

## APPENDICES

### Appendix A - Machine Learning Details (Attribute-BasedMLSemanticTypeDetection.ipynb)

This appendix documents the optional ML classifier pipeline, which complements the rule-based stage by producing per-column format predictions under header-only evidence. It details preprocessing, feature extraction, model training, and evaluation, and accompanies **Algorithm 1**, which specifies where the ML classifier is invoked within the semantic type detection workflow.

For reproducibility, all configuration parameters; including random seed, data-split policy, cross-validation, hyperparameter grids, and persistence of trained models; are listed at the end of this appendix.

**Scope note:** In this study, the ML classifier was run only on the first **100 data sources (50 UCI + 50 Prague) with around 1,400 columns**; all other experiments use the **rule-based** pipeline alone.

#### Preprocessing and Feature Engineering.

Column headers and descriptions are cleaned through normalisation, abbreviation replacement, and camel-case splitting (see **Section 2.3**). The resulting textual data is vectorised using a TF-IDF model [33] built from a combined vocabulary of *DBHeaders* [9] and the *Formats* dictionary. In parallel, **dictionary-derived SourceKeywords are extracted to record which terms triggered candidate formats, supporting explainability, consistency checks, and enrichment when NIL cases are logged**.

#### Addressing Class Imbalance.

Semantic type classification often suffers from skewed distributions, with some types appearing far more frequently than others. To mitigate this imbalance, we apply the Synthetic Minority Over-sampling Technique (SMOTE), generating synthetic minority class samples to improve generalisation while reducing the risk of overfitting.

#### Model Selection and Hyperparameter Optimisation.

Multiple classifiers are evaluated, including Random Forest, Logistic Regression, Gradient Boosting, K-Nearest Neighbours, and LinearSVC. Hyperparameters are tuned via GridSearchCV with stratified cross-validation to ensure fair optimisation across imbalanced classes.

#### Evaluation and Interpretability.

Models are assessed using precision, recall, and F1-score on held-out test data. Feature importance analysis is employed to interpret classification behaviour and identify influential terms.

#### Model Persistence.

All trained classifiers, together with the fitted TF-IDF vectorizer, are saved to support reproducibility and allow flexible deployment scenarios.

#### Algorithm 1 – Attribute-Based ML Semantic Type Detection (applied systematically to all non-ID columns; dictionaries provide features and reconciliation)

##### Inputs:

dataInfo (dataset/table metadata and columns); formatDict (Formats Dictionary); abbrDict (Abbreviations Dictionary); dbHeadersInfo (DB headers corpus); classifiers ({RandomForest, LogisticRegression, GradientBoosting, KNN, LinearSVC}); tfidfVectorizer (trained on Formats+DBHeaders vocabulary).

