

ARISTOTLE UNIVERSITY OF THESSALONIKI
FACULTY OF SCIENCES
SCHOOL OF INFORMATICS
MSC IN Data and Web Science

Eurostat's Knowledge Graph Enrichment

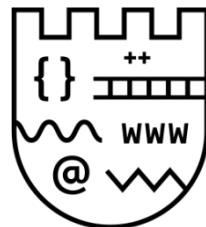
Εμπλουτισμός του γράφου γνώσης της Eurostat

Skaperdas Efstratios

AEM (ID): 96

Supervisor:

Prof. Nick Bassiliades



Date:

07-2024

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ρητά ότι η παρούσα πτυχιακή εργασία, καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στο πλαίσιο αυτής της εργασίας, αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή/και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Abstract

In recent years, the enrichment of knowledge graphs (KGs) has become increasingly important in various domains, providing a robust foundation for data integration, information retrieval, and artificial intelligence applications. This thesis investigates the enrichment of a knowledge graph that was developed in the first half of 2022, using data mainly from Eurostat [1], [2]. This knowledge graph consists of 307.444 triples in total, 171.050 unique entities, with 59.735 of them as subjects and 150.588 as objects, and 43 unique relations.

Knowledge graphs are organized representations of knowledge, including entities, properties, and the connections between them. Knowledge graphs are essential for structuring data in a machine-understandable manner, facilitating complex data processing, and aiding in semantic search and reasoning. The importance of knowledge graphs is that they facilitate the precise and efficient retrieval of information and decision-making by connecting a variety of categories of information.

Machine learning, a subset of artificial intelligence, has become increasingly dependent on structured information representations, such as knowledge graphs, to enhance its capacity to understand and analyze complex data. It plays a crucial role in the development and improvement of knowledge graphs. Through the use of machine learning methods, we can automate the process of extracting, classifying, and predicting connections within the data. This leads to a substantial improvement in the accuracy and scalability of constructing knowledge graphs. This thesis investigates several machine learning approaches to tackle significant obstacles in knowledge graph augmentation.

Graph neural networks (GNNs) have become powerful methods for extracting meaningful representations from knowledge graphs. They enable various tasks such as entity categorization, link prediction, and semantic similarity evaluation. Furthermore, the combination of machine learning and knowledge graphs enables activities such as discovering new information and conducting semantic searches.

In knowledge graphs and machine learning, triple classification and relation prediction problems are critical issues. The objective of triple classification is to ascertain the accuracy or validity of a particular triple (consisting of a subject, predicate, and object) within a knowledge graph. This task entails assigning labels to triples depending on whether they are present or absent in the knowledge graph, such as "true" or "false." It has a vital function in verifying the coherence and comprehensiveness of knowledge graphs.

Conversely, relation prediction is concerned with forecasting the specific sort of connection (predicate) that links two entities (subject and object) in a knowledge graph. This job is critical for tasks such as link prediction and knowledge base completion, where successfully forecasting associations aids in filling in missing links and improving the graph's structure. Both tasks use machine learning methodologies, such as graph neural networks and embedding models, to acquire information about items and connections. This enables precise categorization and prediction inside knowledge graphs.

Combining triple classification and relation prediction tasks entails leveraging machine learning models to make accurate predictions within knowledge graphs. One approach is to use triple classification models that can predict the validity of triples across all possible relations, thereby indirectly performing relation prediction. This method typically involves the following steps: First, encode the subject and object entities, as well as candidate relations, into a model that can produce a probability score for each relation. Second, for each triple (subject, relation, object), the model evaluates the likelihood that the triple is valid under each candidate relation.

Finally, the model selects the relation associated with the highest probability score as the predicted relation for that triple. The triple classification model effectively combines the tasks by determining the likelihood of three things having different connections. This makes it easier to accurately predict relationships in knowledge graphs. By combining these tasks, researchers can enhance the completeness and accuracy of knowledge bases, supporting applications such as semantic search, recommendation systems, and automated reasoning in various domains.

Keywords: Machine learning, Knowledge graphs, Deep learning, Knowledge graph enrichment, Graph neural networks

Περίληψη

Τα τελευταία χρόνια, ο εμπλουτισμός των γράφων γνώσης (KGs) έχει γίνει όλο και πιο σημαντικός σε διάφορους τομείς, παρέχοντας ένα ισχυρό θεμέλιο για την ολοκλήρωση δεδομένων, την ανάκτηση πληροφοριών και τις εφαρμογές τεχνητής νοημοσύνης. Η παρούσα εργασία διερευνά τον εμπλουτισμό ενός γράφου γνώσης που αναπτύχθηκε το πρώτο εξάμηνο του 2022, χρησιμοποιώντας δεδομένα κυρίως από την Eurostat [1], [2]. Ο συγκεκριμένος γράφος γνώσης αποτελείται από 307.444 τριπλέτες συνολικά, 171.050 μοναδικές οντότητες, με 59.735 από αυτές ως υποκείμενα και 150.588 ως αντικείμενα, και 43 μοναδικές σχέσεις.

Οι γράφοι γνώσης είναι οργανωμένες αναπαραστάσεις της γνώσης, συμπεριλαμβάνοντας τις οντότητες, τις ιδιότητες και τις μεταξύ τους συσχετίσεις. Οι γράφοι γνώσης είναι απαραίτητοι για τη δόμηση των δεδομένων με τρόπο κατανοητό από τις μηχανές, τη διευκόλυνση της σύνθετης επεξεργασίας δεδομένων και την υποβοήθηση της σημασιολογικής αναζήτησης και συλλογιστικής. Η ιδιαίτερη σημασία τους έγκειται στο γεγονός ότι διευκολύνουν την ακριβή και αποτελεσματική ανάκτηση πληροφοριών και τη λήψη αποφάσεων συνδέοντας ποικίλες κατηγορίες πληροφοριών.

Η μηχανική μάθηση, ένα υποσύνολο της τεχνητής νοημοσύνης, εξαρτάται όλο και περισσότερο από δομημένες αναπαραστάσεις πληροφοριών, όπως οι γράφοι γνώσης, για να ενισχύσει την ικανότητά της να κατανοεί και να αναλύει πολύπλοκα δεδομένα. Διαδραματίζει κρίσιμο ρόλο στην ανάπτυξη και τη βελτίωση των γράφων γνώσης. Μέσω της χρήσης μεθόδων μηχανικής μάθησης, μπορούμε να αυτοματοποιήσουμε τη διαδικασία εξαγωγής, ταξινόμησης και πρόβλεψης συνδέσεων εντός των δεδομένων. Αυτό οδηγεί σε σημαντική βελτίωση της ακρίβειας και της επεκτασιμότητας της κατασκευής γράφων γνώσης. Η παρούσα εργασία διερευνά διάφορες προσεγγίσεις μηχανικής μάθησης για την αντιμετώπιση σημαντικών εμποδίων στον εμπλουτισμό γράφων γνώσης.

Τα νευρωνικά δίκτυα γράφων (GNN) έχουν ανελιχθεί σε ισχυρές μεθόδους για την εξαγωγή ουσιαστικών πληροφοριών και αναπαραστάσεων από γράφους γνώσης. Επιτρέπουν διάφορες εργασίες όπως η κατηγοριοποίηση οντοτήτων, η πρόβλεψη συνδέσμων και η αξιολόγηση σημασιολογικής ομοιότητας. Επιπλέον, ο συνδυασμός της μηχανικής μάθησης και των γράφων γνώσης επιτρέπει δραστηριότητες όπως η ανακάλυψη νέων πληροφοριών και η διεξαγωγή σημασιολογικών αναζητήσεων.

Τα προβλήματα κατηγοριοποίησης τριπλετών και πρόβλεψης σχέσεων αποτελούν κρίσιμα ζητήματα στους γράφους γνώσης και στη μηχανική μάθηση. Ο στόχος της κατηγοριοποίησης τριπλετών είναι να εξακριβωθεί η ακρίβεια ή η εγκυρότητα μιας συγκεκριμένης τριπλέτας (που αποτελείται από ένα υποκείμενο, ένα κατηγόρημα και ένα αντικείμενο) μέσα σε έναν γράφο γνώσης. Το έργο αυτό συνεπάγεται την ανάθεση ετικετών στις τριπλέτες ανάλογα με το αν είναι παρούσες ή όχι στον γράφο γνώσης, όπως "αληθής" ή "ψευδής". Έχει ζωτική σημασία για την επαλήθευση της συνοχής και της πληρότητας των γράφων γνώσης.

Αντίθετα, η πρόβλεψη σχέσεων ασχολείται με την πρόβλεψη συγκεκριμένης σύνδεσης (κατηγόρημα) που συνδέει δύο οντότητες (υποκείμενο και αντικείμενο) σε έναν γράφο γνώσης. Αυτό είναι κρίσιμο για εργασίες όπως η πρόβλεψη συνδέσεων και η συμπλήρωση βάσεων γνώσης, όπου η επιτυχής πρόβλεψη συνδέσεων βοηθά στη συμπλήρωση των συνδέσεων που λείπουν και στη βελτίωση της δομής του γράφου. Και οι δύο εργασίες χρησιμοποιούν μεθοδολογίες μηχανικής μάθησης, όπως τα νευρωνικά δίκτυα γράφων και τα μοντέλα διανυσματικών αναπαραστάσεων (embeddings), για την απόκτηση πληροφοριών σχετικά με τις οντότητες και τις συνδέσεις. Αυτό επιτρέπει την ακριβή κατηγοριοποίηση και πρόβλεψη εντός των γράφων γνώσης.

Ο συνδυασμός εργασιών ταξινόμησης τριπλετών και πρόβλεψης σχέσεων περιλαμβάνει την αξιοποίηση μοντέλων μηχανικής μάθησης για την επίτευξη ακριβών προβλέψεων, σε διαφόρους τομείς, σε γράφους γνώσης. Μια προσέγγιση είναι η χρήση μοντέλων ταξινόμησης τριπλετών που μπορούν να προβλέψουν την εγκυρότητα των τριπλετών σε όλες τις πιθανές σχέσεις, εκτελώντας έτσι έμμεσα την πρόβλεψη σχέσεων. Αυτή η μέθοδος περιλαμβάνει συνήθως τα ακόλουθα βήματα: πρώτα, κωδικοποίηση των οντοτήτων υποκειμένου και αντικειμένου μαζί με τις υποψήφιες σχέσεις σε ένα μοντέλο ικανό να παράγει μια πρόβλεψη πιθανότητας για κάθε σχέση. Δεύτερον, για κάθε τριπλέτα, το μοντέλο αξιολογεί την πιθανότητα η τριπλέτα να είναι έγκυρη κάτω από κάθε υποψήφια σχέση.

Τέλος, η σχέση που σχετίζεται με την υψηλότερη βαθμολογία πιθανότητας επιλέγεται ως η προβλεπόμενη σχέση για τη συγκεκριμένη τριπλέτα. Αυτή η προσέγγιση ενσωματώνει αποτελεσματικά τις εργασίες πρόβλεψης σχέσεων και κατηγοριοποίησης τριπλετών αξιοποιώντας την ικανότητα του μοντέλου ταξινόμησης τριπλετών να αξιολογεί την πιθανότητα τριπλετών κάτω από διαφορετικές σχέσεις, διευκολύνοντας έτσι την ακριβή πρόβλεψη σχέσεων μέσα σε γράφους γνώσης. Συνδυάζοντας αυτές τις εργασίες, οι ερευνητές μπορούν να ενισχύσουν την πληρότητα και την ακρίβεια των βάσεων γνώσης, υποστηρίζοντας εφαρμογές όπως η σημασιολογική αναζήτηση, τα συστήματα συστάσεων και η αυτοματοποιημένη συλλογιστική σε διάφορους τομείς.

Λέξεις κλειδιά: Μηχανική μάθηση, Γραφοί γνώσης, Βαθιά μάθηση, Εμπλουτισμός γράφων γνώσης, Νευρωνικά δίκτυα γράφων

Acknowledgements

This thesis not only showcases my personal efforts but also the collective support and guidance I have received throughout my academic journey. Above all, I would like to sincerely thank Prof. Nick Bassiliades, my immediate supervisor. Your unwavering support, insightful criticism, and unwavering encouragement have largely shaped my research path. From the first idea for this project until the last draft, your direction has been priceless.

Your outstanding expertise, patience, and attention to my study development, as demonstrated by Prof. Nick Bassiliades, have profoundly impacted me. Your door was always open, and I found excellent guidance in your willingness to share knowledge and provide helpful critique. Whether it was late at night or during your busy schedule, the time and effort you dedicated to our conversations demonstrated a level of dedication that exceeded expectations. Your faith in my talents gave me the courage to aim for perfection and challenge my own efforts.

Furthermore, I am extremely grateful for your coaching outside of this thesis's boundaries. Your focus on the value of intellectual inquiry, critical thinking, and academic integrity has really changed my attitude toward study and professional behavior. You have been a model in showing how to strike a balance between compassion, understanding, and demanding academic work. My personal and professional development under your direction is incalculable; I will take the knowledge gained with me all through my career. Under your guidance, the many meetings, brainstorming sessions, and group projects have enhanced my education and built a community that has made this road intriguing and intellectually stimulating.

To my family and friends, I appreciate your consistent support and empathy during this trying time. Your encouragement kept me motivated and provided the balance I needed to keep going.

Finally, I would like to thank the teaching members, administrative assistants, and other students who have helped me along the academic road. We value your support and help in many different ways.

Table of Contents

Abstract	iv
Περίληψη.....	vi
Acknowledgements	xi
List of Figures.....	xvii
List of Tables.....	xx
1. Introduction.....	1
1.1 Thesis Structure.....	2
2. Machine Learning	5
2.1 Basic concepts of Machine Learning	5
2.1.1 Definition and Overview.....	5
2.1.2 Why to use machine learning?	6
2.1.3 Types of Machine Learning.....	7
2.1.4 Types of data	8
2.1.5 Feature Engineering	9
2.1.6 Overfitting and underfitting	10
2.2 Machine learning algorithms.....	11
2.2.1 Algorithms for Supervised Learning	12
2.2.1.1 Linear Regression.....	12
2.2.1.2 Logistic Regression	13
2.2.1.3 Random Forest	14
2.2.2 Algorithms for Unsupervised Learning.....	15
2.2.2.1 PCA.....	15
2.2.2.2 K-means	16
2.2.3 Deep Learning.....	17
2.2.3.1 Neural Networks Basics	18

2.2.3.2	Deep Neural Networks	19
2.2.3.3	Convolutional Neural Networks	19
2.2.3.4	Recurrent Neural Networks.....	21
2.2.3.5	Training Deep Models.....	22
2.2.3.6	Generative Models	23
2.2.3.7	Graph Convolution Networks	25
2.2.3.8	Graph Attention Networks	25
2.3	Model Training and Evaluation	27
2.3.1	Data Cleaning and Preprocessing	27
2.3.2	Data Splitting for Model Training and Evaluation	29
2.3.3	Hyperparameters in Machine Learning.....	30
2.3.4	Evaluation Metrics.....	32
3.	Semantic Web and Knowledge Graphs	37
3.1	Resource Description Framework (RDF)	38
3.1.1	Definition and Conceptual Framework	38
3.1.2	Core Components of RDF	38
3.1.3	RDF Syntax and Serialization Formats	39
3.1.4	RDF Schema (RDFS)	42
3.1.5	SPARQL	43
3.2	OWL (Web Ontology Language)	45
3.2.1	Definition and Purpose of OWL.....	46
3.2.2	Classes in OWL.....	47
3.2.3	Properties in OWL.....	49
3.2.4	Reasoning in OWL.....	51
3.2.5	OWL Reasoners and Inference Types.....	52
3.2.6	OWL API.....	53
3.3	Knowledge Graphs.....	54
3.3.1	Definition and Conceptual Framework	55
3.3.2	Historical Evolution of Knowledge Graphs	56
3.3.3	Key Components of Knowledge Graphs.....	57
3.3.4	Data Integration for Knowledge Graph	58
3.3.5	Knowledge Extraction Techniques	62
3.3.6	Representing Knowledge Graphs	63
3.3.7	Domain-Specific Knowledge Graphs	65

4.	Problem – Dataset – Tools.....	67
4.1	Description of the problem	67
4.1.1	Relation Prediction	67
4.1.2	Triple classification	69
4.1.3	Relation prediction through triple classification	71
4.1.4	Advantages of Triple Classification.....	72
4.2	Tools used.....	73
4.2.1	Python.....	73
4.2.2	RDFlib.....	74
4.2.3	GraphDB	75
4.2.4	TensorFlow	77
4.2.5	Protégé	78
4.3	Dataset	79
4.3.1	Statistics.....	83
5.	Description of the implementation and the experiments	91
5.1	Pipeline	91
5.2	Preprocessing	92
5.3	Evaluation	97
5.4	Models.....	102
5.4.1	Group of Experiments	103
5.4.2	Individual Experiments	129
6.	Results of the experiments.....	133
6.1	Discussion	152
7.	Conclusions and Future Work	161
7.1	Conclusions.....	161
7.2	Future work	162
8.	References.....	165

A. Detailed Experiments Results	175
I. Group 1	175
II. Group 2	183
III. Group 3	190
IV. Group 4	197
V. Group 5	207
VI. Group 6	214
VII. Group 7	223
VIII. Group 8	231
IX. Group 9	240
X. Group 10	249
XI. Experiment 1.....	258
XII. Experiment 2.....	259
XIII. Experiment 3.....	261
XIV. Experiment 4.....	263
XV. Experiment 5.....	264
XVI. Experiment 6.....	266
XVII. Experiment 7.....	267
XVIII. Experiment 8.....	269
XIX. Experiment 9.....	271
XX. Experiment 10.....	272
XXI. Experiment 11.....	274
XXII. Experiment 12.....	276
XXIII. Experiment 13.....	278
XXIV. Experiment 14.....	279

List of Figures

<i>Figure 2.1: Data produced in a minute on the internet in 2023 [9]</i>	6
<i>Figure 2.2 Underfitting, good fitting and Overfitting [19]</i>	11
<i>Figure 2.3: Fundamental machine learning algorithms [21]</i>	12
<i>Figure 2.4: Illustration of the placement of deep learning with respect to machine learning and artificial intelligence [28]</i>	18
<i>Figure 2.5: Training process in Deep Learning algorithms [33]</i>	23
<i>Figure 2.6: Machine Learning vs. Deep Learning training features [34]</i>	23
<i>Figure 2.7: Missing Value Imputation [47]</i>	28
<i>Figure 2.8: Outlier Detection and Treatment example [48]</i>	28
<i>Figure 2.9: Encoding Categorical Variable example [49]</i>	29
<i>Figure 2.10: Train, test and validation split [51]</i>	30
<i>Figure 2.11: Significance of Hyperparameters [54]</i>	32
<i>Figure 2.12: Confusion matrix [60]</i>	33
<i>Figure 2.13: ROC Curve [61]</i>	34
<i>Figure 2.14: Metrics for Evaluating Machine Learning Models [68]</i>	36
<i>Figure 3.1: The Semantic Web language layer cake [71]</i>	37
<i>Figure 3.2: RDF/XML example</i>	39
<i>Figure 3.3: Example Turtle Syntax</i>	40
<i>Figure 3.4: Example JSON-LD</i>	41
<i>Figure 3.5: Example N-Triples</i>	41
<i>Figure 3.6: N-Quad example</i>	42
<i>Figure 3.7: RDFS Example 1</i>	43
<i>Figure 3.8: RDFS Example 2</i>	43
<i>Figure 3.9: SPARQL query example 1</i>	45
<i>Figure 3.10: PARQL query example 2</i>	45
<i>Figure 3.11: Example of Class Hierarchy in OWL</i>	47
<i>Figure 3.12: Example of necessary & sufficient restriction</i>	47
<i>Figure 3.13: Example of Enumerated Class in OWL</i>	48
<i>Figure 3.14: Example of Intersection and Union of Classes in OWL</i>	48
<i>Figure 3.15: Example of Disjointness of classes in OWL</i>	49
<i>Figure 3.16: Example of Object Property in OWL</i>	50
<i>Figure 3.17: Example of Data Property in OWL</i>	50
<i>Figure 3.18: Example of Inverse Object Property in OWL</i>	50
<i>Figure 3.19: Example of Functional and Inverse Functional Properties in OWL</i>	51
<i>Figure 3.20: JSON-like Property Graph representation</i>	65

<i>Figure 4.1: The link prediction problem [132].....</i>	69
<i>Figure 4.2: GraphDB home page [141].....</i>	77
<i>Figure 4.3: The GlossaryTerm class</i>	80
<i>Figure 4.4: The Content class.....</i>	80
<i>Figure 4.5: The Reference class.....</i>	81
<i>Figure 4.6: The Classification class</i>	81
<i>Figure 4.7: SPARQL Query 1.....</i>	84
<i>Figure 4.8: Results of SPARQL Query 1</i>	84
<i>Figure 4.9: SPARQL Query 2.....</i>	84
<i>Figure 4.10: Results of SPARQL Query 2</i>	84
<i>Figure 4.11: SPARQL Query 3.....</i>	84
<i>Figure 4.12: Results of SPARQL Query 3</i>	85
<i>Figure 4.13: SPARQL Query 4.....</i>	85
<i>Figure 4.14: Results of SPARQL Query 4</i>	85
<i>Figure 4.15: SPARQL Query 5.....</i>	86
<i>Figure 4.16: Results of SPARQL Query 5 (top 15 count).....</i>	86
<i>Figure 4.17: Results of SPARQL Query 5 (lowest 15 count).....</i>	87
<i>Figure 4.18: SPARQL Query 6.....</i>	87
<i>Figure 4.19: Results of SPARQL Query 6</i>	88
<i>Figure 4.20: SPARQL Query 7.....</i>	88
<i>Figure 4.21: Results of SPARQL Query 7</i>	88
<i>Figure 4.22: SPARQL Query 8.....</i>	89
<i>Figure 4.23: Results of SPARQL Query 8</i>	89
<i>Figure 4.24: SPARQL Query 9.....</i>	90
<i>Figure 4.25: Results of SPARQL Query 9</i>	90
<i>Figure 5.1: Pipeline followed</i>	91
<i>Figure 5.2: Preprocessing function</i>	92
<i>Figure 5.3: Production of the node pairs</i>	100
<i>Figure 5.4: Relation prediction process</i>	101
<i>Figure 5.5: Clustering evaluation process.....</i>	101
<i>Figure 5.6: Dense architecture.....</i>	103
<i>Figure 5.7: Conv1D architecture</i>	104
<i>Figure 5.8: TransE Model.....</i>	107
<i>Figure 5.9: TransH Model</i>	108
<i>Figure 5.10: RotatE Model.....</i>	108
<i>Figure 5.11: HolE Model</i>	109
<i>Figure 5.12: DistMult Model.....</i>	109
<i>Figure 5.13: ComplEx Model.....</i>	111
<i>Figure 5.14: GCNLayer</i>	116

