επίσης, σημαντικό είναι να βρούμε ένα καλό συνδυασμό του πλήθους αυτών και του learning rate. Για το λόγο αυτό επιστρατεύσαμε μεθόδους παρόμοιες με αυτές που εξετάσαμε στα προηγούμενα μοντέλα. Στο σχήμα 4.11 φαίνεται η απόδοση του μοντέλου, ως προς τη μετρική *F1*, για διάφορες τιμές των παραμέτρων. Παρόλληλα, η απόδοση του μοντέλου ως προς τις μετρικές *Recall*, *Precision*, και *Accuracy* συνοψίζονται στον Πίνακα 4.6. Τέλος, δίνοντας ξανά περισσότερη βαρύτητα στη μετρική *F1*, οι παράμετροι που επιλέγονται στο τελικό μοντέλο είναι *160 Number of Estimators*, και *Learning Rate 0.6*.

#### 4.7.4.6 Σχολιασμός των Μοντέλων

Στις προηγούμενες υποενότητες αναλύσαμε τρία διαφορετικά μοντέλα ταξινόμησης με σκοπό την επίλυση του προβλήματος της αναγνωσιμότητας. Ξεκινήσαμε με το μοντέλο *KNN* και στη συνέχεια είδαμε τα μοντέλα των *Random Forests* και *AdaBoost*. Αν και στο τελικό σύστημα ο χρήστης μπορεί να επιλέξει οποιοδήποτε μοντέλο επιθυμεί, θεωρήσαμε ότι καλό είναι ένα από αυτά να οριστεί ως προεπιλογή.

Ανάμεσα στα τρία μοντέλα, από όποιη απόδοσης ως προς τη μετρική *F1*, την καλύτερη απόδοση πετυχαίνει ο *Random Forests*. Ωστόσο, πέρα από την απόδοση, ένας σημαντικός παράγοντας είναι και ο χρόνος που απαιτείται για την ολοκλήρωση της ταξινόμησης. Από όποιη χρόνου ταχύτερος είναι ο *KNN*, ο οποίος χρειάζεται λιγότερο από 1 δευτερόλεπτο για να ταξινομήσει 100 τμήματα κώδικα. Ακολουθεί



Σχήμα 4.11: Στην παραπάνω γραφική παράσταση παρουσιάζονται αποτελέσματα του μοντέλου που κάνει προβλέψεις χρησιμοποιώντας τον classifier AdaBoost και δένδρα απόφασης (decision trees) ως εκτιμητή βάσης, για διάφορα πλήθη εκτιμητών (οριζόντιος άξονας), και διαφορετικά learning rates (κάθετος άξονας).

ο AdaBoost, με περίπου 2 δευτερόλεπτα ανά 100 τμήματα κώδικα και τέλος ο Random Forests που χρειάζεται πάνω από 10 δευτερόλεπτα για το ίδιο έργο.

Επειδή το σύστημα πρέπει να παρέχει στο χρήστη μία γρήγορη απόκριση στο query του, το μοντέλο που επιλέγεται ως προεπιλογή είναι αυτό του AdaBoost. Ο classifier αυτός έχει αρκετά καλύτερα απόδοση από τον KNN και ελαφρώς χαμηλότερη από αυτή του Random Forests. Επίσης, η ταχύτητα του επιτρέπει μία «ομαλή» εμπειρία στο χρήστη.

## 4.8 Clusterer

Η επόμενη βασική λειτουργία του συστήματος μας είναι η ομαδοποίηση των αποτελεσμάτων, δηλαδή των τμημάτων κώδικα. Την εργασία αυτή την αναλαμβάνει

το επόμενο κομμάτι που σχεδιάστηκε, ο ομαδοποιητής (Clusterer). Στις επόμενες ενότητες θα δούμε πως κατασκευάστηκε.

#### 4.8.1 Προεπεξεργασία Δεδομένων

Πριν χρησιμοποιήσουμε οποιοδήποτε αλγόριθμο ομαδοποίησης πρέπει να ετοιμάσουμε τα δεδομένα μας ώστε να μπορούν να ομαδοποιηθούν από αυτόν. Μία πρώτη προσέγγιση είναι να εξετάσουμε τα τμήματα κώδικα σαν αρχεία κειμένου και να τα ομαδοποιήσουμε με βάση τις λέξεις (tokens) που εμφανίζονται μέσα σε αυτά. Αν και αυτή η θεώρηση είναι αρκετά αποτελεσματική στο να φέρει κοντά (δηλ. σε ίδιες ομάδες) τμήματα κώδικα που είναι παρόμοια, αποτυγχάνει στο να διαχωρίσει τα τμήματα κώδικα με βάση τις διαφορετικές υλοποιήσεις που αυτά απεικονίζουν. Για παράδειγμα, εξετάζοντας τα δύο τμήματα κώδικα που απεικονίζονται παρακάτω ως κείμενο, παρατηρούμε ότι έχουν περισσότερες ομοιότητες παρά διαφορές. Ωστόσο, αν και τα δύο απαντούν στο ερώτημα *"How to read a file in Java"*, παρουσιάζουν έκαστο μια διαφορετική υλοποίηση, αφού στην πρώτη περίπτωση χρησιμοποιείται μόνο ένας BufferedReader, ενώ στη δεύτερη γίνεται επιπλέον χρήση ενός αντικειμένου InputStream..

```

1 // Snippet 1
2 Charset charset = Charset.forName("US-ASCII");
3 try (BufferedReader reader = Files.newBufferedReader(file, charset)) {
4     String line = null;
5     while ((line = reader.readLine()) != null) {
6         System.out.println(line);
7     }
8 } catch (IOException x) {
9     System.err.format("IOException: %s%n", x);
10 }

1 // Snippet 2
2 try (InputStream in = Files.newInputStream(file);
3     BufferedReader reader =
4     new BufferedReader(new InputStreamReader(in))) {
```

```

5     String line = null;
6
7     while ((line = reader.readLine()) != null) {
8
9         System.out.println(line);
10
11    } catch (IOException x) {
12
13        System.err.format("IOException: %s%n", x);
14    }

```

Το γεγονός ότι σαν κείμενο είναι σε μεγάλο βαθμό όμοια είναι λογικό εφόσον πρόκειται για δύο αποσπάσματα σε μία γλώσσα προγραμματισμού με αυστηρή σύνταξη, τα οποία έχουν τον ίδιο στόχο. Ο προγραμματιστής όμως ενδιαφέρεται να μελετήσει τις διαφορετικές υλοποιήσεις ανάγνωσης ενός αρχείου και να υιοθετήσει στον κώδικα του αυτή που του ταιριάζει περισσότερο. Για το λόγο αυτό, η αναπαράσταση των τμημάτων κώδικα ως απλών εγγράφων δεν αποδίδει στην περίπτωση μας οπότε πρέπει να βρούμε άλλο τρόπο.

Για να μπορέσουμε να διαχωρίσουμε τα τμήματα κώδικα με βάση τις διαφορετικές υλοποιήσεις που αυτά προσφέρουν κάνουμε την εικασία ότι:

”...τμήματα κώδικα που χρησιμοποιούν παρόμοια API calls απεικονίζουν και όμοιες υλοποιήσεις του προβλήματος...”

Άρα θα προσπαθήσουμε να ομαδοποιήσουμε τα τμήματα κώδικα με βάση τα API calls που περιέχουν και όχι με βάση το συνολικό τους κείμενο, έκταση κλπ. Σε προηγούμενο στάδιο, μέσω του *Parser*, είχαμε εξάγει τα στοιχεία των APIs κάθε τμήμα κώδικα.

Ένας αλγόριθμος ομαδοποίησης δεν μπορεί να τροφοδοτηθεί απευθείας με μία ακολουθία συμβόλων. Πρέπει να μετατρέψουμε το κάθε τμήμα κώδικα ως ένα διάνυσμα αριθμητικών τιμών με σταθερό μήκος. Για να διευθετηθεί το πρόβλημα αυτό:

- *Κατακερματίζουμε (tokenizing)* τα τμήματα κώδικα (το οποίο έχει γίνει αφού κάθε API call για εμάς είναι ένα token) και αποδίδουμε μία τιμή (id) σε κάθε token.
- *Καταμετρούμε (counting)* τις εμφανίσεις των tokens σε κάθε τμήμα κώδικα.

- Κανονικοποιούμε και αποδίδουμε βάρη με μειωτική σημασία σε tokens που εμφανίζονται συχνά και στη πλειοψηφία των τμημάτων κώδικα.

Στην αναπαράσταση αυτή, τα χαρακτηριστικά και τα δείγματα ορίζονται με τον ακόλουθο τρόπο:

- Η συχνότητα εμφάνισης του κάθε token μεταφράζεται ως χαρακτηριστικό (*feature*).
- Το διάνυσμα των συχνοτήτων όλων των tokens που περιέχονται σε ένα τμήμα κώδικα θεωρείται δειγμα (*sample*).

Πίνακας 4.7: Αναπαράσταση των τμημάτων κώδικα με την προσέγγιση Bag of Words.

	<i>API call 1</i>	<i>API call 2</i>	<i>API call 3</i>	<i>API call 4</i>	<i>API call 5</i>	<i>API call 6</i>	<i>API call 7</i>	<i>API call 8</i>	<i>API call 9</i>
<i>Snippet 1</i>	1	0	0	3	0	1	1	0	1
<i>Snippet 2</i>	2	2	0	1	0	0	0	0	2
<i>Snippet 3</i>	1	1	0	1	0	0	3	0	0
<i>Snippet 4</i>	0	2	0	0	0	0	1	0	0
<i>Snippet 5</i>	2	0	0	0	0	0	0	0	2

Έτσι, μπορούμε να αναπαραστήσουμε μία συλλογή από τμήματα κώδικα ως ένα πίνακα, με μία γραμμή για κάθε τμήμα κώδικα και μία στήλη για κάθε token (δηλ. API call), όπως παρουσιάζεται στον Πίνακα 4.7. Από τον Πίνακα καταλαβαίνουμε ότι κάθε τμήμα κώδικα μπορεί να αναπαρασταθεί ως ένα διάνυσμα εννέα διαστάσεων σε ένα Μοντέλο Διανυσματικού Χώρου (Vector Space Model). Για παράδειγμα,  $\text{Snippet}_1 = (1, 0, 0, 3, 0, 1, 1, 0, 1)$ . Αυτή η μέθοδος αναπαράστασης ονομάζεται *Bag of Words* [54], και αποτελεί μία από τις πιο συνηθισμένες τεχνικές διανυσματοποίησης εγγράφων.

Ο παραπάνω τρόπος απεικόνισης έχει ένα βασικό ελάττωμα. Σε ένα μεγάλο σύνολο από τμήματα κώδικα, μερικά tokens μπορεί να εμφανίζονται πολύ συχνά (π.χ. `"println"`, `"printStackTrace"`) με αποτέλεσμα να προσδίδουν πολύ λίγη χρήσιμη πληροφορία σχετικά με το πραγματικό περιεχόμενο του τμήματος κώδικα. Άμα τροφοδοτούσαμε τον αλγόριθμο απευθείας με αυτά τα δεδομένα αυτοί οι συχνοί όροι θα επισκίαζαν άλλους πιο ενδιαφέροντες όρους με χαμηλότερες συχνότητες εμφάνισης.

Για να αποφύγουμε κάτι τέτοιο είναι σημαντικό να αποδώσουμε βάρη στα tokens ανάλογα με το πόσο σημαντικά είναι. Μία πολύ χρήσιμη τεχνική είναι ο μετασχηματισμός *tf-idf* [26]. Ο όρος *tf* (term frequency), που ισοδυναμεί με τον αριθμό των εμφανίσεων ενός όρου στο δεδομένο κομμάτι κώδικα (στο σύστημα μας είναι η απόλυτη συχνότητα), πολλαπλασιάζεται με τον όρο *idf* (inverse document frequency), που εκφράζει ένα μέτρο της πληροφορίας που παρέχει ένας όρος. Συνολικά:

$$tfidf(t, d) = tf(t, d) \cdot idf(t)$$

Υπάρχουν αρκετές επιλογές για την απόδοση βαρών μέσω του όρου *idf*. Στο δικό μας σύστημα υπολογίζεται ως

$$idf(t) = \log \frac{1 + n_d}{1 + df(d, t)} + 1,$$

όπου  $n_d$  είναι ο συνολικός αριθμός των τμημάτων κώδικα, και  $df(d, t)$  είναι ο αριθμός των τμημάτων κώδικα που περιέχουν το token  $t$ . Τα *tf-idf* διανύσματα που προκύπτουν, στη συνέχεια κανονικοποιούνται με την Ευκλείδεια νόρμα<sup>25</sup>:

$$v_{norm} = \frac{v}{\|v\|} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \dots + v_n^2}}$$

Αρχικά η μέθοδος αυτή αναπτύχθηκε και χρησιμοποιήθηκε στον τομέα της ανακτησης πληροφορίας<sup>26</sup> (information retrieval) από μηχανές αναζήτησης ως μέθοδος βαθμολόγησης [55], αλλά αποδείχθηκε ότι είναι αποτελεσματική και για ταξινόμηση και ομαδοποίηση εγγράφων.

Αν και στο σημείο αυτό είμαστε έτοιμοι να ομαδοποιήσουμε τα δεδομένα μας, υπάρχει ένα τελευταίο ζήτημα που πρέπει να διευθετήσουμε. Μέσω των παραπάνω ενεργειών το σύνολο των δεδομένων μας έχει εκφραστεί σε ένα πολυδιάστατο χώρο. Συγκεκριμένα, αν για παράδειγμα σε όλο το σύνολο των τμημάτων κώδικα εμφανίζονται 100 διαφορετικά API calls, τότε κάθε διάνυσμα με το οποίο αναπαριστούμε το τμήμα κώδικα ανήκει σε ένα χώρο 100 διαστάσεων! Όπως καταλαβαίνει κανείς κάτι τέτοιο είναι πολύ δύσκολο να αναπαρασταθεί γραφικά.

<sup>25</sup>[https://en.wikipedia.org/wiki/Norm\\_\(mathematics\)](https://en.wikipedia.org/wiki/Norm_(mathematics))

<sup>26</sup>[https://en.wikipedia.org/wiki/Information\\_retrieval](https://en.wikipedia.org/wiki/Information_retrieval)

Για να μπορέσουμε να έχουμε μία καλύτερη αναπαράσταση των τμημάτων κώδικα στο χώρο, μπορούμε να χρησιμοποιήσουμε τεχνικές μείωσης διαστασημότητας<sup>27</sup>. Η τεχνική που χρησιμοποιήθηκε ονομάζεται *Multi-dimensional Scaling* (MDS) και ο στόχος της είναι να βρεθεί μια αναπαράσταση των δεδομένων, λιγότερων διαστάσεων, η οποία θα διατηρεί τη σχέση των αποστάσεων του αρχικού πολυδιάστατου χώρου [56]. Η χρησιμότητα του MDS θα φανεί στις επόμενες ενότητες όπου και θα περιγράψουμε τη διαδικασία ομαδοποίησης.

#### 4.8.2 Επιλογή του Πλήθους των Ομάδων (Clusters)

Στην ομαδοποίηση δεδομένων με μη εποπτευόμενη μάθηση<sup>28</sup> (unsupervised learning) η επιλογή των αριθμών των ομάδων (clusters) αποτελεί ιδιαίτερη πρόκληση. Παρόλο που έχουν προταθεί διάφορες τεχνικές για την εκτίμηση του αριθμού των ομάδων ([57, 58, 59, 60, 61, 62, 63]), δεν υπάρχει απόδειξη άρα και «χρυσός» κανόνας ότι κάποια λειτουργεί απολύτως σωστά.