##### Output:

analysedColumns\_models (per-column predictions per model + aggregate, confidence, NIL/flags, review logs).

```

1  function SemanticTypeDetect(dataInfo, formatDict,
2      abbrDict, dbHeadersInfo,
3      classifiers, tfidfVectorizer, confidenceThreshold)
4      columns ← ExtractAndCleanColumns(dataInfo) // trim,
5      normalise case/punctuation, etc.
6
7      for each column in columns do
8          // Deterministic ID assignment (DB context)
9          if column.belongsToDatabase then
10             pkfk ← GetPrimaryForeignKeyInfo(column, dataInfo)
11             if pkfk.isPK or pkfk.isFK then
12                 column.semanticType ← "ID column"
13                 column.mlConfidence ← 1.0
14                 SavePerModelPredictions(column, {}) // optional
15                 empty per-model record
16                 continue
17             end if
18         end if
19
20         // Dictionary-derived FEATURES (no hard gate)
21         expanded ← ReplaceAbbreviations(column.cleanedName, abbrDict)
22         expanded ← SplitCamelCase(expanded)
23         sourceKeywords ← ExtractKeywords(expanded,
24             formatDict) // e.g., ColumnKeyword, DescriptionKeyword
25
26         // Compose text input and vectorise
27         if column.description ≠ Ø then
28             inputText ← Concatenate(expanded, " ",
29             column.description)
30         else
31             inputText ← expanded
32         end if
33         X ← tfidfVectorizer.transform([inputText])
34
35         // Predict with all classifiers; collect per-model outputs
36         preds ← {} // model → (label,
37         confidence)
38         for each clf in classifiers do
39             y_hat ← clf.predict(X)
40
41     end for
42
43     analysedColumns_models ← analysedColumns_models ∪
44     {column.semanticType, column.mlConfidence, column.NILFlags,
45     column.reviewLog}
46
47     if confidenceThreshold > 0.0 then
48         if column.mlConfidence < confidenceThreshold then
49             column.semanticType ← "NIL"
50             column.mlConfidence ← 0.0
51             column.NILFlags ← true
52             column.reviewLog ← "NIL detected"
53         end if
54     end if
55
56     SavePerModelPredictions(column, analysedColumns_models)
57
58     analysedColumns_models ← analysedColumns_models ∪
59     {column.semanticType, column.mlConfidence, column.NILFlags,
60     column.reviewLog}
61
62     if confidenceThreshold > 0.0 then
63         if column.mlConfidence < confidenceThreshold then
64             column.semanticType ← "NIL"
65             column.mlConfidence ← 0.0
66             column.NILFlags ← true
67             column.reviewLog ← "NIL detected"
68         end if
69     end if
70
71     SavePerModelPredictions(column, analysedColumns_models)
72
73     analysedColumns_models ← analysedColumns_models ∪
74     {column.semanticType, column.mlConfidence, column.NILFlags,
75     column.reviewLog}
76
77     if confidenceThreshold > 0.0 then
78         if column.mlConfidence < confidenceThreshold then
79             column.semanticType ← "NIL"
80             column.mlConfidence ← 0.0
81             column.NILFlags ← true
82             column.reviewLog ← "NIL detected"
83         end if
84     end if
85
86     SavePerModelPredictions(column, analysedColumns_models)
87
88     analysedColumns_models ← analysedColumns_models ∪
89     {column.semanticType, column.mlConfidence, column.NILFlags,
90     column.reviewLog}
91
92     if confidenceThreshold > 0.0 then
93         if column.mlConfidence < confidenceThreshold then
94             column.semanticType ← "NIL"
95             column.mlConfidence ← 0.0
96             column.NILFlags ← true
97             column.reviewLog ← "NIL detected"
98         end if
99     end if
100
101     SavePerModelPredictions(column, analysedColumns_models)
102
103     analysedColumns_models ← analysedColumns_models ∪
104     {column.semanticType, column.mlConfidence, column.NILFlags,
105     column.reviewLog}
106
107     if confidenceThreshold > 0.0 then
108         if column.mlConfidence < confidenceThreshold then
109             column.semanticType ← "NIL"
110             column.mlConfidence ← 0.0
111             column.NILFlags ← true
112             column.reviewLog ← "NIL detected"
113         end if
114     end if
115
116     SavePerModelPredictions(column, analysedColumns_models)
117
118     analysedColumns_models ← analysedColumns_models ∪
119     {column.semanticType, column.mlConfidence, column.NILFlags,
120     column.reviewLog}
121
122     if confidenceThreshold > 0.0 then
123         if column.mlConfidence < confidenceThreshold then
124             column.semanticType ← "NIL"
125             column.mlConfidence ← 0.0
126             column.NILFlags ← true
127             column.reviewLog ← "NIL detected"
128         end if
129     end if
130
131     SavePerModelPredictions(column, analysedColumns_models)
132
133     analysedColumns_models ← analysedColumns_models ∪
134     {column.semanticType, column.mlConfidence, column.NILFlags,
135     column.reviewLog}
136
137     if confidenceThreshold > 0.0 then
138         if column.mlConfidence < confidenceThreshold then
139             column.semanticType ← "NIL"
140             column.mlConfidence ← 0.0
141             column.NILFlags ← true
142             column.reviewLog ← "NIL detected"
143         end if
144     end if
145
146     SavePerModelPredictions(column, analysedColumns_models)
147
148     analysedColumns_models ← analysedColumns_models ∪
149     {column.semanticType, column.mlConfidence, column.NILFlags,
150     column.reviewLog}
151
152     if confidenceThreshold > 0.0 then
153         if column.mlConfidence < confidenceThreshold then
154             column.semanticType ← "NIL"
155             column.mlConfidence ← 0.0
156             column.NILFlags ← true
157             column.reviewLog ← "NIL detected"
158         end if
159     end if
160
161     SavePerModelPredictions(column, analysedColumns_models)
162
163     analysedColumns_models ← analysedColumns_models ∪
164     {column.semanticType, column.mlConfidence, column.NILFlags,
165     column.reviewLog}
166
167     if confidenceThreshold > 0.0 then
168         if column.mlConfidence < confidenceThreshold then
169             column.semanticType ← "NIL"
170             column.mlConfidence ← 0.0
171             column.NILFlags ← true
172             column.reviewLog ← "NIL detected"
173         end if
174     end if
175
176     SavePerModelPredictions(column, analysedColumns_models)
177
178     analysedColumns_models ← analysedColumns_models ∪
179     {column.semanticType, column.mlConfidence, column.NILFlags,
180     column.reviewLog}
181
182     if confidenceThreshold > 0.0 then
183         if column.mlConfidence < confidenceThreshold then
184             column.semanticType ← "NIL"
185             column.mlConfidence ← 0.0
186             column.NILFlags ← true
187             column.reviewLog ← "NIL detected"
188         end if
189     end if
190
191     SavePerModelPredictions(column, analysedColumns_models)
192
193     analysedColumns_models ← analysedColumns_models ∪
194     {column.semanticType, column.mlConfidence, column.NILFlags,
195     column.reviewLog}
196
197     if confidenceThreshold > 0.0 then
198         if column.mlConfidence < confidenceThreshold then
199             column.semanticType ← "NIL"
200             column.mlConfidence ← 0.0
201             column.NILFlags ← true
202             column.reviewLog ← "NIL detected"
203         end if
204     end if
205
206     SavePerModelPredictions(column, analysedColumns_models)
207
208     analysedColumns_models ← analysedColumns_models ∪
209     {column.semanticType, column.mlConfidence, column.NILFlags,
210     column.reviewLog}
211
212     if confidenceThreshold > 0.0 then
213         if column.mlConfidence < confidenceThreshold then
214             column.semanticType ← "NIL"
215             column.mlConfidence ← 0.0
216             column.NILFlags ← true
217             column.reviewLog ← "NIL detected"
218         end if
219     end if
220
221     SavePerModelPredictions(column, analysedColumns_models)
222
223     analysedColumns_models ← analysedColumns_models ∪
224     {column.semanticType, column.mlConfidence, column.NILFlags,
225     column.reviewLog}
226
227     if confidenceThreshold > 0.0 then
228         if column.mlConfidence < confidenceThreshold then
229             column.semanticType ← "NIL"
230             column.mlConfidence ← 0.0
231             column.NILFlags ← true
232             column.reviewLog ← "NIL detected"
233         end if
234     end if
235
236     SavePerModelPredictions(column, analysedColumns_models)
237
238     analysedColumns_models ← analysedColumns_models ∪
239     {column.semanticType, column.mlConfidence, column.NILFlags,
240     column.reviewLog}
241
242     if confidenceThreshold > 0.0 then
243         if column.mlConfidence < confidenceThreshold then
244             column.semanticType ← "NIL"
245             column.mlConfidence ← 0.0
246             column.NILFlags ← true
247             column.reviewLog ← "NIL detected"
248         end if
249     end if
250
251     SavePerModelPredictions(column, analysedColumns_models)
252
253     analysedColumns_models ← analysedColumns_models ∪
254     {column.semanticType, column.mlConfidence, column.NILFlags,
255     column.reviewLog}
256
257     if confidenceThreshold > 0.0 then
258         if column.mlConfidence < confidenceThreshold then
259             column.semanticType ← "NIL"
260             column.mlConfidence ← 0.0
261             column.NILFlags ← true
262             column.reviewLog ← "NIL detected"
263         end if
264     end if
265
266     SavePerModelPredictions(column, analysedColumns_models)
267
268     analysedColumns_models ← analysedColumns_models ∪
269     {column.semanticType, column.mlConfidence, column.NILFlags,
270     column.reviewLog}
271
272     if confidenceThreshold > 0.0 then
273         if column.mlConfidence < confidenceThreshold then
274             column.semanticType ← "NIL"
275             column.mlConfidence ← 0.0
276             column.NILFlags ← true
277             column.reviewLog ← "NIL detected"
278         end if
279     end if
280
281     SavePerModelPredictions(column, analysedColumns_models)
282
283     analysedColumns_models ← analysedColumns_models ∪
284     {column.semanticType, column.mlConfidence, column.NILFlags,
285     column.reviewLog}
286
287     if confidenceThreshold > 0.0 then
288         if column.mlConfidence < confidenceThreshold then
289             column.semanticType ← "NIL"
290             column.mlConfidence ← 0.0
291             column.NILFlags ← true
292             column.reviewLog ← "NIL detected"
293         end if
294     end if
295
296     SavePerModelPredictions(column, analysedColumns_models)
297
298     analysedColumns_models ← analysedColumns_models ∪
299     {column.semanticType, column.mlConfidence, column.NILFlags,
300     column.reviewLog}
301
302     if confidenceThreshold > 0.0 then
303         if column.mlConfidence < confidenceThreshold then
304             column.semanticType ← "NIL"
305             column.mlConfidence ← 0.0
306             column.NILFlags ← true
307             column.reviewLog ← "NIL detected"
308         end if
309     end if
310