<i>Figure 5.15: TransE Model for Group 4.....</i>	117
<i>Figure 5.16: GATLayer</i>	119
<i>Figure 5.17: MarginRankingLoss function</i>	120
<i>Figure 5.18: Improved GCNLayer.....</i>	127
<i>Figure 5.19: Attention Layer for experiment 13</i>	132

List of Tables

<i>Table 4.1: Descriptions of the object properties</i>	82
<i>Table 4.2: Descriptions of the datatype properties</i>	82
<i>Table 5.1: The five randomly selected triples</i>	98
<i>Table 5.2: Parameters for the TransE Model</i>	106
<i>Table 5.3: Parameters for the TransE model in Group 7</i>	124
<i>Table 6.1: Abbreviations for columns</i>	133
<i>Table 6.2: Metrics for threshold 0.5.....</i>	134
<i>Table 6.3: Metrics for threshold 0.6.....</i>	134
<i>Table 6.4: Metrics for threshold 0.7.....</i>	134
<i>Table 6.5: Training times and epochs required for each model.....</i>	135
<i>Table 6.6: Metrics for threshold 0.5.....</i>	135
<i>Table 6.7: Metrics for threshold 0.6.....</i>	135
<i>Table 6.8: Metrics for threshold 0.7.....</i>	136
<i>Table 6.9: Training times and epochs required for each model.....</i>	136
<i>Table 6.10: Metrics for threshold 0.5</i>	136
<i>Table 6.11: Metrics for threshold 0.6</i>	137
<i>Table 6.12: Metrics for threshold 0.7</i>	137
<i>Table 6.13: Training times and epochs required for each model.....</i>	137
<i>Table 6.14: Metrics for threshold 0.5</i>	138
<i>Table 6.15: Metrics for threshold 0.6</i>	138
<i>Table 6.16: Metrics for threshold 0.7</i>	138
<i>Table 6.17: Training times and epochs required for each model.....</i>	138
<i>Table 6.18: Metrics for threshold 0.5</i>	139
<i>Table 6.19: Metrics for threshold 0.6</i>	139
<i>Table 6.20: Metrics for threshold 0.7</i>	139
<i>Table 6.21: Training times and epochs required for each model.....</i>	140
<i>Table 6.22: Metrics for threshold 0.5</i>	140
<i>Table 6.23: Metrics for threshold 0.6</i>	140
<i>Table 6.24: Metrics for threshold 0.7</i>	141
<i>Table 6.25: Training times and epochs required for each model.....</i>	141
<i>Table 6.26: Metrics for threshold 0.5</i>	141
<i>Table 6.27: Metrics for threshold 0.6</i>	142
<i>Table 6.28: Metrics for threshold 0.7</i>	142
<i>Table 6.29: Training times and epochs required for each model.....</i>	142
<i>Table 6.30: Metrics for threshold 0.5</i>	143

<i>Table 6.31: Metrics for threshold 0.6</i>	143
<i>Table 6.32: Metrics for threshold 0.7</i>	143
<i>Table 6.33: Training times and epochs required for each model.....</i>	144
<i>Table 6.34: Metrics for threshold 0.5</i>	144
<i>Table 6.35: Metrics for threshold 0.6</i>	144
<i>Table 6.36: Metrics for threshold 0.7</i>	145
<i>Table 6.37: Training times and epochs required for each model.....</i>	145
<i>Table 6.38: Metrics for threshold 0.5</i>	145
<i>Table 6.39: Metrics for threshold 0.6</i>	146
<i>Table 6.40: Metrics for threshold 0.7</i>	146
<i>Table 6.41: Training times and epochs required for each model.....</i>	146
<i>Table 6.42: Metrics for different thresholds</i>	146
<i>Table 6.43: Training time and epoch required for the model.....</i>	147
<i>Table 6.44: Metrics for different thresholds</i>	147
<i>Table 6.45: Training time and epoch required for the model</i>	147
<i>Table 6.46: Metrics for different thresholds</i>	147
<i>Table 6.47: Training time and epoch required for the model</i>	147
<i>Table 6.48: Metrics for different thresholds</i>	148
<i>Table 6.49: Training time and epoch required for the model</i>	148
<i>Table 6.50: Metrics for different thresholds</i>	148
<i>Table 6.51: Training time and epoch required for the model</i>	148
<i>Table 6.52: Metrics for different thresholds</i>	148
<i>Table 6.53: Training time and epoch required for the model</i>	149
<i>Table 6.54: Metrics for different thresholds</i>	149
<i>Table 6.55: Training time and epoch required for the model</i>	149
<i>Table 6.56: Metrics for different thresholds</i>	149
<i>Table 6.57: Training time and epoch required for the model</i>	149
<i>Table 6.58: Metrics for different thresholds</i>	150
<i>Table 6.59: Training time and epoch required for the model</i>	150
<i>Table 6.60: Metrics for different thresholds</i>	150
<i>Table 6.61: Training time and epoch required for the model</i>	150
<i>Table 6.62: Metrics for different thresholds</i>	151
<i>Table 6.63: Training time and epoch required for the model</i>	151
<i>Table 6.64: Metrics for different thresholds</i>	151
<i>Table 6.65: Training time and epoch required for the model</i>	151
<i>Table 6.66: Metrics for different thresholds</i>	151
<i>Table 6.67: Training time and epoch required for the model</i>	152
<i>Table 6.68: Metrics for different thresholds</i>	152
<i>Table 6.69: Training time and epoch required for the model</i>	152

Chapter 1

Introduction

Two essential technologies that have profoundly affected several fields, including artificial intelligence, data science, and the semantic web, are machine learning (ML) and knowledge graphs (KGs). Within artificial intelligence, machine learning is the study of statistical models and algorithms that, by learning from data, let computers complete tasks without explicit directions. Offering uses ranging from predictive analytics to natural language processing and computer vision, it has evolved into a necessary tool in data-driven decision-making procedures.

Machine learning involves a range of advanced approaches, including neural networks, deep learning, reinforcement learning, and ensemble methods, that surpass traditional statistical models. These approaches enable machine learning to address intricate problems like image identification, natural language processing, and game playing, which require advanced pattern detection and decision-making skills.

Conversely, knowledge graphs show a disciplined approach to arranging data. They comprise edges (relationships) and nodes (entities), creating a graph-based picture of knowledge. This framework makes it possible to effectively store, retrieve, and manipulate linked data, therefore enabling a better knowledge of the connections among many bits of data. Semantic search heavily utilizes recommendation algorithms, information extraction, and knowledge graphs, all of which provide a strong foundation for knowledge management and reasoning.

Machine learning and knowledge graphs working together offer a great way to improve the powers of both disciplines. Combining ML and KGs helps to extract richer and more contextual insights from data. Using the structured information in knowledge graphs, machine learning techniques may increase interpretability and prediction accuracy. Knowledge graphs, on the other hand, benefit from machine learning methods to automatically extract, enhance, and maintain their material. By combining the predictive ability of machine learning with the structured knowledge representation of graphs, this synergy has the potential to progress many applications, including customized recommendations, question-answering systems, and sophisticated data analytics.

In the current thesis, the target is to enrich Eurostat's knowledge graph [1]. Located in Luxembourg, Eurostat is the statistics agency of the European Union tasked with delivering high-quality statistical data to support cross-country and regionally based comparisons [2]. Comprising a key role as a component of the European Commission, Eurostat's goal is to offer objective, consistent data that supports the decision-making processes of governments, companies, and the general public of the EU. It gathers, compiles, and harmonizes data from national statistics institutes across EU members and other participating countries, ensuring uniformity and comparability. Covering a wide spectrum of statistical subjects, including economics, demography, environment, and social affairs, Eurostat supports the creation and assessment of policies meant to promote economic growth, social cohesion, and sustainable development all throughout Europe.

The dataset we use contains 307.444 explicit triples, 43 distinct relations, and 171.050 distinct entities, with 59.735 of them as subjects and 150.588 as objects. We used the entire dataset for all the experiments. We approach the enrichment of the knowledge graph using models specialized for the triple classification task. Triple classification models are better than simple relation prediction models because they understand relationships in knowledge graphs more fully and in greater detail. These models look at the subject, predicate, and object structure of the data. While simple relation prediction models only look at whether there is a binary relationship between two entities, triple classification models do more. For example, they figure out the exact role of each entity in a relationship, separate similar relationships, and add contextual information that makes the data more semantically rich. This enables triple classification models to detect more complex patterns, support reasoning tasks, and improve the accuracy of entity linking and attribute extraction.

Consequently, these models are more appropriate for sophisticated tasks like semantic search, knowledge base completion, and natural language comprehension. They provide more accurate and meaningful predictions compared to simpler models. We categorize the experiments into 10 groups based on the embedding style, model architecture, and other components used. We have a total of 14 individual experiments that explore various methodologies and components, mostly focusing on certain models within the groupings. During the experiments, we develop, present, and recommend deep learning models that successfully accomplish the triple classification challenge. Next, we assess these models with specific evaluation metrics and plots, utilizing them to forecast a correlation between distinct nodes not directly connected within the network. In the triple, we select the relationship with the highest probability.

1.1 Thesis Structure

Each of the seven chapters in the thesis serves a distinct purpose. [Chapter 2](#) sets the theoretical background for machine and deep learning topics. It investigates deep learning and machine learning approaches. It addresses unsupervised learning techniques like PCA and K-Means clustering as well as supervised learning methods including random forest, logistic regression, and linear regression. Deep learning subjects include neural network basics, deep neural networks, convolutional neural networks, recurrent neural networks, training techniques, generative models, Graph Convolution Networks, and Graph Attention Network. This chapter also covers model training and assessment, which includes data cleaning, preprocessing, data splitting, hyperparameter tweaking, and evaluation criteria.

This thesis fully explores the semantic web and knowledge graphs in [Chapter 3](#). The chapter begins with a thorough examination of the Resource Description Framework (RDF), a tool for organizing data on the web through subject-predicate-object triples and various serialization formats. It then presents the Web Ontology Language (OWL), outlining its function in constructing ontologies and supporting advanced reasoning capacity for inferring information. The chapter also delves into knowledge graphs, highlighting their development, primary components, integration strategies for various data sources, and methods for extracting and displaying domain-specific information. Emphasizing their critical relevance in current data management and use paradigms, it ends by emphasizing the pragmatic uses of knowledge graphs in improving information retrieval, artificial intelligence systems, and domain-specific applications.

[Chapter 4](#) explores the synergy between problem, dataset, and tools within the research framework. It starts by looking at the problem statement and focusing on relation prediction, triple classification, and how they can be used together to predict relations through classified triples. It stresses the benefits of this approach in representing knowledge and drawing conclusions. The chapter then introduces the tools utilized in the research, including Python for its versatility, RDFLib for RDF data handling, GraphDB for knowledge graph storage, TensorFlow for deep learning, and Protégé for ontology management. It also discusses the dataset used in the study, providing key statistics to contextualize the data's role in machine learning and semantic technology research. Overall, Chapter 4 establishes a foundation for the methodology and experimental framework employed in the thesis.

[Chapter 5](#) details the implementation and experimental processes undertaken in the research. The chapter commences with a comprehensive overview of the pipeline, detailing the sequential steps from data processing to the execution of the experiment. The chapter then discusses preprocessing techniques applied to the data to ensure it is suitable for analysis. It proceeds to evaluate the performance of models and experiments, outlining the methodologies used for assessment. Additionally, various models are examined, including both group-level experiments and specific individual experiments conducted to validate hypotheses and analyze results. In essence, Chapter 5 provides a comprehensive exploration of the practical implementation and empirical evaluation of the research methodology.

[Chapter 6](#) presents the experiment results and conclusions drawn from the approaches used. The next conversation critically examines these findings and offers an understanding of their relevance within the larger framework of the study goals. Turning now to [Chapter 7](#), it summarizes important discoveries and their ramifications for the study area, thereby providing conclusions derived from the facts and arguments. Chapter 7 also lists possible future paths for research and development based on the gaps and possibilities discovered, thus guiding our efforts. Chapters 6 and 7 together provide a thorough assessment of the study results and ideas for further developments in the subject.

Chapter 2

Machine Learning

Within artificial intelligence (AI), machine learning (ML) is a subfield that lets computers learn from experience and increase their performance without explicit programming. Fundamentally, machine learning is the development of models and algorithms enabling computers to detect trends, provide predictions, and adapt to new data [3].

2.1 Basic concepts of Machine Learning

The fundamental ideas form the cornerstone of machine learning and are essential for understanding and effectively using machine learning methods. Fundamentally important ideas include supervised learning, unsupervised learning, reinforcement learning, model assessment, feature engineering, and overfitting. These ideas provide a structure for using machine learning methods to investigate, evaluate, and solve practical problems [4].

2.1.1 Definition and Overview

Machine learning is the ability of systems to independently detect patterns, make judgments, and improve their performance through data use. Its origins can be traced back to the midway point of the twentieth century. Proposing the idea of intelligent computers able to replicate human learning, Alan Turing and Marvin Minsky laid the foundation for machine learning in the 1940s and 1950s. However, the first learning algorithms did not lead to the subject really flourishing until the 1950s and 1960s [3], [5].

In 1959, Arthur Samuel used the term "machine learning" to characterize computers' ability to choose information from experience without explicit programming. Over the following several decades, researchers investigated substitute methods for machine learning, including symbolic learning and rule-based systems.

Machine learning attracted a lot of attention in the 1980s and 1990s because of developments in computer power and the availability of vast databases. Decision trees, support vector machines, and neural networks are among the basic algorithms that scientists have created and polished, and many modern machine learning approaches draw upon them.

Machine learning has seen an unheard-of explosion in the twenty-first century due to the convergence of massive data [6], better algorithms, and strong processing capability. The internet's introduction and the abundance of digital data have created a particularly favorable environment for the development and use of machine learning systems. Image and audio identification, natural language processing, and many other complex tasks have advanced thanks in significant part to the development of deep learning, a specialist field of machine learning focused on neural networks with several layers.

Nowadays, many fields, including healthcare, finance, marketing, and autonomous systems, critically depend on machine learning. The field is always developing through ongoing research on creative

algorithms, ethical issues, and the formulation of solutions for pragmatic challenges. We anticipate that the increasing frequency of machine learning will extend its impact on technology and society, changing our interactions with and advantages from intelligent technologies.

2.1.2 Why to use machine learning?

In recent years, the utilization of machine learning has become increasingly prevalent across various disciplines, offering unprecedented opportunities for automation, optimization, and insight generation [7].

The Era of Big Data

As shown in Figure 2.1, the spread of digital technology, the rise of linked systems, and the internet have all dramatically expanded the amount, speed, and diversity of data created across many sectors. Often referred to as "big data," the wealth of information consists of both organized and unstructured data sources like sensor data, social media comments, transaction records, and multimedia resources. Because of its enormous volume and complex character, conventional statistical approaches have considerable difficulty digesting, interpreting, and drawing insights from big data. High dimensionality, sparsity, and noise distinguish datasets for which such techniques struggle [8].



Figure 2.1: Data produced in a minute on the internet in 2023 [9]

Limitations of Traditional Statistical Methods

The volume and complexity of large datasets overwhelm conventional statistical methods based on inferential and descriptive statistics. Though in certain cases, linear models, hypothesis testing, and conventional regression approaches might be helpful, they usually fall short of capturing the intricate interrelationships and nonlinear patterns inherent in large datasets. Traditional methods rely on strict assumptions about the distribution, independence, and linearity of the data. Real-world changes and the addition of new data sources may render these assumptions useless [10].

The Promise of Machine Learning

A paradigm shift in data analysis, machine learning offers a scalable and adaptable framework that helps to extract insights and create predictions from large datasets. Unlike traditional statistical methods, which depend on predetermined models and explicit assumptions, machine learning algorithms immediately discover patterns and relationships from the data. This feature helps them to fit complex and dynamic datasets. Mappings between input characteristics and output labels are capable of acquisition by support vector machines, decision trees, and supervised learning algorithms. This enables these methods to support classification and regression projects [7].

Deep Learning

Deep learning, inspired by the design and behavior of the human brain, has emerged as a paradigm-shifting field in large dataset management. Comprising several layers of linked nodes, deep neural networks shine in producing hierarchical data representations. This helps them to identify intricate patterns and traits that might elude more conventional approaches. Convolutional neural networks (CNNs) cause a paradigm shift in computer vision and image recognition, and recurrent neural networks (RNNs) are excellent at processing sequential data, including time series and natural language [11].

Addressing Complexity and Uncertainty

Machine learning techniques' resilient systems help to address the complexity and unpredictability inherent in large volumes. Using the collective knowledge of several models, ensemble techniques—including random forests and gradient boosting—help to improve forecast accuracy and lower the overfitting risk. In the face of uncertainty, probabilistic models—such as Bayesian networks and Gaussian processes—help to enable moral decision-making by quantifying it. Reward-driven learning yields reinforcement learning. Giving autonomous agents interaction with their environment helps them identify the optimal rules more easily [12].

2.1.3 Types of Machine Learning

Generally, machine learning is classified into two primary categories. The objective of predictive or supervised learning is to discover a relationship between inputs x and outputs y from a set of labeled input-output pairs $D = \{(x_i, y_i)\}_{i=1}^N$. In this case, D represents the training set, and N denotes the quantity of training examples [13], [14].

Each training input x_i is, in its most basic form, a D – dimensional vector of numbers, representing, for example, the height and weight of an individual. These are known as covariates, attributes, or features. Nevertheless, x_i may represent any complex structured entity, including but not limited to a time series, molecular shape, graph, or image.

In the same way, while the output or response variable can theoretically take on any form, the majority of methods presuppose that y_i is either a real-valued scalar (such as income level) or a categorical or nominal variable from a finite set, $y_i \in \{1, \dots, C\}$. The problem is referred to as classification or pattern recognition when y_i is categorical, and regression when y_i is a real-valued variable. Ordinal regression is a further variation in which the label space Y, such as grades A–F, has a natural ordering.

Unsupervised learning constitutes the second primary category of machine learning. The provided parameters are the inputs $D = \{x_i\}_{i=1}^N$. Identifying "interesting patterns" within the data is the objective. Sometimes, this is referred to as "knowledge discovery." Unlike supervised learning, which allows for a comparison between the predicted and observed values of y for a given x , this problem lacks a clear definition due to the lack of guidance on specific patterns to search for and an apparent error metric to employ.

A third category of machine learning, reinforcement learning, entails using iterative experimentation to gain knowledge. The system consists of an interactive agent that explores a specified environment with the goal of maximizing the total rewards it receives. Based on its actions, the system provides the agent with feedback in the form of incentives or punishments. This process allows the agent to gradually improve its approach. This approach is particularly notable for its application in intricate decision-making situations, such as game playing, robotic control, and autonomous systems. Reinforcement learning models create ideal policies that strike a balance between short-term rewards and long-term objectives, frequently under the direction of exploration-exploitation strategies.

2.1.4 Types of data

Data is the fuel behind the development and enhancement of prediction models in many diverse disciplines at the core of machine learning. In machine learning, both researchers and practitioners rely on a deep awareness of the many types of data used in their discipline [6], [7].

Structured Data

Structured data is the most common and well-known type of data in machine learning applications. Columns exhibit qualities or attributes according to a preset framework; rows represent specific data instances. Structured data comes in spreadsheets, CSV files, and databases. Classification and regression are common applications of machine learning for structured data; the aim is to utilize input properties to project a target variable.

Unstructured Data

The absence of a predetermined format or arrangement in unstructured data makes it different from structured data and has its own set of challenges. This can make processing and analysis more complex and demanding. Within the realm of data modalities, this classification encompasses various forms such as text, images, audio, and video. Among these are surveillance films, audio records, medical imaging, and social media entries. Unstructured data-specific machine learning methods like computer vision, audio processing, and natural language processing (NLP) let one get significant insights from these many and abundant sources.

Semi-Structured Data

Semi-structured data is a subset of information between structured and unstructured data. Semi-structured data preserves some degree of order while still maintaining simplicity of presentation. Commonly utilized protocols such as HTML, XML (Extensible Markup Language), and JSON (JavaScript Object Notation) fit semi-structured data forms. Web-based applications primarily use semi-structured data due to its ability to facilitate flexible data storage and exchange. Techniques to govern semi-structured data are crucial in the domains of web crawling, information retrieval, and data integration.

Temporal Data

Temporal data—that is, observations or occurrences over a period of time—is information that is essentially time-dependent. Temporal data sources include financial transactions, event diaries, time series data, and event series data. Temporal data analysis presents special difficulties due to temporal patterns and dependencies, including but not limited to trends, seasonality, and anomalies. By customizing machine learning methods to examine temporal data—such as anomaly detection, sequential modeling, and time series forecasting—we may extract valuable insights for a wide range of uses, including predictive maintenance and financial projections.

2.1.5 Feature Engineering

Data analysis and machine learning both critically depend on feature engineering. To increase model performance and interpretability, it means choosing and developing additional features from raw data. Machine learning algorithms improve their learning capacity by extracting extensive information from unprocessed data. This work calls for understanding area knowledge, identifying important traits, and presenting data in a way that best maximizes the ability of the model to detect basic trends and relationships [14], [15], [16].

Feature engineering draws on several basic ideas. Relevance ensures that the characteristics contain important facts directly related to the prediction target. Unrelated or repetitive elements could add pointless information and lower the model's performance. Representation ensures that the encoded data accurately captures the problem area's basic framework. This might call for standardizing numerical features, translating category data into numerical forms, or efficiently handling missing values.

Complexity refers to the meticulous selection of features that are required to gather the necessary data without causing an overfit. While depicting non-linear interactions may require complex transformations, we typically prefer simple, clearly comprehensible properties. The goal of interpretability is to enhance the model's clarity and comprehensibility, enabling stakeholders to comprehend the fundamental elements that impact decisions and forecasts.

Feature engineering is the use of a wide range of techniques intended for various kinds of data, problem areas, and modeling methodologies. Several common approaches consist of:

- Usually, in order to enable its storage, transport, or processing in a specific way, encoding is the act of converting data from one form to another. Many times, machine learning techniques translate categorical data into numerical form. One-hot encoding, label encoding, and target encoding are among the often-used techniques for changing category properties.
- Standardizing and normalizing numerical values help to guarantee similar ranges and variations. Often used techniques for numerical data preparation include standard scaling, min-max scaling, and resilient scaling.
- Management data imputation provides numerous ways to handle missing values in the dataset: mean imputation, median imputation, and more intricate algorithms include predictive modeling or k-nearest neighbors (KNN).
- Techniques such as polynomial, logarithmic, and box-cox transformations capture nonlinear interactions and improve a model's performance, as well as feature transformation outcomes.