Μία λογική πορεία είναι να γίνουν αρκετές δοκιμές, για διαφορετικά πλήθη ομάδων ενώ παράλληλα τροποποιούνται τα δεδομένα (π.χ. μείωση θορύβου), έως ότου το αποτέλεσμα που προκύπτει να έχει κάποια λογική. Ωστόσο, για τη δική μας υλοποίηση δεν υφίσταται αυτή η «πολυτέλεια» διότι το σύνολο των δεδομένων δεν είναι σταθερό αλλά μεταβάλλεται ανάλογα με το ερώτημα του χρήστη. Επίσης, δεν είναι δυνατόν να εξετάζουμε τα δεδομένα κάθε φορά πριν τα χωρίσουμε σε ομάδες. Το σύστημα πρέπει να επιλέγει αυτόματα των αριθμό των ομάδων και να εκτελεί τη διαδικασία της ομαδοποίησης.

Προς την αντιμετώπιση των παραπάνω περιορισμών, το σύστημα μας επιλέγει τον αριθμό των ομάδων λαμβάνοντας υπόψιν το αποτέλεσμα από ποικίλες διαφορετικές μεθόδους που έχουν προταθεί στη βιβλιογραφία. Δηλαδή, πριν γίνει η τελική ομαδοποίηση, εφαρμόζεται ένα πλήθος μεθόδων που αποσκοπούν στην εκτίμηση του πραγματικού αριθμού των ομάδων, και η τελική ομαδοποίηση γίνεται με βάση το συγκερασμό των αποτελεσμάτων των μεθόδων αυτών. Στις ακόλουθες ενότητες θα δούμε ποιες είναι αυτές οι μέθοδοι που χρησιμοποιούνται.

<sup>27</sup>[https://en.wikipedia.org/wiki/Dimensionality\\_reduction](https://en.wikipedia.org/wiki/Dimensionality_reduction)

<sup>28</sup>[https://en.wikipedia.org/wiki/Unsupervised\\_learning](https://en.wikipedia.org/wiki/Unsupervised_learning)

#### 4.8.2.1 Εκτίμηση Πλήθους Ομάδων από Ιεραρχική Ομαδοποίηση

Αρχικά, για να εκτιμήσουμε το πλήθος των ομάδων που μπορούν να σχηματίσουν τα τμήματα κώδικα στο δοθέν ερώτημα, εφαρμόζουμε έναν αλγόριθμο ιεραρχικής ομαδοποίησης<sup>29</sup> (hierarchical clustering), και συγκεκριμένα προσέγγισης *bottom-up* (agglomerative).

---

#### Αλγόριθμος 4: Ψευδοκώδικας Hierarchical (Agglomerative) Clustering

---

**Input:** Distance matrix  $d$  of  $n \times n$  of pairwise distances between points

- 1 Form  $n$  clusters each with one element
- 2 Construct a graph  $T$  by assigning one vertex to each cluster
- 3 **while** there is more than one cluster **do**
- 4     Find the two closest clusters  $C_1$  and  $C_2$
- 5     Merge  $C_1$  and  $C_2$  into a new cluster  $C$  with  $|C_1| + |C_2|$  elements
- 6     Compute distance from  $C$  to all other clusters
- 7     **if** they are close **then**
- 8         Add a new vertex  $C$  to  $T$  and connect to vertices  $C_1$  and  $C_2$
- 9         Remove rows and columns of  $d$  corresponding to  $C_1$  and  $C_2$
- 10         Add row & column to  $d$  corresponding to the new cluster  $C$
- 11 **end**
- 12 **return**  $T$

---

Η ιεραρχική ομαδοποίηση ξεκινά τοποθετώντας κάθε δείγμα στο δικό του cluster. Οπότε αρχικά έχουμε τόσα clusters όσες είναι και οι παρατηρήσεις μας (singleton clusters). Στη συνέχεια, χρησιμοποιείται ένα από τα ακόλουθα κριτήρια για να συνενωθούν διαδοχικά τα clusters σε νέα μεγαλύτερο.

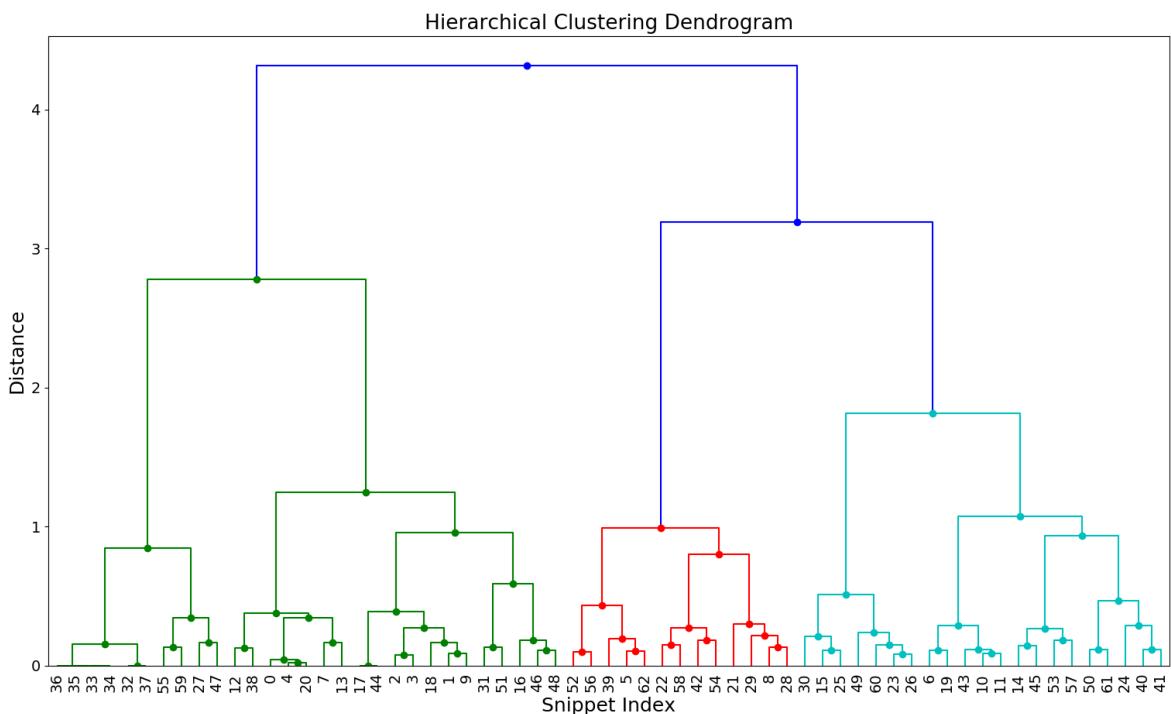
- *Ward*. Ελαχιστοποιεί το τετραγωνικό σφάλμα εσωτερικά του κάθε cluster. Αυτή είναι και η μετρική που επιλέγουμε για το σύστημα μας.
- *Maximum* ή *Complete Linkage*. Ελαχιστοποιεί τη μέγιστη απόσταση μεταξύ των παρατηρήσεων ανά ζευγάρι από clusters.
- *Average Linkage*. Ελαχιστοποιεί τη μέση τιμή των αποστάσεων μεταξύ όλων των παρατηρήσεων ανά ζευγάρι από clusters.

Η διαδικασία συνένωσης σταματά όταν σχηματιστεί ένα συνολικό cluster το οποίο περιέχει όλες τις αρχικές παρατηρήσεις. Ο τρόπος που απεικονίζεται η ιεραρχική δομή που σχηματίστηκε είναι συνήθως μέσω ενός δενδρογράμματος. Για

---

<sup>29</sup>[https://en.wikipedia.org/wiki/Hierarchical\\_clustering](https://en.wikipedia.org/wiki/Hierarchical_clustering)

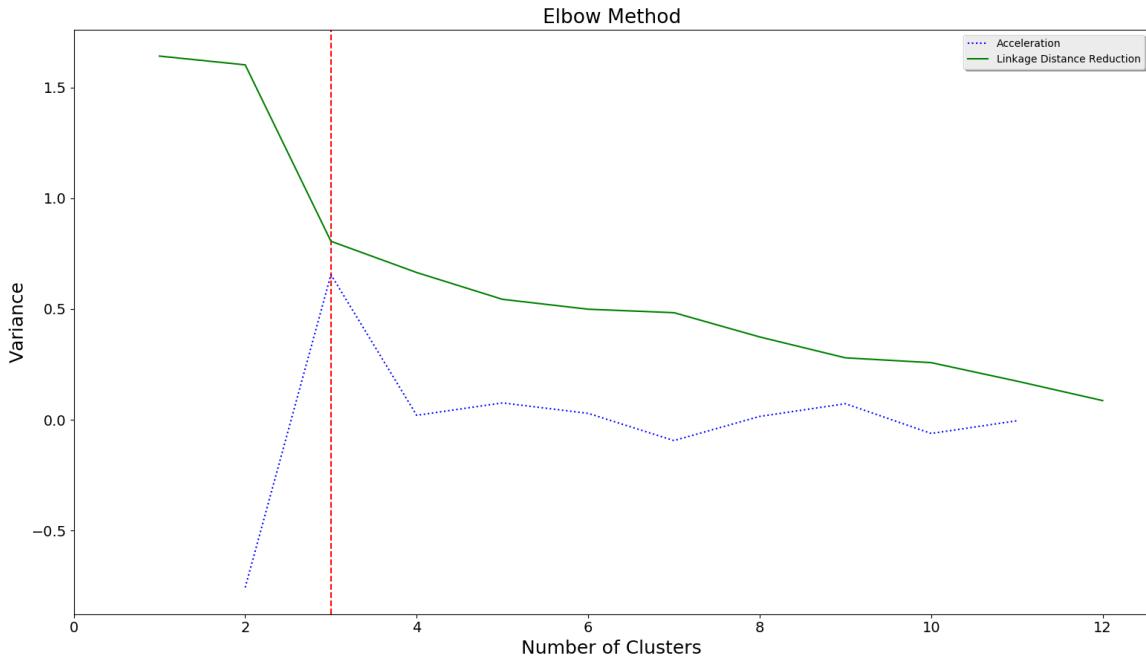
παράδειγμα στο ερώτημα *"How to play audio file"* το δενδρόγραμμα που σχηματίζεται φαίνεται στο Σχήμα 4.12. Πριν τον σχηματισμό του τελευταίου μεγάλου cluster, θεωρούμε ένα κατώφλι στην απόσταση στο οποίο σταματά η διαδικασία της ομαδοποίησης. Το κατώφλι αυτό τίθεται στο 70% της μέγιστης απόστασης στον πίνακα αποστάσεων που δίνεται ως είσοδος (Αλγόριθμος 4). Στο σημείο αυτό εξάγουμε τον εκτιμώμενο αριθμό των clusters με βάση το πλήθος που έχει σχηματιστεί έως τότε.



Σχήμα 4.12: Δενδρόγραμμα ιεραρχικής (agglomerative) ταξινόμησης. Σχηματίζονται τρία clusters τα οποία απεικονίζονται με διαφορετικά χρώματα (πράσινο, κόκκινο, μπλε).

#### 4.8.2.2 Εκτίμηση Πλήθους Ομάδων με τη μέθοδο του Γονάτου (Elbow Method)

Η μέθοδος του γονάτου (elbow method) προσπαθεί να βρει το σημείο εκείνο στο οποίο η προσθήκη ενός επιπλέον cluster δεν προσδίδει ιδιαίτερη πληροφορία στο μνήμελο [64]. Συγκεκριμένα, σχεδιάζοντας την διακύμανση του μοντέλου σε συνάρτηση με τον αριθμό των clusters, τα πρώτα clusters προσθέτουν περισσότερη πληροφορία, αλλά σε κάποιο σημείο το κέρδος της πληροφορίας μειώνεται σχηματίζοντας μία γωνία (ή γόνατο) στη γραφική παράσταση. Ο αριθμός των clusters επιλέγεται από εκείνο το σημείο του γονάτου και εξ αυτού και το όνομα της μεθόδου.



Σχήμα 4.13: Η μέθοδος του γονάτου (elbow method). Η συνεχόμενη γραμμή (πράσινη) συμβολίζει την απόσταση μεταξύ των clusters, ενώ η διακεκομμένη (μπλε) είναι η επιτάχυνση στη μείωση της προηγούμενης απόστασης. Η διακεκομμένη (κόκκινη) επισημαίνει το σημείο όπου εμφανίζεται το γόνατο.

Για παράδειγμα, για το ερώτημα που είχαμε και προηγουμένως, εάν θέλουμε να κάνουμε εκτίμηση των αριθμό των ομάδων με τη μέθοδο του γονάτου προκύπτει το Σχήμα 4.13. Παρατηρούμε ότι το γόνατο εμφανίζεται για αριθμό ομάδων ίσο με τρία. Το σύστημα μας επιλέγει το σημείο του γονάτου αυτόματα υπολογίζοντας τη μέγιστη τιμή της δεύτερης παραγώγου των αποστάσεων σύνδεσης (δηλ. της επιτάχυνσης). Η δημιουργία και τέταρτου cluster δεν αποδίδει ιδιαίτερη μείωση στη διακύμανση του μοντέλου οπότε και θεωρείται περιττή. Το σημείο του γονάτου εντοπίζεται από το σύστημα μας αυτόματα.

#### 4.8.2.3 Εκτίμηση Πλήθους Ομάδων από Ανάλυση Σιλουέτας (Silhouette Analysis)

Η επόμενη μέθοδος που θα χρησιμοποιήσουμε για να κάνουμε εκτίμηση του αριθμού των clusters είναι η ανάλυση του διαγράμματος σιλουέτας<sup>30</sup> (silhouette analysis). Η μέθοδος αυτή μελετά την απόσταση που έχουν μεταξύ τους τα σχηματιζόμενα clusters. Το διάγραμμα σιλουέτας απεικονίζει πόσο κοντά βρίσκονται τα σημεία ενός cluster με αυτά των γειτονικών του και έτσι μας βοηθά να εκτιμήσουμε τον

<sup>30</sup>[https://en.wikipedia.org/wiki/Silhouette\\_\(clustering\)](https://en.wikipedia.org/wiki/Silhouette_(clustering))

αριθμό των συνολικών clusters.

Η σταθερά σιλουέτας<sup>31</sup> (silhouette coefficient) ορίζεται για κάθε δείγμα ξεχωριστά και εξαρτάται από δύο ποσότητες:

1. Τη μέση απόσταση μεταξύ ενός δείγματος και όλων των άλλων σημείων που βρίσκονται στην ίδια ομάδα, που συμβολίζεται με  $a$ .
2. Τη μέση απόσταση μεταξύ ενός δείγματος και όλων των άλλων σημείων που βρίσκονται στην κοντινότερη ομάδα, που συμβολίζεται με  $b$ .