311     SavePerModelPredictions(column, analysedColumns_models)
312
313     analysedColumns_models ← analysedColumns_models ∪
314     {column.semanticType, column.mlConfidence, column.NILFlags,
315     column.reviewLog}
316
317     if confidenceThreshold > 0.0 then
318         if column.mlConfidence < confidenceThreshold then
319             column.semanticType ← "NIL"
320             column.mlConfidence ← 0.0
321             column.NILFlags ← true
322             column.reviewLog ← "NIL detected"
323         end if
324     end if
325
326     SavePerModelPredictions(column, analysedColumns_models)
327
328     analysedColumns_models ← analysedColumns_models ∪
329     {column.semanticType, column.mlConfidence, column.NILFlags,
330     column.reviewLog}
331
332     if confidenceThreshold > 0.0 then
333         if column.mlConfidence < confidenceThreshold then
334             column.semanticType ← "NIL"
335             column.mlConfidence ← 0.0
336             column.NILFlags ← true
337             column.reviewLog ← "NIL detected"
338         end if
339     end if
340
341     SavePerModelPredictions(column, analysedColumns_models)
342
343     analysedColumns_models ← analysedColumns_models ∪
344     {column.semanticType, column.mlConfidence, column.NILFlags,
345     column.reviewLog}
346
347     if confidenceThreshold > 0.0 then
348         if column.mlConfidence < confidenceThreshold then
349             column.semanticType ← "NIL"
350             column.mlConfidence ← 0.0
351             column.NILFlags ← true
352             column.reviewLog ← "NIL detected"
353         end if
354     end if
355
356     SavePerModelPredictions(column, analysedColumns_models)
357
358     analysedColumns_models ← analysedColumns_models ∪
359     {column.semanticType, column.mlConfidence, column.NILFlags,
360     column.reviewLog}
361
362     if confidenceThreshold > 0.0 then
363         if column.mlConfidence < confidenceThreshold then
364             column.semanticType ← "NIL"
365             column.mlConfidence ← 0.0
366             column.NILFlags ← true
367             column.reviewLog ← "NIL detected"
368         end if
369     end if
370
371     SavePerModelPredictions(column, analysedColumns_models)
372
373     analysedColumns_models ← analysedColumns_models ∪
374     {column.semanticType, column.mlConfidence, column.NILFlags,
375     column.reviewLog}
376
377     if confidenceThreshold > 0.0 then
378         if column.mlConfidence < confidenceThreshold then
379             column.semanticType ← "NIL"
380             column.mlConfidence ← 0.0
381             column.NILFlags ← true
382             column.reviewLog ← "NIL detected"
383         end if
384     end if
385
386     SavePerModelPredictions(column, analysedColumns_models)
387
388     analysedColumns_models ← analysedColumns_models ∪
389     {column.semanticType, column.mlConfidence, column.NILFlags,
390     column.reviewLog}
391
392     if confidenceThreshold > 0.0 then
393         if column.mlConfidence < confidenceThreshold then
394             column.semanticType ← "NIL"
395             column.mlConfidence ← 0.0
396             column.NILFlags ← true
397             column.reviewLog ← "NIL detected"
398         end if
399     end if
400
401     SavePerModelPredictions(column, analysedColumns_models)
402
403     analysedColumns_models ← analysedColumns_models ∪
404     {column.semanticType, column.mlConfidence, column.NILFlags,
405     column.reviewLog}
406
407     if confidenceThreshold > 0.0 then
408         if column.mlConfidence < confidenceThreshold then
409             column.semanticType ← "NIL"
410             column.mlConfidence ← 0.0
411             column.NILFlags ← true
412             column.reviewLog ← "NIL detected"
413         end if
414     end if
415
416     SavePerModelPredictions(column, analysedColumns_models)
417
418     analysedColumns_models ← analysedColumns_models ∪
419     {column.semanticType, column.mlConfidence, column.NILFlags,
420     column.reviewLog}
421
422     if confidenceThreshold > 0.0 then
423         if column.mlConfidence < confidenceThreshold then
424             column.semanticType ← "NIL"
425             column.mlConfidence ← 0.0
426             column.NILFlags ← true
427             column.reviewLog ← "NIL detected"
428         end if
429     end if
430
431     SavePerModelPredictions(column, analysedColumns_models)
432
433     analysedColumns_models ← analysedColumns_models ∪
434     {column.semanticType, column.mlConfidence, column.NILFlags,
435     column.reviewLog}
436
437     if confidenceThreshold > 0.0 then
438         if column.mlConfidence < confidenceThreshold then
439             column.semanticType ← "NIL"
440             column.mlConfidence ← 0.0
441             column.NILFlags ← true
442             column.reviewLog ← "NIL detected"
443         end if
444     end if
445
446     SavePerModelPredictions(column, analysedColumns_models)
447
448     analysedColumns_models ← analysedColumns_models ∪
449     {column.semanticType, column.mlConfidence, column.NILFlags,
450     column.reviewLog}
451
452     if confidenceThreshold > 0.0 then
453         if column.mlConfidence < confidenceThreshold then
454             column.semanticType ← "NIL"
455             column.mlConfidence ← 0.0
456             column.NILFlags ← true
457             column.reviewLog ← "NIL detected"
458         end if
459     end if
460
461     SavePerModelPredictions(column, analysedColumns_models)
462
463     analysedColumns_models ← analysedColumns_models ∪
464     {column.semanticType, column.mlConfidence, column.NILFlags,
465     column.reviewLog}
466
467     if confidenceThreshold > 0.0 then
468         if column.mlConfidence < confidenceThreshold then
469             column.semanticType ← "NIL"
470             column.mlConfidence ← 0.0
471             column.NILFlags ← true
472             column.reviewLog ← "NIL detected"
473         end if
474     end if
475
476     SavePerModelPredictions(column, analysedColumns_models)
477
478     analysedColumns_models ← analysedColumns_models ∪
479     {column.semanticType, column.mlConfidence, column.NILFlags,
480     column.reviewLog}
481
482     if confidenceThreshold > 0.0 then
483         if column.mlConfidence < confidenceThreshold then
484             column.semanticType ← "NIL"
485             column.mlConfidence ← 0.0
486             column.NILFlags ← true
487             column.reviewLog ← "NIL detected"
488         end if
489     end if
490
491     SavePerModelPredictions(column, analysedColumns_models)
492
493     analysedColumns_models ← analysedColumns_models ∪
494     {column.semanticType, column.mlConfidence, column.NILFlags,
495     column.reviewLog}
496
497     if confidenceThreshold > 0.0 then
498         if column.mlConfidence < confidenceThreshold then
499             column.semanticType ← "NIL"
500             column.mlConfidence ← 0.0
501             column.NILFlags ← true
502             column.reviewLog ← "NIL detected"
503         end if
504     end if
505
506     SavePerModelPredictions(column, analysedColumns_models)
507
508     analysedColumns_models ← analysedColumns_models ∪
509     {column.semanticType, column.mlConfidence, column.NILFlags,
510     column.reviewLog}
511
512     if confidenceThreshold > 0.0 then
513         if column.mlConfidence < confidenceThreshold then
514             column.semanticType ← "NIL"
515             column.mlConfidence ← 0.0
516             column.NILFlags ← true
517             column.reviewLog ← "NIL detected"
518         end if
519     end if
520
521     SavePerModelPredictions(column, analysedColumns_models)
522
523     analysedColumns_models ← analysedColumns_models ∪
524     {column.semanticType, column.mlConfidence, column.NILFlags,
525     column.reviewLog}
526
527     if confidenceThreshold > 0.0 then
528         if column.mlConfidence < confidenceThreshold then
529             column.semanticType ← "NIL"
530             column.mlConfidence ← 0.0
531             column.NILFlags ← true
532             column.reviewLog ← "NIL detected"
533         end if
534     end if
535
536     SavePerModelPredictions(column, analysedColumns_models)
537
538     analysedColumns_models ← analysedColumns_models ∪
539     {column.semanticType, column.mlConfidence, column.NILFlags,
540     column.reviewLog}
541
542     if confidenceThreshold > 0.0 then
543         if column.mlConfidence < confidenceThreshold then
544             column.semanticType ← "NIL"
545             column.mlConfidence ← 0.0
546             column.NILFlags ← true
547             column.reviewLog ← "NIL detected"
548         end if
549     end if
550
551     SavePerModelPredictions(column, analysedColumns_models)
552
553     analysedColumns_models ← analysedColumns_models ∪
554     {column.semanticType, column.mlConfidence, column.NILFlags,
555     column.reviewLog}
556
557     if confidenceThreshold > 0.0 then
558         if column.mlConfidence < confidenceThreshold then
559             column.semanticType ← "NIL"
560             column.mlConfidence ← 0.0
561             column.NILFlags ← true
562             column.reviewLog ← "NIL detected"
563         end if
564     end if
565
566     SavePerModelPredictions(column, analysedColumns_models)
567
568     analysedColumns_models ← analysedColumns_models ∪
569     {column.semanticType, column.mlConfidence, column.NILFlags,
570     column.reviewLog}
571
572     if confidenceThreshold > 0.0 then
573         if column.mlConfidence < confidenceThreshold then
574             column.semanticType ← "NIL"
575             column.mlConfidence ← 0.0
576             column.NILFlags ← true
577             column.reviewLog ← "NIL detected"
578         end if
579     end if
580
581     SavePerModelPredictions(column, analysedColumns_models)
582
583     analysedColumns_models ← analysedColumns_models ∪
584     {column.semanticType, column.mlConfidence, column.NILFlags,
585     column.reviewLog}
586
587     if confidenceThreshold > 0.0 then
588         if column.mlConfidence < confidenceThreshold then
589             column.semanticType ← "NIL"
590             column.mlConfidence ← 0.0
591             column.NILFlags ← true
592             column.reviewLog ← "NIL detected"
593         end if
594     end if
595
596     SavePerModelPredictions(column, analysedColumns_models)
597
598     analysedColumns_models ← analysedColumns_models ∪
599     {column.semanticType, column.mlConfidence, column.NILFlags,
600     column.reviewLog}
601
602     if confidenceThreshold > 0.0 then
603         if column.mlConfidence < confidenceThreshold then
604             column.semanticType ← "NIL"
605             column.mlConfidence ← 0.0
606             column.NILFlags ← true
607             column.reviewLog ← "NIL detected"
608         end if
609     end if
610
611     SavePerModelPredictions(column, analysedColumns_models)
612
613     analysedColumns_models ← analysedColumns_models ∪
614     {column.semanticType, column.mlConfidence, column.NILFlags,
615     column.reviewLog}
616
617     if confidenceThreshold > 0.0 then
618         if column.mlConfidence < confidenceThreshold then
619             column.semanticType ← "NIL"
620             column.mlConfidence ← 0.0
621             column.NILFlags ← true
622             column.reviewLog ← "NIL detected"
623         end if
624     end if
625