- Feature selection techniques help find the most relevant traits for a prediction assignment while removing pointless or repetitive features. Typical approaches include choosing features individually, removing traits recursively, and ranking features according to their importance.
- The process of creating new features based on existing ones in order to gather additional data or improve the model's predictive capacity is known as feature generation. These might be a mix of traits, unique area-specific changes, or the engineering construction of time-related properties.

Success in machine learning models across numerous fields and applications depends on effective feature engineering. Creating educational elements enables experts to accomplish the following tasks:

- Improve Model Performance: Skillful feature design helps machine learning algorithms identify complex patterns and relationships in the data, thereby improving prediction performance and enabling generalization capability.
- Improve Interpretability: By providing insights into the components that influence forecasts, feature engineering helps models become more comprehensible. Easy understanding and interpretation of the model's characteristics helps stakeholders trust its conclusions.
- Feature engineering helps to lower the dimensionality of the feature space by selecting relevant properties and removing unnecessary ones. Simplifying the model reduces computing complexity and lessens the overfitting risk.
- Activate Transfer Learning: This speeds up the development of new models and applications by transferring and reusing well-designed features from other machine learning tasks and domains.

2.1.6 Overfitting and underfitting

Overfitting and underfitting are two fundamental subjects in machine learning that relate to models' generalizing ability. They reflect different challenges, each having unique effects on a model's effectiveness in handling fresh, unseen input [17], [18].

When a machine learning model not only detects the basic trends in the training data but also learns about the noise and random fluctuations inherent in that data, it produces overfitting results. The model, in essence, becomes too complex and overfits the training data, reducing its ability to generalize to fresh, unseen data. This could cause the model to show poor performance on fresh, real-world data while showing high performance on the training set. Figure 2.2 demonstrates all three differentiations of a machine learning model fit.

Models that are too complicated, contain too many parameters relative to the volume of training data, or run too many training cycles often result in overfitting. Usually shown as a clear rise in accuracy when training the data, indications of overfitting often show a significant drop in performance when tested on another validation or test set. Many techniques for handling overfitting include regularization, dropout (in neural networks), and cross-validation. For example, regularizing methods penalize excessively complex models, thereby preventing overfitting to noise.

On the other hand, underfitting results from a model that is too basic to faithfully depict the underlying patterns seen in the training data. The model performs poorly not only on the training set but also

when dealing with unknown data due to its inability to comprehend the complexities of the data. Underfit models frequently exhibit low variance and significant bias.

Simple models, insufficient model training, or insufficient relevant data might all lead to underfitting. The reduced accuracy on the training and validation datasets indicates underfitting. One may consider employing more complex models, incorporating relevant traits, or increasing the training duration to help the model detect more subtle patterns and prevent underfitting.

The construction of a model that shows high generalization to fresh data depends on achieving the ideal balance between overfitting and underfitting. Usually referred to as the "bias-variance tradeoff," the balance between bias and variance is crucial. Models with high variance (overfitting) are too intricate, capturing noise in the training data, while models with high bias (underfitting) are too simple, failing to capture the underlying patterns in the data. The aim is to find the optimal balancing point that reduces both variance and bias, thereby producing a model with outstanding performance on both the training and test datasets.

Regularization techniques, adequate cross-validation, and attentively observing performance metrics on validation or test sets help to strike a compromise. The main objective is to create models with excellent generalizing capacity, thus minimizing the influence of noise and proving their capacity.

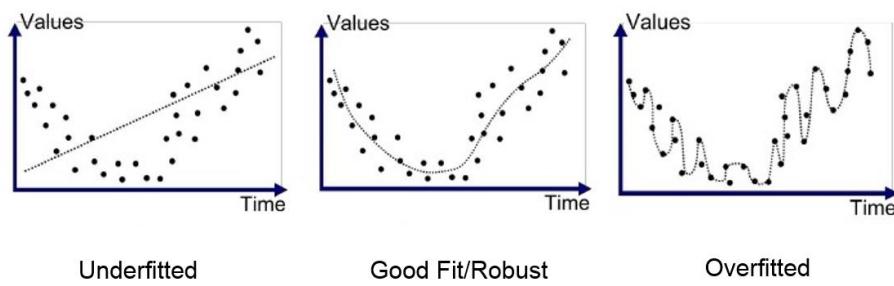


Figure 2.2 Underfitting, good fitting and Overfitting [19]

2.2 Machine learning algorithms

Machine learning algorithms allow computers to learn from data and make wise decisions. Predictive analytics and image recognition are only two of the many uses for which these techniques provide the basic foundation. These algorithms fall generally into three groups: supervised, unsupervised, and reinforcement learning. Figure 2.3 shows the main ideas for every group. As time goes on, they apply mathematical ideas to see trends, improve knowledge, and independently increase performance. They use linear regression for prognosis, k-means clustering for classification of similar data points, and deep neural networks for complex projects such as image and speech recognition. Achieving success in machine learning operations depends on deliberate choice and use of appropriate algorithms [20].

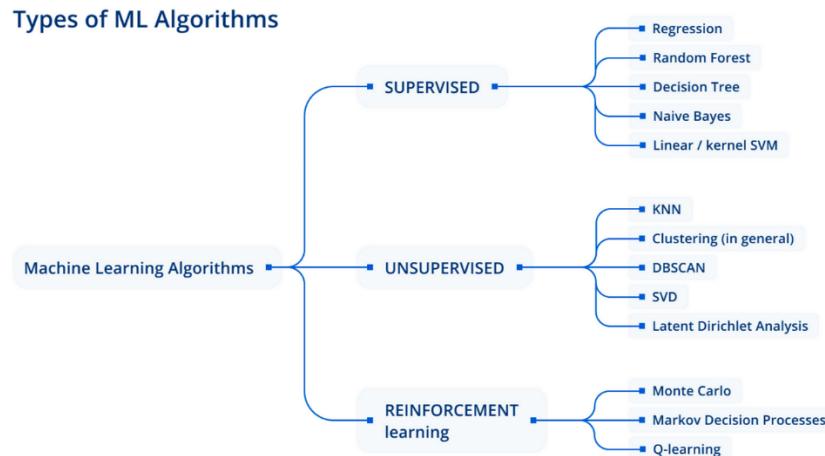


Figure 2.3: Fundamental machine learning algorithms [21]

2.2.1 Algorithms for Supervised Learning

In the field of machine learning, essential elements include algorithms for supervised learning that allow computers to forecast based on labeled input data. These methods, ranging from linear regression to neural networks to decision trees, form the foundation of predictive analytics. Examining their theoretical foundations, pragmatic applications, and performance across many datasets helps one to fully appreciate their usefulness in addressing practical challenges. Strong arguments and thorough investigation expose the advantages, drawbacks, and potential developments of these supervised learning methods, underscoring their indispensable contribution to the discipline of machine learning.

2.2.1.1 Linear Regression

Linear Regression is a basic and extensively used supervised learning technique. A range of applications favor this choice due to its simplicity and interpretability, particularly when anticipating a linear relationship between input variables and the target variable. The main goal of linear regression is to establish a mathematical model that represents the linear correlation between independent variables (features) and a dependent variable (target) by fitting a linear equation to the available data [22], [23].

The fundamental structure of a linear regression model may be represented as:

$$y = mx + b$$

In this regard, x denotes the variable unaffected by other elements, y the variable impacted by other elements, m the line's rate of change, and b the point of intersection of the line with the y-axis. Within the scope of multiple linear regression, which considers numerous independent variables, the equation becomes broader:

$$y = b_0 + b_1x_1 + b_2x_2 + \dots + b_nx_n$$

One term in the equation, b_0 represents the y-intercept. Coefficients, b_1, b_2, \dots, b_n , also show how each independent variable, x_1, x_2, \dots, x_n , shapes the dependent variable.

Finding the best suitable values for these coefficients is the main emphasis of the training process for a linear regression model, hence minimizing the difference between the projected and real values in the training data. Often, the Ordinary Least Squares (OLS) method—which seeks to reduce the sum of squared deviations between expected and actual outcomes—helps to optimize.

Many fields use linear regression. In finance, we may use it to forecast stock prices using historical data analysis; in healthcare, it helps us project patient recovery length using pertinent health parameters. The interpretability of the model is a major benefit, as it helps stakeholders understand how every independent variable affects the projected outcome.

Still, linear regression comes with limitations. The model depends on a linear correlation between variables, so failure to satisfy this assumption could affect its performance. Furthermore, this model is rather sensitive to extreme values; so, the presence of multicollinearity among the predictor variables may influence the coefficient estimate accuracy.

2.2.1.2 Logistic Regression

Particularly for tasks requiring binary classification, logistic regression is a widely utilized and flexible method. We employ logistic regression for classification problems rather than regression, despite its deceptive moniker. It is particularly appropriate in cases where the dependent variable is binary, therefore signaling two different categories. Underlying logistic regression is the basic idea of building a model to forecast the probability of a given event falling into a certain class [22], [24].

The logistic regression model converts the linear combination of input features into a range between 0 and 1 by using the logistic function, also known as the sigmoid function. We can formulate the mathematical basis of logistic regression as follows:

$$P(Y = 1) = \frac{1}{1 + e^{-(b_0 + b_1x_1 + b_2x_2 + \dots + b_nx_n)}}$$

The equation describes the probability of an occurrence belonging to class 1, denoted as $P(Y = 1)$. The base of the natural logarithm is represented by e , and the coefficients $b_0, b_1, b_2, \dots, b_n$ are related to the independent variables x_1, x_2, \dots, x_n . The logistic function guarantees that the result is a legitimate probability.

The process of training a logistic regression model entails optimizing the coefficients in order to maximize the probability of the observed data. For this objective, we frequently use conventional optimization methods like gradient descent.

The straightforwardness and ease of understanding of logistic regression make it particularly advantageous. The model's coefficients provide valuable information on the direction and magnitude of the association between each independent variable, as well as the logarithm of the probability of the expected occurrence. In situations where understanding the elements that impact the outcome is just as critical as generating precise forecasts, the ability to interpret the results is very beneficial.

Many fields use logistic regression. In the healthcare field, it forecasts the likelihood of a patient suffering from a specific medical condition by analyzing several clinical characteristics. In the field of marketing, it can predict the likelihood of a buyer purchasing a product, considering their demographic

data. The algorithm's ability to generate probabilities makes it extremely useful in situations where decision-makers require a comprehensive understanding of the level of confidence linked to forecasts.

Nevertheless, logistic regression does possess some constraints. It assumes a linear correlation between the independent variables and the logarithm of the probability of occurrence, which may not always be valid in complex real-world situations. Furthermore, logistic regression is susceptible to the influence of outliers.

2.2.1.3 Random Forest

Ensemble learning techniques have emerged as potent instruments in machine learning, with the goal of improving predictive performance by amalgamating numerous base models. Random Forest, one of these methods, has amassed considerable traction due to its resilience and adaptability when applied to a variety of datasets. This well-known ensemble learning algorithm achieves robust generalization by aggregating the predictions of a large number of decision trees. It can also efficiently reduce overfitting by utilizing a variety of models that generate uncorrelated errors [22], [23].

Decision trees are multifunctional models for machine learning that partition the feature space into hierarchical node structures, with each node representing a class label, feature, or decision rule. Splitting criteria like information gain or Gini impurity to recursively divide the feature space creates a tree structure that makes classification and regression tasks easier. Using the Gini measure, we can assign a value to a node in a decision tree that represents impurity or uncertainty. The mathematical expression to determine the Gini impurity for a node with samples and classes is as follows:

$$I_G(t) = 1 - \sum_{k=1}^K p(k|t)^2$$

In order to derive the final prediction for instance i , the Random Forest algorithm aggregates the predictions from all trees.

$$\hat{y}_i = \frac{1}{|T|} \sum_{t \in T} \hat{y}_i^t$$

where T denotes the set of decision trees in the Random Forest, and \hat{y}_i^t denotes the prediction of tree t , for instance i .

The Random Forest algorithm functions by executing a series of critical stages. Bootstrap sampling kicks off the process, extracting replacement random samples from the initial dataset to create numerous subsets that ensure diversity among the training instances. Each node of the decision tree evaluates a random subset of features for splitting, preventing any one feature from dominating the model and promoting diversity. Then, we use the bootstrapped samples and random feature subsets to make multiple decision trees. We train each one separately using splitting criteria to make sure that every node is as pure as possible. At the end of the process, we obtain the final output of the random forest by aggregating (for regression) or majority voting (for classification) the predictions from individual trees.

Random Forest offers several advantages, including resistance to overfitting, the capacity to handle high-dimensional datasets, integrated feature selection within the model, and interpretability through decision tree visualization. However, its hyperparameter sensitivity, its inability to interpret large

ensembles, and its computational complexity can all have an impact on its performance. Additionally, it performs less effectively on imbalanced datasets.

2.2.2 Algorithms for Unsupervised Learning

Designed to find latent patterns and structures in unlabeled data, algorithms for unsupervised learning are essential instruments in the area of machine learning. Fundamentally, these algorithms—k-means clustering, hierarchical clustering, and principal component analysis—are essential for data exploration and dimensionality reduction. One may gain a strong awareness of their capacity to detect inherent data features by analyzing their theoretical underpinnings, pragmatic implementations, and performance across many datasets.

2.2.2.1 PCA

A robust statistical tool, principal component analysis (PCA), lowers dimensionality and improves data presentation. It finds great use in many fields, including but not limited to image processing, finance, and bioinformatics [11], [13], [25].

PCA essentially aims to maintain much of the relevant variation in high-dimensional data while lowering its dimension. Denoted X , the dataset has n observations and p characteristics or variables. The intended result is principal components—a new collection of orthogonal variables meant to maximize data variance.

Denoted as PC_1 , the first principal component is a linear combination of the original variables:

$$PC_1 = a_{11}X_1 + a_{21}X_2 + \cdots + a_{p1}X_p,$$

where $a_{11}, a_{21}, \dots, a_{p1}$ are the loadings or coefficients associated with each variable. The loading coefficients are chosen such that the variance of PC_1 is maximized, subject to the constraint that $\sum_{j=1}^p a_{j1}^2 = 1$ to ensure unit length. Subsequent principal components, PC_2, PC_3, \dots, PC_k are computed similarly, with the additional constraint that they are orthogonal to the preceding components. This orthogonality ensures that each principal component captures a unique source of variance in the data.

The PCA computational process comprises the following phases:

- **Standardization:** In the dataset, apply a zero mean and one standard deviation. This phase essentially reduces the impact of variables with more variances on the PCA by making all variables on the same scale.
- **Covariance Matrix:** Find the covariance matrix of the standardized data. Since it offers a clear description of the pairwise covariances between variables, the covariance matrix is a necessary instrument for understanding the connections within the data.
- **Eigenvalue Decomposition:** Eigenvalue decomposition aids in obtaining the covariance matrix's eigenvectors and related eigenvalues. The eigenvectors indicate the orientations of maximum variance, while the eigenvalues indicate the degree of variation along the previously indicated dimensions.

- **Principal Component Selection:** Sort eigenvectors in decreasing order based on eigenvalues. Principal components are the eigenvectors with the largest eigenvalues, hence explaining variance.
- **Projection:** Project the original data onto the selected primary components to get a lower-dimensional picture of the data. This projection is formed by multiplying the conventional data matrix with the chosen eigenvectors' matrix.

PCA has a variety of useful advantages for data presentation and analysis:

- **Dimensionality Reduction:** PCA reduces high-dimensional data to a lower-dimensional space, thereby preserving much of the data's variability. This simplifies the implementation of further studies, including the detection of correlations, trends, clusters, and visualizations.
- **Noise Reduction:** By focusing on the main components covering the highest degree of variability, PCA is a rather effective technique for removing noise and superfluous data. Reducing noise helps make the results of the data analysis more comprehensible.
- **Feature Extraction:** PCA allows us to extract features wherein the principal components reflect fresh features derived from linear combinations of the original variables. Using the generated features as input for the next machine learning technique helps reduce the model's complexity.

2.2.2.2 K-means

K-means clustering, a fundamental method in unsupervised learning, divides data into discrete groups based on degree of similarity. From data mining to image segmentation to pattern recognition, its simplicity, effectiveness, and simplicity have established it as a fundamental component in many fields. The reduction of the within-cluster sum of squares (WCSS), often referred to as inertia, fundamentally guides K-means clustering. An example dataset $X = \{x_1, x_2, \dots, x_n\}$ searches to partition n data points in d-dimensional space into k clusters $C = \{C_1, C_2, \dots, C_k\}$, thereby minimizing the total intra-cluster variance. Mathematically, this is expressed as [13], [22]:

$$\underset{C}{\operatorname{argmin}} \sum_{i=1}^k \sum_{x \in C_i} \|x - \mu_i\|^2$$

where μ_i denotes the centroid of cluster C_i .

There are several iterative stages in the K-means algorithmic workflow. The algorithm commences by selecting k data points at random from the dataset to serve as initial centroids. The algorithm then designates each data point for the centroid closest to it, leading to the formation of k clusters. The process then revises the centroids by recalculating the cluster centroids using the mean of the coordinates associated with each cluster. The assignment-update procedure is iterated until the convergence criteria are satisfied, usually after a predetermined number of iterations or when the centroids no longer undergo substantial change.

By optimizing the total intra-cluster variance, K-means generates compact and well-separated clusters. Still, the algorithm may converge to local optima given its sensitivity to the starting point. Several

optimization methods (e.g., Lloyd's algorithm) and initialization strategies (e.g., K-means+) have been proposed to handle this issue.

K-means is particularly flexible, straightforward, and scalable. The method is suitable for managing large datasets because it exhibits linear complexity as the number of data points increases. Professionals in a variety of disciplines may access the algorithm because of its simplicity of implementation and understanding. K-means may also handle a broad spectrum of data types and be relevant for many different clustering projects.

Nevertheless, K-means is not without its constraints. Initialization is crucial, as the initial centroids significantly influence the cluster quality, potentially leading to suboptimal solutions. Furthermore, K-means' assumption of spherical and isotropic clusters might not hold true for all datasets. Because K-means are deterministic, they always converge to a local optimal. However, the initialization of the algorithm can alter the final result.

When implementing K-means clustering, there are numerous pragmatic factors to consider. It is critical to perform preprocessing operations, such as scale and normalization of features, to guarantee that every dimension makes an equal contribution to the distance calculation. We frequently employ domain expertise, silhouette analysis, or the elbow method to optimize hyperparameters, specifically the selection of k , which represents the number of clusters. External indices (such as purity and F-measure) and internal indices (such as silhouette coefficient and Davies-Bouldin index) may be utilized to assess the quality of clustering outcomes.

2.2.3 Deep Learning

Deep learning, a powerful subset of machine learning, as shown in Figure 2.4, utilizes artificial neural networks with multiple layers, commonly referred to as deep neural networks. Deep learning systems replicate the complex structure of the human brain by automatically learning patterns and representations from vast amounts of data. Establishing itself as a basic component in present artificial intelligence research and practical implementations, this novel approach has spurred major breakthroughs in numerous disciplines, including computer vision, natural language processing, and voice recognition. Deep neural networks can independently extract features and expose hidden patterns because of their hierarchical structure. This quality helps them reach advanced capabilities and unmatched performance in handling complex problems [26], [27].

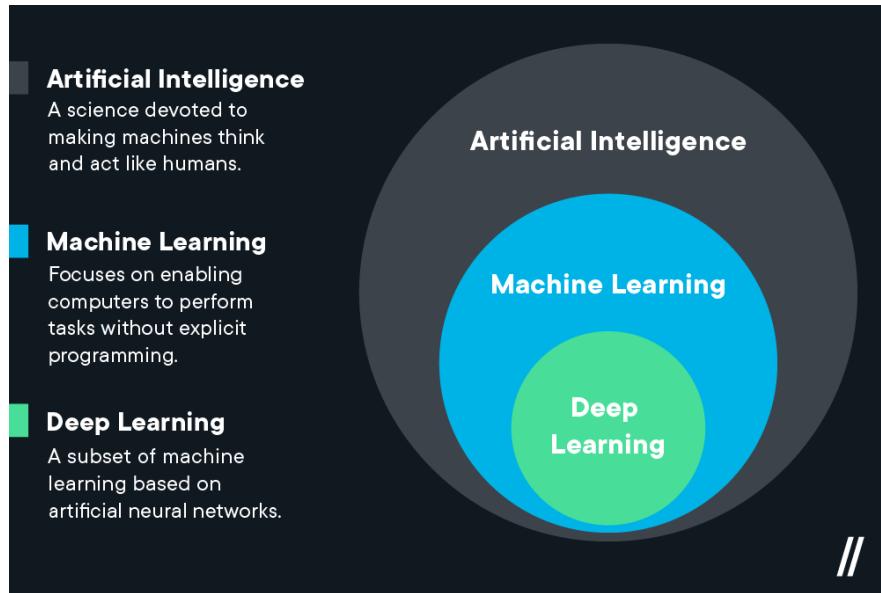


Figure 2.4: Illustration of the placement of deep learning with respect to machine learning and artificial intelligence [28]

2.2.3.1 Neural Networks Basics

The building blocks of deep learning are neural networks, which provide a versatile and adaptable framework for machine learning applications. The complex architecture and function of the human brain, which consists of linked nodes layered in layers, have a significant impact on neural networks. One must explore basic subjects like architecture, activation functions, and the training process via backpropagation if one is to understand the foundations of neural networks [29], [30].

Three main layer types define a neural network: the input layer, the hidden layer, and the output layer. With every node in the input layer representing a unique characteristic, this layer is in charge of getting the starting data. Between the input and output layers, hidden layers are essential for obtaining hierarchical data representations. The output layer creates the last predictions or classifications from the gathered patterns. The training process alters the weights, which define the degree of links and reflect the connections between nodes.

Activation functions provide neural networks with non-linearities, allowing them to learn about intricate interactions in the input. Two common activation functions are the rectified linear unit (ReLU), which lets only positive values pass through, and the sigmoid function, which compresses output values into the range between 0 and 1. The choice of activation function greatly influences the network's ability to recognize and show intricate data patterns.

Training a neural network involves iteratively using the backpropagation algorithm. During each iteration, the network generates predictions and calculates the discrepancy between these forecasts and the actual outcomes. Following that, the network adjusts the weights to minimize analogous errors and propagates the error in the other direction. Gradient descent or its variations often drive the optimization process, enabling the network to converge towards optimal weights and enhance its predictive capability.

These basic ideas form the foundation for more complex structures, such as convolutional neural networks (CNNs) for image processing and recurrent neural networks (RNNs) for sequential data.

Highly successful in tackling complex problems in many disciplines, neural networks have the capacity to adapt and automatically acquire hierarchical properties. As a result, deep learning has emerged.