Με βάση τα  $a$ ,  $b$ , το silhouette coefficient  $s$  για ένα δείγμα ισούται με:

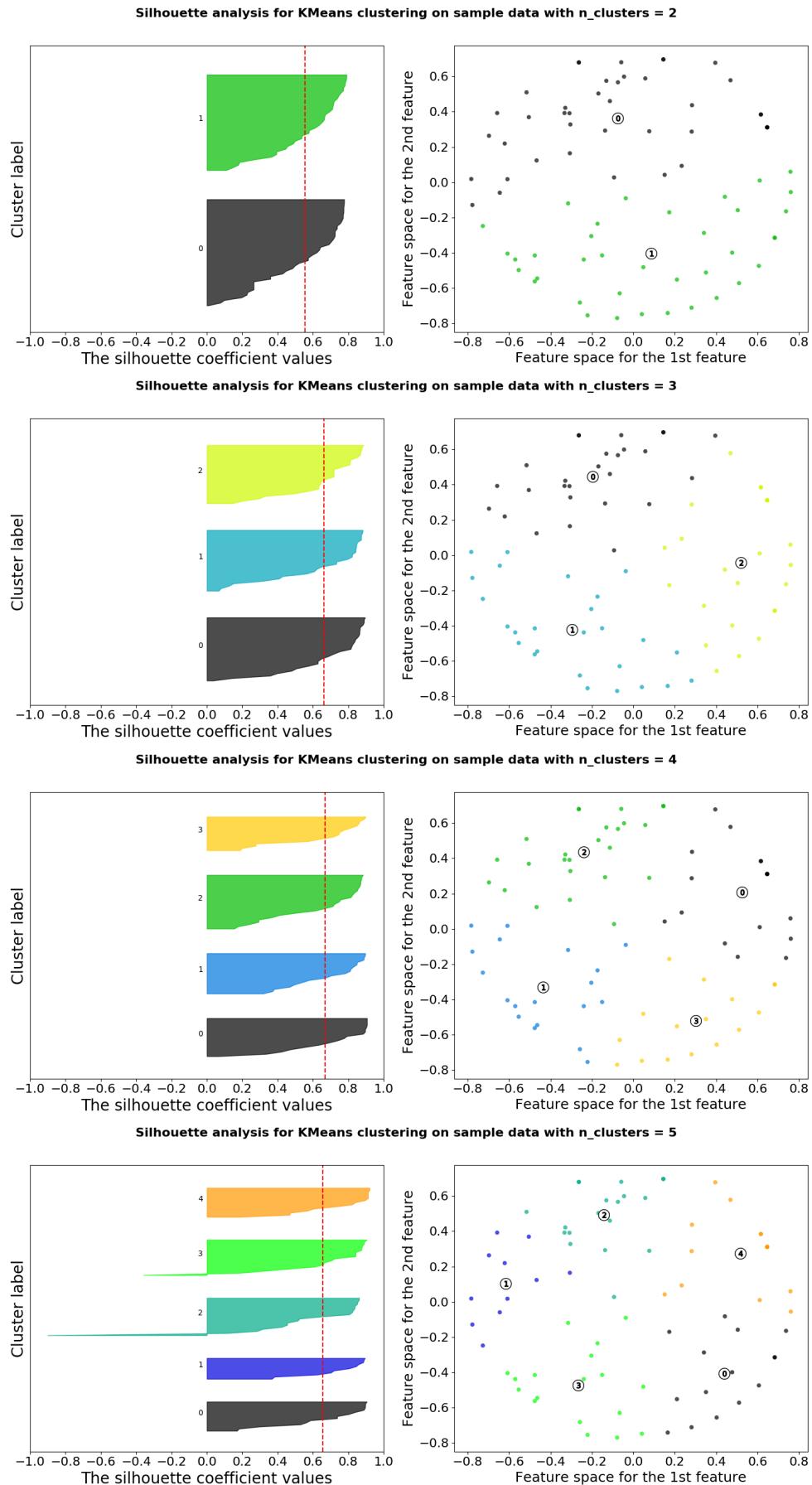
$$s = \frac{b - a}{\max(a, b)}$$

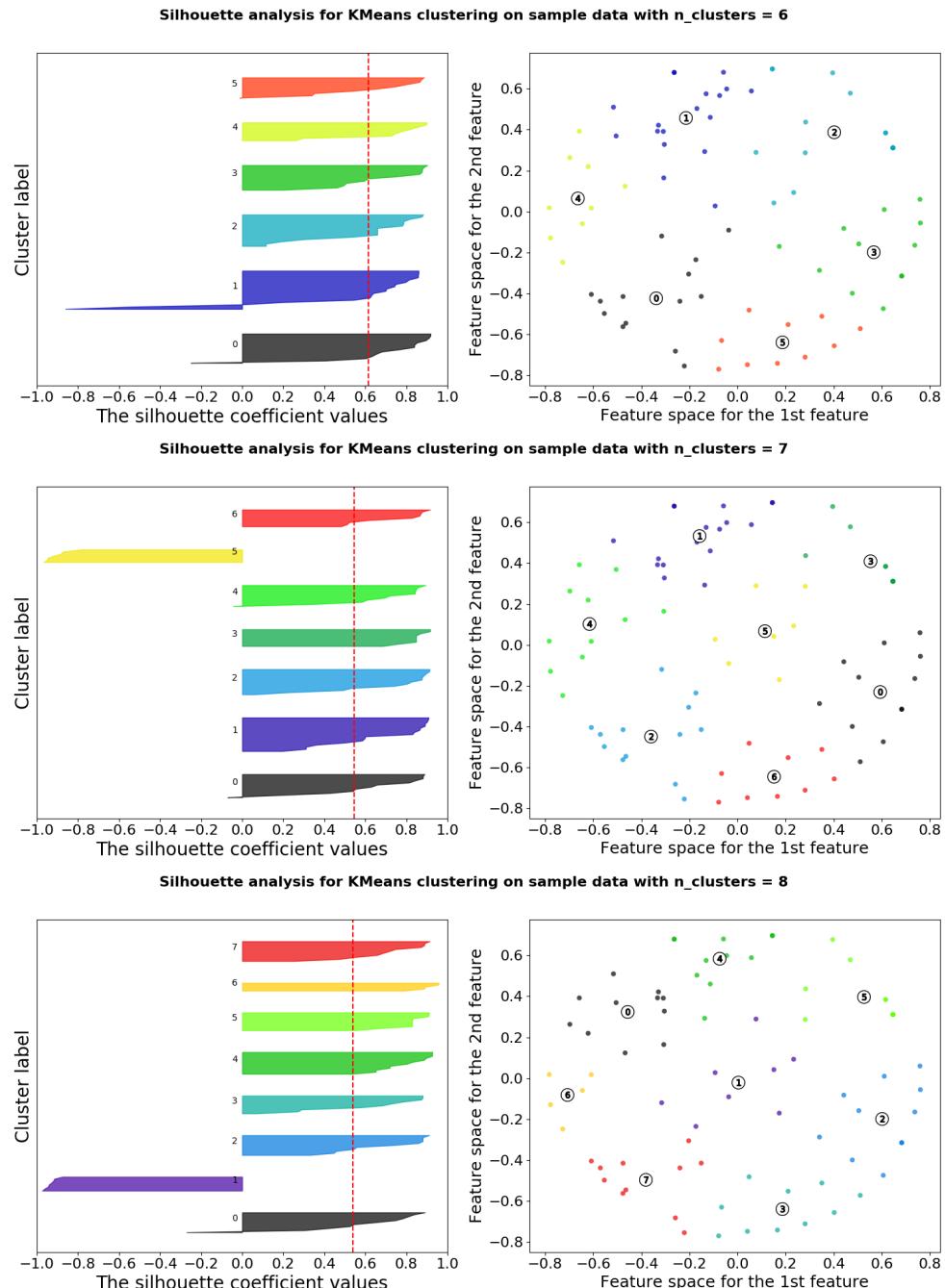
Η μετρική αυτή κυμαίνεται μεταξύ  $-1$ , όταν η ομαδοποίηση είναι λανθασμένη, και  $+1$ , όταν έχουν σχηματιστεί πυκνές και καλά διαχωρισμένες ομάδες. Τιμές κοντά στο μηδέν υποδεικνύουν επικαλυπτόμενες ομάδες.

Γενικά, μία καλή ομαδοποίηση προκύπτει όταν τα δείγματα στην πλειοψηφία τους έχουν θετικά silhouette scores, και η μέση τιμή όλων των scores είναι υψηλή. Στο Σχήμα 4.14 παρουσιάζονται τα αποτελέσματα (το ερώτημα παραμένει ίδιο με τις προηγούμενες μεθόδους). Αρχικά, για αριθμό clusters ίσο με 2, παρατηρούμε ότι ναι μεν έχουμε θετικές τιμές για κάθε δείγμα στα silhouette scores, αλλά δεν είναι ιδιαίτερα υψηλές με αποτέλεσμα ο μέσος όρος να είναι χαμηλός (κόκκινη γραμμή). Στη συνέχεια, παρατηρούμε ότι για αριθμό clusters 3 και 4 έχουμε και στις δύο περιπτώσεις καλή ομαδοποίηση. Ωστόσο, για πλήθος 3 η μέση τιμή είναι ελάχιστα υψηλότερα. Τέλος, για αριθμό από 5 και πάνω παρατηρούμε ότι λόγω των αρνητικών τιμών η ομαδοποίηση χειροτερεύει.

Το σύστημα μας, μέσω της παραπάνω ανάλυσης, έχει σχεδιαστεί ώστε να επιλέγει αυτόματα το εκτιμώμενο πλήθος των ομάδων, που είναι αναγκαίο αφού δεν είναι δυνατόν όπως είπαμε ο χρήστης να εξετάζει κάθε φορά τις γραφικές παραστάσεις μία προς μία. Ειδικότερα, επιλέγεται το πλήθος αυτό για το οποίο τα δείγματα έχουν την υψηλότερη μέση τιμή στα silhouette scores μιας και κάτι τέτοιο υποδεικνύει συνήθως και καλύτερες μεμονωμένες τιμές.

<sup>31</sup>[https://en.wikipedia.org/wiki/Silhouette\\_\(clustering\)](https://en.wikipedia.org/wiki/Silhouette_(clustering))





Σχήμα 4.14: Ανάλυση διαγράμματος σιλουόετας. Στα παραπάνω διαγράμματα, στα αριστερά, απεικονίζονται οι τιμές των silhouette scores για όλα τα δείγματα ανά cluster στα οποία έχουν καταταγεί, ενώ στα δεξιά φαίνεται η κατανομή των δειγμάτων στο δισδιάστατο χώρο και ο διαχωρισμός τους σε clusters.

#### 4.8.2.4 Εκτίμηση Πλήθους Ομάδων από Affinity Propagation

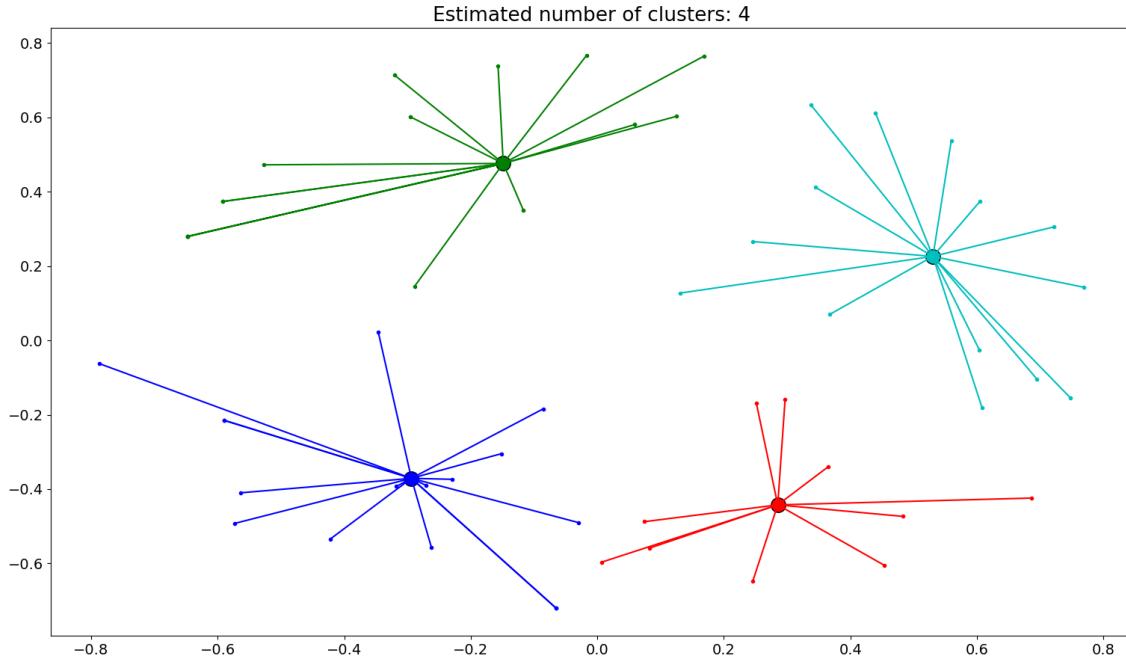
Η τελευταία μέθοδος που θα χρησιμοποιήσουμε για την εκτίμηση του πλήθους των ομάδων είναι το *Affinity Propagation* [65]. Η μέθοδος αυτή δημιουργεί clusters στέλνοντας μηνύματα μεταξύ των ζευγαριών των δειγμάτων μέχρι να υπάρξει σύγκλιση. Το σύνολο των δειγμάτων περιγράφεται από ένα μικρό αριθμό αντιπροσωπευτικών δειγμάτων (exemplars), τα οποία αντιπροσωπεύουν όλα τα άλλα δείγματα. Τα μηνύματα που στέλνονται μεταξύ των ζευγαριών εκφράζουν την καταλληλότητα ενός δείγματος να είναι το exemplar του άλλου. Τα μηνύματα ανανεώνονται με βάση τις αποκρίσεις από άλλα ζευγάρια. Η διαδικασία σταματά όταν υπάρξει σύγκλιση, δηλαδή στο σημείο όπου τα τελικά exemplars έχουν επιλεχθεί, και ως εκ τούτου έχει προκύψει η τελική ομαδοποίηση.

Τα μηνύματα που στέλνονται μεταξύ των δειγμάτων ανήκουν σε μία από τις ακόλουθες κατηγορίες:

- Η πρώτη κατηγορία είναι η αξιοπιστία (responsibility)  $r(i, k)$ , η οποία εκφράζει κατά πόσο το δείγμα  $k$  πρέπει να τεθεί ως exemplar του δείγματος  $i$ .
- Η δεύτερη κατηγορία είναι η διαθεσιμότητα (availability)  $a(i, k)$ , η οποία δηλώνει κατά πόσο το δείγμα  $i$  πρέπει να επιλέξει το δείγμα  $k$  ως το exemplar του, λαμβάνοντας υπόψιν κατά πόσο το  $k$  δύναται να γίνει exemplar των υπόλοιπων δειγμάτων.

Η μέθοδος αυτή είναι αρκετά ενδιαφέρουσα γιατί έχει την ικανότητα εκτίμησης του αριθμού των clusters βασιζόμενη μόνο στα δεδομένα. Για παράδειγμα, για το ερώτημα που εξετάσαμε και σε όλες τις προηγούμενες μεθόδους, η εκτίμηση του Affinity Propagation για τη μορφή και το πλήθος των clusters δίνεται στο Σχήμα 4.15. Όπως παρατηρούμε η πρόβλεψη της μεθόδου είναι ότι σχηματίζονται τέσσερα clusters.

Το κύριο μειονέκτημα της μεθόδου αυτής είναι η πολυπλοκότητα της. Ο αλγόριθμος έχει χρονική πολυπλοκότητα της τάξης  $O(N^2T)$ , όπου  $N$  είναι το πλήθος των δειγμάτων και  $T$  είναι το πλήθος των επαναλήψεων που χρειάζονται μέχρι τη σύγκλιση. Αυτό κάνει τη μέθοδο του Affinity Propagation κυρίως κατάλληλη για μικρούς έως μέσου μεγέθους datasets.



Σχήμα 4.15: Η μέθοδος Affinity Propagation. Με μικρές τελείες απεικονίζονται τα exemplars μετά τη σύγκλιση του αλγορίθμου, ενώ με μεγάλες τελείες τα κέντρα των clusters.

#### 4.8.2.5 Τελική Επιλογή του Πλήθους των Ομάδων

Στις παραπάνω υποενότητες είδαμε μεθόδους που αντιμετωπίζουν το πρόβλημα επιλογής του αριθμού των ομάδων. Όπως διαπιστώνεται και από τα αποτελέσματα των μεθόδων, δεν υπάρχει ένας και μοναδικός τρόπος διαχωρισμού των τμημάτων κώδικα. Ωστόσο, παρατηρούμε ότι οι εκτιμήσεις που προκύπτουν είναι κοντά μεταξύ τους.