626     SavePerModelPredictions(column, analysedColumns_models)
627
628     analysedColumns_models ← analysedColumns_models ∪
629     {column.semanticType, column.mlConfidence, column.NILFlags,
630     column.reviewLog}
631
632     if confidenceThreshold > 0.0 then
633         if column.mlConfidence < confidenceThreshold then
634             column.semanticType ← "NIL"
635             column.mlConfidence ← 0.0
636             column.NILFlags ← true
637             column.reviewLog ← "NIL detected"
638         end if
639     end if
640
641     SavePerModelPredictions(column, analysedColumns_models)
642
643     analysedColumns_models ← analysedColumns_models ∪
644     {column.semanticType, column.mlConfidence, column.NILFlags,
645     column.reviewLog}
646
647     if confidenceThreshold > 0.0 then
648         if column.mlConfidence < confidenceThreshold then
649             column.semanticType ← "NIL"
650             column.mlConfidence ← 0.0
651             column.NILFlags ← true
652             column.reviewLog ← "NIL detected"
653         end if
654     end if
655
656     SavePerModelPredictions(column, analysedColumns_models)
657
658     analysedColumns_models ← analysedColumns_models ∪
659     {column.semanticType, column.mlConfidence, column.NILFlags,
660     column.reviewLog}
661
662     if confidenceThreshold > 0.0 then
663         if column.mlConfidence < confidenceThreshold then
664             column.semanticType ← "NIL"
665             column.mlConfidence ← 0.0
666             column.NILFlags ← true
667             column.reviewLog ← "NIL detected"
668         end if
669     end if
670
671     SavePerModelPredictions(column, analysedColumns_models)
672
673     analysedColumns_models ← analysedColumns_models ∪
674     {column.semanticType, column.mlConfidence, column.NILFlags,
675     column.reviewLog}
676
677     if confidenceThreshold > 0.0 then
678         if column.mlConfidence < confidenceThreshold then
679             column.semanticType ← "NIL"
680             column.mlConfidence ← 0.0
681             column.NILFlags ← true
682             column.reviewLog ← "NIL detected"
683         end if
684     end if
685
686     SavePerModelPredictions(column, analysedColumns_models)
687
688     analysedColumns_models ← analysedColumns_models ∪
689     {column.semanticType, column.mlConfidence, column.NILFlags,
690     column.reviewLog}
691
692     if confidenceThreshold > 0.0 then
693         if column.mlConfidence < confidenceThreshold then
694             column.semanticType ← "NIL"
695             column.mlConfidence ← 0.0
696             column.NILFlags ← true
697             column.reviewLog ← "NIL detected"
698         end if
699     end if
700
701     SavePerModelPredictions(column, analysedColumns_models)
702
703     analysedColumns_models ← analysedColumns_models ∪
704     {column.semanticType, column.mlConfidence, column.NILFlags,
705     column.reviewLog}
706
707     if confidenceThreshold > 0.0 then
708         if column.mlConfidence < confidenceThreshold then
709             column.semanticType ← "NIL"
710             column.mlConfidence ← 0.0
711             column.NILFlags ← true
712             column.reviewLog ← "NIL detected"
713         end if
714     end if
715
716     SavePerModelPredictions(column, analysedColumns_models)
717
718     analysedColumns_models ← analysedColumns_models ∪
719     {column.semanticType, column.mlConfidence, column.NILFlags,
720     column.reviewLog}
721
722     if confidenceThreshold > 0.0 then
723         if column.mlConfidence < confidenceThreshold then
724             column.semanticType ← "NIL"
725             column.mlConfidence ← 0.0
726             column.NILFlags ← true
727             column.reviewLog ← "NIL detected"
728         end if
729     end if
730
731     SavePerModelPredictions(column, analysedColumns_models)
732
733     analysedColumns_models ← analysedColumns_models ∪
734     {column.semanticType, column.mlConfidence, column.NILFlags,
735     column.reviewLog}
736
737     if confidenceThreshold > 0.0 then
738         if column.mlConfidence < confidenceThreshold then
739             column.semanticType ← "NIL"
740             column.mlConfidence ← 0.0
741             column.NILFlags ← true
742             column.reviewLog ← "NIL detected"
743         end if
744     end if
745
746     SavePerModelPredictions(column, analysedColumns_models)
747
748     analysedColumns_models ← analysedColumns_models ∪
749     {column.semanticType, column.mlConfidence, column.NILFlags,
750     column.reviewLog}
751
752     if confidenceThreshold > 0.0 then
753         if column.mlConfidence < confidenceThreshold then
754             column.semanticType ← "NIL"
755             column.mlConfidence ← 0.0
756             column.NILFlags ← true
757             column.reviewLog ← "NIL detected"
758         end if
759     end if
760
761     SavePerModelPredictions(column, analysedColumns_models)
762
763     analysedColumns_models ← analysedColumns_models ∪
764     {column.semanticType, column.mlConfidence, column.NILFlags,
765     column.reviewLog}
766
767     if confidenceThreshold > 0.0 then
768         if column.mlConfidence < confidenceThreshold then
769             column.semanticType ← "NIL"
770             column.mlConfidence ← 0.0
771             column.NILFlags ← true
772             column.reviewLog ← "NIL detected"
773         end if
774     end if
775
776     SavePerModelPredictions(column, analysedColumns_models)
777
778     analysedColumns_models ← analysedColumns_models ∪
779     {column.semanticType, column.mlConfidence, column.NILFlags,
780     column.reviewLog}
781
782     if confidenceThreshold > 0.0 then
783         if column.mlConfidence < confidenceThreshold then
784             column.semanticType ← "NIL"
785             column.mlConfidence ← 0.0
786             column.NILFlags ← true
787             column.reviewLog ← "NIL detected"
788         end if
789     end if
790
791     SavePerModelPredictions(column, analysedColumns_models)
792
793     analysedColumns_models ← analysedColumns_models ∪
794     {column.semanticType, column.mlConfidence, column.NILFlags,
795     column.reviewLog}
796
797     if confidenceThreshold > 0.0 then
798         if column.mlConfidence < confidenceThreshold then
799             column.semanticType ← "NIL"
800             column.mlConfidence ← 0.0
801             column.NILFlags ← true
802             column.reviewLog ← "NIL detected"
803         end if
804     end if
805
806     SavePerModelPredictions(column, analysedColumns_models)
807
808     analysedColumns_models ← analysedColumns_models ∪
809     {column.semanticType, column.mlConfidence, column.NILFlags,
810     column.reviewLog}
811
812     if confidenceThreshold > 0.0 then
813         if column.mlConfidence < confidenceThreshold then
814             column.semanticType ← "NIL"
815             column.mlConfidence ← 0.0
816             column.NILFlags ← true
817             column.reviewLog ← "NIL detected"
818         end if
819     end if
820
821     SavePerModelPredictions(column, analysedColumns_models)
822
823     analysedColumns_models ← analysedColumns_models ∪
824     {column.semanticType, column.mlConfidence, column.NILFlags,
825     column.reviewLog}
826
827     if confidenceThreshold > 0.0 then
828         if column.mlConfidence < confidenceThreshold then
829             column.semanticType ← "NIL"
830             column.mlConfidence ← 0.0
831             column.NILFlags ← true
832             column.reviewLog ← "NIL detected"
833         end if
834     end if
835
836     SavePerModelPredictions(column, analysedColumns_models)
837
838     analysedColumns_models ← analysedColumns_models ∪
839     {column.semanticType, column.mlConfidence, column.NILFlags,
840     column.reviewLog}
841
842     if confidenceThreshold > 0.0 then
843         if column.mlConfidence < confidenceThreshold then
844             column.semanticType ← "NIL"
845             column.mlConfidence ← 0.0
846             column.NILFlags ← true
847             column.reviewLog ← "NIL detected"
848         end if
849     end if
850
851     SavePerModelPredictions(column, analysedColumns_models)
852
853     analysedColumns_models ← analysedColumns_models ∪
854     {column.semanticType, column.mlConfidence, column.NILFlags,
855     column.reviewLog}
856
857     if confidenceThreshold > 0.0 then
858         if column.mlConfidence < confidenceThreshold then
859             column.semanticType ← "NIL"
860             column.mlConfidence ← 0.0
861             column.NILFlags ← true
862             column.reviewLog ← "NIL detected"
863         end if
864     end if
865
866     SavePerModelPredictions(column, analysedColumns_models)
867
868     analysedColumns_models ← analysedColumns_models ∪
869     {column.semanticType, column.mlConfidence, column.NILFlags,
870     column.reviewLog}
871
872     if confidenceThreshold > 0.0 then
873         if column.mlConfidence < confidenceThreshold then
874             column.semanticType ← "NIL"
875             column.mlConfidence ← 0.0
876             column.NILFlags ← true
877             column.reviewLog ← "NIL detected"
878         end if
879     end if
880
881     SavePerModelPredictions(column, analysedColumns_models)
882
883     analysedColumns_models ← analysedColumns_models ∪
884     {column.semanticType, column.mlConfidence, column.NILFlags,
885     column.reviewLog}
886
887     if confidenceThreshold > 0.0 then
888         if column.mlConfidence < confidenceThreshold then
889             column.semanticType ← "NIL"
890             column.mlConfidence ← 0.0
891             column.NILFlags ← true
892             column.reviewLog ← "NIL detected"
893         end if
894     end if
895
896     SavePerModelPredictions(column, analysedColumns_models)
897
898     analysedColumns_models ← analysedColumns_models ∪
899     {column.semanticType, column.mlConfidence, column.NILFlags,
900     column.reviewLog}
901
902     if confidenceThreshold > 0.0 then
903         if column.mlConfidence < confidenceThreshold then
904             column.semanticType ← "NIL"
905             column.mlConfidence ← 0.0
906             column.NILFlags ← true
907             column.reviewLog ← "NIL detected"
908         end if
909     end if
910
911     SavePerModelPredictions(column, analysedColumns_models)
912
913     analysedColumns_models ← analysedColumns_models ∪
914     {column.semanticType, column.mlConfidence, column.NILFlags,
915     column.reviewLog}
916
917     if confidenceThreshold > 0.0 then
918         if column.mlConfidence < confidenceThreshold then
919             column.semanticType ← "NIL"
920             column.mlConfidence ← 0.0
921             column.NILFlags ← true
922             column.reviewLog ← "NIL detected"
923         end if
924     end if
925
926     SavePerModelPredictions(column, analysedColumns_models)
927
928     analysedColumns_models ← analysedColumns_models ∪
929     {column.semanticType, column.mlConfidence, column.NILFlags,
930     column.reviewLog}
931
932     if confidenceThreshold > 0.0 then
933         if column.mlConfidence < confidenceThreshold then
934             column.semanticType ← "NIL"
935             column.mlConfidence ← 0.0
936             column.NILFlags ← true
937             column.reviewLog ← "NIL detected"
938         end if
939     end if