2.2.3.2 Deep Neural Networks

Deep neural networks (DNNs) are an advanced version of standard neural networks. They have numerous hidden layers that allow them to learn complex representations of data. Because these networks are very deep, they can automatically pull out hierarchical features. This lets them capture complex patterns and relationships that would be challenging for structures with fewer layers. To comprehend deep neural networks, one must explore the advantages they provide, the difficulties they present, and the fundamental components that form their structure [11], [31].

The depth of neural networks is critical to their ability to extract complex characteristics and representations from data. Deep architectures are highly effective at capturing hierarchical abstractions, enabling them to identify progressively intricate patterns as the data passes through the levels. This skill has greatly contributed to their achievements in various fields, including picture recognition, natural language processing, and speech comprehension.

Nevertheless, the complexity of neural networks also presents difficulties. As neural networks increase in depth, problems such as disappearing gradients and exploding gradients can impede the training process. Vanishing gradients refer to the phenomenon where gradients decrease exponentially as they move backwards through the layers, which makes it difficult to update the weights in the earlier layers. On the other hand, bursting gradients refer to gradients that are overly large and can cause instability in the training process. To address these issues, it is necessary to employ meticulous weight initialization, normalization approaches, and specific topologies such as residual networks (ResNets) that feature skip links.

Multiple essential components contribute to the functionality of deep neural networks. Image processing applications frequently utilize convolutional layers, which use filters to autonomously acquire spatial hierarchies and local patterns. Recurrent layers, found in recurrent neural networks (RNNs), handle sequential data by remembering previous inputs to observe connections over time. Fully linked layers create connections between all nodes in the previous layer and every node in the current layer, allowing for extensive interactions.

We train a deep network using activation functions like rectified linear units (ReLUs) and optimization algorithms like stochastic gradient descent. ReLUs introduce non-linearities. By standardizing activations within each layer, batch normalization improves and accelerates training.

Deep neural networks have revolutionized the field of machine learning by achieving remarkable achievements in previously considered difficult problems. Their profoundness and ability to autonomously acquire knowledge have driven progress in artificial intelligence, opening up new possibilities for comprehending and deciphering intricate data.

2.2.3.3 Convolutional Neural Networks

Emerging as a potent class of deep learning architectures used in numerous computer vision, image recognition, and related domain applications are convolutional neural networks (CNNs). CNNs started out as simple structures that could copy how the visual cortex works in animals. Since then, they've

grown into more complex structures that can learn complex hierarchical representations right away from raw input [29], [31].

Convolutional processes fundamentally build CNNs and serve as the foundation for their hierarchical feature learning capacity. In convolution, we pass an input picture across the spatial dimensions after first subjecting it to a filter—also called a kernel. We compute the element-wise product of the overlapping input area and the filter. This technique generates a feature map by emphasizing certain patterns or traits clearly seen in the input.

The fundamental architectural components of CNNs consist of convolutional layers, pooling layers, and fully linked layers. Many filters in convolutional layers cooperate with the input to generate spatial characteristics at many levels of hierarchy. Pooling layers guarantee that the learned features are translationally invariant by lowering the spatial dimensions of the feature maps, hence simplifying computation complexity. To accomplish classification or regression, fully linked layers combine the obtained characteristics.

CNN architectures show somewhat different degrees of profundity and complexity. Among the notable examples are LeNet-5, AlexNet, VGGNet, GoogLeNet (Inception), ResNet, and—more recently—EfficientNet and Vision Transformers (ViTs). The designs differ in relation to the layer count, kernel width, activation mechanism, and connection pattern. Together, these elements affect the model's ability to obtain discriminative features and extend to unobserved data.

Modern CNNs may combine batch normalization, skip connections, dropout regularization, and attention algorithms. We use these methods to increase training stability, reduce overfitting, and enhance feature representation. Furthermore, developments in hardware accelerators such as GPUs and TPUs have made training CNN models with increasing complexity and depth simpler. This leads to significant performance improvements on a variety of real-world workloads and benchmark datasets.

CNNs often use gradient-based optimization techniques such as SGD, Adam, or RMSprop to eliminate as much as possible of a given kind of loss during training. For classification tasks, CNNs, for instance, use cross-entropy loss. CNNs change the weights and biases of the fully connected layers and convolutional filters in training to reduce the difference between their forecasts and real-world outcomes.

We frequently use a variety of methods to address problems such as overfitting and vanishing gradients. These methods include early stopping, dropout, weight regularization, data augmentation, and dropout. Moreover, a method that leverages task-specific datasets to improve pre-trained CNN models—transfer learning—has become very popular as a way to hasten model convergence and apply knowledge from big model datasets.

In many different fields, including object identification, semantic segmentation, picture production, and medical image analysis, CNNs have shown extraordinary performance. On ImageNet and other benchmark databases, CNNs have outperformed humans in picture classification. This indicates that individuals can pick up hierarchical visual data representation skills.

Furthermore, included in the actual applications of CNN-based approaches are augmented reality, driverless cars, security systems, and healthcare diagnostics. Real-time video streams, for example,

use CNNs for image augmentation using generative adversarial networks (GANs), medical picture segmentation for illness diagnosis, and object recognition and classification.

2.2.3.4 Recurrent Neural Networks

A subclass of artificial neural networks designed especially to capture temporal connections within sequential data, recurrent neural networks (RNNs). Feedforward neural networks process data as it passes through a succession of layers. RNNs, on the other hand, show changes in time through directed cycles built by connections. Their unique design allows them to effectively identify patterns and relationships as time goes on and handle varied-length input sequences [31], [32].

The basis for RNNs is the recurrent neuron, which preserves an internal state or memory across time steps. The recurrent neuron generates an output depending on both the current input and the past state at every time step. We can mathematically express the calculation inside an RNN cell as follows:

$$h_t = f(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$$

where:

- x_t is the input at time step t
- h_t is the output (hidden state) at time step t
- W_{hx} is the weight matrix for the input-to-hidden connections
- W_{hh} is the weight matrix for the recurrent connections
- b_h is the bias vector
- f is the activation function, typically a non-linear function such as the hyperbolic tangent (tanh) or the rectified linear unit (ReLU)

The network is able to preserve a memory of prior inputs by means of the recurrent connections $W_{hh}h_{t-1}$, therefore enabling it to acquire insights from past data and provide predictions about future inputs. RNNs differ from other types of neural networks primarily in their ability to represent sequential data—that is, this recurrent feedback loop.

Typically, we optimize a loss function in relation to network parameters like weights and biases when training RNNs. Backpropagation through time (BPTT), an extension of the backpropagation algorithm tailored for recurrent architectures, is a prevalent method. To use BPTT, you first view the network as a deep feedforward network with shared weights. Then, you use the standard backpropagation algorithm to update the parameters and find the gradients.

In contrast, the problem of vanishing and exploding gradients can make training RNNs difficult. The gradients of the loss function with respect to the parameters either get smaller exponentially or bigger explosively when you go back in time. This makes it harder to train networks that depend on things that are far away. There are many ideas that have been put forward to solve this problem, such as gradient clipping, orthogonal initialization, and activation functions like long short-term memory (LSTM) and gated recurrent unit (GRU) cells.

Because RNNs can represent sequential data, they have become very popular in many different fields. Among some such uses are:

- Natural Language Processing (NLP): Tasks like language modeling, machine translation, sentiment analysis, and text synthesis frequently use RNNs.
- Speech Recognition: Using RNNs, automatic speech recognition (ASR) applications have effectively modeled the temporal relationships in voice signals.
- Time Series Prediction: RNNs are well suited for tasks including time series prediction—stock price forecasting, weather prediction, and energy demand forecasting.
- Sequence-to-Sequence Learning: RNNs may map input sequences to output sequences of varied lengths for tasks like machine translation, picture captioning, and video analysis.
- Reinforcement Learning: In reinforcement learning, RNNs have been used for sequential decision-making tasks wherein the agent learns to interact with an environment over time to maximize a total reward.

Even though their effectiveness in sequential data modeling is remarkable, RNNs still present numerous difficulties. These include handling long-range dependencies, handling erratic or partial inputs, and maximizing inputs of different diameters. Future RNN research may focus on improving architectures, including attention mechanisms to concentrate on relevant information as well as techniques for unsupervised and self-supervised learning to reduce the need for labeled data. Also, mixing RNNs with other deep learning architectures like transformers and CNNs could possibly make the system much more scalable and improve its performance.

2.2.3.5 Training Deep Models

Contemporary machine learning research heavily relies on the development of complex models, which also serve as the foundation for several other fields of study. Deep models, characterized by their hierarchical organization of interconnected neurons, need a meticulous approach to optimization. Figure 2.5 depicts the process of training deep models, which involves iteratively adjusting model parameters to minimize a predefined loss function. Gradient-based approaches, which are frequently used in machine learning, calculate the loss function's gradients with respect to the model parameters. We then use these gradients to update the parameters to minimize the loss function [32].

Important components of training deep models include the design of the neural network, loss functions, and optimization techniques. The model's ability to detect intricate patterns in the data depends on the design of the neural network. Loss functions guide the optimization process toward positive outcomes by measuring the difference between expected and actual outputs. While classification tasks usually utilize cross-entropy loss, regression tasks often use mean squared error (MSE). Stochastic gradient descent (SGD) and its derivatives (e.g., Adam, RMSprop) coordinate changes to parameters in order to minimize the loss function as rapidly as feasible.

A multitude of strategies and techniques optimize the training process and alleviate obstacles like overfitting. In an effort to prevent overfitting, L1 and L2 regularization, among other regularizing methods, penalize overly complex models. Data augmentation methods improve the model's generalizability by adding modifications such as rotation, scaling, and inversion to the training set. In situations where annotated data is scarce, transfer learning bootstraps the training process by utilizing pre-trained models on large datasets.

Despite the advancements in deep model training, a variety of challenges persist. Computational complexity, overfitting, and disappearing or bursting gradients are all difficult problems. Techniques like dropout, model pruning, and gradient clipping are used to reduce these challenges and improve the training process's stability and efficacy.

Deep learning, as shown in Figure 2.6, requires large datasets and machines with GPUs and faces these challenges more prominently than traditional machine learning. Machine learning, on the other hand, performs well with small to medium datasets and can work on low-end machines. It focuses on understanding the features and representation of the data, making it more versatile with a variety of algorithms and shorter training times. While deep learning may take hours or weeks to train, machine learning often takes only seconds or hours. Additionally, interpreting the results of deep learning models is more difficult compared to some machine learning algorithms, which are easier to interpret.

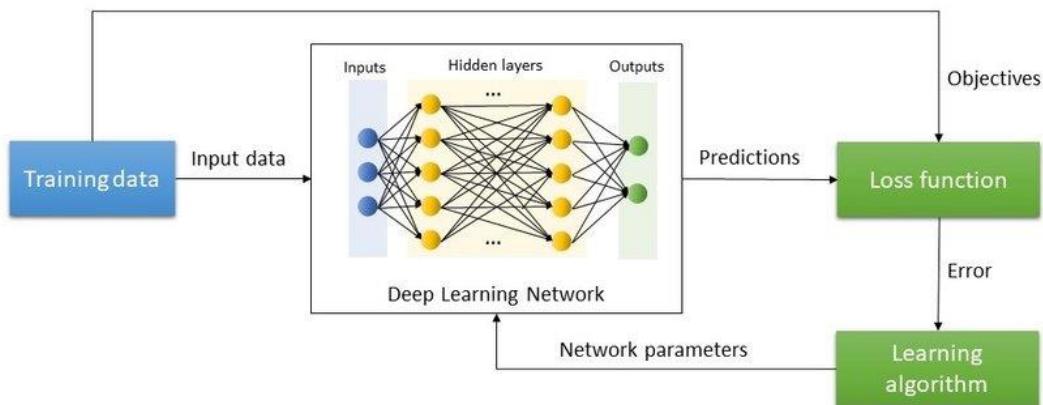


Figure 2.5: Training process in Deep Learning algorithms [33]

	Deep Learning	Machine Learning
Data	Needs a big dataset	Performs well with a small to a medium dataset
Hardware requirements	Requires machines with GPU	Works with low-end machines
Engineering peculiarities	Needs to understand the basic functionality of the data	Understands the features and how they represent the data
Training time	Long	Short
Processing time	A few hours or weeks	A few seconds or hours
Number of algorithms	Few	Many
Data interpretation	Difficult	Some ML algorithms are easy to interpret, whereas some are hardly possible

Figure 2.6: Machine Learning vs. Deep Learning training features [34]

2.2.3.6 Generative Models

In artificial intelligence and machine learning, generative models are crucial. They allow computers to grasp and generate complex data distributions. Unlike discriminative models, they seek to

comprehend the fundamental structure of the data and provide new samples that, rather than just learning the boundary between different classes, mirror the original data distribution [35], [36].

Generative models are algorithms meant especially to learn the probability distribution of a given dataset. These models may generate fresh samples indistinguishable from the original dataset by understanding the natural patterns and structures in the data. In several fields, including computer vision, natural language processing, and healthcare, the ability to generate synthetic data has major consequences.

Generative models include a large range of techniques, each with unique means for learning and producing data distributions. Among the notable instances of these kinds are variational inference models, flow-based models, autoencoders, probabilistic graphical models, and generative adversarial networks (GANs).

By using graphical frameworks such as Bayesian networks or Markov random fields, probabilistic graphical models capture the aggregate probability distribution of data. Trained to precisely replicate the input data, an encoder and a decoder network together constitute autoencoders. Using probabilistic inference, variational autoencoders (VAEs) provide a large spectrum of outputs. A generator and a discriminator network engaged in a two-player minimax game make up generative adversarial networks (GANs). While the discriminator's job is to distinguish between actual data and the generator's generated data, the generator's goal is to provide samples that very nearly reflect real data. Variational inference models, like variational Bayes and expectation maximization, try to find the best solution to a lower-bound problem by guessing how complex the posterior distributions are. Flow-based models have the capacity to execute reversible transformations, turning data from a simple distribution into a more complex one. This enables the rapid creation of fresh samples as well as the evaluation of the likelihood of already existing data.

Applied in several fields, generative models drastically change the ways we assess and generate data. Among the noteworthy uses are image creation and synthesis, text generation and natural language processing, drug development and molecular design, anomaly detection, and data augmentation.

The image-generating and synthesis fields widely use GANs and VAEs to produce realistic pictures. This has enabled the creation of many applications, including image super-resolution, style transfer, and picture-to-image translation. Transformers abound in text production and natural language processing; recurrent neural networks (RNNs) serve as language models. They support duties like language translation, conversation generation, and summarizing, as well as assisting to produce cohesive content. Drug discovery and molecular design use generative models to generate novel molecular structures with specific properties, thereby accelerating research in material science and pharmacology. They also make training discriminative models better by creating fake samples and finding outliers by comparing those outliers to the learned distribution of data.

Although generative models have achieved impressive results, they encounter many obstacles, such as mode collapse in GANs, posterior collapse in VAEs, and the need for assessment metrics to check sample quality. Future research endeavors seek to tackle these difficulties while enhancing the scalability, interpretability, and resilience of generative models.

2.2.3.7 Graph Convolution Networks

Graph convolutional networks (GCNs) have become rather effective mechanisms for learning from graph-structured data. Unlike traditional convolutional neural networks (CNNs), which work on normal grids (like pictures), GCNs apply the convolution operation to graph data. This helps them understand node features and relationships. Information aggregation from a node's neighborhood is the fundamental concept of graph convolution [37], [38]. This can be formally expressed as:

$$H^{(l+1)} = \sigma(\hat{A}H^{(l)}W^{(l)})$$

where $H^{(l)}$ is the node feature matrix at layer l , with $H^{(0)} = X$; \hat{A} is the normalized adjacency matrix; $W^{(l)}$ is the trainable weight matrix at layer l ; and σ is a non-linear activation function, such as ReLU. To ensure numerical stability and improve convergence, the adjacency matrix is often normalized. A common normalization method is:

$$\hat{A} = D^{-1/2}AD^{-1/2}$$

where D is the degree matrix with $D_{ii} = \sum_j A_{ij}$.

GCNs can be categorized into spectral and spatial methods. Spectral methods rely on the graph Fourier transform. The convolution operation is defined in the spectral domain using the graph Laplacian $L = I - D^{-1/2}AD^{-1/2}$. The spectral convolution is defined as:

$$H^{(l+1)} = \sigma(Ug_\theta(\Lambda)U^TH^{(l)})$$

where U is the matrix of eigenvectors and Λ is the diagonal matrix of eigenvalues of L , and g_θ is a learnable function.

The convolution is directly defined in the node domain by spatial algorithms, which aggregate features from neighboring nodes. Diverse GCN variants may result from variations in the aggregation function. For instance, ChebNet reduces computational complexity by employing Chebyshev polynomials to approximate the spectral filters. To aggregate features from a fixed-size neighborhood, GraphSAGE uses functions such as mean, LSTM, or pooling. Graph Attention Networks (GAT) implement attention mechanisms to assign distinct weights to neighbors based on their significance.

Graph Convolutional Networks (GCNs) have shown effective applications across several disciplines. Social networks use community identification, connection prediction, and node categorization. Biological networks use GCNs to forecast protein functions and interactions. Knowledge graphs enhance the process of categorizing entities and making predictions about relationships.

Graph Convolutional Layers are a notable breakthrough in deep learning, allowing for efficient learning from intricate, non-Euclidean data structures. Generalizing convolution processes to graphs allows for a broad variety of applications, making graph convolutional networks (GCNs) an essential element of contemporary machine learning systems.

2.2.3.8 Graph Attention Networks

Extending on the ideas presented by Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs) are a fresh method of learning from graph-structured data. Using attention processes, Graph

Attention Networks (GATs) dynamically allocate weights to immediate proximity nodes. This allows the network to concentrate on the most pertinent sections of the graph during the learning process. This method simplifies the model's understanding of intricate data linkages and eliminates certain obstacles associated with GCNs, including the static weights of adjacent nodes [39], [40], [41].

The core concept underlying graph attention is to calculate a collection of attention coefficients that signify the significance of the neighbors of each node. For each node u_i , these coefficients e_{ij} are computed for every neighbor $u_j \in N(i)$, where $N(i)$ denotes the set of neighboring nodes of u_i .

The attention coefficients are computed using a shared attention mechanism, typically parameterized by a learnable weight vector a . For nodes u_i and u_j with features h_i and h_j , respectively, the attention coefficient e_{ij} is given by:

$$e_{ij} = \text{LeakyReLU}(a^T [Wh_i || Wh_j])$$

where $||$ denotes the concatenation, W is a weight matrix, and *LeakyReLU* is a non-linear activation function.

To ensure the coefficients are comparable across different nodes, a softmax function is applied:

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in N(i)} \exp(e_{ik})}$$

where a_{ij} represents the normalized attention coefficient.

The final step involves aggregating the features from neighboring nodes, weighted by the attention coefficients. The aggregated feature for node u_i in the next layer is:

$$h_i^{(l+1)} = \sigma \left(\sum_{j \in N(i)} a_{ij} Wh_j^{(l)} \right)$$

where σ is a non-linear activation function, such as ReLU.

To enhance the model's capacity and stabilize the learning process, GATs often employ multi-head attention. In this approach, K independent attention mechanisms (heads) are applied, and their outputs are concatenated:

$$h_i^{(l+1)} = \left[\sum_{j \in N(i)} a_{ij}^k W^k h_j^{(l)} \right]_K$$

Alternatively, the outputs can be averaged to reduce dimensionality and computational complexity.

Graph Attention Networks have been employed in a variety of fields as a result of their capacity to dynamically evaluate the significance of neighboring nodes. Social networks employ GATs to classify nodes, infer links, and identify communities. In biological networks, they help predict gene regulatory networks and protein-protein interactions. GATs also improve personalized recommendation systems by efficiently capturing user-item interactions.

By integrating attention mechanisms that dynamically modify the influence of neighboring nodes, Graph Attention Layers represent a significant advancement in graph-based learning. This adaptability enables GATs to capture more intricate and nuanced relationships within graph-structured data,

rendering them a potent instrument for a variety of applications. The ability to concentrate on the most relevant components of the graph improves the performance and interpretability of graph neural networks, thereby solidifying GATs as a critical element in the evolving deep learning landscape.

2.3 Model Training and Evaluation

Model training and assessment are crucial steps in the life cycle of a machine learning project, serving as the fundamental processes that ascertain the efficacy and dependability of predictive models. Model training in machine learning entails instructing an algorithm to discern patterns and correlations within data, while evaluation gauges its performance and ability to apply knowledge to new instances. This critical stage involves the meticulous selection and preparation of data, the creation of suitable model architectures, and the fine-tuning of hyperparameters. To design effective and trustworthy intelligent systems, it is crucial to have a deep understanding and mastery of model training and evaluation, as machine learning applications become increasingly prevalent in various domains [42], [43].

2.3.1 Data Cleaning and Preprocessing

Data cleansing and preprocessing are critical components in the pipeline of machine learning, significantly impacting the effectiveness and caliber of predictive models. These critical procedures include converting unprocessed data into a structure suitable for analysis, resolving concerns about the quality of the data, and preparing features for model training [15], [16], [44], [45], [46].

Data Wrangling

Data management consists of a variety of tasks that prepare unprocessed data for analysis. This part encompasses the process of normalizing and transforming data to guarantee consistency and uniformity. It also involves handling inconsistencies, duplication, and mistakes. Data aggregation, filtering, and transformation methods are crucial in determining the quality and structure of a dataset. These methods provide the foundation for later modeling and analytic efforts.

Missing Value Imputation

In real-world datasets, missing data calls for careful attention and handling. Different kinds of missing data—including missing totally at random (MCAR), missing at random (MAR), and missing not at random (MNAR)—demand customized imputation methods. Mean, median, and mode imputation, in addition to more advanced methods like forward and backward infill and interpolation, are typical approaches. As illustrated in Figure 2.7, these imputation techniques fill in the missing data, converting incomplete datasets into more complete ones. Nevertheless, the particular demands of the modeling endeavor and the underlying data distribution should determine the selection of an imputation technique.



Figure 2.7: Missing Value Imputation [47]

Outlier Detection and Treatment

Anomalies—also called outliers—may have a major effect on the resilience and performance of machine learning models. Finding data points that show a significant departure from the expected distribution and using appropriate techniques for correction or elimination would help one find and handle outliers. Statistical tools like the Z-score (shown in Figure 2.8) and the interquartile range (IQR) are often used to find outliers. To treat outliers, transformation and winsorization can be used. Maintaining valuable information while removing outliers is of the utmost importance, as an excessive emphasis on outlier removal may result in the forfeiture of crucial insights.