Κατά τη χρήση του συστήματος, ο χρήστης έχει την επιλογή να ορίσει με ποια μέθοδο θα επιλεχθεί ο αριθμός των ομάδων. Βέβαια, σε περίπτωση που δεν επιθυμεί να μπει σε αυτή τη διαδικασία, το σύστημα μπορεί να επιλέξει λαμβάνοντας υπόψιν το αποτέλεσμα και από τις τέσσερις μεθόδους. Συγκεκριμένα, σε αυτό το ενδεχόμενο, το σύστημα επιλέγει τη συνηθέστερη εκτίμηση και σε περίπτωση που υπάρχει ισοψηφία τότε επιλέγει θέτοντας βάρη στο αποτέλεσμα κάθε μεθόδου. Ως μέθοδο με το μεγαλύτερο βάρος επιλέξαμε την ιεραρχική ομαδοποίηση, με την ανάλυση σιλουέτας, τη μέθοδο γονάτου, και τέλος του *affinity propagation* να ακολουθούν με την αναφερθείσα σειρά.

### 4.8.3 Ομαδοποίηση (Clustering)

Εφόσον καταφέραμε να καταλήξουμε με μία επιλογή για το πλήθος των ομάδων, αυτό που απομένει για το κομμάτι του Clusterer είναι ο σχηματισμός των τελικών ομάδων. Για το σκοπό επιλέχθηκε ο αλγόριθμος *K-Means* [66]. Ο αλγόριθμος αυτός χρησιμοποιείται σε μεγάλο εύρος εφαρμογών σε διαφορετικούς τομείς.

Ο K-Means ομαδοποιεί τα δεδομένα προσπαθώντας να χωρίσει το σύνολο των δειγμάτων σε ομάδες ίσης διακύμανσης, ελαχιστοποιώντας ένα κριτήριο γνωστό ως *Sum of Squared Errors (SSE)*. Κάθε cluster  $C$  που σχηματίζεται, μπορεί να περιγραφεί από τους μέσους  $\mu_j$  των δειγμάτων που ανήκουν σε αυτό. Οι παραπάνω μέσοι συνήθως αποκαλούνται cluster *centroids*. Αξίζει να σημειωθεί ότι γενικά τα centroids δεν είναι σημεία από τα  $X$ , ωστόσο είναι σημεία που ανήκουν στον ίδιο χώρο με αυτά. Ο αλγόριθμος K-Means στοχεύει να βρει centroids που ελαχιστοποιούν το κριτήριο SSE, το οποίο ισούται με:

$$\sum_{i=0}^n \min_{\mu_j \in C} (\|x_j - \mu_i\|^2)$$

Το κριτήριο SSE αποτελεί ένα μέτρο του πόσο συναφή είναι τα clusters στο εσωτερικό τους (cohesion).

Σε γενικές γραμμές ο αλγόριθμος αποτελείται από τρία βήματα. Στο πρώτο βήμα επιλέγονται τα αρχικά centroids, με τη πιο απλή μέθοδο να επιλέγει  $k$  δείγματα τυχαία από το σύνολο των  $X$ . Μετά την αρχικοποίηση, ο αλγόριθμος επαναλαμβάνεται μεταξύ των άλλων δύο βημάτων. Το πρώτο βήμα αναθέτει κάθε δείγμα στο πλησιέστερο του centroid. Το δεύτερο βήμα δημιουργεί νέα centroids υπολογίζοντας τη μέση τιμή όλων των δειγμάτων που ανατέθηκαν σε κάθε παλιό centroid. Υπολογίζεται η διαφορά μεταξύ των νέων και των παλιών centroids και ο αλγόριθμος επαναλαμβάνει τα δύο τελευταία βήματα μέχρις ότου η διαφορά είναι μικρότερη από ένα όριο. Με άλλα λόγια, επαναλαμβάνεται έως ότου τα centroids μένουν σχετικά σταθερά.

Δοσμένου αρκετού χρόνου, ο K-Means συγκλίνει πάντα, ωστόσο αυτό μπορεί να γίνει σε ένα τοπικό ελάχιστο. Αυτό εξαρτάται σε μεγάλο βαθμό από την αρχικοποίηση των centroids. Ως αποτέλεσμα, η εκτέλεση του αλγορίθμου εκτελείται αρκετές φορές, εκκινώντας με διαφορετικά centroids. Μία μέθοδος που χρησιμοποιείται για

---

**Αλγόριθμος 5:** Ψευδοκάδικας αλγορίθμου K-Means
 

---

**Input :**  $E = \{e_1, e_2, \dots, e_n\}$  (set of entities to be clustered)  
 $k$  (number of clusters)  
 $MaxIters$  (limit of iterations)

**Output:**  $C = \{c_1, c_2, \dots, c_k\}$  (set of cluster centroids)  
 $L = \{l(e)|e = 1, 2, \dots, n\}$  (set of cluster labels of E)

```

1 for  $c_i \in C$  do
2   |  $c_i \leftarrow e_j \in E$  (random selection)
3 end
4 for  $e_i \in E$  do
5   |  $l(e_i) \leftarrow argminDistance(e_i, c_j), j \in \{1\dots k\}$ 
6 end
7 do
8   | for  $c_i \in C$  do
9     |   |  $UpdateCluster(c_i)$ 
10    | end
11   | for  $e_i \in E$  do
12     |   |  $minDist \leftarrow argminDistance(e_i, c_j), j \in \{1, \dots, k\}$ 
13     |   | if  $minDist \neq l(e_i)$  then
14       |     |   |  $l(e_i) \leftarrow minDist$ 
15       |     |   |  $changed \leftarrow true$ 
16     |   | end
17   |   |  $iter = iter + 1$ 
18 while  $changed = true$  and  $iter \leq MaxIters$ ;

```

---

την αντιμετώπιση του παραπάνω προβλήματος είναι η χρήση του K-Means++ [67], ο οποίος είναι μία παραλλαγή του K-Means. Ο αλγόριθμος αυτός, επιλέγει αρχικά centroids τα οποία βρίσκονται σχετικά μακριά το ένα από το άλλο, καταλήγοντας σε καλύτερα αποτελέσματα από την τυχαία επιλογή. Αυτή είναι και η μέθοδος που χρησιμοποιούμε στην υλοποίηση του συστήματος μας.

## 4.9 Presenter

Ο Presenter είναι το τελευταίο δομικό στοιχείο του συστήματος μας, και είναι αυτό που είναι υπεύθυνο για την κατάταξη των ομαδοποιημένων τμημάτων κώδικα, και την παρουσίαση τους στο χρήστη.

#### 4.9.1 Κατάταξη των Αποτελεσμάτων

Η κατάταξη (ranking) των τμημάτων κώδικα πριν την παρουσίαση τους στο χρήστη είναι πολύ σημαντική γιατί έτσι ο χρήστης θα βρει πιο σύντομα το αποτέλεσμα που τον ενδιαφέρει γλιτώνοντας του πολύτιμο χρόνο. Για το σκοπό αυτό πρέπει να δώσουμε μία βαθμολογία σε κάθε τμήμα κώδικα η οποία θα καθορίσει τη σειρά παρουσίασης του ανάμεσα στα υπόλοιπα. Για την βαθμολόγηση των τμημάτων κώδικα στηριζόμαστε στα δύο παρακάτω κριτήρια.

**Κριτήριο 1:** *Μικρότερη απόσταση από το centroid του cluster → Μεγαλύτερη βαθμολογία*

Το κριτήριο αυτό βασίζεται στην παρατήρηση ότι τα τμήματα κώδικα που βρίσκονται κοντύτερα στο centroid του cluster περιέχουν πιο αντιπροσωπευτικά API calls από άλλα που βρίσκονται πιο μακριά.

Έτσι, τμήματα κώδικα που βρίσκονται κοντά στο centroid λαμβάνουν μεγαλύτερα μπόνους ενώ όσο απομακρύνονται από αυτό το μπόνους μειώνεται με απότομο τρόπο. Το συνολικό βάρος  $D$  που δίνεται με βάση αυτό το κριτήριο της απόστασης ισοδυναμεί με το άθροισμα των tf-idf βαρών για κάθε API call που εμφανίζεται στο τμήμα κώδικα.

Εξετάζοντας τα πιο αντιπροσωπευτικά API calls στην αρχή της λίστας ο χρήστης μπορεί να καταλάβει γρήγορα το περιεχόμενο της συγκεκριμένης ομάδας. Έτσι, μπορεί να συνεχίσει στα επόμενα αποτελέσματα της συγκεκριμένης ομάδας ή να μεταβεί σε άλλη αν η τρέχουσα δεν τον ενδιαφέρει.

**Κριτήριο 2:** *Μικρότερο μήκος κώδικα → Μεγαλύτερη βαθμολογία*

Το δεύτερο κριτήριο έχει να κάνει με την διαπίστωση ότι οι προγραμματιστές προτιμούν μικρότερα τμήματα κώδικα όταν φάχνουν παραδείγματα για το πως θα επιλύσουν ένα πρόβλημα. Έτσι, προωθούμε τα μικρότερα τμήματα κώδικα σε υψηλότερες θέσεις. Τα τμήματα κώδικα λαμβάνουν ποινή  $P$  με εκθετικό τρόπο όσο αυξάνονται οι γραμμές κώδικα τους σύμφωνα με τη συνάρτηση:

$$P(LoC) = e^{\left(\frac{LoC}{MeanLoC}\right)}$$

όπου  $LoC$  είναι το πλήθος των γραμμών κώδικα του τρέχοντος τμήματος κώδικα, και  $MeanLoC$  είναι ο μέσος όρος του πλήθος των γραμμών κώδικα για όλα τα τμήματα

κώδικα. Έτσι snippets που έχουν σχετικά μικρή έκταση λαμβάνουν μικρές ποινές, ενώ αυτά που έχουν περισσότερες γραμμές κώδικα δέχονται και με απότομο τρόπο μεγαλύτερες ποινές. Η τελική βαθμολογία  $FS$  του εκάστοτε τμήματος κώδικα είναι ο συγκερασμός των δύο παραπάνω τιμών ως:

$$FS = D - P$$

όπου  $D$  η τιμή που προέκυψε με βάση τις αποστάσεις και  $P$  η ποινή λόγω των γραμμών κώδικα. Τέλος, οι βαθμολογίες των τμημάτων κώδικα γίνονται scaling στο διάστημα  $[0, 1]$ .

Τέλος, κάθε cluster λαμβάνει μία γενική βαθμολογία, η οποία προκύπτει από το μέσο όρο των ατομικών βαθμολογιών των τμημάτων κώδικα που περιέχει λαμβάνοντας υπόψη μόνο το κριτήριο της απόστασης  $D$ . Ο λόγος που στο σημείο αυτό που αγνοούμε το μήκος των τμημάτων κώδικα  $P$  είναι επειδή θέλουμε να προωθήσουμε ψηλότερα τα clusters που περιέχουν πιο συνεκτικά παραδείγματα. Αυτή η βαθμολόγηση των clusters είναι βοηθητική ώστε ο χρήστης να έχει μία ιδέα της ποιότητας κάθε cluster πριν περιηγηθεί στο εσωτερικό του. Επίσης, σε περιπτώσεις που ο χρήστης έχει ελάχιστη γνώση για το αντικείμενο για το οποίο φάχνει να βρει παραδείγματα, μπορεί απευθείας να περιηγηθεί στο cluster με την υψηλότερη βαθμολογία μιας και αυτό είναι πιθανό να περιέχει μία επαρκή λύση.

#### 4.9.2 Παρουσίαση των Αποτελεσμάτων

Εφόσον τα αποτελέσματα έχουν ομαδοποιηθεί και βαθμολογηθεί το μόνο που μένει είναι η παρουσίαση τους στο χρήστη. Στο πλαίσιο της εργασίας, δημιουργήθηκε μια διεπαφή γραμμής εντολών (command line interface) για την υποβολή ερωτημάτων και την παρουσίαση των αποτελεσμάτων.

Το μόνο που απαιτείται από το χρήστη είναι η εισαγωγή του ερωτήματος, και προαιρετικά η ρύθμιση των παραμέτρων για ιδιαίτερες επιλογές (επιλογή classifier στον Readability Evaluation, μεθόδου στην εκτίμηση του πλήθους των clusters στον Clusterer, εμφάνιση διαγραμμάτων ομαδοποίησης κ.α.). Ο τρόπος χρήσης και οι επιλογές του συστήματος παρουσιάζονται παρακάτω:

Το μόνο που απαιτείται από το χρήστη είναι η εισαγωγή του ερωτήματος, και

προαιρετικά η ρύθμιση των παραχάτω παραμέτρων σε περίπτωση που δεν επιθυμεί να χρησιμοποιήσει αυτές που έχουν τεθεί ως προεπιλογή.

- *download log*. Μπορεί να ενεργοποιηθεί σε περίπτωση που επιθυμεί να του εμφανίζονται πληροφορίες κατά το κατέβασμα των τμημάτων κώδικα από το Internet. *Προεπιλογή* ανενεργό.
- *readability clf*. Μέσω αυτής της παραμέτρου μπορεί να επιλέξει ποιος ταξινομητής θα χρησιμοποιηθεί στη διαδικασία αξιολόγησης αναγνωσιμότητας. *Διαθέσιμες επιλογές*: knn, randomforest, adaboost (προεπιλογή).
- *method*. Μέσω αυτής της παραμέτρου μπορεί να επιλέξει ποια μέθοδος θα χρησιμοποιηθεί στην επιλογή του πλήθος των ομάδων κατά τη διαδικασία της ομαδοποίησης. *Διαθέσιμες επιλογές*: hc (Hierarchical Clustering), sa (Silhouette Analysis), em (Elbow Method), ap (Affinity Propagation), boa (Best of All - Προεπιλογή).
- *dendrogram*. Μπορεί να ενεργοποιηθεί σε περίπτωση που επιθυμεί να του εμφανιστεί το δενδρόγραμμα από την ιεραρχική ομαδοποίηση. *Προεπιλογή* ανενεργό.
- *elbow*. Μπορεί να ενεργοποιηθεί σε περίπτωση που επιθυμεί να του εμφανιστεί το διάγραμμα από τη μέθοδο του γονάτου. *Προεπιλογή* ανενεργό.
- *silhouette*. Μπορεί να ενεργοποιηθεί σε περίπτωση που επιθυμεί να του εμφανιστούν τα διαγράμματα από τη μέθοδο ανάλυσης σιλουέτας. *Προεπιλογή* ανενεργό.
- *affinity propagation*. Μπορεί να ενεργοποιηθεί σε περίπτωση που επιθυμεί να του εμφανιστεί το διάγραμμα από τη μέθοδο του affinity propagation. *Προεπιλογή* ανενεργό.

```

1 usage: codecatch.py [-h] [--download_log DOWNLOAD_LOG]
2                               [--readability_clf READABILITY_CLF] [--method METHOD]
3                               [--dendrogram] [--elbow] [--silhouette]
4                               [--affinity_propagation]
5                               query
6
7 positional arguments:
```

```

8     query          The query string to download code snippets.
9
10    optional arguments:
11      -h, --help           show this help message and exit
12      --download_log DOWNLOAD_LOG
13                  Show logging while downloading new snippets. Default
14                  is '0' (off).
15      --readability_clf READABILITY_CLF
16                  Specify which classifier will be used for readability
17                  evaluation. Available options: knn, randomforest,
18                  adaboost (default).
19      --method METHOD      Specify which method will be used for selecting the
20                  number of clusters. Available options: (1) hc
21                  (Hierarchical Clustering), (2) sa (Silhouette
22                  Analysis), (3) em (Elbow Method), (4) ap (Affinity
23                  Propagation), (5) boa (Best of All). Run all above
24                  methods and choose the most common result.
25      --dendrogram        Show the dendrogram plot after clustering.
26      --elbow             Show the elbow method plot after clustering.
27      --silhouette         Show the silhouette analysis plot after clustering.
28      --affinity_propagation
29                  Show the affinity propagation plot after clustering.