```

```

30     p_max ← PredictConfidence(clf, X, y_hat)           // calibrated prob / softmax / decision fn
31     preds[clf.name] ← (y_hat, p_max)
32 end for

33 // Aggregate predictions (e.g., weighted by validation F1)
34 (aggType, aggConf) ← AggregatePredictions(preds, weights=ValidationScores(classifiers))
35 column.semanticType ← aggType
36 column.mlConfidence ← aggConf

37 // Reconciliation & review signals
38 column.ruleSignals ← sourceKeywords                  // retained for explainability
39                                         column.consistencyFlag ← Consistency(ruleHint=BestRuleHint(sourceKeywords),
40                                         mlType=aggType) // AGREE / DISAGREE / NONE

41 // Low-confidence handling → NIL + enrichment queue
42 if aggConf < confidenceThreshold then
43     column.semanticType ← "NIL"
44     LogForReview(column, sourceKeywords, TopK(preds, k=3)) // store inputs + top candidates
45     AppendToEnrichmentQueue(sourceKeywords, column) // drive dictionary/abbr updates
46 end if

47 SavePerModelPredictions(column, preds)               // persist all models' outputs
48 end for

49 return analysedColumns_models(columns)
50 end function

```

#### Reproducibility details.

We fix the random seed to **42** across all steps. Data are split with **80/20 stratified hold-out** (`train_test_split(test_size=0.2, stratify=labels)`), and model selection uses **GridSearchCV** with **3-fold CV** (`cv=3, n_jobs=-1`).

**Imbalance handling:** we apply **SMOTE** with `k_neighbors=nn` (as in the released code) and a task-specific sampling\_strategy.

**Feature extraction:** a custom **TF-IDF** vectorizer built from the **union** of **DBHeaders** tokens and the **Formats** dictionary (as normalised in §2.3).

#### Evaluated models & grids:

**RandomForest** (`n_estimators ∈ {100,200}, max_depth ∈ {None,30}, min_samples_split ∈ {2,5}`),  
**LogisticRegression** (`C ∈ {0.1,1}, solver='lbfgs', max_iter=20000),  
GradientBoosting (n_estimators=100, learning_rate=0.1, max_depth=3),  
KNN (n_neighbors ∈ {3,5}, weights ∈ {uniform,distance}),  
LinearSVC (C ∈ {0.1,1}, max_iter=20000, dual='auto').`

**Persistence:** the selected model per classifier and the TF-IDF vectorizer are exported via **joblib** to support exact reproduction.

**Repository:** code, dictionaries (Formats/Abbreviations), range tables, and the FinalFormat→KG mapping tables are released in [38].

#### Appendix B - Bounded Numerical types

This appendix lists the bounded numerical formats adopted throughout the rule-based and data-quality pipelines. Each of these formats defines a numeric attribute whose valid range is defined a priori, enabling automatic detection of out-of-range and physically impossible values during data quality assessment (Section 2.7). The bounds shown below were derived from domain standards and dataset statistics and are applied uniformly across all evaluated data sources. These limits are **not fixed thresholds**, they are intentionally subjective and can be easily altered to reflect alternative domain assumptions or updated business rules, for example, different accepted ranges for *year* or *age*.

Table A

Bounded Numerical	Minimum	Maximum	Bounded Numerical	Minimum	Maximum
acidity	0	7	normalized	0	1
age	0	130	numerical between 0 and 360	0	360
alkalinity	7	14	numerical between 0 and 60	0	60
bloodpressure	0	250	percentage	0	100
day	1	366	ph	0	14
heartrate	40	200	saltiness	0	40
hour	0	24	tannins	0	100
latitude	-90	90	week	1	53
longitude	-180	180	year	1800	2100

#### Appendix C - TOP 100 list of NIL tokens ordered by amount

#	CleanedColumn	Count	#	CleanedColumn	Count
1	no	278	21	sa	30
2	so	149	22	f	30
3	Nil	98	23	wp	28
4	min	80	24	gd	26
5	ga	73	25	st	25
6	sv	73	26	rc	24
7	+	71	27	off	22
8	to	68	28	s no	21
9	lg	64	29	pl	21
10	s	60	30	m	21
11	er	56	31	maf	21
12	gf	49	32		21
13	cg	49	33		20
14	sf	45	34	1b	19
15	from	40	35	c	19
16	pp	36	36	ch	19
17	dp	34	37	zero	19
18	po	33	38	pb	19
19	b	32	39	yc	18
20	ba	32	40	tav	18

#	CleanedColumn	Count
41	shg	18
42	3	17
43	fc	17
44	bf	17
45	x	17
46	out	17
47	sl no	16
48	bs	16
49	new	16
50	lvl	16
51	streak	16
52	for	15
53	us	15
54	medal	15
55	otl	15
56	designation	15
57	lost	14
58	fs	14
59	vote	14
60	chest	14
61	gw	14
62	ta	14
63	extension	14
64	stats	14
65	ff	14
66	nat	14
67	tries	13
68	tdp	13
69	dep	13
70	defense	13

#	CleanedColumn	Count
71	replies	13
72	mat	13
73	prd	13
74	may	13
75	toi	13
76	rf	13
77	ot	13
78	ra	13
79	away	13
80	str	12
81	warp	12
82	generosity	12
83	stds	12
84	winnings	12
85	thailand	12
86	bpg	12
87	guild	12
88	arr	12
89	bop	12
90	wpa	12
91	pt	12
92	joined	12
93	realm	11
94	station	11
95	delta	11
96	@bat	11
97	previous	11
98	adult	11
99	subcategory	11
100	msrp	11

## Appendix D - Preprocessing and Semantic Type Assignment: Detailed Workflow

Our semantic type detection pipeline employs a comprehensive preprocessing and mapping approach that transforms raw column headers into interpretable semantic annotations via the *FinalFormat* type system. This process involves a series of systematic steps, each designed to maximize robustness, flexibility, and explainability.

Our semantic type detection pipeline executes the following steps, closely mirroring the production implementation:

### 1. Initial Parsing and Metadata Extraction:

- Extract numeric prefixes (e.g., “1.ID”) for traceability.
- Parse column names by splitting at colons, slashes, or parentheses to isolate core terms.
- Extract free-text descriptions from split points, or from parentheses if none exists, ensuring use of all available metadata.

### 2. Header Normalization:

- Convert all header tokens to lowercase.
- Remove or replace punctuation and special characters (underscores, hyphens, asterisks).
- Split compound words using spaces, dots, underscores, hyphens, or camelCase patterns (e.g., “AvgPitStopTime” → “avg pit stop time”).
- Explicitly tokenize symbols (e.g., “%”) for percentage detection.
- Match plural and singular forms (e.g., “chlorides” → “chloride”).

### 3. Abbreviation and Variant Expansion

- Replace abbreviations using a dictionary of over 1000 entries (e.g., DOB → date of birth; bp → blood pressure).
- Apply regular expressions to tokenize and substitute complex patterns, including units (mg/dL, kg/m<sup>2</sup>), and non-alphabetic tokens like °C, cm<sup>2</sup>, etc.