Detecting Outliers with z-Scores

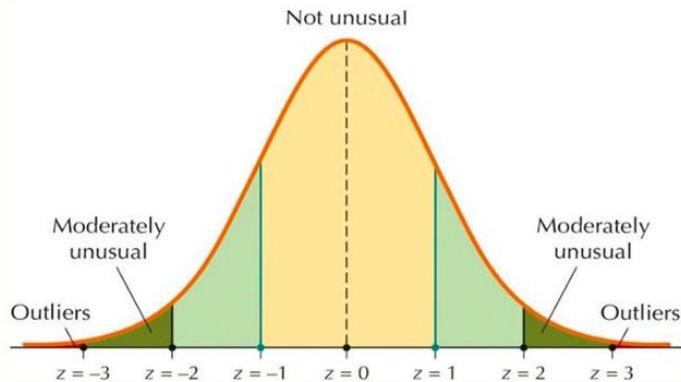


Figure 2.8: Outlier Detection and Treatment example [48]

Feature Scaling

Feature scaling is an essential preprocessing operation that prevents certain features from overpowering others on account of magnitude differences by ensuring that features are on a comparable scale. We frequently employ scaling techniques such as min-max scaling and standardization (z-score normalization). Min-max scaling linearly scales features to a specified range, while standardization centers features around their mean and scales them by their standard deviation. When it comes to algorithms that use gradient descent-based optimization and are sensitive to feature magnitudes, feature scaling is especially important.

Encoding Categorical Variables

In machine learning, categorical variables present unique challenges, as the majority of algorithms require numerical inputs. Encoding categorical variables entails converting categorical data into a numerical format suitable for training models. One-hot encoding, as shown in Figure 2.9, generates binary columns for each category, while label encoding allocates a distinct integer to each category. There is also target encoding and frequency encoding, which utilize information about the target variable or category frequencies to improve the encoding process. These methods help transform categorical data into a suitable format for machine learning models.

The diagram shows a table on the left with four rows labeled 'Color' (top), 'Red' (second), 'Green' (third), and 'Blue' (bottom). An arrow labeled 'One-hot encoding' points from this table to a larger table on the right. The right table has three columns labeled 'd1', 'd2', and 'd3'. The 'd1' column contains values 1, 0, 0 respectively for Red, Green, and Blue. The 'd2' column contains values 0, 1, 0 respectively. The 'd3' column contains values 0, 0, 1 respectively. This represents a one-hot encoded version of the original categorical data.

Color	d1	d2	d3
Red	1	0	0
Green	0	1	0
Blue	0	0	1

Figure 2.9: Encoding Categorical Variable example [49]

Dimensionality Reduction

Techniques for dimensionality reduction aim to decrease the number of features in a dataset, ensuring that no pertinent information escapes. This approach overcomes dimensionality problems in addition to simplifying computation, thereby enhancing model interpretability. Finding the orthogonal axes with the highest data fluctuation helps one reduce linear dimensionality using Principal Component Analysis (PCA). Two other methods to capture complex, non-linear relationships in high-dimensional data are uniform manifold approximation and projection (UMAP) and t-distributed Stochastic Neighbor Embedding (t-SNE).

2.3.2 Data Splitting for Model Training and Evaluation

Data splitting is a crucial technique in machine learning that plays a key role in ensuring the development and evaluation of robust models. It has two primary functions: training the model and evaluating its performance. In the absence of a correct division, a model faces the danger of overfitting. On the other hand, without a separate test set, the model's evaluation lacks strictness, potentially leading to misleading results regarding its effectiveness [45], [50].

To develop a proficient data splitting method, it is necessary to partition the dataset into three fundamental subsets: the training set, the validation set, and the test set.

The training set serves as the foundation for model development. The model uses this subset, which represents the largest portion of the data, to train its parameters. Throughout this collection, the model learns patterns and connections, adjusting its internal weights to minimize the selected loss function. The training set serves as the driving force behind the machine learning process, directing the model towards achieving optimal performance for the given job.

The validation set functions as a milestone throughout the training process. During the process of parameter refinement on the training set, the validation set serves as an autonomous evaluation of the model's performance. By assessing the model's ability to generalize to unseen data not used for training, this subset serves as a measure to prevent overfitting. We typically carry out hyperparameter optimization, which involves modifying the model's architecture or regularization approaches, by evaluating the model's performance on the validation set. This set serves as a precautionary measure, guaranteeing that the model is not simply memorizing the training data but genuinely comprehending the fundamental patterns.

The test set is the final measure of a model's performance. We only use it after deciding whether the model is suitable for assessment, usually when a model finishes the training process, leaving it unmodified throughout the training and hyperparameter tuning procedures. Emulating real-life events, the test set assesses the model's ability to extend to wholly new and unseen data. Using a distinct test set, researchers guarantee an objective assessment free of any inadvertent changes depending on test results. Including accuracy, precision, recall, and F1 score, the performance indicators from the test set provide a complete assessment of the model's strengths.

We frequently employ a general guideline, such as a division of 80-10-10 or 70-15-15, to denote the allocation of data among training, validation, and test sets. However, the precise division ratio may vary depending on the characteristics of the problem and the magnitude of the dataset. When there is inadequate labeled data, we may implement techniques such as cross-validation. This entails dividing the data into a multitude of folds for both training and validation in order to optimize the utilization of the current information.

As illustrated in Figure 2.10, a common data splitting technique is the holdout method. Here, the initial split separates the data into training and testing sets. We use the training set to train the model and reserve the testing set to evaluate its performance on unseen data.

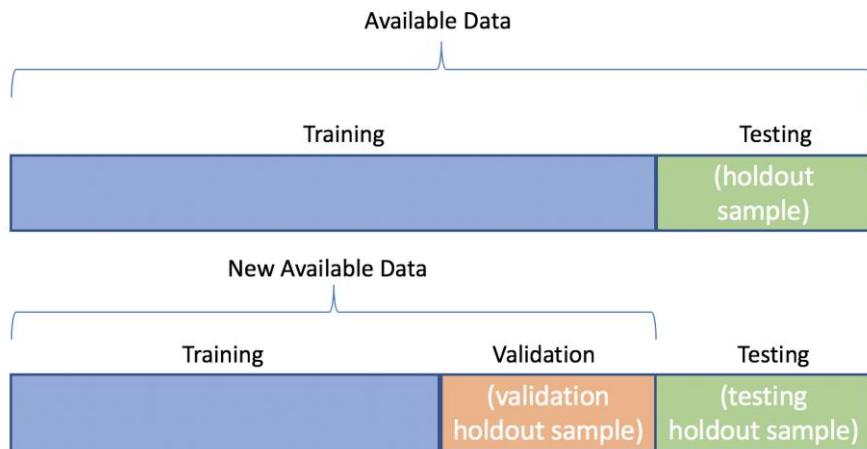


Figure 2.10: Train, test and validation split [51]

2.3.3 Hyperparameters in Machine Learning

Hyperparameters are critical elements in the development and refinement of machine learning models, having a significant impact on their efficacy and generalizability. Contrary to model

parameters, which we acquire through training data, hyperparameters are predetermined configurations that shape the model's behavior and complexity [52], [53].

Types of Hyperparameters

Hyperparameters comprise a wide range of configurations that influence the structure, operation, and learning processes of machine learning models. In neural networks, model-specific hyperparameters comprise architectural decisions including layer count, layer thickness, and activation function. Hyperparameters specific to the algorithm control algorithmic behavior and regularization mechanisms. These hyperparameters include regularization strength, tree depth in the case of decision trees, and kernel parameters in the case of support vector machines. Hyperparameters of optimization algorithms, such as the learning rate, block size, and convergence criteria, also determine the dynamics of learning during training.

Hyperparameter Tuning Techniques

Hyperparameter tuning, in addition to hyperparameter optimization, necessitates a deliberate search for the best hyperparameter combination that generates the most model performance. Researchers have developed numerous techniques to modify hyperparameters, each with its own advantages and disadvantages. Grid search involves systematically investigating predefined hyperparameter values and evaluating every conceivable combination through cross-validation. By using pre-defined distributions to sample hyperparameter values, random search offers a more effective replacement for grid search. By using probabilistic models, Bayesian optimization guides the search process and iteratively changes the search space depending on past assessments.

There are also automated frameworks for hyperparameter tuning, such as Hyperopt, AutoML, Bayesian Optimization, and Hyperband (BOHB), which use advanced optimization algorithms and meta-learning techniques to speed up the process of hyperparameter tuning.

Hyperparameter Tuning in Deep Learning

Hyperparameter optimization is becoming increasingly important in deep learning due to the complexity and magnitude of neural network architectures. The architectural hyperparameters of neural networks, including but not limited to the number of layers, layer diameters, and activation functions, have a significant impact on their capacity for representation and expressiveness. Optimization hyperparameters, comprising the learning rate, sample size, and optimizer selection, determine convergence dynamics and training stability. Regularization hyperparameters, including dropout rates and weight decay, are of paramount importance in regulating model complexity and preventing overfitting. In deep learning applications spanning various domains, it is critical to optimize performance and generalization through the precise calibration of these hyperparameters, as illustrated in Figure 2.11.

Hyperparameter Importance and Interpretability

Sensitivity analysis methods provide an insightful study of the relative relevance of hyperparameters and their subsequent effect on a model's performance. Sensitivity analysis carefully alters some hyperparameters while maintaining others constant in order to reveal the degree to which changes in hyperparameter values impact a model's performance. Adapted feature significance methods, such as

permutation importance and SHAP values, take into account hyperparameters and provide objective measures of their impact on the model's performance. Still, the process of determining the significance of hyperparameters within the context of complex machine learning models remains difficult.

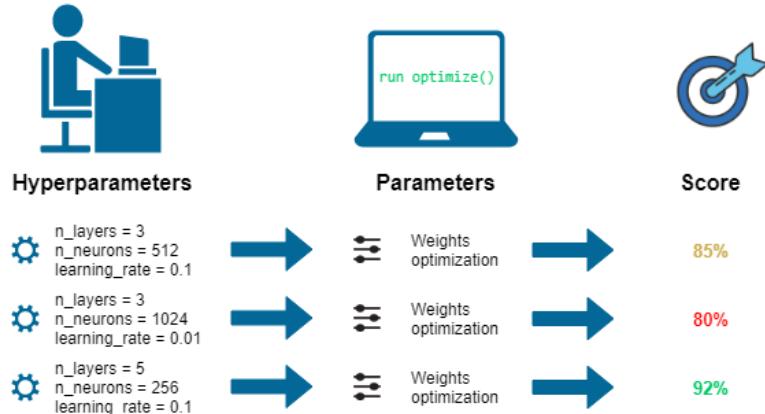


Figure 2.11: Significance of Hyperparameters [54]

2.3.4 Evaluation Metrics

Evaluation criteria, primarily shown in Figure 2.14, largely determine the efficiency and usefulness of machine learning models in a variety of fields and applications [55], [56]. There are examples in the test set that the learning algorithm has never come across before. Therefore, if our model accurately predicts the labels of the instances in the test set, we say that it generalizes well or is good.

Classification Metrics

Several assessment measures provide insights on the performance of the model in classification problems, in which the aim is to assign instances to predetermined groups or categories. A basic statistic, accuracy, gauges the percentage of correctly categorized events out of all the others. Let TP be true positives; TN be true negatives; FP be false positives; and FN be false negatives [57], [58]. Accuracy can be determined as:

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

While accuracy is easy to understand, in datasets include a dominating class it might be misleading. A model might simply predict the majority class all the time and achieve high accuracy, despite failing to identify the minority class effectively.

Precision and recall provide a more nuanced view, capturing the trade-off between identifying relevant instances (precision) and minimizing false negatives (recall).

$$\begin{aligned} Precision &= \frac{TP}{TP + FP} \\ Recall &= \frac{TP}{TP + FN} \end{aligned}$$

A trade-off often exists between precision and recall. A model emphasizing precision might miss true positives (high false negatives) to ensure its positive predictions are accurate. Conversely, a recall-

focused model might capture many true positives but also include a significant number of false positives (low precision).

The F1 score computes the harmonic mean of precision and recall and is useful when both metrics are equally important.

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Matthews Correlation Coefficient (MCC) is a binary classifier quality performance metric. The MCC offers a fair evaluation that is applicable even if the classes differ substantially in size by evaluating all four categories of the confusion matrix—true positives, true negatives, false positives, and false negatives. Regarding mathematics, we define the MCC as follows:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

The MCC ranges from -1 to 1, where +1 indicates perfect prediction, 0 indicates random chance, and -1 indicates total disagreement between predictions and actual observations.

ROC curves and the Area Under the Curve (AUC) let practitioners graphically show the balance between the false positive and real positive rates, as shown in Figure 2.13. This enables their evaluation of a model's performance over many thresholds [59]. By means of a comprehensive study of true positives, true negatives, false positives, and false negatives, the confusion matrix clarifies model mistakes and misclassifications, therefore enabling a more deep awareness of them, as illustrated in Figure 2.12.

		Predicted	
		Negative (N) -	Positive (P) +
Actual	Negative -	True Negative (TN)	False Positive (FP) Type I Error
	Positive +	False Negative (FN) Type II Error	True Positive (TP)

Figure 2.12: Confusion matrix [60]



Figure 2.13: ROC Curve [61]

Regression Metrics

Regression problems predict continuous values instead of separate classes. Among the usually utilized regression measures with an average size of errors between predicted and actual values are Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE). Higher R-squared (R^2) denotes a better model fit; this metric shows the percentage of variance the model clarifies [62].

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

MAE is less sensitive to outliers than MSE, but it provides less interpretability in terms of data units.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

MSE is sensitive to outliers, as squaring large errors amplifies their impact.

$$RMSE = \sqrt{MSE}$$

RMSE is a statistic that is more intuitive and easier to grasp than MSE. It quantifies the error's average size.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

R-squared may be misleading because, even when it is not statistically significant, it increases with an increasing number of predictors.

Clustering Metrics

Clustering methods categorize instances that have similar characteristics, employing evaluation criteria that assess the accuracy of cluster allocations [63].

The Silhouette Score quantifies the level of cluster cohesiveness and separation by taking into account the similarity within clusters and the dissimilarity across clusters. Mathematically, the Silhouette Score $s(i)$ is defined for each instance i as:

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

where $a(i)$ is the intra-cluster distance, or typical distance between i and all other points in the same cluster, and $b(i)$ is the minimum average distance between i and points in a different cluster (inter-cluster distance).

The Silhouette Score ranges from -1 to 1:

- $s(i) \approx 1$: The instance is well-matched to its own cluster and poorly matched to neighboring clusters.
- $s(i) \approx 0$: The instance lies on or near the decision boundary between two clusters.
- $s(i) \approx -1$: The instance might have been assigned to the wrong cluster.

The Davies-Bouldin Index provides a measure of cluster separation, considering both intra-cluster compactness and inter-cluster separation. It is defined as:

$$DB = \frac{1}{k} \sum_{i=1}^k \max_{j \neq i} \left(\frac{\sigma_i + \sigma_j}{d_{ij}} \right)$$

where:

- k is the number of clusters.
- The average distance between every point in cluster i and its centroid is σ_i .
- The distance between the centroids of clusters i and j is denoted as d_{ij} .

The Davies-Bouldin Index ranges from 0 to infinity:

- Lower values indicate better clustering, with well-separated and compact clusters.
- Higher values suggest poor clustering with significant overlap between clusters.

The Calinski-Harabasz Index (also known as the Variance Ratio Criterion) evaluates the ratio of the sum of between-cluster dispersion to within-cluster dispersion. It is defined as:

$$CH = \frac{\text{tr}(B_k)}{\text{tr}(W_k)} * \frac{n - k}{k - 1}$$

where:

- $\text{tr}(B_k)$ is the trace of the between-cluster dispersion matrix.
- $\text{tr}(W_k)$ is the trace of the within-cluster dispersion matrix.
- n is the total number of instances.

- k is the number of clusters.

Higher values on the Calinski-Harabasz Index indicate better-defined clusters.

Adjusted Rand Index (ARI) and Normalized Mutual Information (NMI) quantify the similarity between true and predicted cluster assignments, adjusted for chance and normalized for cluster size imbalances. ARI modifies the Rand Index to account for chance, while NMI measures the mutual information between the true and predicted assignments, scaled by the entropy of the clusterings.

Other Evaluation Metrics

Beyond classification, regression, and clustering tasks, many more evaluation metrics serve certain uses and fields. Ranked lists are common in information retrieval and recommendation systems; MAP is a measure of their performance. Intersect over Union (IoU) computes segmentation mask or anticipated and real bounding box overlap. When searching for items or discrete elements of a picture, this is very crucial. Commonly used Cohen's Kappa in inter-rater reliability studies to assess annotator agreement adjusted for chance [64], [65], [66], [67].

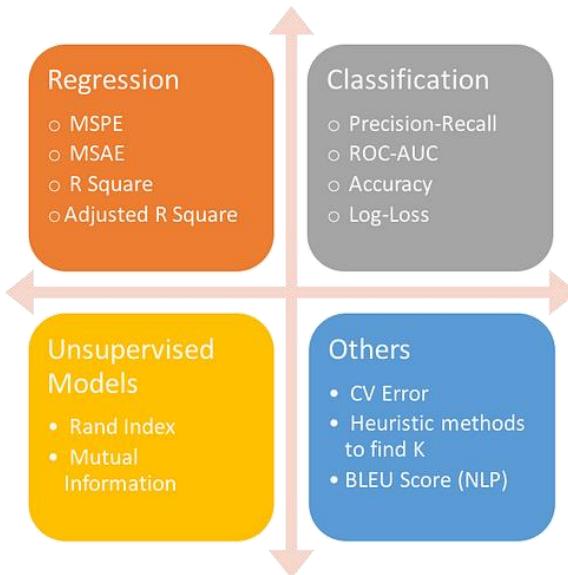


Figure 2.14: Metrics for Evaluating Machine Learning Models [68]

Chapter 3

Semantic Web and Knowledge Graphs

Knowledge graphs and the Semantic Web are two revolutionary advancements in organization and online data retrieval. Originally creating the Internet, Sir Tim Berners-Lewley also suggested the Semantic Web, which gives data a layer of semantic relevance, thereby improving the present online architecture. This integration enables more complex and contextually informed user interactions. Standardized ontologies, metadata, and linked data concepts facilitate this process [69].

The "Semantic Web Layer Cake" is a commonly used representation of semantic web languages' vision. Tim Berners-Lee, the director of the World Wide Web Consortium (W3C), introduced the initial layer cake¹ at XML 2000. This version consisted of a rule's language added on top of the ontology language. Researchers have discovered that adding rule languages to the Web Ontology Language OWL is not an effortless task [70]. Figure 3.1 modifies the layer cake diagram by placing rules alongside OWL on a shared layer. Tim Berners-Lee introduced the updated version of layer cake during his keynote speech at the WWW2005 conference.

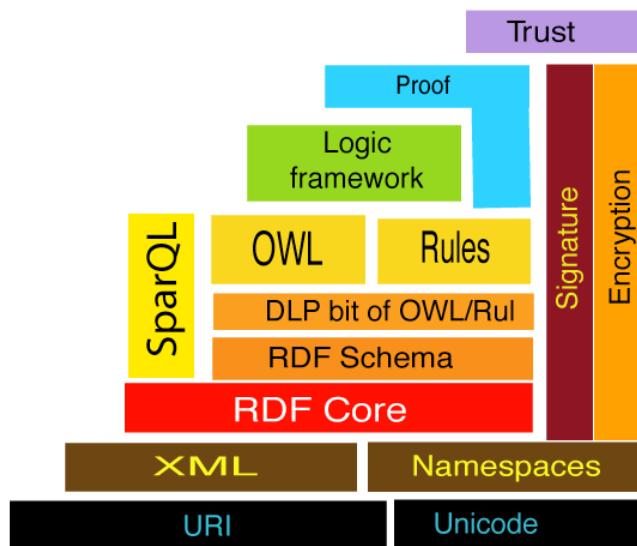


Figure 3.1: The Semantic Web language layer cake [71]

Complementing the Semantic Web, knowledge graphs are dynamic structures that interconnect information in a way that simulates human thought [72]. Automated systems outperform traditional databases by establishing substantial associations between items, enabling them to not only store data but also deduce and navigate complex knowledge graphs. Knowledge graphs

¹ <http://www.w3.org/2000/Talks/1206-xml2k-tbl/slides10-0.html>

and semantic webs are leading the effort to turn the web into a more intelligible and linked realm of knowledge.

3.1 Resource Description Framework (RDF)

The Resource Description Framework (RDF) is an essential element of the Semantic Web, serving a pivotal function in the representation and facilitation of data flow. RDF offers a standardized approach to describing resources and their relationships, facilitating the exchange and integration of data across various systems [73], [74].

3.1.1 Definition and Conceptual Framework

RDF, created by the W3C, is a standardized framework for representing and describing online resources in a manner that enables computers to understand their meaning. Triples, composed of subject-predicate-object assertions, form the foundation of RDF, a data paradigm. These triples indicate relationships between resources. The triples serve as the fundamental components of RDF graphs, with nodes representing resources and edges indicating the connections between them. RDF uses triples to depict a wide range of information, showcasing their inherent adaptability and scalability. RDF enables the construction of linked data graphs, which may represent many kinds of information in a cohesive and expandable way [75].

3.1.2 Core Components of RDF

The essential elements that provide the foundation for describing and exchanging data on the Semantic Web are resources, properties, and assertions. These components are the core building blocks of RDF [75], [76]. Understanding these basic components is essential for effectively using RDF's potential to enable intelligent data management, knowledge representation, and semantic interoperability on the internet. Research and development in RDF have the potential to enhance the capabilities of the semantic web and open up new avenues for data-driven creativity and discovery. The following text provides a concise summary of the essential components that make up RDF:

- **Resources:** The concept of resources, which includes all entities that a Uniform Resource Identifier (URI) can identify, is the fundamental element of RDF. Resources include tangible entities, such as people, places, or physical goods, as well as ethereal components, such as abstract concepts, documents, or web pages. In RDF, resources serve as the subjects or objects of assertions, forming the basis for constructing a data model using graphs. RDF facilitates unambiguous referencing and linking of data from diverse sources and domains.
- **Properties:** Properties refer to the characteristics or attributes of resources in RDF and establish relationships between them. RDF statements represent characteristics as predicates, connecting subjects and objects to provide assertions or descriptions about the resources. Properties may range from fundamental attributes like names or dates to more complex relationships such as "*isPartOf*" or "*hasAuthor*." Uniform resource identities (URIs) identify characteristics in the Resource Description Framework (RDF).

This makes it possible to build a large semantic network that goes beyond the simple syntax of data.

- **Statements:** RDF statements, also referred to as triples, are the fundamental components of RDF data models. They consist of assertions that include a subject, predicate, and object. Each statement posits a correlation between a subject resource, a predicate property, and an object resource or literal value. An example of a triple is `<ex:John ex:hasAge "30">`, which indicates that the resource "`ex:John`" has an age of "30". In this case, "ex" is used as a namespace prefix. The structure and semantics of RDF graphs together specify the representation of large knowledge systems in a distributed and decentralized way.