```

Για παράδειγμα, αν επιθυμούμε να βρούμε παραδείγματα για το ερώτημα "*How to play audio file*" δεν έχουμε παρά να εκτελέσουμε την εντολή:

```
1 $ python3 codecatch.py "how to play audio file"
```

Μετά από περίπου 15 δευτερόλεπτα που είναι και ο μέσος χρόνος απόκρισης του συστήματος μας, προβάλλονται στο χρήστη το πλήθος των clusters που έχουν σχηματιστεί, καθώς επίσης η μέση βαθμολογία κάθε cluster και το πλήθος των τμημάτων κώδικα που περιέχει. Επίσης, για κάθε cluster προβάλλονται τα 8 πιο αντιπροσωπευτικά API calls που αυτό περιέχει. Αυτή η πληροφορία είναι χρήσιμη γιατί βοηθά το χρήστη να σχηματίσει μία πρώτη ιδέα για το τι περιέχει κάθε cluster.

```

1 Query: how to play audio file
2
3 Top APIs per cluster:
4
5 Cluster 1 | Avg. Cluster Score 2.51  14 snippets:
6 play, round, add, create, getDefaultSensor, getAudioFileFormat, printStackTrace, getData
7
8 Cluster 2 | Avg. Cluster Score 14.04  17 snippets:
9 open, getaudioinputstream, start, getResource, write, getClip, getFormat, read

```

```

10
11 Cluster 3 | Avg. Cluster Score 5.08 11 snippets:
12 drain, stop, isPlaying, pause, setContentView, prepare, setDataSource, getClass
13
14 Enter ID number of cluster to see snippets. To exit enter -1.
15 Cluster ID:

```

Στη συνέχεια ο χρήστης μπορεί να επιλέξει το cluster του οποίου θέλει να εξετάσει τα τμήματα κώδικα. Έστω ότι επιλέγει το δεύτερο το οποίο έχει και την υψηλότερη βαθμολογία. Τότε αρχίζει να βλέπει τμήματα κώδικα που ανήκουν στο cluster αυτό με δυνατότητες μετάβασης στο επόμενο ή προηγούμενο τμήμα κώδικα και αλλαγής cluster. Για παράδειγμα το πρώτο τμήμα κώδικα που βλέπει μετά την παραπάνω επιλογή του είναι το ακόλουθο:

```

1 Snippet 1 MethodInvocations:
2
3 ['open', 'getClip', 'getAudioInputStream', 'printStackTrace', 'start']
4
5 Snippet 1 / 17 code:
6
7 ****CODE*****
8
9 import java.io.*;
10 import java.net.URL;
11 import javax.sound.sampled.*;
12 import javax.swing.*;
13
14 // To play sound using Clip, the process need to be alive.
15 // Hence, we use a Swing application.
16 public class SoundClipTest extends JFrame {
17
18     public SoundClipTest() {
19         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20         this.setTitle("Test Sound Clip");
21         this.setSize(300, 200);
22         this.setVisible(true);
23
24         try {
25             // Open an audio input stream.
26             URL url = this.getClass().getClassLoader().getResource("gameover.wav");
27             AudioInputStream audioIn = AudioSystem.getAudioInputStream(url);
28             // Get a sound clip resource.
29             Clip clip = AudioSystem.getClip();
30             // Open audio clip and load samples from the audio input stream.
31             clip.open(audioIn);
32             clip.start();

```

```

33     } catch (UnsupportedAudioFileException e) {
34         e.printStackTrace();
35     } catch (IOException e) {
36         e.printStackTrace();
37     } catch (LineUnavailableException e) {
38         e.printStackTrace();
39     }
40 }
41
42 public static void main(String[] args) {
43     new SoundClipTest();
44 }
45 }
46
47 ****
48 LOC Penalty = 3.330
49 API Weight = 17.924
50 SCORE = 0.755
51 URL Position: 3
52 In Page Order: 4
53 URL: https://stackoverflow.com/questions/26305/how-can-i-play-sound-in-java

```

Μαζί με τον κώδικα του τμήματος κώδικα παρουσιάζονται κάποια επιπλέον στοιχεία, όπως τα API calls που χρησιμοποιούνται σε αυτό, η βαθμολογία του, η ιστοσελίδα από την οποία προέρχεται κλπ. Με ευκολία μπορεί να περιηγηθεί και στα επόμενα τμήματα κώδικα του cluster ή να μεταβεί σε άλλο cluster για να εξετάσει διαφορετικές λύσεις.

## 4.10 Σύνοψη Κεφαλαίου

Σε αυτό το κεφάλαιο μελετήσαμε την υλοποίηση ενός συστήματος προτάσεων στην τεχνολογία λογισμικού (RSSE). Συγκεκριμένα, αφού είδαμε τη δομή του συστήματος μακροσκοπικά στη συνέχεια αναλύσαμε το κάθε κομμάτι του αναλυτικά. Παράλληλα αναφερθήκαμε στις δυσκολίες που αντιμετωπίσαμε κατά τη σχεδίαση και περιγράψαμε τα βήματα που ακολουθήσαμε για την επίλυση τους. Τέλος, αφού συνδέσαμε τα διάφορα κομμάτια μεταξύ τους είδαμε παραδείγματα από τον τρόπο λειτουργίας του. Στο επόμενο κεφάλαιο θα θέσουμε το σύστημα σε πειραματισμό και θα μελετήσουμε τη χρησιμότητα του.

# Κεφάλαιο 5

## Αξιολόγηση και Πειράματα

### 5.1 Γενικά

Στο προηγούμενο κεφάλαιο μελετήσαμε τον τρόπο σχεδίασης και κατασκευής ενός συστήματος προτάσεων στην τεχνολογία λογισμικού. Το επόμενο βήμα είναι η αξιολόγηση του συστήματος μας. Για το σκοπό αυτό, θα διεξάγουμε κάποια πειράματα τα οποία θα μας βοηθήσουν να συγκρίνουμε το σύστημα μας με άλλες υπάρχουσες λύσεις και επιλογές που έχει ένας μηχανικός λογισμικού.

Ένας προφανής τρόπος αξιολόγησης του συστήματος μας θα ήταν η άμεση σύγκριση του με άλλα παρόμοια συστήματα. Δυστυχώς όμως κάτι τέτοιο δεν ήταν εφικτό. Αν και κατά καιρούς έχουν αναπτυχθεί παρόμοια συστήματα, τις περισσότερες φορές έχει συμβεί στα πλαίσια ερευνών χωρίς να γίνουν διαθέσιμα προς το ευρύτερο κοινό. Ακόμη, συστήματα που είχαν δημοσιοποιηθεί, σταμάτησαν να συντηρούνται με αποτέλεσμα να σταματήσει και η λειτουργία τους.

Επίσης, η άμεση σύγκριση με πολλά συστήματα θα ήταν αδύνατη γιατί τα περισσότερα από αυτά διαφέρουν ως προς το στόχο τους. Συγκεκριμένα, τα περισσότερα από αυτά δεν έχουν ως στόχο την πρόταση ακέραιων παραδειγμάτων κώδικα στο χρήστη, όσο να βρουν και να προτείνουν κάποια μοτίβα. Ένα σύστημα που έχει αντίστοιχη λειτουργία με το δικό μας, και παραμένει διαθέσιμο και λειτουργικό προς το κοινό, είναι το Bing Code Search που εξετάσαμε και στη βιβλιογραφία. Ωστόσο, το σύστημα αυτό έχει σχεδιαστεί για να προτείνει παραδείγματα στη

γλώσσα προγραμματισμού *C#*<sup>1</sup>. Κάτι τέτοιο καθιστά αδύνατη τη σύγκριση με το δικό μας σύστημα που στοχεύει στη γλώσσα Java.

Λόγω των παραπάνω, αποφασίστηκε η αξιολόγηση του συστήματος μας να γίνει συγκρίνοντας το με τη μηχανή αναζήτησης της Google. Συγκεκριμένα, καταστρώθηκαν πειράματα τα οποία αντιπροσωπεύουν ρεαλιστικά σενάρια χρήσης που θα ακολουθούσε ένας προγραμματιστής ή μηχανικός λογισμικού στην προσπάθεια του να βρει λύσεις σε κάποιο πρόβλημα. Στις επόμενες ενότητες θα περιγράψουμε τα σενάρια αυτά και μέσω μετρικών θα συγκρίνουμε τα αποτελέσματα του συστήματος μας έναντι της μηχανής αναζήτησης της Google.

Τέλος, ένα σημείο που οφείλουμε να διευκρινίσουμε είναι ο τρόπος με τον οποίο έγινε η αξιολόγηση. Όπως καταλαβαίνει κανείς, στον χλάδο της ανάκτησης πληροφορίας υπάρχει έντονα η υποκειμενική κρίση. Ακόμη περισσότερο σε μία περίπτωση όπως η δική μας, όπου δεν υπάρχει ένα και μοναδικός τρόπος επίλυσης των προβλημάτων, ο παραγόντας αυτός μπορεί να επηρεάσει την εγκυρότητα των αποτελεσμάτων. Ένα τρόπος για να μειωθεί η υποκειμενικότητα θα ήταν να ζητήσουμε από ένα σεβαστό πλήθος ατόμων να κάνει χρήση του συστήματος μας σε πραγματικές συνθήκες και έτσι να συλλέξουμε δεδομένα. Όμως, κάτι τέτοιο δεν ήταν δυνατό, λόγω έλλειψης ανθρωπίνων πόρων, οπότε προσπαθήσαμε να επιλέξουμε πειράματα που απαιτούν ουδέτερη κρίση ώστε τα αποτελέσματα μας να είναι όσο γίνεται αντικειμενικότερα.

## 5.2 Συγκέντρωση Συνόλου Ερωτημάτων

Πριν ξεκινήσουμε, πρέπει να συγκεντρώσουμε ένα σύνολο από ερωτήματα (queries) τα οποία θα αποτελέσουν τη βάση με την οποία θα διεξαχθούν όλα τα πειράματα μας. Τα ερωτήματα αυτά πρέπει να είναι ρεαλιστικά (δηλ. να είναι ερωτήματα που έντονας χρήστης θα μπορούσε να θέσει στο σύστημα) και να προέρχονται από διαφορετικά αντικείμενα της γλώσσας Java. Λαμβάνοντας υπόψιν τα παραπάνω κριτήρια, κατασκευάστηκαν 15 queries τα οποία αναγράφονται στον Πίνακα 5.1. Μαζί εμφανίζονται και το πλήθος των ομάδων που σχηματίστηκαν από το σύστημα μας για καθένα από τα ερωτήματα, ο αριθμός των τμημάτων κώδικα που αξιολο-

<sup>1</sup>[https://en.wikipedia.org/wiki/C\\_Sharp\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/C_Sharp_(programming_language))

Πίνακας 5.1: Το σύνολο των ερωτημάτων που θα χρησιμοποιηθούν για την αξιολόγηση του συστήματος μοζί με το πλήθος των τελικών ομάδων που σχηματίστηκαν (Clusters), το πλήθος των τελικών τμημάτων κώδικα (Snippets), και το πλήθος των τμημάτων κώδικα ανά ομάδα (Snippets/Cluster).

Index	Query	Clusters	Snippets	Snippets/Cluster
1	How to read CSV file	3	44	14.7
2	How to generate MD5 hash code	4	43	10.8
3	How to upload file to FTP	3	13	4.3
4	How to split string	5	53	10.6
5	How to draw text graphics	3	40	13.3
6	How to play audio file	5	79	15.8
7	How to substitute string	3	31	10.3
8	How to convert collection to an array	4	49	12.3
9	How to send email	3	46	15.3
10	How to connect to a JDBC database	5	44	8.8
11	How to execute select statement JDBC database	3	59	19.7
12	How to initialize thread	3	33	11.0
13	How to write binary data	3	35	11.7
14	How to read ZIP archive	2	32	16.0
15	How to send packet via UDP	2	31	15.5

γήθηκαν ως κατάλληλα για να ενταχθούν στη λίστα αποτελεσμάτων, και τέλος ο μέσος όρος των τμημάτων κώδικα που αντιστοιχούν ανά ομάδα.

Όπως φαίνεται τα ερωτήματα που συγκροτήθηκαν είναι απλά, επιτρέποντας έτσι την συγκέντρωση πληθώρας διαφορετικών τρόπων υλοποίησης, και επίσης έχουν να κάνουν με χρήσιμα αντικείμενα της Java που προέρχονται από ποικίλους τομείς.

Σημειώνεται πως για όλα τα παραπάνω ερωτήματα στα πειράματα που ακολουθούν χρησιμοποιήθηκαν οι προεπιλεγμένες ρυθμίσεις για το σύστημα μας. Δηλαδή, στον αξιολογητή αναγνωσιμότητας χρησιμοποιείται ο ταξινομητής AdaBoost, και στην επιλογή του πλήθους των ομάδων χρησιμοποείται η μέθοδος Best of All.

### 5.3 Πείραμα 1o - Ακρίβεια Αποτελεσμάτων

Με το πρώτο πείραμα έχουμε σκοπό να μελετήσουμε την ακρίβεια των αποτελεσμάτων του συστήματος μας και να τη συγχρίνουμε με την ακρίβεια της μηχανής αναζήτησης της Google. Για το σκοπό αυτό θα επιστρατεύσουμε μετρικές αξιολόγησης συστημάτων ανάκτησης πληροφορίας, και συγκεκριμένα τις μετρικές Average Precision καθώς και Mean Average Precision<sup>2</sup>.

**Average Precision.** Μετρικές όπως το precision και το recall απαιτούν ολόκληρο το σύνολο των επιστρεφόμενων εγγράφων για να εφαρμοστούν. Κάτι τέτοιο πολλές φορές δεν είναι λειτουργικό σε συστήματα που επιστρέφουν μεγάλο αριθμό

<sup>2</sup>[https://en.wikipedia.org/wiki/Evaluation\\_measures\\_\(information\\_retrieval\)](https://en.wikipedia.org/wiki/Evaluation_measures_(information_retrieval))

αποτελεσμάτων. Ακόμη, σε συστήματα που επιστρέφουν έγγραφα με βάση κάποια βαθμολόγηση, είναι επιθυμητό η σειρά αυτή να ληφθεί υπόψιν στην αξιολόγηση, κάτι που δεν συμβαίνει για τις μετρικές precision και recall. Ωστόσο, υπολογίζοντας το precision και το recall σε κάθε θέση μίας διατεταγμένης λίστας, μπορούμε να σχεδιάσουμε την  $p(r)$ , δηλαδή το precision  $p$  ως συνάρτηση του recall  $r$ . Το ολοκλήρωμα της  $p(r)$  στο διάστημα από  $r = 0$  ως  $r = 1$  ισούται με το Average Precision.

$$AveP = \int_0^1 p(r)dr$$

Το παραπάνω ολοκλήρωμα σε πρακτικές εφαρμογές μπορεί να αντικατασταθεί από το πεπερασμένο άθροισμα:

$$AveP = \sum_{k=1}^n P(k)\Delta r(k)$$

όπου  $k$  είναι η θέση στην ακολουθία των εγγράφων,  $n$  ο αριθμός των εγγράφων που επιστράφηκαν,  $P(k)$  το precision στο σημείο  $k$ , και  $\Delta r(k)$  η μεταβολή στο recall από το σημείο  $k - 1$  στο  $k$ . Τέλος, το παραπάνω άθροισμα ισούται τελικά με:

$$AveP = \frac{\sum_{k=1}^n (P(k) \cdot rel(k))}{\text{number of relevant documents}}$$

όπου το  $rel(k)$  ισούται με 1 άμα το έγγραφο στη θέση  $k$  είναι σχετικό με το query, και με 0 σε αντίθετη περίπτωση.

Πίνακας 5.2: Τα δέκα πρώτα αποτελέσματα ενός ερωτήματος. Μεταξύ τους υπάρχουν σχετικά (1) και μη σχετικά έγγραφα (0).