### 4. SourceKeywords Extraction

- Identify key tokens (SourceKeywords) that capture semantic meaning (e.g., “age”, “date”, “amount”).
- Scan descriptions for known tokens, including “has”, “is”, or numeric/temporal keywords.
- Use primary/foreign key metadata to override assignments where appropriate.

### 5. Semantic Type Assignment (Rule-Based and ML classifier Logic)

- Match tokens to a curated dictionary (2,800 mappings) to assign one of 39 FinalFormat types.
- Apply priority logic:
  - “binary” overrides all if “has”, “is”, or “or not” is present.
  - “percentage” takes precedence if “%” or related terms detected.
  - Hierarchical rules resolve conflicts between categorical, numerical, and domain types.
  - Special logic for “name” columns in certain contexts (city, state, etc.).
- If no match is found:
  - Retry abbreviation expansion and check individual tokens (with plural handling).
  - If still unresolved, use ML-based prediction.

### 6. Ambiguity Resolution

- Use ML classifier (Random Forest, Logistic Regression, etc.) with weighted TF-IDF features.
- Assign confidence scores to all assignments for human-in-the-loop review.

## 7. Knowledge Graph Annotation

- Perform syntactic and semantic matching (FastText) between normalized headers and KG property/class labels.
- Use substring and similarity logic for best match selection (DBpedia, Schema.org).

## 8. Final Type Selection and Logging

- Consolidate assignments if column and description differ, prioritizing by semantic importance (e.g., prefer “datetime”, “IDcolumn”, etc.).
- Attribute assignments to specific source keywords for explainability.
- Assign “Nil” and log cases for manual review if no match is found.

Our semantic type detection pipeline applies a comprehensive sequence of normalization, expansion, and mapping steps that transform raw column headers into interpretable semantic annotations via the FinalFormat type system. The workflow (detailed above) enables robust, transparent, and flexible type assignment across real-world tabular data.

### Practical Examples of Semantic Type Assignment

**Table B** below demonstrates, on a large and diverse set of real columns, how our pipeline’s dictionary logic, abbreviation expansion, and contextual rules deliver precise, explainable semantic types.

### Key Interpretation Patterns and Highlights

#### Binary Columns:

Binary-type columns are often signaled by common words such as “has”, “is”, or phrases like “or not”. The pipeline detects these cues and classifies such columns under the binary FinalFormat, ensuring boolean attributes are annotated correctly even when explicit True/False values are absent.

#### Disambiguating Categorical Types:

Columns containing the word “range” (e.g., “price range”, “date range”) are accurately identified as categorical, despite potentially misleading co-occurring terms like “date” or “price” that might otherwise point toward temporal or financial types. This illustrates the nuanced, context-aware mapping the system achieves via dictionary-based disambiguation.

#### Geographic and Demographic Mapping:

Headers with words like “nationality”, “province”, or “capital” are consistently mapped to country, state, or city FinalFormats. Synonym handling ensures that “ProvinceName” is linked to state, and “capital name” to city, even if these terms don’t match dictionary entries exactly.

#### Temporal and Timestamp Recognition:

Fields such as “updated”, “day of week”, and “timestamp” are mapped to their appropriate FinalFormats:

“updated” → date  
“request timestamp” → datetime  
“day of week” → weekday

This allows precise differentiation between temporal and generic date columns.

#### Financial and Monetary Attributes:

Lexical cues like “cash”, “currency”, “usd”, and “price” reliably trigger the assignment to the money FinalFormat, enabling wide coverage of financial information.

#### Communication and Identifier Columns:

Terms like “fax”, “cellular phone”, and “author channel id” are normalized to phone and IDcolumn, respectively, with common abbreviations (e.g., “nr” for number) mapped to numerical.

#### Scientific and Measurement Values:

Abbreviations and specialized terms, such as “wgt avg” (weight average) or “trestbps” (blood pressure), are interpreted using expansion rules, with assignments to numerical $>=0$  and bloodpressure, reflecting the system’s capacity to handle both generic and domain-specific semantics.

#### Web and Address Columns:

Patterns like “web address”, “link”, or “href” result in URLformat annotations, unifying various forms of URL-related fields.

Postal and address information, including “zipcode”, “provider zip code”, and “purchase address”, is reliably mapped to postalcode or street.

#### String and Free-Text Columns:

Terms such as “abstract”, “desc 1”, and “spec abstract” are interpreted as free text and assigned the string FinalFormat.

#### Other Notable Patterns:

- Medical/Health: Medical headers like “bp” or “trestbps” are mapped to bloodpressure.
- Percentage: Any header containing the “%” symbol or “percent” is assigned percentage.
- Temporal granularity: “Month”, “hour”, “week number”, “year” are all mapped to their corresponding temporal FinalFormats.

Through these patterns, **Table B** provides a transparent view of our pipeline’s ability to handle abbreviation, synonymy, compound phrases, and real-world header messiness, delivering precise, actionable semantic types for data quality assessment.

**Table B**

Original Header	Normalized	SourceKeywords	FinalFormat
Members with age 5 - 17 years old	members with age 5 17 years old	age	age
Ankle_Ground_Angle	ankle ground angle	angle	angle
has_birth_date	has birth date	has	binary
IsBasedOnRealStory	is based on real story	is	binary
BookedHotelOrNot	booked hotel or not	or not	binary
bp	bp	blood pressure	bloodpressure
trestbps	trestbps	blood pressure	bloodpressure
Reported Influenza Activity	reported influenza activity	activity	categorical
Aircraft Manufacturer	aircraft manufacturer	aircraft	categorical
Pclass_1	pclass 1	class	categorical
ATTEND_DEPT	attend dept	department	categorical
Date Range*	date range*	range	categorical
price_range	price range	range	categorical
Father's Birth Place	father's birth place	birth place	city
CapitalName	capital name	capital	city
Author, Country	author, country	country	country
team_one_player_one_nationality	team one player one nationality	nationality	country
BIRTHDATE	birthdate	birthdate	date
dtRef	dt ref	date	date
Data Last Updated	data last updated	updated	date
Order Date and Time	order date and time	date and time	datetime
Request timestamp	request timestamp	timestamp	datetime
school_day	school day	day	day
FLAG_EMAIL	flag email	email	E-mailformat
order_hour_of_day	order hour of day	hour	hour
authorChannelId	author channel id	id	IDcolumn
Nameserver IP Address	nameserver ip address	ip	IPformat
src_ip_country_code	src ip country code	ip	IPformat
vehicle_gps_latitude	vehicle gps latitude	latitude	latitude
Delivery_location_longitude	delivery location longitude	longitude	longitude
Cash amount	cash amount	cash	money
discount_price_currency	discount price currency	currency	money
cons.price.idx	cons price idx	price	money
Values in Billions USD	values in billions usd	usd	money
Date_Of_Death_Month	date of death month	month	month
Head Coach	head coach	coach	name
Artistic Director	artistic director	director	name
fullname words	fullname words	fullname	name
LASTNAME	lastname	lastname	name
Person Baptised	person baptised	person	name
Allowed Amount	allowed amount	amount	numerical
Trip_Distance_km	trip distance km	distance	numerical
Ground Elevation	ground elevation	elevation	numerical
Flug-Nr.	flug nr	number	numerical
Deforestation_Area_Ha	deforestation area ha	area	numerical>=0
free_throw_attempts	free throw attempts	attempts	numerical>=0
ViewDepth	view depth	depth	numerical>=0
DirectoryEntryImportSize	directory entry import size	size	numerical>=0
Wgt. Avg.	wgt avg	weight	numerical>=0
YearsInCurrentRole	years in current role	years	numerical>=0
% of Total Supply Owned	% of total supply owned	%	percentage
Percent of State employment	percent of state employment	percent	percentage
Number of Cellular phone	number of cellular phone	cellular phone	phone
Fax Numbers	fax numbers	fax	phone
Customer_Postal_Code	customer postal code	postal code	postalcode
Provider Zip Code	provider zip code	zip code	postalcode
ProvinceName	province name	province	state
Purchase Address	purchase address	address	street
Spec. abstract	spec abstract	abstract	string
desc_1	desc 1	description	string
AvgPitStopTime	avg pit stop time	time	time
goods-title-link-jump href	goods title link jump href	href	URLformat
Free Download Link	free download link	link	URLformat
Web Address	web address	web address	URLformat
arrival_date_week_number	arrival date week number	week number	week
Day of Week	day of week	day of week	weekday
athlete_year_birth	athlete year birth	year	year

These examples illustrate how the system transforms raw, often noisy or abbreviated, column headers into precise semantic annotations. By combining normalization, robust abbreviation expansion, and hierarchical rule logic, our pipeline ensures reliable type assignment even in the face of ambiguity, domain-specific terminology, or non-standard header formats. This approach not only supports accurate data profiling but also underpins the effectiveness of our automated data quality assessment framework. Columns that remain unresolved after this rule-based process are passed to the **machine-learning classifier** described in *Appendix A*, which provides detailed configuration and reproducibility information for that stage.