3.1.3 RDF Syntax and Serialization Formats

RDF serializing formats are essential parts of the Semantic Web ecosystem, as they provide specific ways to encode RDF data in many syntactic forms. An efficient RDF data transfer and storage approach improves communication and interoperability across many systems and applications. RDF/XML, Turtle, JSON-LD, N-Triples, and RDFa are the main RDF serializing models. Each one of these techniques has special benefits for displaying RDF triples in a form that computers and humans can grasp [75], [77], [78].

- **RDF/XML:** RDF/XML is a serialization standard that has played a vital role in the development of the Semantic Web. It creates a structured representation of RDF triples using an XML-based grammar. The syntax's hierarchical structure is characterized by the storage of subject-predicate-object triples inside XML components. This enables the representation of semantic connections in a uniform way. Although RDF/XML has historical importance, it has received criticism for being excessively wordy and intricate, resulting in a reduced demand for it in modern Semantic Web applications.

The RDF/XML syntax necessitates the inclusion of certain XML elements, such as `<rdf:RDF>`, `<rdf:Description>`, and `<rdf:datatype>`, to include resource descriptions and data types. The purpose of this hierarchical structure is to precisely depict the interdependence of resources. The RDF/XML form shown in Figure 3.2 demonstrates the connection between the resource "`http://example.org/John`" and the attribute "`ex:hasAge`." The attribute has a specific value of "30" and is associated with a clear explanation of its data type.

```

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ex="http://example.org/">
  <rdf:Description rdf:about="http://example.org/John">
    <ex:hasAge rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">30</ex:hasAge>
  </rdf:Description>
</rdf:RDF>

```

Figure 3.2: RDF/XML example

The preceding RDF/XML example indicates compliance with the RDF specifications; however, it underscores the challenges associated with verbosity and complexity. This accentuates the need to explore serialization techniques that are more succinct.

- **Turtle (Terse RDF Triple Language):** Turtle emerges as a serialization format that targets the difficulties posed by RDF/XML, with an emphasis on human comprehensibility and succinctness. The utilization of punctuation and indentation to represent triples in the syntax results in a more concise and comprehensible illustration of RDF data. This format's widespread adoption is due to its accessibility, making it a viable option for both manual editing and automated processing.

Turtle's succinct syntax is evident in its representation of RDF triples: it links the resource "<http://example.org/John>" to the property "ex:hasAge" and assigns it the literal value "30." Figure 3.3 demonstrates how effectively Turtle communicates semantic relationships in a succinct and easily comprehensible fashion.

```

@prefix ex: <http://example.org/> .

ex:John ex:hasAge 30 .

```

Figure 3.3: Example Turtle Syntax

The simplicity and aesthetic allure of Turtle's syntax are significant elements contributing to its popularity as a user-friendly serialization format within the Semantic Web ecosystem. The clear and precise representation of the data facilitates human comprehension as well as supports computational analysis.

- **JSON-LD:** JSON-LD is a sophisticated serialization method for RDF that leverages the widely used JSON format to facilitate the transmission of linked data. In contrast to RDF/XML and Turtle, JSON-LD integrates the concepts of context and framing, which facilitate the association of words with URIs and structured data. The notable advantage of JSON-LD is its compatibility with contemporary JSON applications, in addition to providing precise and unambiguous definitions for linked data.

The following JSON-LD example in Figure 3.4 showcases the application of a context specification to establish a link between the URI prefix and the resource "<http://example.org/John>." By integrating this contextual information, the representation becomes more streamlined while maintaining the inherent legibility of JSON.

```
{
  "@context": {
    "ex": "http://example.org/"
  },
  "@id": "ex:John",
  "ex:hasAge": 30
}
```

Figure 3.4: Example JSON-LD

By effectively interacting with JSON applications and providing a foundation for linked data semantics, JSON-LD underscores its significance in contemporary Semantic Web applications. Context enriches the representation by imparting explicit significance, thereby facilitating the data's comprehension by both humans and machines.

- **N-Triples:** N-Triples represent RDF triples using a simple textual format consisting of subject-predicate-object statements, separated by whitespace, and terminated with a period. The RDF data model represents each component of the triple as an IRI (Internationalized Resource Identifier), blank node, or literal value. Machines commonly use N-Triples for data exchange and serialization, as they offer a lightweight, human-readable representation of RDF data. Figure 3.5 is an illustration that succinctly conveys the relationship between the resource "<http://example.org/John>" and the literal value "30," demonstrating the simplicity of N-Triples.

```
<http://example.org/John> <http://example.org/hasAge> "30".
```

Figure 3.5: Example N-Triples

Because of its plain syntax, N-Triples is particularly effective in situations where a simple and easily understandable representation of individual triples is required. The format is especially useful for debugging and data transmission, prioritizing simplicity and clarity.

- **N-Quads:** N-Quads incorporate named graphs into the representation of RDF data, enabling the storage of quads composed of subject-predicate-object-context tuples. In addition to subject, predicate, and object, N-Quads include a fourth component, as shown in Figure 3.6, the context or named graph, which identifies the specific graph to which the quad belongs. N-Quads are particularly useful for representing RDF data in scenarios where context or provenance information is essential, such as data integration, versioning, and metadata annotation.

```
<http://example.org/books/TheGreatGatsby>
<http://schema.org/author> "F. Scott Fitzgerald"
<http://example.org/sources/BookData> .
```

Figure 3.6: N-Quad example

3.1.4 RDF Schema (RDFS)

The RDF schema (RDFS) is a crucial element of the semantic web ecosystem. It provides a consistent foundation for building vocabularies and ontologies defining RDF data structure and meaning. RDFS improves the Resource Description Framework (RDF) by letting you make hierarchies, constraints, and inference rules that make it easier to understand what RDF data means [77], [79].

Class-subclass connections, property hierarchies, and domain/range limitations form the fundamental foundation of RDF Schema. Classes denote entity categorization, whereas properties elucidate the connections between entities or attribute-value combinations. RDFS incorporates elements such as *rdf:type*, *rdfs:subClassOf*, *rdfs:subPropertyOf*, *rdfs:domain*, and *rdfs:range* to establish relationships and limitations inside RDF vocabularies. RDFS facilitates interoperability and knowledge exchange across many applications and domains by offering a systematic language for expressing the semantics of RDF data.

The software provides a variety of capabilities to assist with ontology development, schema validation, and inferencing on RDF data. Class hierarchies categorize items into taxonomies or ontologies, enabling the establishment of subtypes and supertypes within a certain field. Property hierarchies enable the establishment of complex connections between entities, allowing for the definition of subproperties and property attributes. Domain and range restrictions limit the scope of attributes to specific categories of subjects and objects, thereby improving data consistency and integrity.

RDF Schema facilitates interoperability and integration by providing a defined lexicon and framework for delineating the organization and meaning of RDF data. Various applications and domains have the ability to use, recycle, and expand RDFS vocabularies, promoting cooperation and the sharing of knowledge. RDFS also easily combines other Semantic Web standards like OWL (Web Ontology Language) and SPARQL (SPARQL Protocol and RDF Query Language), which lets you do complex querying and deduction over RDF data.

Figures 3.7 and 3.8 show scenarios emphasizing RDF's strengths. Figure 3.7 specifies the *Animal*, *Mammal*, and *Cat* classes, wherein *Cat* is a subclass of *Mammal* and *Mammal* is a subclass of *Animal*. Given the transitive nature of subclass connections in RDFS, we can infer that *Cat*, despite her designation as a subclass of *Mammal*, is also a subclass of *Animal*.

```

@prefix ex: <http://example.org/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

ex:Animal a rdfs:Class .
ex:Mammal a rdfs:Class ;
    rdfs:subClassOf ex:Animal .
ex:Cat a rdfs:Class ;
    rdfs:subClassOf ex:Mammal .

```

Figure 3.7: RDFS Example 1

We define the property *hasParent* in Figure 3.8, restricting its domain and range to instances of the class *Person*, signifying its exclusive use in describing relationships between persons. The property *isSiblingOf* is a subproperty of *hasParent*, indicating that the relationship of being a sibling is a specialized form of the parent-child relationship.

```

@prefix ex: <http://example.org/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

ex:hasParent a rdf:Property ;
    rdfs:domain ex:Person ;
    rdfs:range ex:Person .
ex:isSiblingOf a rdf:Property ;
    rdfs:subPropertyOf ex:hasParent .

```

Figure 3.8: RDFS Example 2

3.1.5 SPARQL

The standard query language, SPARQL, allows users to query and manipulate Resource Description Framework (RDF) data. The semantic web is developing and requires quick and effective approaches for data searching and retrieval from RDF databases. SPARQL satisfies this requirement by providing a language that is both articulate and robust, enabling users to construct queries against RDF data. This enables users to extract valuable insights and information from extensive semantic databases [69], [80], [81].

SPARQL has a comprehensive and articulate syntax specifically designed for interrogating RDF databases. SPARQL queries consist of three primary components: SELECT, WHERE, and OPTIONAL clauses. The WHERE clause establishes the graph patterns to compare with the RDF data, whereas

the SELECT clause determines the variables to include in the query results. In addition, SPARQL has many query modifiers, including ORDER BY, LIMIT, and OFFSET, which allow for customization of the query results. Graph pattern matching is the process of comparing patterns supplied in the WHERE clause with the RDF graph. This process is fundamental to the meaning of SPARQL queries. To depict graph patterns, SPARQL uses triple patterns, including subject-predicate-object triples. By using the OPTIONAL keyword, queries can include optional patterns and process data that may or may not be present.

It has some fundamental attributes that distinguish it as a resilient query language for accessing semantic data. The traits include:

- SPARQL provides a wide range of query structures, such as basic graph patterns, optional patterns, filters, subqueries, and federated queries. These allow for the simple articulation of complex inquiry forms.
- RDF Data Manipulation: SPARQL provides functionality for not only retrieving RDF data, but also modifying, adding, and deleting RDF triples, simplifying data manipulation tasks.
- Many SPARQL implementations include inference capabilities, allowing users to use ontological axioms and rules to uncover latent information inside RDF data. The SPARQL protocol offers a standardized approach to executing queries on remote RDF endpoints, facilitating interoperability and distributed querying scenarios.

It also serves as a useful tool in various applications and scenarios across various fields:

- SPARQL facilitates the integration of diverse data sources and allows for the execution of intricate queries for activities related to semantic search and information retrieval.
- Semantic Data Integration: SPARQL enables the integration of diverse data sources by offering a consolidated query interface for accessing and querying RDF data.
- SPARQL plays a fundamental role in ontology-driven applications, which include tasks like integrating semantic data, constructing knowledge graphs, and performing semantic reasoning.
- SPARQL facilitates the discovery and querying of linked data resources on the semantic web. This promotes interoperability and the reuse of data.

The following are a few practical examples that demonstrate the use of the SPARQL language:

```

SELECT ?name ?birthPlace

WHERE {
    GRAPH ?g {
        ?person foaf:name ?name ;
            dbo:birthPlace ?birthPlace .
    }
    FILTER (?g =
<http://example.com/graph1>)
}

```

Figure 3.9: SPARQL query example 1

In Figure 3.9, the query retrieves people's names and birthplaces from a specific named graph. The GRAPH keyword specifies the graph pattern, enabling the retrieval of data from a specific named graph. The FILTER clause restricts the results to a specific graph identified by its URI.

```

SELECT ?name ?birthPlace

WHERE {
    GRAPH ?g {
        ?person foaf:name ?name ;
            dbo:birthPlace ?birthPlace .
    }
    FILTER (?g =
<http://example.com/graph1>)
}

```

Figure 3.10: PARQL query example 2

The query in Figure 3.10 calculates the total count of individuals belonging to the RDF dataset's class "person" using the COUNT aggregate function. It counts the occurrences of the variable ?person that are instances of the specified class.

3.2 OWL (Web Ontology Language)

OWL is an essential tool in semantic web technologies. It facilitates the representation, dissemination, and exchange of knowledge across several fields of study. OWL is a language for ontology that provides a systematic framework for defining concepts, relationships, and axioms in a certain domain. This allows robots to comprehend and evaluate the significance of internet content. OWL, produced by the W3C, is a standard that greatly enhances the goal of the semantic web: to enhance the web with data that can be read by machines [76], [82], [83].

Based on formal logic and knowledge representation paradigms, OWL provides powerful tools for representing complicated domains, allowing for the modeling of everything from basic taxonomies to elaborate ontologies. OWL is a knowledge representation language that is built on Description Logic (DL). It offers a diverse set of tools for defining classes, attributes, persons, and logical restrictions. This enables the creation of comprehensive and semantically robust knowledge models. This feature enables a wide range of individuals, including researchers, practitioners, and domain experts, to convert and distribute domain information in a format that can be understood by machines. This promotes compatibility and the merging of knowledge across different systems and applications.

The development of OWL demonstrates a progressive path characterized by continuous improvement and growth to better meet the changing needs of the semantic web environment. Building on its predecessor, RDF Schema (RDFS), OWL has developed via a sequence of changes to produce its most current variant, OWL 2. The development process includes new functionality and bug-fixing software to enhance OWL's performance and capabilities. Separate profiles inside OWL 2—OWL 2 EL, OWL 2 QL, OWL 2 RL, and OWL 2 Full—define one of the fundamental aspects of this progression. Each of these profiles provides a distinct equilibrium between the capacity to convey ideas effectively and the ease of computational analysis, catering to a diverse array of application situations. The growth of OWL highlights its critical role in allowing knowledge representation and reasoning in the semantic web field. This positions OWL as a fundamental technology for enabling the integration and interoperability of intelligent data across different systems and domains.

The key feature of OWL is its commitment to formal semantics, which gives ontologies clear and exact meanings, making it easier for automated reasoning and inference. By using automated reasoning engines, OWL enables computers to deduce implicit knowledge, identify inconsistencies, and validate data against domain constraints. This enhances the dependability and credibility of knowledge-based systems. In addition, OWL's compatibility with open standards and concepts of interoperability promotes its use in several fields, including healthcare, life sciences, e-commerce, and cultural heritage.

3.2.1 Definition and Purpose of OWL

OWL is a formal language used primarily for ontologies creation and management. Ontologies are structured models of knowledge that include concepts, relationships, and constraints within a certain domain. Ontologies written in OWL are much more semantically and logically complex than less informal vocabularies or taxonomies. They enable the precise description of domain entities and their interconnections. The use of a well-defined structure and semantics facilitates machine comprehension and manipulation of ontological information [84].

The primary goal of the Web Ontology Language (OWL) is to establish a formalized and standardized structure for structuring and representing ontologies across the Internet. It seeks to enable descriptions of domain knowledge that are comprehensible to machines. This promotes improved interoperability, automation, and reasoning capabilities in web-based systems. OWL makes online information resources easier to understand and easier to use by allowing users to define complex conceptual frameworks, connections, and limitations in a way that is both

semantically complete and logically sound. Therefore, the primary objective of OWL is to enable machines and humans to effectively communicate, collaborate, and employ ontological knowledge; in doing so, it contributes to the development of a more intelligent and interconnected web ecosystem.

3.2.2 Classes in OWL

Classes are essential in the Web Ontology Language (OWL) since they provide the foundation for organizing and defining entities in ontologies. OWL classes serve the purpose of categorizing ideas, as well as providing important reasoning and inference skills that are essential for knowledge representation and semantic web applications.

Defining Classes and Subclasses

In order to formally represent domain knowledge and concepts, defining classes and subclasses in the OWL is a fundamental aspect of ontology architecture. The OWL facilitates class definitions and their interrelationships. The "*subClassOf*" and "*equivalentTo*" properties denote the combination of subclass, necessary, and equivalent restrictions. Necessary conditions specify the minimum requirements for class membership, while sufficient conditions ensure that all instances that meet these requirements are indeed class members. The "*subClassOf*" relation states that all instances of the subclass are also instances of the superclass. The "*equivalentTo*" relation, in contrast, asserts that two classes possess identical definitions and semantics, which translates to the same instances and features.

Figure 3.11 shows the hierarchical arrangement of classes, wherein "*Mammal*" is a subordinate category of "*Animal*" and "*Cat*" is a subordinate category of "*Mammal*" [79].

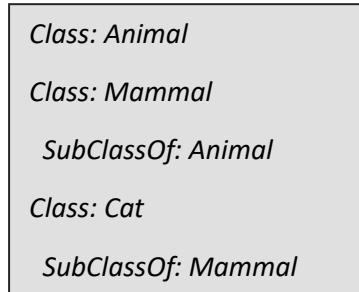


Figure 3.11: Example of Class Hierarchy in OWL

Class "*Human*" in Figure 3.12 is defined as a subclass of "*Mammal*" with "*has Brain*." It may be defined using both required and sufficient requirements.

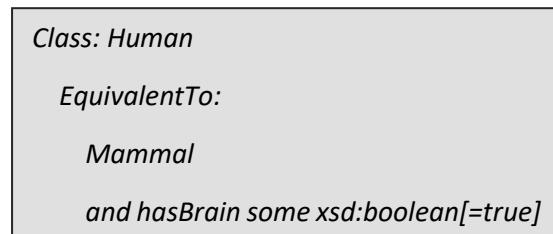


Figure 3.12: Example of necessary & sufficient restriction

Enumerated Classes

OWL integrates enumerated classes, a concept that requires the explicit enumeration of individuals to define a class. This facilitates the representation of classes comprising a restricted number of individuals, thereby providing a more accurate and comprehensive depiction. For example, we illustrate in Figure 3.13 the "*PrimaryColors*" class, which includes the elements "*Red*," "*Blue*," and "*Yellow*" [79].

<i>Class: PrimaryColors</i>
<i>EquivalentTo: {Red, Blue, Yellow}</i>

Figure 3.13: Example of Enumerated Class in OWL

Intersection and Union of Classes

Beyond rudimentary class hierarchies, OWL facilitates the construction of complex class expressions through the implementation of class intersection and union. The process of intersection involves the consolidation of multiple classes in order to generate a novel class consisting of individuals who are members of all the original classes. Conversely, the union establishes a category comprising individuals from each of the merged classes. Figure 3.14 demonstrates how the intersection class "*CarnivorousMammal*" fuses instances of "*Mammal*" with those that consume "*Meat*," while the union class "*DomesticPet*" combines instances of "*Pet*" with whose owner is a "*Person*" [79].

<i>Class: CarnivorousMammal</i>
<i>EquivalentTo: Mammal and eats some Meat</i>
<i>Class: DomesticPet</i>
<i>EquivalentTo: Pet or hasOwner some Person</i>

Figure 3.14: Example of Intersection and Union of Classes in OWL

Disjointness of classes

In OWL, disjointness of classes is defined as the absence of common instances between two or more classes, thereby representing separate and non-overlapping groups of people within a domain. Formally, a disjoint class axiom states that the intersection of the sets comprising the disjoint classes is empty. This concept is fundamental to ontology engineering, as it enables precise classification and reasoning about entities based on their distinct characteristics and properties [79].

The practical implications of disjointness in OWL are manifold:

- **Enhanced Semantic Clarity:** Disjointness assertions provide clarity about the distinctiveness of classes within an ontology, thereby preventing ambiguity and facilitating precise knowledge representation.

- **Better Reasoning:** Disjointness axioms make reasoning more efficient by reducing the search space for inference engines. This makes it possible to answer queries and check ontologies more quickly.
- **Facilitated Knowledge Integration:** Disjointness assertions help bring together different ontologies by making sure that concepts from different domains that overlap are compatible and make sense.

In Figure 3.15, we claim that the class `ex:Cat` is disjoint with both `ex:Dog` and `ex:Fish`, therefore suggesting that there are no cases that fit both `ex:Cat` and either `ex:Dog` or `ex:Fish`. Similarly, there are no instances that belong to both `ex:Dog` and `ex:Fish`.

```

@prefix ex: <http://example.org#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .

# Declare classes
ex:Cat rdf:type owl:Class .
ex:Dog rdf:type owl:Class .
ex:Fish rdf:type owl:Class .

# Assert disjointness
ex:Cat owl:disjointWith ex:Dog .
ex:Cat owl:disjointWith ex:Fish .

```

Figure 3.15: Example of Disjointness of classes in OWL

3.2.3 Properties in OWL

The study of properties in OWL (Web Ontology Language) constitutes a fundamental exploration into the structure and semantics of knowledge representation. Properties serve as crucial elements in defining relationships between entities, enabling the formalization and inference of complex knowledge structures.

Object Properties

In OWL, object properties indicate the relationships between instances or individuals. These characteristics facilitate the creation of connections between classes, allowing for the depiction of complex relationships within an ontology. The characteristics of an object's property, such as transitivity, symmetry, or reflexivity, impart additional significance to connections. The "`hasParent`" property in Figure 3.16 establishes a relationship between individuals who belong to the "`Person`" class. Its domain and range indicate that the property can only establish connections between individuals who belong to the "`Person`" class [84], [85].

ObjectProperty: hasParent

Domain: Person

Range: Person

Figure 3.16: Example of Object Property in OWL

Data Properties

Data properties represent attributes or characteristics of individuals that are associated with data values, such as strings, numbers, or dates. These properties define relationships between instances and literal values, providing a means to describe and categorize domain-specific information. Each data property can have a domain and a range, specifying the types of individuals it applies to and the types of values it can take. Also, they support cardinality constraints, which specify the minimum and maximum number of values that can be associated with an instance, and they are uniquely identified within the ontology, allowing for precise specification and referencing [84], [85].

For example, the data property "*hasAge*" (shown in Figure 3.17) has a domain of "*Person*". This indicates that the "*hasAge*" property can only be used to describe instances of the "*Person*" class. A data property's range specifies the type of data value it can be associated with. The range for "*hasAge*" is "*xsd:integer*," which means it can only be associated with integer numbers.

DataProperty: hasAge

Domain: Person

Range: xsd:integer

Figure 3.17: Example of Data Property in OWL

Inverse Object Properties

The OWL framework includes the concept of inverse object properties, which allow for the depiction of reciprocal connections between entities. An object property that establishes a connection between individual A and individual B will have an inverse property that links B back to individual A. The inverse relationship between the object characteristics "*hasParent*" in Figure 3.18 and "*isChildOf*" establishes a two-way link between individuals expressing parent-child relationships [84], [85].

ObjectProperty: hasParent

InverseOf: isChildOf

Figure 3.18: Example of Inverse Object Property in OWL

Functional and Inverse Functional Properties

Functional characteristics in OWL guarantee that each individual is associated with just one other individual via the attribute. Inverse functional properties, on the other hand, state that each individual has a distinct inverse property value [84], [85].