Document Index	1	2	3	4	5	6	7	8	9	10
Document Relevance	1	0	1	0	1	0	0	1	0	1

Για παράδειγμα, το Average Precision της ακολουθίας στον Πίνακα 5.2 ισούται με  $(\frac{1}{1} + \frac{2}{3} + \frac{3}{5} + \frac{4}{8} + \frac{5}{10})/5 = 0.65$

**Mean Average Precision.** Η μετρική αυτή δεν είναι τίποτα παραπάνω από τον μέσο όρο των average precisions για ένα πλήθος από διαφορετικά queries. Δηλαδή,

$$MAP = \frac{\sum_{q=1}^Q AveP(q)}{Q}$$

όπου  $Q$  είναι το πλήθος των queries.

Τα πλεονεκτήματα αυτών των μετρικών είναι ότι μπορούν να υπολογιστούν σχετικά εύκολα με παρατήρηση των αποτελεσμάτων, καθώς επίσης και ότι λαμβάνουν υπόψιν τους την κατάταξη που έχει προκύψει. Δηλαδή, μία κατάταξη που έχει ίσα σχετικά έγγραφα με μία δεύτερη, αλλά αυτά βρίσκονται σε υψηλότερες θέσεις, θα έχει καλύτερο Average Precision από τη δεύτερη της οποίας τα ισάριθμα σχετικά έγγραφα είναι τοποθετημένα στις τελευταίες θέσεις.

**Σχετικά Έγγραφα.** Πριν προχωρήσουμε στην παρουσίαση του πειράματος, πρέπει να ορίσουμε ποια έγγραφα θα θεωρούμε σχετικά. Ως σχετικά έγγραφα (δηλ. snippets) θεωρούνται αυτά που είναι σχετικά με το ερώτημα και προσπαθούν να επιλύσουν το πρόβλημα που περιγράφεται από αυτό. Π.χ. στο πρώτο ερώτημα της λίστας μας (Πίνακας 5.1) "*How to read CSV file*", σχετικά θεωρούνται όλα τα snippets που έχουν να κάνουν με ανάγνωση αρχείου τύπου Comma-Separated Values (CSV). Αν ένα snippet περιέχει κώδικα σχετικό με CSV αρχεία αλλά για κάποια άλλη διαδικασία, π.χ. γράψιμο, τότε θεωρείται μη σχετικό. Αν όμως ένα snippet περιέχει κώδικα για γράψιμο αλλά και για ανάγνωση τότε είναι σχετικό για το πείραμα μας. Τέλος, όλες οι πιθανές λύσεις (διάφοροι τρόποι υλοποίησης, με διαφορετικά API calls κλπ.) που μπορούν να κάνουν ανάγνωση αρχείων τέτοιου τύπου θεωρούνται σχετικές, άσχετα αν είναι βέλτιστες ή προτιμώμενες από το χρήστη.

Το σενάριο χρήσης του συστήματος μας για το πείραμα αυτό έχει ως εξής:

- Ο χρήστης πυροδοτεί το σύστημα στέλνοντας ένα ερώτημα και περιμένει τα αποτελέσματα.
- Ο χρήστης εξετάζει την προεπισκόπηση των ομάδων όπου εμφανίζονται τα δημοφιλέστερα API calls σε καθεμία από αυτές.
- Ο χρήστης επιλέγει με βάση την προεπισκόπηση σωστή ομάδα και αρχίζει να περιηγείται στα αποτελέσματα.

Για τις ανάγκες του πειράματος εξετάζουμε τα 15 πρώτα snippets και σημειώνουμε ποια από αυτά είναι σχετικά και ποια μη σχετικά. Σε περίπτωση που τα snippets του cluster εξαντληθούν, μεταφερόμαστε στην επόμενη ομάδα κ.ο.κ.

Στη συνέχεια, για τα ίδια ερωτήματα μαζεύουμε μετρήσεις μέσω της μηχανής αναζήτησης. Συγκεκριμένα:

1. Ο χρήστης πυροδοτεί τη μηχανή αναζήτησης στέλνοντας ένα ερώτημα και περιμένει τα αποτελέσματα της αναζήτησης.
2. Ο χρήστης μπαίνει επισκέπτεται τη πρώτη ιστοσελίδα των αποτελεσμάτων.
3. Ο χρήστης εξετάζει τα snippets στη σελίδα από πάνω έως κάτω.

Εξετάζουμε τα 15 πρώτα τμήματα κώδικα και σημειώνουμε αν είναι σχετικά ή όχι. Σε περίπτωση που δεν υπάρχουν άλλα τμήματα κώδικα στην ιστοσελίδα, αν ακόμη δεν έχουμε συμπληρώσει 15, μεταφερόμαστε στην επόμενη ιστοσελίδα κ.ο.κ.

Με βάση τα παραπάνω υπολογίστηκε το *Average Precision* σε τρεις θέσεις για το κάθε ερώτημα ( $k = 5, 10, 15$ ). Τα αποτελέσματα του πειράματος απεικονίζονται στο Σχήμα 5.1. Όπως παρατηρούμε, στα περισσότερα ερωτήματα το σύστημα μας (Codecatch) υπερισχύει έναντι της μηχανής αναζήτησης (Google). Ακόμη και στις περιπτώσεις που η μηχανή αναζήτησης είναι καλύτερη, το σύστημα μας ακολουθεί πολύ κοντά στο σκορ. Ας δούμε λιγάκι πιο συγκεκριμένα τι συμβαίνει στις περιπτώσεις όπου υπάρχει διαφορά μεταξύ των αποτελεσμάτων. Π.χ. στο ερώτημα Q3 ("How to upload file to FTP") ο λόγος της κακής απόδοσης της μηχανής αναζήτησης οφείλεται ίσως στο γεγονός της βαρύτητας που προσδίδει ο όρος FTP σε συνδυασμό με το μικρό πλήθος διαφορετικών τμημάτων κώδικα για το ερώτημα αυτό. Έτσι, σε αρκετά αποτελέσματα στην κατάταξη της Google εμφανίζονται τμήματα κώδικα που έχουν μεν σχέση με το πρωτόκολλο FTP αλλά όχι αποκλειστικά για το ανέβασμα αρχείων (συναντώνται τμήματα κώδικα για σύνδεση μέσω FTP, για κατέβασμα αρχείων κλπ.). Αντίθετα, επειδή το σύστημα μας ομαδοποιεί τα αποτελέσματα με βάση τα API calls, δίνει μεγάλη βαρύτητα στη μέθοδο *upload* και έτσι συγκεντρώνει μαζί τα τμήματα κώδικα που την περιέχουν και τα κατατάσσει μαζί. Στο ερώτημα Q4 ("How to split string"), το οποίο είναι ένα πολύ συνηθισμένο ερώτημα (2.6 εκατομμύρια προβολές στο StackOverflow<sup>3</sup>) υπάρχει μεγάλη πληθώρα τμημάτων κώδικα. Ωστόσο, επειδή πολλές από αυτές βρίσκονται σε ιστοσελίδες συζητήσεων ή ερωτοαπαντήσεων (π.χ. StackOverflow), όπου η κάθε μία απάντηση πρέπει να προσφέρει κάτι διαφορετικό από τις υπόλοιπες ώστε να συμφωνεί με τη φιλοσοφία της ιστοσελίδας, μαζί με τις σχετικές λύσεις πολλές φορές το θέμα παρεκκλίνει με αποτέλεσμα να σχολιάζονται και άλλες υλοποιήσεις όχι άμεσα σχε-

<sup>3</sup><https://stackoverflow.com/questions/tagged/java?page=4&sort=votes&pagesize=15>

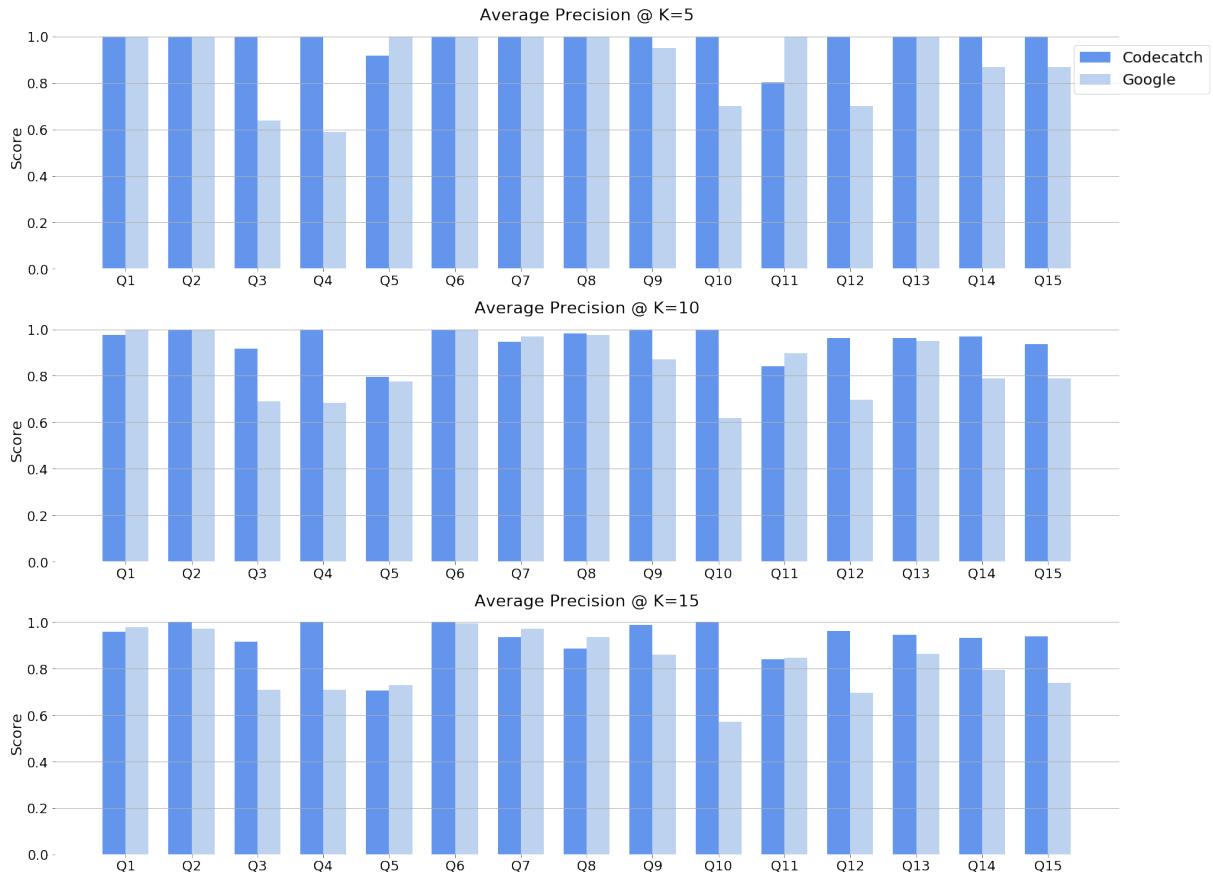
τικές. Επίσης, πολλές φορές σε τέτοιες ιστοσελίδες συναντώνται κομμάτια κώδικα που δίνονται από τους χρήστες με κενά επειδή οι τελευταίοι δεν γνωρίζουν πως να υλοποιήσουν το ζητούμενο από το ερώτημα και επιθυμούν να ζητήσουν βοήθεια από τους υπόλοιπους. Ένας άλλος λόγος της διαφοράς ανάμεσα στα συστήματα, π.χ. στο Q12 ("How to initialize thread"), μπορεί να οφείλεται στο γεγονός ότι πολλές ιστοσελίδες έχουν εκπαιδευτικό χαρακτήρα που είναι γενικότερου ενδιαφέροντος γύρω από το Java thread programming<sup>4</sup> (π.χ. tutorials). Οι ιστοσελίδες αυτές έχουν σκοπό να παρουσιάσουν πληροφορία γενικά για τα τμήματα στη γλώσσα Java και όχι απλώς για την εκκίνηση τους. Τέλος, στα ερωτήματα που το σύστημα μας υστερεί, π.χ. Q11 ("How to execute select stament JDBC database"), αυτό οφείλεται κατά ένα μέρος στα API calls που εμφανίζονται. Στη συγκεκριμένη περίπτωση το API του JDBC παρουσιάζει συχνά όμοια ονόματα μεθόδων για διαφορετικές διαδικασίες<sup>5</sup> (π.χ. createStatement αντί για createSelectStatement, executeQuery αντί για executeSelectQuery). Σε μία τέτοια περίπτωση το σύστημα μας είναι πιθανό να ομαδοποιήσει μαζί τμήματα κώδικα που έχουν ίδια API calls αλλά εξυπηρετούν διαφορετικούς σκοπούς, π.χ. "stmt.executeQuery('SELECT id FROM people WHERE ...')” και "stmt.executeQuery('DELETE FROM people WHERE id ...')".

Έπειτα, για να αποκτήσουμε μία συνολικότερη εικόνα της απόδοσης του συστήματος μας ως προς αυτή τη μετρική υπολογίζουμε το *Mean Average Precision* στις θέσεις  $k = 5, 10, 15$ . Τα αποτελέσματα παρουσιάζονται στο Σχήμα 5.2. Παρατηρούμε ότι και στις τρεις θέσεις το σύστημα μας αποδίδει καλύτερα από τη μηχανή αναζήτησης.

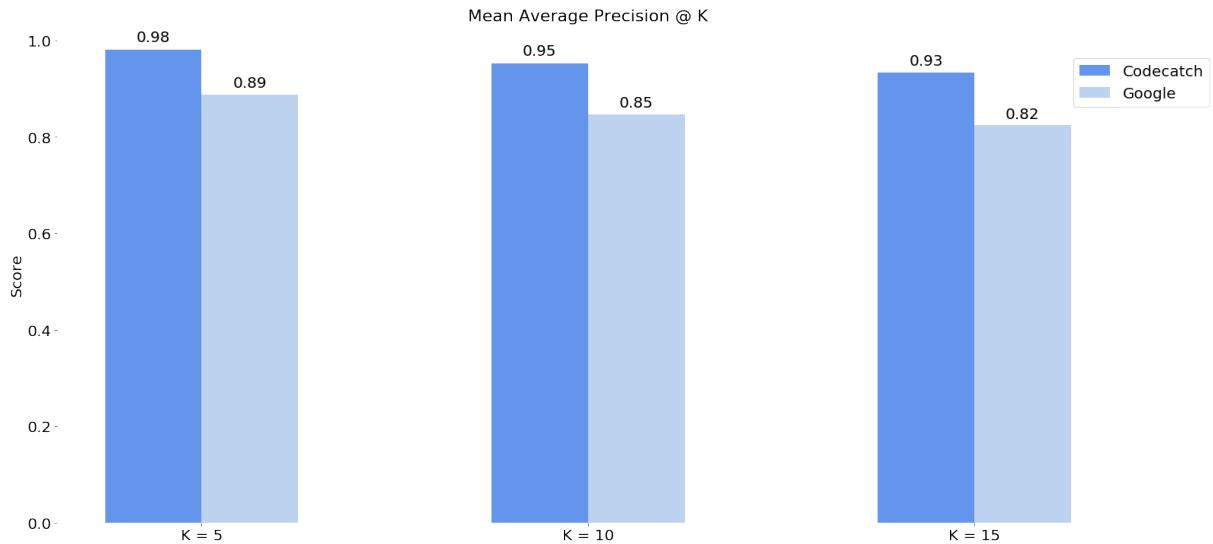
Γενικά, από το πείραμα αυτό μπορούμε να συμπεράνουμε ότι όταν ο χρήστης επιλέγει τη σωστή ομάδα, κάτι που δεν είναι δύσκολο να συμβεί μιας και τα δημοφιλή APIs που του εμφανίζονται στην προεπισκόπηση είναι αρκετά καθοδηγητικά, είναι πιθανότερο να βρει περισσότερα σχετικά τμήματα κώδικα και σε καλύτερες θέσεις στη διάταξη από ότι αν χρησιμοποιούσε τη μηχανή αναζήτησης.