## Appendix E - Full SemTab 2024 Metadata-to-KG Track [12] Analysis

### E.1 Methods (Human Review Protocol)

One author, **blinded to official GT labels and to all cell values**, adjudicated each prediction using only the **column header** and public ontology documentation (definitions and official redirects). Each case was assigned to one of four categories:

- **C1 (GT-CONF):** Predicted URI equals the GT URI or an **official redirect to the same canonical entity**. (*None were found beyond the official script; C1 contributes 0 in our audit.*)
- **C2 (GT-REFINE):** Predicted URI is **not** the GT entity but is a **more appropriate** entity/property for the header (change of granularity/scope/type that better reflects the header). We submit these as **candidate GT issues**.
- **C3 (ALT):** Predicted URI is **not** the GT entity and **not necessarily better**, but remains **plausible** under header-only interpretation (synonym/alias/generalisation).
- **C4 (INC):** Remaining cases (NIL, non-English unresolved, unit/numeric-only, ambiguous, lexical confusion, etc.).

We report raw counts and, where space permits, **95% binomial Wilson confidence intervals** for proportions.

### E.2 Results Overview (n = 141)

**Official (GT-strict).** Using exact URI equality for the same canonical entity (no manual flags):

- **Hit@1 (strict):**  $63/141 \approx 0.45$  (95% CI  $\approx [0.36, 0.53]$ )
- **Hit@5 (strict):**  $67/141 \approx 0.48$  (95% CI  $\approx [0.40, 0.56]$ )

**Audit reclassification (ours only; header-only rubric).** From the adjudicated set:

- **C1 (GT-CONF, additional): 0** (*no extra same-entity corrections beyond strict*)
- **C2 (GT-REFINE): 43/141 (30.5%)**
- **C3 (ALT): 6/141 (4.0%)**
- **C4 (INC): 24/141 (17.0%)**

**Diagnostic Hit@1 (ours only; not for cross-system ranking).**

- **B1: Strict:**  $63/141 = 0.447$  ( $\approx 0.45$ )
- **B2: B1 + C2 (GT-REFINE):**  $(63+43)/141 = 106/141 = 0.752$  (95% CI  $\approx [0.67, 0.81]$ )
- **B3: B2 + C3 (ALT):**  $(106+6)/141 = 112/141 = 0.794$  (95% CI  $\approx [0.72, 0.85]$ )

**Diagnostic Hit@5 (ours only; header-only audit)**

Starting from the strict Top-5 match count ( $67/141 = 0.475$ ), we apply the same cumulative reconciliation used for Hit@1:

- **B1 — Strict:**  $67/141 = 0.475$  (95% CI  $\approx [0.40, 0.56]$ )
- **B2 — B1 + C2 (GT-REFINE):**  $(67 + 43)/141 = 110/141 = 0.781$  (95% CI  $\approx [0.71, 0.85]$ )
- **B3 — B2 + C3 (ALT):**  $(110 + 6)/141 = 116/141 = 0.823$  (95% CI  $\approx [0.76, 0.88]$ )

#### Notes.

1. **C1 (GT-CONF)** contributed **0** cases; therefore Panel B starts from the strict baseline.
2. These figures are **diagnostic** and apply to **our outputs only**; they are **not used for cross-system ranking**.
3. Under this diagnostic view, **C2/C3 are counted as correct** for Top-5 irrespective of whether the GT URI appears in the model's Top-5 list, because the human audit judged them ontology-consistent and header-appropriate under the rubric.

### E.3 Tables

**Table C - GT-Refinement (Author-Proposed Replacement)**

Column values	GT Annotation	Our Annotation	Items	Justification
Year	dbo:releaseDate	dbo:year	20	<b>Granularity:</b> header denotes general temporal attribute; releaseDate presupposes an event/product.
ISO(2)/ISO(3)	dbo:iso31661Code	dbo:isoCode	2	<b>Scope:</b> header indicates generic ISO code; 3166-1 scope not evidenced by header alone.
Length	dbo:duration	dbo:length	1	<b>Type:</b> physical extent vs time interval.
Government	dbo:governmentType	dbo:government	4	<b>Entity vs attribute.</b>
State	dbo:location	dbo:state	4	<b>Geopolitical specificity.</b>
GDP	dbo:grossDomesticProduct	dbo:gdpPerCapita	1	<b>Indicator specificity</b> (flag as refinement with caveat).
Alphabeticcode	dbo:currencyCode	dbo:code	4	<b>Generalisation:</b> currency scope not evidenced.
GrantingInstitution	dbo:almaMater	dbo:institution	1	<b>Relation vs institution.</b>
Height(ft), HEIGHTINMETERS	dbo:elevation	dbo:height	5	<b>Domain shift:</b> non-geographical height likely.
Board	dbo:owner	dbo:board	1	<b>Role mismatch.</b>
		<b>TOTAL</b>	43	<b>30.5%</b>

**Table D - Ontology-Consistent Alternative (Plausible, not GT)**

Column values	GT Annotation	Our Annotation	Items	Justification
Size	dbo fileSize	dbo collectionSize	1	Header ambiguity; plausible alternative.
Label	dbo developer	dbo distributingLabel	1	Label semantics vs developer.
System	dbo computingPlatform	dbo systemRequirements	3	Ambiguous "System"; both plausible.
watergauge	dbo elevation	dbo water	1	Water-level semantics plausible.
		<b>TOTAL</b>	6	<b>4%</b>

**Table E - Incorrect (incl. NIL / multilingual / unit-only / lexical confusions)**

Column values	GT Annotation	Our Annotation	Items	Issue
Basincountries, Staat(en)	country	Nil	2	Multilingual/compound words issues
8.848, Feets, Meters, H?he	elevation	Nil	5	Numeric/unit headers lacking context
GNI, GDPNominal(US\$M)	giniCoefficient, grossDomesticProduct	Nil	2	Abbreviations and economic indicators
Ashton-under-Lyne, Editeur, Endroit, Stadt	location, owner	Nil	5	Non-English terms, ambiguous without context
Europe, Japan, Rel.	releaseDate	Nil	5	Contextual mismatch with GT annotation
NorthAmerica	releaseDate	northWestPlace	1	Contextual mismatch with GT annotation
Rationale	knownFor	ratio	2	Semantic misunderstanding due to lexical similarity
1T	iataAirlineCode	Nil	1	headers lacking context
Max.-Tiefe-(m)	depth	max	1	headers lacking context
		<b>TOTAL</b>	24	<b>17%</b>

#### E.4 Successes and Challenges (Header-Only CTA)

**Scope.** This section interprets results under the **header-only** constraint and uses the audit rubric from §E.1: **C.1 GT-Confirmed** (same canonical entity), **C.2 GT-Refinement** (author-proposed replacement for GT), **C.3 Ontology-Consistent Alternative** (plausible but not GT), and **C.4 Incorrect** (incl. NIL/ambiguous/multilingual/unit-only).

##### E.4.1 Success cases (C.1 / C.3)

A substantial fraction of columns with clear headers are annotated reliably:

- **Examples (C.1 GT-Confirmed where applicable).** Headers such as “Government”, “Year”, and “Height(ft)” tend to map to intuitive DBpedia targets. When the **same canonical URI** as GT is reached (identical URI or official redirect), these are counted under **C.1**.
- **Plausible alternatives (C.3).** In some cases the model selects ontology-consistent alternatives that remain defensible under header-only semantics, e.g., elevation → height for non-geographic contexts, developer → distributingLabel for “Label”, or computingPlatform → systemRequirements for “System”. These are recorded as **C.3** because they are plausible yet **not** the GT entity.

Takeaway. Under the header-only premise, the system is robust to minor label differences and often converges to semantically sound choices even when GT uses a different—but related—entity/property.