<i>ObjectProperty: hasSocialSecurityNumber</i>
<i>Characteristics: Functional</i>
<i>ObjectProperty: isIdentifiedBySSN</i>
<i>Characteristics: InverseFunctional</i>

Figure 3.19: Example of Functional and Inverse Functional Properties in OWL

As demonstrated in Figure 3.19, the functional declaration "*hasSocialSecurityNumber*" guarantees that every user is assigned a distinct Social Security Number. On the contrary, the inverse functional property "*isIdentifiedBySSN*" indicates that every Social Security Number is associated with a distinct individual.

Symmetric and Transitive Properties

Symmetric properties indicate that if a particular relationship exists between two individuals, A and B, then that relationship also exists between B and A. Transitive properties extend this, suggesting that if A has a connection to B and B has a connection to C, then A must also connect to C. These attributes enhance the semantics of ontology relationships [84], [85].

3.2.4 Reasoning in OWL

Reasoning is an essential component within the realm of ontologies, as it enhances the capabilities of ontological models by enabling automated deduction and inference. Reasoning plays a critical role in the OWL framework because it uncovers latent knowledge, ensures the integrity of ontologies, and facilitates the integration and investigation of more intricate data. The primary goal of reasoning in ontologies is to integrate [86]:

- **Consistency checking:** This is the process of reasoning to verify the logical coherence of ontologies. It ensures that there are no contradictions or conflicts in the stated axioms and statements. Erroneous conclusions may arise from inconsistent ontologies, and the process of reasoning aids in identifying and resolving such problems.
- **Inference Generation:** By using clearly stated axioms, reasoning enables the production of inferred knowledge. It extends beyond the expressly mentioned data to infer implicit connections, categorizations, and limitations within the ontology.
- **Question responding:** Reasoning facilitates effective question responding by providing replies that go beyond the expressly stated facts. It facilitates the logical extraction of information from the ontology, thereby improving data retrieval and exploration.
- **Ontology development:** Reasoning enables the development of ontologies by determining the consequences of modifications to preexisting knowledge. It aids in guaranteeing that changes to the ontology maintain logical coherence and avoid unforeseen repercussions.

3.2.5 OWL Reasoners and Inference Types

By automating complicated reasoning tasks, OWL reasoners improve the usability and expressiveness of ontological models. They are essential parts of knowledge representation and semantic technologies [84].

Pellet

As a high-performance OWL reasoner that facilitates OWL DL reasoning, Pellet stands out. Developed using the Java programming language, Pellet demonstrates effective reasoning capabilities that make it well-suited for the development of extensive ontologies. For instance, in instance checking, consistency checking, and classification, it employs optimized algorithms that guarantee scalability and computational efficiency. The Pellet reasoner is very reliable for many types of ontology modeling tasks because it fully supports standard OWL features like transitivity and inverse properties [86].

Hermit

Reasoner Hermit is renowned for its strict adherence to the OWL 2 specification. The software was constructed using the Hermit OWL reasoner system and is written in Java. It provides robust support for all OWL 2 reasoning characteristics, such as OWL 2 DL and OWL 2 RL. Due to its emphasis on validity and completeness, Hermit is a dependable option for applications that demand rigorous semantic inference. Its open-source characteristics and seamless integration with a multitude of ontology development tools contribute to its extensive utilization in the research and development sector [87].

FaCT++

Fact++ is notable for its C++ implementation, which enables it to offer an efficient and platform-independent environment for semantic reasoning execution. This implementation language's selection is consistent with its focus on scalability and performance, making it well-suited for applications that handle extensive ontologies and intricate knowledge structures.

By conforming to the OWL 2 specification, Fact++ guarantees compatibility with the most recent benchmarks in ontology representation. The fact that the software supports OWL 2 features, like the OWL 2 DL and OWL 2 RL reasoning profiles, makes it a reliable choice for uses that need to follow modern semantic web standards [88].

OWL offers a diverse range of inference types that provide deduction, categorization, and querying of ontological knowledge representations. We will define and examine each of the sorts of inferences that are accessible in OWL [69]:

- **Class Subsumption (Subsumption Inference):** In OWL, class subsuming inference—often referred to as subclass reasoning—is the primary kind of inference. By may accurately establish a class (subclass) within a class (superclass) by using a careful analysis of the axioms and deduced facts of the ontology. Consequently, the reasoner can deduct from the hierarchical link among "Dog," "Mammal," and "Animal" that "Dog" belongs to the class of "Animal."

- **Property Inference:** In the ontology, property inference is the process of deducing property links between individuals or classes using stated qualities and inferred information. OWL supports a variety of property inference categories, including subproperty, inverse property, symmetric property, and transitive property inference. When claiming, for instance, "*isParentOf*" as a subproperty of "*isAncestorOf*," the reasoner might deduce that an individual *x* must also be an ancestor of *y* if *x* is a parent of an individual *y*.
- **Instance Classification:** In ontology, instance categorization is the process of assigning individuals to a class. Based on their claimed class memberships and inferred knowledge, the reasoner may classify individuals into appropriate categories, taking into account their qualities and relationships. Should the ontology indicate that "*Fido*" has the attribute "*isDog*," and "*isDog*" is a subclass of "*Animal*," the reasoner can categorize "*Fido*" as an instance of the "*Animal*" class.
- **Property Characteristics (Property Characteristics Inference):** OWL allows the deduction of property characteristics such as reflexivity, symmetry, transitivity, and functionality. This inference process uses data from the ontology and explicitly defined properties to automatically determine these characteristics. For example, given that the ontology defines "*isParentOf*" as a transitive relation and states that "*Mary*" is a parent of "*Tom*," the reasoner can infer that "*John*" is also a parent of "*Tom*."
- **Consistency Checking:** Consistency checking is a critical inference type in OWL that involves determining whether the ontology contains any logical contradictions or inconsistencies.
- **Query Answering (Query Inference):** Query answering inference enables the retrieval of relevant information from the ontology in response to user queries. OWL reasoners use query inference to match user queries against asserted axioms and inferred knowledge, resulting in accurate and efficient responses. Query inference supports various query types, including class membership queries, property queries, and instance queries, enhancing the ontology's utility for knowledge retrieval and decision-making processes.
- **Property Chain (Property Chain Inference):** Property chain inference involves deducing complex property relationships through the composition of simpler property chains. OWL describes property chains using property chain axioms. This enables the reasoner to determine composite property paths using the claimed axioms and inferred knowledge. Property chain inference makes it easier to model complex relationships between people and classes, which helps with more complex semantic reasoning and querying tasks.

3.2.6 OWL API

Semantic web technologies' extensive use has raised ontology engineering's requirement for efficient tools and frameworks. For researchers and developers wishing to programmatically create and administer OWL ontologies, the OWL API has evolved as their first option [89], [90].

An extensive array of features is available for the manipulation of OWL ontologies through the OWL API, a Java-based toolkit. The software provides a variety of classes and methods that facilitate the parsing, manipulation, reasoning, and serialization of ontologies. This simplifies the development of sophisticated applications that depend on ontologies. The API is compatible with

current ontology tools and frameworks, as it adheres to the standards established by the World Wide Web Consortium (W3C) for OWL representation and reasoning. Some notable characteristics and capabilities of this product are as follows:

- The OWL API helps to handle OWL files in Turtle, RDF/XML, and OWL/XML, among other formats. The program provides parsers that could import ontologies from several sources—local files, URLs, input streams, etc. This facilitates a simple interface with various data sources.
- Programmatically generating, changing, and removing ontology elements like classes, properties, individuals, and axioms allows developers to use the OWL API to control ontologies. The API offers a wide spectrum of classes and interfaces to describe OWL constructs, therefore allowing exact control over ontological content and architecture.
- Reasoners such as Pellet, HermiT, and FaCT++ provide ontological reasoning and inference that is compatible with the OWL API. These reasoners may assist developers in responding to questions about OWL ontologies, classifying them, and confirming consistency. As such, their uses show higher semantic expressiveness and are stronger for inferencing.
- The OWL API allows the serialization of ontologies into various formats, such as RDF/XML, OWL/XML, Turtle, and Manchester Syntactic forms. The program exports ontologies to various formats, such as files, output streams, and string representations. This guarantees compatibility with other ontologies, tools, and systems.

The OWL API has various benefits when compared to alternative tools and frameworks for manipulating ontologies. These advantages include:

- The OWL API offers a wide range of features for parsing, manipulating, reasoning, and serializing ontologies, making it a viable option for ontology building.
- Java serves as the foundation for the OWL API, enabling its use across various operating systems and seamless integration with Java-based applications and frameworks. Developers benefit from a diverse ecosystem of Java libraries and development tools that allow them to create applications driven by ontologies.
- A dynamic community of developers, academics, and ontology practitioners actively supports the OWL API. The software undergoes continuous upgrades, bug corrections, and contributions from the community, which guarantees its ongoing relevance and usefulness in dynamic semantic web settings.

3.3 Knowledge Graphs

In the expansive realm of information available on the internet, the skill to effectively arrange, retrieve, and comprehend material is of utmost importance. Conventional techniques for arranging data often fail to capture the intricate connections and meanings that exist in real-world information. Developers have developed knowledge graphs as a strong paradigm to address the challenges of organizing and exploiting organized and unstructured data in a linked and relevant manner [91], [92].

3.3.1 Definition and Conceptual Framework

A knowledge graph is a structured representation of knowledge that collects and organizes things, ideas, and their connections. Usually, we record it in graph format. Knowledge graphs differ from standard databases or ontologies by not following a strict structure. Instead, they use a flexible and dynamic approach to describing data. Knowledge graphs have the ability to include several forms of information, including factual knowledge, contextual correlations, and inferential connections, due to their flexibility [93].

Knowledge graphs are not just storage spaces for data; they function as robust frameworks for representing knowledge that facilitate sophisticated reasoning, inference, and exploration. Knowledge graphs enable the execution of intricate queries, entity resolution, and knowledge inference using graph-based algorithms and semantic technologies. This allows for the extraction of useful insights from linked data sources.

The core components of knowledge graphs are entities, relationships, and characteristics.

- **Entities:** Entities are the fundamental components of knowledge graphs. They represent tangible items, abstract ideas, or specific examples that have unique identities in the actual world. The graph structure often represents entities as nodes, which have a key function in organizing and structuring the underlying knowledge. Entities within a knowledge graph might include a wide range of elements, such as persons (e.g., Albert Einstein), organizations, locations (e.g., Eiffel Tower), events, concepts, and abstract ideas (e.g., democracy). Identifying and categorizing items is critical for constructing the semantic basis of the knowledge graph and allowing for meaningful data representation and retrieval.
- **Relationships:** Relationships establish the connections and linkages between things in the knowledge graph, representing the semantic interconnections and dependencies that exist in the actual world. Relationships include the diverse range of interactions, dependencies, and linkages that define the connections between things. Every relationship is defined by a certain type or label, indicating the nature of the link between items and providing context for understanding their semantic importance. Relationships play an important role in illustrating the intricate interdependence and interconnections that exist within the field of knowledge. The knowledge graph enables the display of extensive and interconnected information.
- **Attributes:** Attributes serve to give more descriptive details on entities or relationships included in the knowledge graph, including their attributes, traits, and metadata. They enhance the depiction of entities and connections, enabling a more comprehensive and nuanced comprehension of the underlying data. Features may comprise a wide range of information, such as textual descriptions, numerical values, category designations, temporal features, and geographical qualities. By including characteristics, the knowledge graph becomes more expressive and contextual, allowing for more advanced data modeling, querying, and analytical capabilities.

The combination of these components creates a complex and linked network of knowledge that encompasses the meaning, context, and connections within the data domain. Knowledge graphs

provide effective data management, discovery, and analysis by arranging information in an organized and linked way. This empowers users to extract important insights and make educated choices in many domains and applications.

3.3.2 Historical Evolution of Knowledge Graphs

Knowledge graphs have evolved over time to reflect the increasing complexity and sophistication of efforts to arrange and link data. Knowledge graphs, which originated as early artificial intelligence and database ideas in the 20th century, have evolved through major turning points such as the explosion of massive data and the development of semantic web technologies.

Early Foundations: Relational Models and Semantic Networks (Pre-2000s)

The development of relational databases laid the foundation for knowledge graphs. In 1970, Edgar Codd's relational model [94] laid the groundwork for the organization of data in tables defined by relationships. Peter Chen's entity-relationship model (ERM) in the 1970s further developed this concept [95]. ERMs introduced the concept of entities, or real-world objects, and the relationships between them, laying the foundation for the graphical representation of knowledge [96], [97].

The Rise of Knowledge Graphs (2000s - 2010s)

With the Semantic Web, the Resource Description Framework (RDF) became a standardized format for online data expression and connection. By expressing knowledge using subject-predicate-object triples, RDF laid the foundation for building knowledge graphs. The process of transforming large amounts of unorganized data into interconnected open datasets was started by projects like DBpedia (2007) [98] and Linked Data, which contributed to the growth of the semantic web.

Knowledge graphs have evolved from simple data structures to include advanced semantics and inferential capabilities. Ontologies are crucial for the formalization of knowledge relevant to a particular domain and for establishing the meaning of things and connections. Reasoning engines and inference mechanisms increased the capacity of knowledge graphs to draw implicit knowledge and form conclusions based on explicit facts.

Graph Databases and Knowledge Graph Construction (2010s)

Graph databases provide scalable solutions for storing and accessing vast knowledge graphs. Technologies such as Neo4j [99], AllegroGraph [100], and Amazon Neptune [101] helped provide effective representation and traversal of graph-structured data. Knowledge graph building approaches emerged from the combination of hand curation and automated extraction methods to create comprehensive knowledge libraries.

Industry Adoption and Applications (2010s - Present)

Several sectors and applications have widely used knowledge graphs. Google, Microsoft, and Facebook have incorporated knowledge graphs into their platforms to improve search, recommendation algorithms, and natural language processing. Google introduced the Knowledge Graph in 2012 [102], Microsoft developed the Microsoft Academic Graph in 2016 [103], and

Facebook implemented the Social Graph [97]. The healthcare, banking, and e-commerce industries have used domain-specific knowledge graphs to optimize the process of integrating data, doing analytics, and providing decision assistance.

The development of knowledge graphs throughout history has been characterized by ongoing innovation and improvement. Knowledge graphs have revolutionized the fields of information management and artificial intelligence, from their early conceptualizations to their current uses. The resolution of critical obstacles and the application of developing technology have positioned knowledge graphs to assume a progressively crucial role in the representation of knowledge, logical thinking, and decision-making processes.

3.3.3 Key Components of Knowledge Graphs

Knowledge graphs consist of essential elements that effectively organize and depict complex data landscapes, facilitating the integration, analysis, and exchange of data [104]. These key components —[entities](#), [relationships](#), [attributes](#), literals, and schema/ontologies—form the foundation of knowledge graphs.

Relationships

Common relationship types include:

- Is-A (hyponymy/hypernymy): This relationship specifies a hierarchical association, where one entity is a subclass (hyponym) of another (hypernym). For example, "cat" is-a "mammal."
- Has-A (meronymy/holonymy): This relationship captures the "*part-of*" association. For instance, "wheel" has-a "car."
- Other Relations: To capture more nuanced connections, numerous domain-specific relationships can be defined. Examples include "wrote" (author-book), "directed" (director-movie), or "*located in*" (city-country).

Literals and Data Types

Literals express the exact values associated with characteristics or properties. They possess the capacity to incorporate a diverse array of data categories, including text, numbers, dates, and URLs. The definition of data types uniformly understands the information in the knowledge graph.

Schemas and Ontologies

Even though they are not exact components, schemas and ontologies play a fundamental role in structuring knowledge graphs. A schema is a blueprint defining the allowed data types, characteristics, relationships, and entities for the KG organization. Ontologies, which are formal, machine-readable standards, capture domain knowledge and connections between items within a given domain. By creating a common language and set of restrictions, they help a wide range of KGs be interoperable.

3.3.4 Data Integration for Knowledge Graph

Creating and maintaining a complete knowledge graph is a complicated task that requires coordinating different formats, structures, and semantics, as well as adding data from different sources. The integration of data is critical in guaranteeing the precision, comprehensiveness, and uniformity of knowledge graphs, thereby empowering them to function as reliable information sources for a multitude of applications [96], [105], [106].

Difficulties and Opportunities

The successful integration of data into knowledge graphs requires addressing several challenges. One major challenge is data heterogeneity—that is, the presence of different data forms, structures, and meanings. A strong focus on data cleansing and quality testing helps to guarantee the consistency and correctness of integrated data. Constructing a consistent representation of entities in the knowledge graph depends critically on entity matching. It entails identifying and linking entities from various datasets that have similarities. Relationship extraction is an essential part of building a complete and integrated knowledge graph. It involves identifying important links between items in text or structured data.

Despite these challenges, the incorporation of data into knowledge graphs presents several opportunities for progress and innovation. There are ongoing efforts to provide advanced frameworks and approaches for data integration in order to address the challenges of combining different types of data. Techniques for aligning ontologies are being improved to simplify the integration of information from different knowledge sources. We are using machine learning methods to deduce absent links and improve the overall coherence of the knowledge graph.

Data Integration Frameworks and Methods

In the context of analysis, decision-making, and knowledge discovery, data integration is the act of combining data from several sources to provide a single viewpoint. Data integration in knowledge graphs helps to create thorough and linked knowledge libraries covering many spheres and points of view. Still, the integration of data for Knowledge Graphs (KGs) offers certain difficulties [107].

Different schemas and data architectures complicate schema heterogeneity—the process of creating mappings between entities and relationships across many data sources [108]. Differences in data formats, nomenclature standards, and entity representations can lead to errors and inconsistencies in the Knowledge Graph, also known as data inconsistency. Knowledge Graph dependability and trustworthiness may suffer from data quality problems, such as erroneous, incomplete, or outdated data from source systems. Furthermore, scalability is an important topic, as the integration of large volumes of data necessitates the use of efficient frameworks and techniques to properly control the complexity and size of the information.

Many models provide methodical ways for adding data to KGs to help overcome these challenges. Retrieving data from source systems, standardizing it, and then loading it into the KG makes up the ETL (Extract, Transform, Load) method. Frameworks like Apache Kafka and Luigi enable ETL processes. By putting data into a staging area before transformation, the ELT (Extract, Load,

Transform) version offers parallel processing and flexibility. Cloud systems like Amazon Web Services (AWS) and Google Cloud Platform (GCP) include tools for ELT pipelines. Data lakes act as centralized stores for raw data from multiple sources, providing a staging ground for later integration into the KG and transformation. Apache Spark and other frameworks help to handle massive amounts of data housed in data lakes. Furthermore, certain systems, such as Google Cloud and OpenKE, assist in managing large volumes of data. The Knowledge Graph Construction Service provides tools designed specifically for knowledge graph maintenance and construction, including data integration features.

Various methods address the challenges of data integration for KGs. Schema matching techniques, such as entity matching and schema alignment, enable the mapping of data into the KG schema by identifying corresponding entities and relationships from a variety of data sources. Data cleansing and normalization methods are employed to resolve inconsistencies in data formats and values, thereby guaranteeing a consistent structure for entities and relationships. Instruments such as Trifecta Wrangler and OpenRefine can help with data cleansing tasks. Entity linking techniques connect entities within the KG to external knowledge bases or databases, thereby improving the KG's interoperability and enriching it with additional information. Tools like the Wikidata Query Service and DBpedia Spotlight can facilitate entity linking. Probabilistic and machine learning techniques, such as TensorFlow and PyTorch, can improve data integration accuracy, resolve data uncertainty, and identify relationships between entities.

These frameworks and methods facilitate the efficient management of data integration for Knowledge Graphs, thereby facilitating the development of dependable and comprehensive knowledge repositories.

Triplestores

Triplestores are a critical technology in semantic data administration, providing efficient storing and querying capabilities for RDF data. As the amount and complexity of structured data increases, triplestores become more important. They let companies use semantic technologies for knowledge representation and integration [69], [109], [110].

RDF triples are the fundamental notion of triplestores. They encapsulate semantic information using subject-predicate-object declarations. These triples are the fundamental units of knowledge representation in triplestores, offering a versatile and expandable framework for defining connections between items. Triplestores adhere to the concepts of the Semantic Web, which include using open standards and ontologies to enable the exchange of data and enhance its meaning.

A triplestore's architecture often consists of many essential components, each fulfilling a distinct function in the storing, indexing, and querying of RDF data. The storage component oversees the actual manifestation of RDF triples, enhancing storage efficiency and access patterns. Indexing structures are used to expedite rapid searches and retrievals of triples based on different criteria, such as subject, predicate, or object. Query processing engines run SPARQL queries on the stored RDF data, allowing for intricate graph-based inquiries and analytics.

A range of technologies and frameworks, including both open-source solutions and commercial products, implement triplestores. Apache Jena, Stardog, and RDF4J are examples of open-source triplestores that provide adaptable and expandable solutions for handling RDF data in various settings. Commercial triplestore solutions provide extra functionalities and assistance, specifically designed for corporations and organizations with strict demands for speed, scalability, and support.

Retrieving and evaluating semantic data stored in RDF format is an essential operation in triplestores, enabling users to execute complex queries and gain meaningful insights from their data. The agreed query language for RDF data is SPARQL, often referred to as SPARQL Protocol or RDF Query Language. For querying triples and graph patterns, it provides a thorough vocabulary and semantics. SPARQL searches simplify many operations, such as basic graph pattern matching, filtering, aggregation, and inference.

Triplestores are critical for data integration and interoperability because they allow companies to combine various types of data sources into a single semantic representation. By following open standards and ontologies, triplestores provide effortless integration and data interchange across different systems and domains. They function as primary storage and retrieval systems for consolidated knowledge graphs, facilitating comprehensive analysis, exploration, and search across different domains.

Knowledge Graph Embedding

Knowledge graph embedding (KGE) methods revolutionize knowledge representation by enabling the transformation of symbolic information into continuous vector representations. KGE enables efficient processing and reasoning over large-scale knowledge graphs by representing items and connections in low-dimensional vector spaces [111], [112], [113].

The notion of representation learning is at the heart of KGE, which seeks to incorporate semantic information into continuous vector representations. KGE approaches use the topology and structure of knowledge graphs to acquire embeddings that maintain semantic similarities and links among entities and relationships. KGE, or Knowledge Graph Embedding, allows for the extraction of underlying meanings in knowledge graphs. This technique facilitates many tasks, such as predicting links between entities, classifying entities, and completing knowledge graphs.

Various cutting-edge methodologies have been suggested by researchers for knowledge graph embedding, each presenting distinct methods for acquiring embeddings of items and connections. Translational models, such as *TransE*, *TransH*, and *TransR*, represent relationships as translations between entity embeddings in the vector space. Semantic matching approaches such as *DistMult* and *ComplEx* use bilinear operators to capture intricate interactions between items and connections. Neural network models like *ConvE*, *ConvKB*, and *NTN* use deep learning structures to acquire hierarchical representations of things and connections.