<sup>4</sup>[https://www.tutorialspoint.com/java/java\\_multithreading.htm](https://www.tutorialspoint.com/java/java_multithreading.htm)

<sup>5</sup>[https://www.tutorialspoint.com/jdbc/jdbc\\_select\\_records.htm](https://www.tutorialspoint.com/jdbc/jdbc_select_records.htm)



Σχήμα 5.1: Αποτελέσματα Average Precision στις θέσεις  $K = 5, 10, 15$  για καθένα από τα 15 ερωτήματα.



Σχήμα 5.2: Αποτελέσματα Mean Average Precision στις θέσεις  $K = 5, 10, 15$ .

## 5.4 Πείραμα 2o - Μέσο Μήκος Αναζήτησης

Ως επόμενο πείραμα θα εξετάσουμε το μήκος αναζήτησης (Search Length (SL)) ανάμεσα στο σύστημα μας και στη μηχανή αναζήτησης της Google, και συγκεκρι-

μένα το μέσο μήκος αναζήτησης (Mean Search Length (MSL)). Πρόκειται, για ένα πολύ χρήσιμο πείραμα γιατί μέσω αυτού θα ελέγξουμε αν έχουμε άφθονα σχετικά αποτελέσματα αλλά και σε καλές θέσεις στην κατάταξη. Ως μήκος αναζήτησης ορίζεται:

*...το πλήθος των μη σχετικών αποτελεσμάτων που εμφανίζονται στο χρήστη έως ότου του εμφανιστούν  $N$  σχετικά [68].*

Το σενάριο χρήσης του συστήματος μας για το πείραμα του μήκους αναζήτησης συνοψίζεται στα παρακάτω βήματα.

1. Ο χρήστης πυροδοτεί τη μηχανή αναζήτησης στέλνοντας ένα ερώτημα και περιμένει τα αποτελέσματα της αναζήτησης.
2. Ο χρήστης επιλέγει κατευθείαν την ομάδα των αποτελεσμάτων με την υψηλότερη βαθμολογία, χωρίς να εξετάσει τα σημαντικότερα API calls κάθε ομάδας, και αρχίζει να περιηγείται στα αποτελέσματα.
3. Ανά πάσα στιγμή, καθώς ο χρήστης περιηγείται σε ένα cluster, αν ο αριθμός των σχετικών snippets που έχει δει σε αυτό το cluster γίνει μικρότερος των μη σχετικών snippets, τότε μεταβαίνει στο cluster με την επόμενη καλύτερη βαθμολογία και συνεχίζει από εκεί.

Για να γίνει πιο σαφής ο τρόπος πλοήγησης του χρήστη στα αποτελέσματα με βάση το σενάριο ας εξετάσουμε ένα παράδειγμα. Έστω η ομαδοποίηση του Πίνακα 5.3, όπου με 1 συμβολίζονται τα σχετικά αποτελέσματα και με 0 τα μη σχετικά, και επίσης έστω ότι ο χρήστης αναζητεί 5 σχετικά τμήματα κώδικα. Τότε θα είχαμε:

$$A_1(1), A_2(0), A_3(1), A_4(0), A_5(0), B_1(0), C_1(1), C_2(1), C_3(1)$$

Ο χρήστης έχει δει 4 μη σχετικά τμήματα κώδικα μέχρι να του παρουσιαστούν 5 σχετικά. Άρα το μήκος αναζήτησης σε αυτή την περίπτωση ισούται με 4. Σημειώνεται πως τα κριτήρια για τη σχετικότητα των τμημάτων κώδικα παραμένουν ίδια με το προηγούμενο πείραμα.

Εφόσον έχουμε υπολογίσει το μήκος αναζήτησης για κάθε ερώτημα ξεχωριστά,

Πίνακας 5.3: Παράδειγμα εύρεσης μήκους αναζήτησης. Για κάθε ομάδα δίνεται μία ακολουθία από σχετικά (1) και μη σχετικά (0) αποτελέσματα.

Index	1	2	3	4	5	6	7	8
Cluster A	1	0	1	0	0	1	1	0
Cluster B	0	1	0	1	1	0	1	0
Cluster C	1	1	0	1	0	1	1	1

στη συνέχεια υπολογίζουμε το μέσο μήκος αναζήτησης (MSL) εύκολα ως:

$$MSL = \frac{\sum_{q=1}^Q SL(q)}{Q}$$

όπου  $SL$  το μήκος αναζήτησης των ερωτημάτων 1, ..., 15 και  $Q$  ο συνολικός αριθμός των ερωτημάτων.

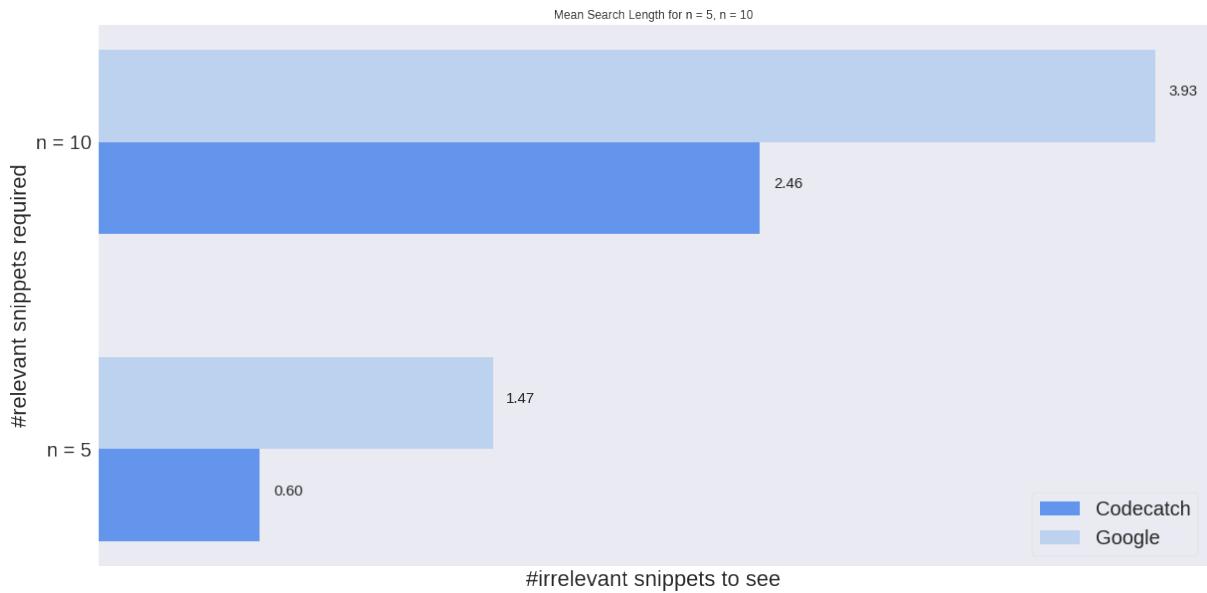
Με βάση τα παραπάνω παρουσιάζονται τα αποτελέσματα του πειράματος για  $MSL N = 5, 10$  στο Σχήμα 5.3. Επίσης, παρουσιάζονται τα  $MSL$  για κάθε  $N$  από 1, ..., 10 στο Σχήμα 5.4. Στο σχήμα αυτό, παρατηρώντας τις τάσεις των γραμμών, διαπιστώνουμε πως σχεδόν πάντα τα 2 αποτελέσματα στο σύστημα μας είναι σχετικά (αφού το  $SL$  είναι σχεδόν ίσο με μηδέν). Μια μικρή ανοδική πορεία του  $SL$  παρατηρείται για το Codecatch μόνο μετά το 5ο ζητούμενο αποτέλεσμα, ωστόσο στη συνέχεια πάλι ομαλοποιείται. Αντίθετα όσον αφορά την κατάταξη της μηχανής αναζήτησης, το  $SL$  αυξάνεται σταθερά στην αρχή, ομαλοποιείται κάπου στη μέση και έπειτα αυξάνεται και πάλι.

Μέσω των αποτελεσμάτων παρατηρούμε ότι μέσω του συστήματος μας απαιτούνται να εξεταστούν λιγότερα μη σχετικά αποτελέσματα κάθε φορά. Κάτι τέτοιο είναι πολύ σημαντικό αφού μπορεί να γλιτώσει πολύτιμο χρόνο στον χρήστη κατά την εκτέλεση μίας αναζήτησης.

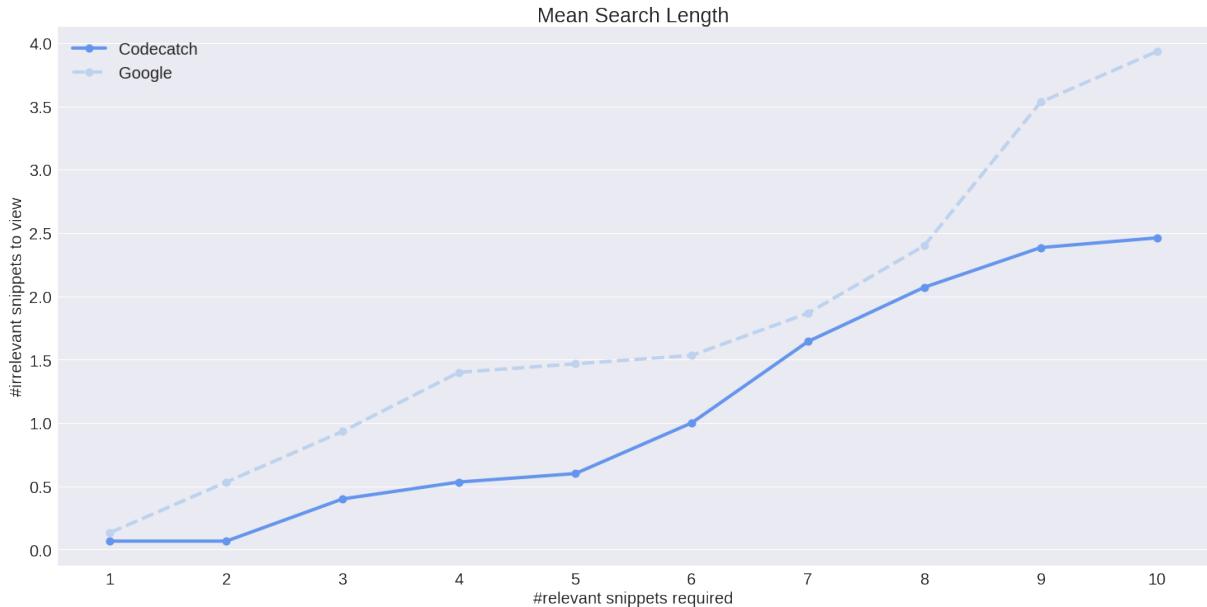
## 5.5 Πείραμα 3ο - Ποιότητα Ομαδοποίησης

Στο πείραμα αυτό θα εξετάσουμε την ποιότητα της ομαδοποίησης του συστήματος μας. Δεν θα γίνει σύγκριση με τη μηχανή αναζήτησης στο συγκεκριμένο σημείο διότι δεν μπορούμε να μετρήσουμε τα μεγέθη αυτά σε ενιαία λίστα αποτελεσμάτων.

Αρχικά, θέλουμε να εξετάσουμε κατά πόσο υπάρχει καλός διαχωρισμός (separation) και συνεκτικότητα (cohesion) στην ομαδοποίηση μας. Για το λόγο αυτό, θα χρησιμοποιήσουμε τη μετρική που είδαμε και στο Κεφάλαιο 4, το *silhouette coefficient*. Ο



Σχήμα 5.3: Μέσο μήκος αναζήτησης (MSL) για  $n = 5, 10$  για το σύστημα μας (Codecatch) και τη μηχανή αναζήτησης (Google)



Σχήμα 5.4: Μέσος μήκος αναζήτησης (MSL) για  $N = 1, \dots, 10$  για το σύστημα μας (Codecatch) και τη μηχανή αναζήτησης (Google).

λόγος που επιλέγουμε τη μετρική αυτή είναι ότι συνδυάζει το separation και το cohesion αποδίδοντας μία συνολικότερη εικόνα για την ομαδοποίηση [69].

Επίσης, μία ακόμη ποσότητα που χρησιμοποιείται στην αξιολόγηση ομαδοποίησης, και συγκεκριμένα ιεραρχικής, μη εποπτευόμενης μάθησης (unsupervised learning) είναι το CoPhenetic Correlation Coefficient (CPCC)<sup>6</sup>. Οι τιμές της σταθεράς αυτής ανήκουν στο διάστημα  $[0, 1]$  με υψηλότερες τιμές να υποδεικνύουν καλύτερες ομαδοποιήσεις.

<sup>6</sup>[https://en.wikipedia.org/wiki/Cophenetic\\_correlation](https://en.wikipedia.org/wiki/Cophenetic_correlation)

ήσεις. Το CPCC είναι ένα μέτρο του πόσο πιστά αναπαριστώνται στο δενδρόγραμμα οι ανομοιότητες μεταξύ των παρατηρήσεων.

Συλλέξαμε τις ποσότητες των παραπάνω μετρικών για καθένα από τα 15 ερωτήματα του Πίνακα 5.1. Τα αποτελέσματα της έρευνας αναγράφονται στον Πίνακα 5.4. Γενικά παρατηρούμε ότι οι τιμές και των δύο μετρικών είναι αρκετά αξιόλογες σε όλα τα ερωτήματα. Κάτι τέτοιο είναι ιδιαίτερα ενθαρρυντικό μιας και δηλώνει ότι οι ομάδες που σχηματίζονται είναι καλά διαχωρισμένες και επίσης περιέχουν ομοιογενή αποτελέσματα. Έτσι, ο χρήστης μπορεί εύκολα να συμπεράνει το περιεχόμενο κάθε ομάδας και να βρει ποικιλία παρόμοιων τμημάτων κώδικα μέσα σε καθεμία από αυτές. Τέλος, το γεγονός οι τιμές των μετρικών είναι παρόμοιες για όλα τα ερωτήματα είναι επίσης θετικό διότι υποδηλώνει πως το σύστημα είναι σταθερό.

Πίνακας 5.4: Αποτελέσματα των μετρικών Silhouette Coefficient και CoPhenetic Correlation Coefficient.

Query Index	Silhouette Coefficient	CPCC
1	0.66	0.65
2	0.63	0.61
3	0.71	0.68
4	0.73	0.84
5	0.60	0.69
6	0.69	0.65
7	0.65	0.68
8	0.74	0.75
9	0.63	0.64
10	0.69	0.64
11	0.67	0.63
12	0.69	0.69
13	0.65	0.66
14	0.62	0.63
15	0.61	0.63
Mean Value	0.66	0.67

## 5.6 Σύνοψη Κεφαλαίου

Στο κεφάλαιο αυτό διεξήγαμε ένα σύνολο πειραμάτων μέσω των οποίων εξετάσαμε την απόδοση του συστήματος μας. Επίσης, έγινε σύγκριση της αποτελεσματικότητας της σχεδίασης μας έναντι στην εξαιρετικά δημοφιλή μηχανή αναζήτησης της Google. Τα αποτελέσματα που προέκυψαν παρουσιάστηκαν στον αναγνώστη μέσω των κατόλληλων σχημάτων για την καλύτερη εποπτεία τους.

Στη συνολική εικόνα τους τα αποτελέσματα ήταν ιδιαίτερα θετικά για το σύστημα μας. Παρόλο που ακολουθήσαμε διαφορετική προσέγγιση αξιολόγησης και παρουσίασης των αποτελεσμάτων από αυτόν της μηχανής αναζήτησης, το σύστημα που προέκυψε αποδείχθηκε αποτελεσματικό. Συγκεκριμένα, μέσω της χρήσης του ένας μηχανικός ή προγραμματιστής μπορεί να βρει αποτελέσματα καλής ακρίβειας και σύντομα. Επίσης, η ποιότητα ομαδοποίησης βοηθά στον εντοπισμό της επιθυμητής λύσης μέσω της συνεκτικότητας και του διαχωρισμού των αποτελεσμάτων που προσφέρει.