##### E.4.2 Error and challenge cases (C.4)

Most failures arise from classes of headers that are inherently fragile under header-only interpretation:

- **Multilingual/compound or abbreviated headers.** E.g., “Basincountries”, “Staat(en)”, “Editeur”, “Rel.”
- **Unit-only or numeric-only headers.** E.g., “8.848”, “Feets”, “Meters”, “Max.-Tiefe-(m)”.
- **Code/identifier without scope.** E.g., “1T” (IATA airline code).
- **Ambiguous geography vs. temporal GT.** E.g., “Europe”, “Japan”, “NorthAmerica” marked as releaseDate in GT; such mappings appear to rely on **cell values**, not headers.

We record these under **C.4 Incorrect** (including **NIL** when the system declines to guess). In header-only evaluation, **NIL** can be desirable behaviour for high-precision operation.

#### E.5 Human-Informed Diagnostic of GT Inconsistencies (C.2 / C.3)

**Motivation.** Strict GT requires the predicted URI to match the **same canonical entity** as the ground truth. However, benchmarks may encode **granularity** or **context** that is not present in column headers. We therefore performed a blinded, header-only audit to separate **(i)** genuine model errors from **(ii)** GT idiosyncrasies.

##### E.5.1 GT-Refinement (C.2 author-proposed replacements)

We identified cases where the model’s URI appears **more appropriate** for the header than the GT entity (candidate GT issues). Representative examples:

- **Year:** releaseDate → year (n=20). The header denotes a general temporal attribute; releaseDate presupposes an event/product.
- **ISO(2)/ISO(3):** iso31661Code → isoCode (n=2). The header does not evidence the 3166-1 scope; a scope-neutral ISO code is header-faithful.
- **Length:** duration → length (n=1). Physical extent vs. time interval.
- **Government:** governmentType → government (n=4). Institution vs. classification attribute.
- **State:** location → state (n=4). Geopolitical specificity.
- **Alphabeticcode:** currencyCode → code (n=4). General “code” without currency scope in the header.
- **GrantingInstitution:** almaMater → institution (n=1). Granting body vs. alumni relation.
- **Height(ft), HEIGHTINMETERS:** elevation → height (n=5). Non-geographical height signalled by header.
- **Board:** owner → board (n=1). Governance/advisory body vs. ownership.

**Classification.** All such cases are recorded as **C.2 GT-Refinement** (not counted as “Correct” under strict GT), and fed into **Panel B** diagnostics to quantify potential GT noise in header-only settings.

##### E.5.2 Ontology-Consistent Alternatives (C.3)

We also found plausible alternatives that are defendable under header-only semantics, but without claiming they “correct” GT. Examples include “Size: fileSize → collectionSize”, “System: computingPlatform → systemRequirements”, and “watergauge: elevation → water”. These are recorded as **C.3**. Diagnostic use only. Neither **C.2** nor **C.3** is used for cross-system ranking; they solely inform **Panel B** (ours only) to show how strict scores shift when GT granularity/aliasing is accounted for.

#### E.6 Consolidated Implications (Header-Only CTA)

- **Strict baseline preserved.** All comparative conclusions in the paper rely on **official GT-strict** results (Panel A).
- **Quantified GT noise.** The audit suggests that ~30.5% of residual strict-GT “errors” reflect **GT-Refinement** (C.2) and ~4% reflect **Plausible Alternatives** (C.3) under header-only interpretation, lifting diagnostic Hit@1 from **0.45** to **≈0.76** (E.2) and **≈0.80** (C.2+C.3).
- **Benchmark design.** We recommend that future benchmarks distinguish **header-only** targets from **context-dependent** targets and maintain synonym/alias sets per entity/property to reduce avoidable mismatches.

## E.7 Researcher-Proofing and Artifacts

- **Uncertainty reporting.** For our measures, we report **raw counts** and **95% Wilson CIs**; baseline CIs are omitted if not provided in the official report.
- **Reproducibility files.** We release a XLSX with one row per column:  
SEMTAB\_2024\_dbpedia\_match\_results.xlsx [38]
- **Namespace coverage.** Some DBpedia concepts exist only as capitalised **classes** (e.g., dbo:Airport) without a lowercase property; systems restricted to properties may miss such cases. We index **both class and property namespaces** to mitigate false negatives.
- **Task alignment.** CTA is a **column-level** assignment task; benchmarks like **t2Dv2** provide **table-level** classes and are not directly comparable to column-level CTA outcomes.

## Appendix F - Comparative Analysis of Semantic Type Detection Approaches

This appendix presents an **extended** comparative analysis of representative systems for semantic type detection. **Table F** consolidates the survey of Liu et al. [23] and **updates/extends** it with recent approaches (including **LLM-based metadata** methods) and **this header-centric framework**.

Dimensions covered include system class, header usage, **header-only capability**, knowledge-graph integration (DBpedia/Schema.org), benchmark datasets, **SemTab setting/participation** (including **offline comparisons**), and key techniques; offering a concise view of recent trends and gaps.

**Table F - Extended comparative analysis of semantic type detection approaches,  
adapted and expanded from Table 1 in Liu et al. [23].**

System (Algorithm)	Year	Class	Header Usage	Header-only Capable?	KG	Data Source	SemTab?	Key Technique
Wang et al. [44]	2012	Heuristic, Lookup	Primary	No	Probbase	Custom Wikipedia Tables	Baseline	Probbase + rules
C <sup>2</sup> [19]	2021	Heuristic, Lookup	Probabilistic	No	DBpedia, Wikidata	Limaye, ISWC2017, SemTab 2019, T2D	Winner CTA 2020	Ensemble/statistics
Magic [40]	2021	Heuristic, Lookup	Direct compare	No	DBpedia, Wikidata	SemTab 2021	Participant	INK embeddings
Alobaid et al. [3]	2022	Heuristic, Lookup	Main input	Yes	DBpedia	SemTab 2021, T2D	Participant	String similarity & normalization
TableMiner+ [47]	2017	Heuristic, Iterative	Lexical	No	Freebase	Limaye, IMDB, MusicBrainz	Baseline	Lexical/iterative
CSV2KG [41]	2019	Heuristic, Iterative	Fallback	No	DBpedia	SemTab 2019	Baseline	Heuristic
Mtab [29, 30, 31]	2019	Heuristic, Iterative	Ensemble	No	DBpedia, Wikidata	SemTab 2019–2021	Winner	Lookup, NLP ensemble
Limaye et al. [22]	2010	Feature Engineering	Strong	No	YAGO	Limaye	Baseline	Prob. graphical model
Mulwad et al. [27,28]	2010	Feature Engineering	Limited	No	Wikitology	Limaye	Baseline	Heuristic, SVM
DAGOBAH Embeddings [6]	2019	Deep Learning, KG Modelling	Central	No	DBpedia, Wikidata	SemTab 2019	Baseline (CTA)	ML+heuristic ensemble
Sherlock [14]	2019	Deep Learning, Table Modelling	Minimal (opt.)	No	DBpedia	T2D, VizNet	Baseline	Deep CNN
Sato [45]	2020	Deep Learning, Table Modelling	No header	No	DBpedia	VizNet	Baseline	TabNet/CRF/BERT
TURL [10]	2020	Deep Learning, Table Modelling	Limited	No	DBpedia	WikiGS, WikiTable, T2D	No	Tabular transformer
Doduo/BERT-CTA [42]	2022	Deep Learning, Table Modelling	Limited	No	-	WikiTable, VizNet	Participant	Fine-tuned BERT (multi-task)
RECA Sun et al [39]	2023	Deep Learning, Table Modelling	Limited	No	DBpedia	T2Dv2, SemTab, Wikipedia	No	Related table context + features
ADWAN	2024	LLM-based, Metadata	Prompts LLM	Yes	Dbpedia, Schema.org	SemTab2024 Metadata Only track	Winner	RAG + CoT + Self-Consistency + RRF
CVA/Metalinker(2024)	2024	LLM-based, Metadata	Main input	Yes	Dbpedia, Schema.org	SemTab2024 Metadata Only track	Participant	Zero-shot LLMs + RAG + SemanticBERT
Anonymized [37]	2024	Heuristic, Lookup	Exclusive header	Yes	-	UCI	No	Dynamic, feedback-driven, hybrid fallback
This paper	2026	Heuristic, Lookup	Exclusive header dynamic	Yes	Dbpedia, Schema.org	Kaggle, VizNet, Sato, UCI, Prague, SemTab 2024 Metadata Only track	2024 Offline comparison	Rule-based + feedback/ML fallback

*Notes:* (i) “Header-only” indicates operation without cell values; (ii) “SemTab note” distinguishes official participation from **offline** evaluations; (iii) rows and sources were cross-checked against cited papers; the curation criteria and change log are available in [38] for reproducibility.