# Κεφάλαιο 6

## Συμπεράσματα και Μελλοντική Εργασία

### 6.1 Συμπεράσματα

Ενώ υπάρχει αφθονία πληροφορίας και λύσεων στο Διαδίκτυο, η ανάπτυξη λογισμικού παραμένει μία δύσκολη υπόθεση και ως ένα σημείο αποθαρρυντική προς νέους προγραμματιστές. Ακόμη, και άτομα που έχουν εμπειρία στο χώρο διστάζουν να επεκτείνουν τις γνώσεις τους προς νέα αντικείμενα. Ένας από τους κύριους λόγους που οφείλεται αυτό είναι ο χρόνος που απαιτείται στην ανακάλυψη καλών παραδειγμάτων.

Το παραπάνω πρόβλημα έρχονται να το λύσουν τα συστήματα προτάσεων στην τεχνολογία λογισμικού (RSSEs). Το προφανές πλεονέκτημα που παρουσιάζουν τα συστήματα αυτά είναι ότι ο χρήστης δεν χρειάζεται να ανατρέχει συνεχώς στο φυλλομετρήτη (browser) του ώστε να ψάξει παραδείγματα μέσω μίας μηχανής αναζήτησης και να αναζητεί σε ένα πλήθος σελίδων για να βρει το κατάλληλο παράδειγμα. Τη δουλειά αυτή την αναλαμβάνουν τα συστήματα RSSEs. Πολλά από τα υπάρχοντα συστήματα είναι αποτελεσματικά σε ορισμένες περιπτώσεις, ωστόσο παρουσιάζουν αδυναμίες. Συγκεκριμένα, τα περισσότερα συστήματα απαιτούν αρκετό χρόνο στην εκμάθηση τους και στη χρήση τους. Επίσης, πολλά από αυτά δεν παρέχουν έτοιμες προς χρήση λύσεις με αποτέλεσμα να περιορίζεται η χρηστικότητα τους.

Για το λόγο αυτό σχεδιάσαμε ένα σύστημα RSSE το οποίο εξειδικεύεται στην

πρόταση παραδειγμάτων κώδικα τα οποία πληρούν κάποια κριτήρια. Τα σημαντικότερα από τα κριτήρια αυτά είναι να έχουν υψηλή αναγνωσιμότητα καθώς επίσης και να είναι δημοφιλή σε πολλά έργα ανοιχτού λογισμικού. Επιπλέον, επικεντρωθήκαμε στην όσο το δυνατό καλύτερη παρουσίαση των αποτελεσμάτων ώστε ο προγραμματιστής να μπορεί να διακρίνει τον τρόπου υλοποίησης που επιθυμεί και να επιλέξει στη συνέχεια το πιο χρήσιμο τμήμα κώδικα για την περίπτωση του.

Μέσω της σχεδίασης του συστήματος μας συμπεράναμε ότι η ομαδοποίηση των snippets πολλές φορές οδηγεί σε καλύτερα αποτελέσματα από την απλή παρουσίαση τους ως μία ενιαία λίστα. Αυτό το διαπιστώσαμε συγκρίνοντας το σύστημα μας, το οποίο κάνει ομαδοποίηση στα αποτελέσματα, με τη μηχανή αναζήτησης της Google. Πολλές φορές μέσω της ομαδοποίησης ο χρήστης εύρισκε ταχύτερα τα αποτελέσματα που έφαχνε, ενώ παράλληλα μέσω του διαχωρισμού μπορούσε να βρει επιπλέον παραπλήσια παραδείγματα που αφορούσαν εναλλακτικές υλοποιήσεις για το ίδιο πρόβλημα. Τέλος, μέσω του ελέγχου της αναγνωσιμότητας και της συχνότητας χρήσης των API calls ο χρήστης ήταν εγγυημένος με καλής ποιότητας τμήματα κώδικα.

## 6.2 Μελλοντική Εργασία

Τέλος, χρήσιμη κρίνεται η αναφορά σε πιθανές επεκτάσεις που θα μπορούσαν να γίνουν στο σύστημα μας μελλοντικά.

Μία προσθήκη στο κομμάτι του *Downloader* θα μπορούσε να είναι νέες μηχανές αναζήτησης πέραν αυτής της Google. Επίσης, θα μπορούσε να προστεθεί η δυνατότητα δημιουργίας τοπικής βάσης δεδομένων η οποία ανάλογα με το χρήστη θα διατηρεί κάποια συχνά χρησιμοποιούμενα αποτελέσματα έτσι ώστε να μην χρειάζεται εκ νέου το κατέβασμα τους.

Στο κομμάτι του *API Miner* ίσως είναι χρήσιμη η ανίχνευση συχνών ακολουθιών στον κώδικα του χρήστη και σε έργα ανοιχτού λογισμικού. Έτσι, σε περίπτωση που ο χρήστης γράψει μία ακολουθία που δεν συναντάται συχνά να ειδοποιείται για πιθανό bug στον κώδικα. Επίσης, χρήσιμη θα ήταν η ανανέωση του λεξιλογίου των APIs από νέα έργα που προστίθενται.

Επίσης, θα ήταν εξαιρετικά χρήσιμη η συλλογή δεδομένων κατά τη χρήση του

συστήματος μέσω των οποίων θα μπορούσε να εκπαιδευτεί κάποιος ranker ώστε να υπάρχει καλύτερη βαθμολόγηση των αποτελεσμάτων αναλόγως με τις προτιμήσεις του κάθε χρήστη. Με ανάλογο τρόπο θα μπορούσε να βελτιωθεί και ο *Readability Evaluator* ταξινομώντας τα snippets ανάλογα με τον εκάστοτε χρήστη.

Επιπρόσθετα, η ανάπτυξη γραφικής διεπαφής και ενσωμάτωσης του σε ένα σύγχρονο προγραμματιστικό περιβάλλον (π.χ. Eclipse) θα άνοιγε νέους ορίζοντες για το σύστημα. Έτσι, το σύστημα θα μπορούσε να αξιοποιεί το γενικότερο πλαίσιο του κώδικα του χρήστη συνθέτοντας πιο εύστοχα ερωτήματα ή ακόμη συνθέτοντας ερωτήματα αυτόματα (π.χ. αξιοποιώντας τα σχόλια στον κώδικα).

Τέλος, η αναζήτηση παραδειγμάτων θα μπορούσε να επεκταθεί και σε άλλες γλώσσες προγραμματισμού (π.χ. Python, Go, C++ κλπ.)

# Βιβλιογραφία

- [1] Buse Raymond P., L. and Westley Weimer. Learning a metric for code readability. *TSE SPECIAL ISSUE ON THE ISSTA 2008 BEST PAPERS*, 2008.
- [2] Overfitting. <https://en.wikipedia.org/wiki/Overfitting>, August 2017.
- [3] Cross-validation. [https://en.wikipedia.org/wiki/Cross-validation\\_\(statistics\)](https://en.wikipedia.org/wiki/Cross-validation_(statistics)), August 2017.
- [4] Lombard Hill Group. What is software reuse. [http://lombardhill.com/what\\_reuse.htm](http://lombardhill.com/what_reuse.htm).
- [5] S. McConnel. Rapid development: Taming wild software schedules. *Microsoft Press*, 1996.
- [6] DocForge. "code reuse". [https://web.archive.org/web/20110710143019/http://docforge.com/wiki/Code\\_reuse](https://web.archive.org/web/20110710143019/http://docforge.com/wiki/Code_reuse), July 2011.
- [7] Tyler Bletsch. *Code-reuse attacks: new frontiers and defenses*. PhD thesis, North Carolina State University, 2011.
- [8] Tyler Bletsch, Xuxian Jiang, Freeh Vince W., and Zhenkai Liang. Jump-oriented programming: a new class of code-reuse attack. *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 30–40, 2011.
- [9] Jonathan Edwards. Example centric programming. *OOPSLA*, 2004.
- [10] Wei Yi, Chandrasekaran Nirupama, Gulwani Sumit, and Hamandi Youssef. Building bing developer assistant, 2015.

- [11] Brandt Joel, Dontcheva Mira, Weskamp Marcos, and Klemmer Scott, R. Example-centric programming: Integrating web search into the development environment. *CHI, Atlanta, Georgia, USA*, 2010.
- [12] Wightman D., Ye Z., and Vertegaal R. Snipmatch: Using source code context to enhance snippet retrieval and parameterization, Cambridge, MA, USA, 2012.
- [13] Lingxiao J., Ghassan M., Zhendong S., and Glondu S. Deckard: Scalable and accurate tree-based detection of code clones, California 2007.
- [14] Xie T. and Pei J. Mapo: Mining api usages from open source repositories, Shanghai 2006.
- [15] Thummalapenta S. and Xie T. Parseweb: A programmer assistant for reusing open source code on the web, Atlanta, Georgia, USA, 2007.
- [16] Russ Cox. Regular expression matching with a trigram index (or: How google code search worked). <https://swtch.com/rsc/regexp/regexp4.html>, January 2012.
- [17] Neamtiu Iulian, Jeffrey S. Foster, and Hicks Michael. Understanding source code evolution using abstract syntax tree matching. *CiteSeerX*, 2005.
- [18] Bang Jensen and Jørgen. *Digraphs: Theory, Algorithms and Applications*, chapter 5.7. Springer-Verlag, 2008.
- [19] S. Shoham A., Mishne and Yahav E. Typestate-based semantic code search over partial programs. *OOPSLA, Tucson, Arizona, USA*, 2012.
- [20] Sahavechaphan N. and Claypool K. Xsnippet: Mining for sample code. *OOPSLA, Portland, Oregon, USA*, 2006.
- [21] Leiserson, Schardl Charles E., and Tao B. A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers). *ACM Symp. on Parallelism in Algorithms and Architectures*, 1961.
- [22] Holmes R., J., Walker R., J., and Murphy G., C. Strathcona example recommendation tool, Lisbon, Portugal, 2005.

- [23] Bruch M., Schafer T., and Mezini M. Fruit: Ide support for framework understanding. *OOPSLA, Portland, Oregon, USA*, 2006.
- [24] Robillard M., P., Maalej W., and Walker R., J. *Recommendation Systems in Software Engineering*. Berlin, Springer.
- [25] Dumais Susan T. Latent semantic analysis. *Annual Review of Information Science and Technology*, 38(230):188–230, 2005.
- [26] Rajaraman A. and Ullman J., D. Mining of massive datasets. *Data Mining*, pages 1–17, 2011.
- [27] Raychev V., Vechev M., and Yahav E. Code completion with statistical language models, Edinburgh, United Kingdom, 2014.
- [28] Craig Tim. What is language modeling? <http://trimc-nlp.blogspot.gr/2013/04/language-modeling.html>, April 2013.
- [29] What are n-grams. <http://text-analytics101.rxnlp.com/2014/11/what-are-n-grams.html>, November 2014.
- [30] Denny Britz. Recurrent neural networks. <http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/>, September 2015.
- [31] Galenson J., Reames P., Bodik R., and Hartmann B. Codehint: Dynamic and interactive synthesis of code snippets. *ICSE, Hyderabad, India*, 2014.
- [32] Fast E., Stefee D., Wang L., Brandt J., and Bernstein M., S. Emergent, crowd-scale programming practice in the ide, Toronto, Ontario, Canada, 2014.
- [33] Brown P., F., D. Pietra V., J., D. Pietra S., A., and Mercer R., L. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics*, 19(2):263–311, 1993.
- [34] Wang J. and Jiawei H. Bide: Efficient mining of frequent closed sequences, 2004.
- [35] Hoffmann, Fogarty R., J., and Weld D., S. Assieme: Finding and leveraging implicit references in a web search. In *Proceeding of UIST: ACM Symposium on User Interface and Technology*, 2007.

- [36] Stylos, Myers J., A., and Myers B., A. Mica: A web-search tool for finding api components and examples. In *Proceedings of VL/HCC: IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 195–202, 2006.
- [37] Terence Parr. *The definitive ANTLR4 Reference*. Pragmatic Bookshelf, 2007.
- [38] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley. <http://docs.oracle.com/javase/specs/jls/se8/html/>, February 2015.
- [39] Michail Papamichail, Themistoklis Diamantoulos, and Andreas Symeonidis. User-perceived source code quality estimation based on static analysis metrics. In *IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2016.
- [40] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. <http://is.muni.cz/publication/884893/en>.
- [41] Boehm B. and Basili V., R. Software defect reduction top 10 list. *Computer*, 34:135–137, 2001.
- [42] Deimel Jr. L., E. The uses of program reading. *SIGCSE Bull*, 17:146–153, 1985.
- [43] Raymond D., R. Reading source code. In IBM Press, editor, *Conference of the Centre for Advanced Studies on Collaborative Research.*, pages 3–16, 1991.
- [44] Rugaber S. The use of domain knowledge in program understanding. *Ann. Softw. Eng*, 9:143–192, 2000.
- [45] Buse Raymond P., L. *Automatically Describing Program Structure and Behavior*. PhD thesis, University of Virginia, May 2012.
- [46] Wes McKinney. pandas: a foundational python library for data analysis and statistics.
- [47] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

- [48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [49] Ron Kohavi. A study of crossvalidation and bootstrap for accuracy estimation and model selection. In *Appears in the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.
- [50] Arlot S. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4(1935-7516):40–79, 2010.
- [51] Fukunaga K. and Narendra P., M. A branch and bound algorithm for computing k-nearest neighbors. In *IEEE transactions on computers*, 1975.
- [52] Breiman L. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- [53] Ji Zhu, Hui Zou, Saharon Rosset, and Trevor Hastie. Multi-class adaboost. *Statistics and Its Interface*, 2:349–360, 2009.
- [54] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: A statistical framework. *CiteSeerX*, 2006.
- [55] Cambridge University. *Introduction to Information Retrieval*. University Press, 2009.
- [56] Borg I. and Groenen P. Modern multidimensional scaling - theory and applications. *Springer*, 1997.
- [57] Sugar Catherine A. and James Gareth M. Finding the number of clusters in a data set, an information theoretic approach. Master’s thesis, Marshall School of Business, University of Southern California, 2011.
- [58] Pelleg D. and Moore A. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.
- [59] Rousseeuw Peter J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Computational and Applied Mathematics*, 20:53–65, 1987.

- [60] Lleti R., Ortiz M., C., Sarabia L., A., and Sánchez M., S. Selecting variables for k-means cluster analysis by using a genetic algorithm that optimises the silhouettes. *Analytica Chimica Acta*, 515:87–100, 2004.
- [61] de Amorim R., C. and Hennig C. Recovering the number of clusters in data sets with noise features using feature rescaling factors. *Information Sciences*, 324:126–145, 2015.
- [62] "How to decide on the correct number of clusters?". <https://stats.stackexchange.com/questions/23472/how-to-decide-on-the-correct-number-of-clusters>, February 2012.
- [63] Tibshinari R., Walther G., and Hastie T. Estimating the numbers of clusters in a data set via the gap statistic. *J. R. Statist. Soc. B*, 63:411–423, 2001.
- [64] Thorndike Robert L. Who belongs in the family? *Psychometrika*, 18:267–276, December 1953.
- [65] Brenda Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.
- [66] Khaled Alsabti, Sanjay Ranka, and Vineet Singh. *An Efficient K-Means Clustering Algorithm*. UT Computer Science, 1998.
- [67] Arthur, David, and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007.
- [68] Cooper William S. Expected search length: A single measure of retrieval effectiveness based on the weak ordering action of retrieval systems. *Journal of the Association for Information Science and Technology*, 19:30–41, January 1968.
- [69] *Cluster Analysis: Basic Concepts and Algorithms*. University of Minnesota Twin Cities, 2010.