To assess the effectiveness of KGE approaches, suitable assessment measures must be established. Typical assessment metrics for Knowledge Graph Embedding (KGE) consist of link prediction measures, including hits@N, Mean Reciprocal Rank (MRR), and Mean Average Precision (MAP). These metrics evaluate the capacity of KGE models to anticipate absent or

unobserved connections in the knowledge graph. In addition, entity classification tasks use measures such as precision, recall, and F1-score to assess the accuracy of entity embeddings.

A wide range of fields and scenarios, including semantic search, recommendation systems, question answering, and knowledge discovery, use Knowledge Graph Embedding (KGE) approaches. KGE enables applications to use the comprehensive semantic framework of knowledge graphs in order to enhance the accuracy and effectiveness of information retrieval, recommendation systems, and decision-making processes. It does this by instructing apps on how to describe entities and connections in a continuous manner. Industries such as healthcare, e-commerce, banking, and scientific research have used information Graph Embedding (KGE) techniques to extract useful insights and information from structured data.

Although KGE approaches provide benefits, they also encounter many problems and restrictions. Scalability problems occur when working with extensive knowledge graphs that consist of millions or billions of items and interactions. These problems require the use of effective training methods and distributed computing frameworks. Limited data and an uneven distribution of information in knowledge graphs make it difficult to obtain precise representations of infrequent or unfamiliar items and connections. Furthermore, effectively managing dynamic and developing knowledge graphs necessitates the use of resilient methods for gradual learning and adjustment.

Knowledge Integration Pipelines

The process of integrating data for knowledge graphs often follows a structured pipeline, ensuring a systematic and efficient way of absorbing information from different sources. The pipeline often consists of many phases.

At this stage, we first gather data from many sources, including structured databases, unstructured text documents, and web APIs. After collection, we prepare and cleanse the data to address any issues related to data quality, disparities, or missing information. This ensures that the data is in a suitable condition for further work.

Entity linking and resolution aim to provide a cohesive representation of entities by connecting related entities from different databases. This stage is essential for establishing a coherent and integrated dataset. The process of constructing a knowledge graph entails using the integrated data to create nodes that represent things and edges that reflect connections. This establishes the fundamental framework of the knowledge graph.

Finally, the knowledge graph enhancement entails integrating missing information, rectifying discrepancies, and broadening connections. The ongoing enhancement process guarantees the accuracy, comprehensiveness, and use of the knowledge graph across diverse applications.

Continuous Integration of Data

Knowledge graphs evolve in response to the introduction of new data. Ongoing data integration procedures are critical for preserving the precision, comprehensiveness, and applicability of knowledge graphs. As part of this process, data sources are constantly checked for any new additions or updates. Data transformation rules are also changed to reflect changes in data

formats or structures. The knowledge graph is checked for errors and fixed on a regular basis, and entity linking, and relationship extraction models are updated to reflect changing data.

3.3.5 Knowledge Extraction Techniques

Information extraction refers to the transformation of unprocessed and raw data into structured and meaningful information. The technique involves deriving significant insights, correlations, and patterns from a substantial volume of sometimes disorganized data. This intricate approach encompasses a diverse range of strategies and techniques, all of which contribute to the retrieval of valuable information [114], [115].

Natural Language Processing (NLP)

Fundamental in nature, NLP methods extract information from textual data. These strategies include:

- **Named Entity Recognition (NER):** This process involves grouping and categorizing entities found in text into predetermined categories, such as people, companies, sites, dates, and so on. NER methods have changed over time from rule-based systems to machine learning models and, more recently, deep learning methods like BERT (Bidirectional Encoder Representations from Transformers), which made entity recognition much more accurate by using context in both directions.
- **Relation Extraction:** After identifying entities, the next step is to determine the connections between them. Using supervised learning—where classifiers are trained from a labeled dataset—relationship extraction is accomplished. Typical methods include semantic role labeling and dependency parsing. Particularly when labeled data is limited, unsupervised and semi-supervised techniques—including clustering and bootstrapping—are frequently used.
- **Coreference Resolution:** This method of resolution helps one find when many phrases refer to the same thing within a text. These days, contextual embeddings and neural networks are used to improve the accuracy of coreference resolution while keeping the completeness and coherence of knowledge graphs.

Information Extraction (IE)

Techniques for information extraction (IE) concentrate on spotting and gathering certain kinds of data from text, including entities, relationships, and events. Named entity recognition (NER), a fundamental component of IE, facilitates the finding and categorization of text's referenced named entities. Relation extraction seeks to find and extract semantic links between entities—that is, "*born-in*" or "*works-for*" links between individuals and companies. Event extraction finds and compiles structured representations of events detailed in text, including temporal information, event triggers, and participants.

For instance, financial news analysis uses IE methods to extract important data like market movements, stock prices, and firm names from news items. IE lets traders and analysts track market changes, spot new trends, and make wise investment choices by automatically generating structured representations of financial events.

Machine Learning (ML) Approaches

Leveraging statistical models and algorithms, machine learning (ML) methods automatically identify patterns and correlations from data, thereby allowing information extraction from many sources. Supervised learning techniques include classification and regression training models for labeled job data, as well as sentiment analysis and named entity identification. Without labeled examples, unsupervised learning systems—such as topic modeling and clustering—discover latent patterns and structures in data. For tasks like language modeling, text production, and document summarizing [116], deep learning models—transformers and recurrent neural networks—RNNs—have shown potential.

In social media analytics, for example, ML methods draw insights from user-generated material such as tweets, posts, and comments. By automatically categorizing social media posts as good, negative, or neutral, sentiment analysis algorithms let companies track consumer mood and react immediately to comments. Latent Dirichlet Allocation (LDA) and other topic modeling methods help marketers find common themes and subjects in social media dialogues, therefore guiding their message to reflect newly developing trends.

Challenges and Considerations

Data quality, scalability, domain-specificity, and interpretability are among the various difficulties and issues knowledge extraction methods must address. Accurate knowledge extraction suffers from noise, uncertainty, and inconsistency found in unstructured data sources. Scalability issues arise when dealing with large amounts of data, necessitating the use of effective algorithms and distributed computing systems. Different fields may have domain-specific information and terminology, which calls for particular models and tools for efficient knowledge extraction. Building confidence and understanding in the decision-making process also depends on guarantees of the interpretability and transparency of information extraction methods.

3.3.6 Representing Knowledge Graphs

Knowledge graphs—complex visual representations of linked data—rely on certain representations to properly express relevance and links. Whether the articulation of things and their interactions is clear and understandable depends in substantial part on the selected representation.

Graphical Representations

The most powerful methods for displaying the complex structures and interactions found in knowledge graphs are graphical representations. Graphical representations help humans perceive complicated data by converting abstract ideas and connections into visual aspects [117], [118].

Many methods create graphic representations of knowledge graphs, each offering a unique way to view the underlying data. Node-link diagrams let users visually investigate the connectivity of elements inside the network [119] by representing things as nodes and connections as links between nodes. Matrix representations, which show the adjacency matrix of the knowledge graph, provide a compact summary of entity connections [120]. Semantic embeddings allow one

to see semantic clusters and network similarity by projecting objects and connections into low-dimensional vector spaces.

Knowledge graphs' graphic depictions find utility in knowledge discovery, data exploration, and decision support, among other fields and use cases. Graphs enable bioinformatics researchers to see how proteins interact with one another, how genes regulate these connections, and how metabolic pathways function [121]. Finding biomarkers and pharmacological targets is, therefore, simpler. Graphical representations allow social network researchers to explore social networks, influence propagation, and community discovery, thereby allowing marketers and sociologists to investigate online communities and social behavior [122], [123]. Graphical representations of user-item interaction networks help to visualize recommendation systems, thereby supporting customized suggestions and content discovery.

Graphical depictions of knowledge graphs provide various issues and concerns, notwithstanding their benefits, including scalability, layout optimization, and information overload. As scalability problems arise, visualizing large-scale knowledge graphs with millions or billions of things and connections calls for effective rendering and interaction approaches. Layout optimization methods aim to improve the readability and aesthetics of graphical representations by minimizing edge crossings, maximizing node spacing, and preserving the structural integrity of the graph. Techniques such as clustering, filtering, and summarizing become required to focus on relevant subsets of the network when the graph contains too much information for efficient display.

RDF and Triple-Based Representation

[RDF](#), or Resource Description Framework, often represents knowledge graphs in a triple-based style. RDF arranges data into subject-predicate-object triples, each of which expresses a claim about a connection between two entities. For example, we may display "*Alice knows Bob*" as a triple: (Alice, knows, Bob). This triple-based approach enables highly flexible and expandable knowledge graphs capable of encapsulating intricate linkages and semantics [124].

Property Graph Representation

In graph databases, the Property Graph model is an adaptable and user-friendly representation utilized to capture and categorize intricate relationships and attributes within a connected dataset. In contrast to conventional relational databases, which are based on tables and rows, the Property Graph model capitalizes on the characteristics of graphs—properties, edges, and nodes—instead [124].

In this model, nodes serve as placeholders for entities like places, people, or concepts, and edges serve as indicators of their relationships. The Property Graph is distinguished by the presence of properties, which are key-value pairs that are linked to both nodes and edges. These attributes enable the graph to retain supplementary data pertaining to entities and connections, thereby transforming the graph into an extensive repository of interrelated information.

For example, consider a social networking site. A Property Graph is a graphical representation in which nodes represent individuals and edges represent relationships such as "*friendship*" or "*work*" (see Figure 3.20). Properties such as "*name*," "*age*," or "*job title*" can be linked to both

nodes and edges. The adaptable and articulate nature of this representation enables the construction of models for practical situations, rendering Property Graphs appropriate for an extensive array of uses, including supply chain management and social networks.

The Property Graph model demonstrates exceptional performance in situations where the interconnections among entities are equally crucial to the entities themselves and where supplementary attributes or metadata have a substantial impact. Graph databases, such as Neo4j [99], which are constructed using the Property Graph model, take advantage of this representation to provide streamlined query functionalities. This empowers users to navigate the graph and reveal significant observations within the interconnected network of data.

```
// JSON-like Property Graph representation
{
  "nodes": [
    { "id": "Person1", "label": "Person" },
    { "id": "Person2", "label": "Person" }
  ],
  "edges": [
    { "id": "friendship", "label": "hasFriend", "source": "Person1", "target": "Person2" }
  ]
}
```

Figure 3.20: JSON-like Property Graph representation

3.3.7 Domain-Specific Knowledge Graphs

Domain-specific knowledge graphs (DSKGs) have surfaced as highly effective instruments for structuring and representing knowledge that is particular to a given industry or domain. They empower organizations to establish all-encompassing repositories of knowledge by documenting domain-specific concepts, relationships, and semantics. These repositories streamline the processes of information retrieval, analysis, and decision-making [125].

The domain specificity principle, which prioritizes tailoring knowledge representation to a specific industry or domain, forms the foundation of DSKGs. Domain-specific ontologies, which define the entities, properties, and relationships pertinent to the domain, form the foundation of DSKGs. Through the utilization of domain-specific vocabularies and semantics, they guarantee the captured knowledge's consistency, interoperability, and relevance.

Among other important processes, DSKG building calls for the establishment of an ontology, data integration, enrichment of semantic information, and creation of knowledge graphs. Working collaboratively, data scientists, knowledge architects, and domain specialists build the domain ontology. This ontology contains the knowledge graph's basic schematics. Data integration methods are used to help absorb and harmonize many data sources, including structured databases, unstructured documents, and outside knowledge bases. Adding semantic relationships, annotations, and extra metadata to the original data is how semantic enrichment procedures add domain-specific semantics and contextual information to the knowledge graph.

Scientific research, healthcare, finance, and e-commerce are among the domains and use cases that utilize DSKGs. DSKGs facilitate the incorporation of patient data from clinical trial repositories, electronic health records (EHRs), and medical imaging databases. This integration supports the implementation of personalized medicine, disease modeling, and healthcare analytics [120], [121]. DSKGs enable financial institutions to detect fraudulent activities, assess risk exposure, and analyze market trends through the integration of financial data, regulatory filings, and market news. They facilitate consumer segmentation, personalized recommendations [122], and product categorization in e-commerce through the integration of product catalogs, user behavior data, and external market data [123].

Despite their benefits, DSKGs face several challenges and considerations, including data quality, scalability, and domain expertise. It is critical for the knowledge graph's precision and credibility to guarantee the quality and dependability of the underlying data sources. When dealing with large volumes of data or combining data from multiple sources, scalability issues arise and necessitate the deployment of efficient systems for indexing, storing, and querying. For ontology construction, semantic enrichment, and the interpretation of domain-specific semantics—all of which depend on domain expertise—data scientists and domain specialists must work together.

Chapter 4

Problem – Dataset – Tools

4.1 Description of the problem

Knowledge graphs function as organized information repositories, depicting items and their connections in a graph-based fashion. Although knowledge graphs are useful, they often lack completeness, which impairs their usefulness in tasks involving knowledge representation and inference. The task of connection prediction is a crucial effort to deduce missing relationships and enhance the comprehensiveness of knowledge graphs [126].

4.1.1 Relation Prediction

In knowledge graphs, the challenge of relation prediction, illustrated in Figure 4.1, involves the difficulty of anticipating absent links between nodes using available information. The main goal is to predict the probability of the presence of relationships between pairs of nodes that are not explicitly included in the knowledge graph $G = (E, R)$, where E denotes the collection of entities and R represents the collection of relationships. In a formal context, the task is to anticipate the relationship (denoted as r) between two entities (denoted as e_1 and e_2) or determine whether there is no relationship between them. The entities e_1 and e_2 belong to a set E , while the relationship r belongs to a set R [127].

Methods

Experts have presented a wide range of algorithmic approaches for the goal of anticipating connections within knowledge graphs: symbolic methods, statistical models, deep learning techniques, and so forth. Logic inference rules and explicit knowledge representation are important parts of symbolic approaches that help predict relationships. These include rule-based reasoning and logic inference. On the other hand, by means of the analysis of available data, statistical models—including matrix factorization and probabilistic graphical models—gain implicit knowledge about relationships and objects through the analysis of available data. As a result, these models predict missing links using these representations. Furthermore, there are two types of deep learning approaches that learn to provide exact predictions by using the structure of graphs and connections: deep relational learning models and graph neural networks (GNNs) [128], [129], [130].

Evaluation

The evaluation of connection prediction methods consists of determining their ability to correctly project missing connections within a knowledge graph. Accuracy, recall, F1-score, area under the receiver operational characteristic curve (AUC-ROC), and mean average precision (MAP) define standard evaluation criteria. These tests provide important new perspectives on the efficacy of predictive models across many fields and assess their accuracy, comprehensiveness, and capacity to differentiate between them [131].

Obstacles and constraints

Predicting relations within knowledge graphs encounters several obstacles, such as limited data, semantic variation, scalability concerns, and data noise and uncertainty. To begin with, the limited number of links between entities in sparse data hampers the accuracy of prediction models trained on only a fraction of the available data. Additionally, the different types of links and their interpretation in different domains lead to semantic heterogeneity, complicating the creation of universally applicable predictive models. Moreover, scalability becomes an issue when dealing with extensive knowledge graphs that consist of millions or billions of entities and interactions. This challenge necessitates the use of efficient algorithms and distributed computing frameworks to manage the computational complexity. Finally, noise and ambiguity in the data lead to mistakes and inconsistencies in observed relationships, which, in turn, affect the dependability and accuracy of prediction models [130].

Cutting-edge methods

Recent advancements in relation prediction for knowledge graphs have resulted in the creation of cutting-edge algorithms that effectively tackle the issues and limits described before. The methods include knowledge graph embeddings, graph neural networks, attention mechanisms, and multi-task learning methodologies. Knowledge graph embeddings are procedures that provide concise vector representations of entities and relationships within a knowledge graph. Specifically, these representations aim to capture both the semantic similarities and relational semantics between objects and relationships. Graph neural networks are an expansion of conventional neural networks that analyze data using graph topologies. They are capable of acquiring comprehensive knowledge about node and edge representations, enabling them to make accurate connections. Attention mechanisms enhance the accuracy and interpretability of models by allowing them to concentrate on relevant sections of the graph during prediction. Multi-task learning methods use extra tasks, such as entity prediction and connection categorization, to help relation prediction models do better and generalize better [129], [130].

Applications and Use Cases

Relation prediction has several applications and use cases in diverse disciplines, such as the semantic web, biomedical informatics, natural language processing, and social network research. In the semantic web, relation prediction makes it possible to align ontologies, match schemas, and add to ontologies, thereby making it easier for different information sources to work together and share information. Similarly, biomedical informatics uses relationship prediction to facilitate drug development, predict protein-protein interactions, and analyze disease-gene associations. These applications contribute significantly to advancements in personalized medicine and precision healthcare. In natural language processing, relation prediction also makes tasks like answering questions, linking entities, and extracting relations easier. This improves the accuracy and depth of textual data comprehension. Finally, relationship prediction in social network analysis aids in the identification of concealed links, community detection, and anticipation of user interactions, thus augmenting our understanding of social networks and online communities [129], [130].

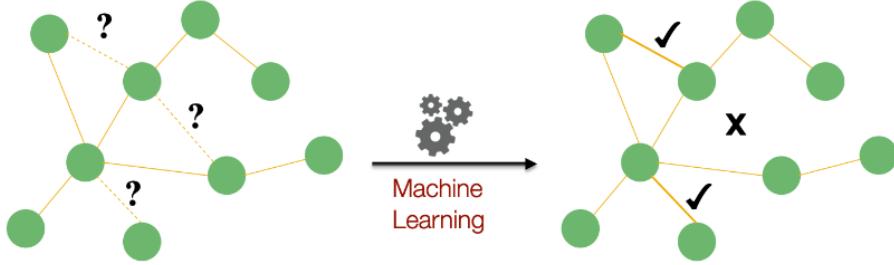


Figure 4.1: The link prediction problem [132]

4.1.2 Triple classification

In knowledge graphs, the triple classification challenge entails giving category labels to triples based on their semantic implications. The primary objective inside a vast and varied knowledge graph is to accurately categorize triples into pre-established classifications, such as kind, connection, or attribute [133].

Mathematical Formulation

A KG can be mathematically represented as a set of triples [133]:

$$KG = \{(h, r, t) \mid h \in E, r \in R, t \in E\}$$

where:

- E is the set of entities in the KG
- R is the set of relations in the KG
- (h, r, t) represents a triple, where h is the head entity, r is the relation between h and t , and t is the tail entity.

The goal of triple classification is to develop a function f that maps a triple (h, r, t) to a binary label indicating its validity within the KG:

$$f: (E \times R \times E) \rightarrow \{0, 1\}$$

where:

- 0 denotes an invalid triple (not present in the KG).
- 1 denotes a valid triple (present in the KG).

Difficulties and unresolved issues

The efficacy and usability of triple classification encounter major obstacles and unresolved questions in real-world contexts. Understanding and addressing these difficulties is critical for progressing to the cutting-edge in knowledge graph completion and improving the resilience and dependability of triple classification techniques [129], [130], [134].

- **Data Sparsity and Incompleteness:** Knowledge graphs frequently face challenges related to data sparsity and incompleteness, which result in the omission or inadequate representation of various entities and connections. The insufficient density of the data hinders the ability of triple classification models to efficiently generalize and generate accurate predictions,

particularly for infrequent items and connections. In order to tackle the problem of data sparsity, it is imperative to employ knowledge graph completion techniques that possess the capability to precisely infer absent connections. Domain-specific expertise and external information sources can accomplish this objective.

- **Class Imbalance and Skewed Distributions:** Triple classification frequently encounters the issue of class imbalance when specific connections exhibit a significantly higher frequency in the knowledge graph than others. The aforementioned imbalance gives rise to imbalanced distributions within the training data, which subsequently causes models to prioritize frequent associations over rare ones. In order to rectify class imbalance, it is imperative to employ strategies including algorithmic modifications, data augmentation, and sampling techniques. These strategies balance the representation of diverse connections and reduce prejudice against dominant classes.
- **Noisy and Erroneous Data:** The use of knowledge graphs to extract and integrate data from disparate sources may lead to the introduction of inaccurate or erroneous information. Triple classification models may become perplexed, and their efficacy may be compromised by erroneous data, resulting in incorrect predictions and flawed inferences. In order to detect and mitigate noise in knowledge graphs, quality assurance measures, data cleansing and validation procedures, and anomaly detection methods are required.
- **Cold Start Problem for New Entities and Relationships:** The frigid start problem arises when triple classification models confront novel items or connections that were not present in the training data. Predicting outcomes in such scenarios may prove challenging for existing models due to their limited understanding of the attributes and meanings of the novel entities or connections. We must implement domain adaptation, zero-shot learning, and transfer learning to overcome the frigid start challenge. These methodologies enable the extrapolation of knowledge from established entities and associations to novel ones, even in the absence of explicit training data.
- **Interpretability and Explainability:** Triple classification models' interpretability and explainability have a significant impact on comprehension and confidence in their predictions. This is especially true in high-stakes scenarios where openness and responsibility take center stage. Black-box models lack a significant reason for their judgments, even if they can provide accurate forecasts. As a result, users encounter difficulties in comprehending the deduced correlations and the fundamental rationales behind them. In order to make triple classification clearer and easier to understand, it is necessary to use methods that let you look at how the model works, figure out how important different features are, and see how much uncertainty there is. These methodologies aid in elucidating the reasoning behind the model's prognostications and bolster user confidence.

Methods and Algorithms

Generally, we can divide triple classification procedures into two fundamental categories: knowledge-driven methods and data-driven methods. The objective of knowledge-driven approaches is to detect gaps in understanding by employing domain-specific ontologies, taxonomies, or rules to represent explicit knowledge. Methods like rule-based reasoning and OWL-based inference often use symbolic reasoning and logical inference to extract new insights from established facts and norms. Conversely,

data-driven methodologies employ machine learning or deep learning techniques to divine relational patterns from the data itself. Graph neural networks (GNNs), logistic regression, and support vector machines (SVMs) are methods that utilize annotated training data to generate prediction models. We subsequently apply these models to unobserved triples and entities. Combining data-driven and knowledge-driven techniques, hybrid methods—including Knowledge Graph Embeddings and Neuro-Symbolic AI—have lately attracted a lot of favor. These methodologies provide a smooth amalgamation of the ability to interpret and predict outcomes [130].

Evaluation

Evaluation metrics are essential in evaluating the efficacy and efficiency of triple classification techniques for knowledge graphs. Widely used measures, including accuracy, precision, recall, and the F1 score, provide valuable insights about the model's capacity to accurately anticipate the absence of connections between things. In addition, there are specific metrics designed for knowledge graph completion tasks that provide more detailed evaluations. These metrics include mean reciprocal rank (MRR), hits@k, and area under the precision-recall curve (AUC-PR). They consider the ranking of the correct relationship prediction, as well as the balance between precision and recall. These metrics allow for thorough benchmarking and comparison of various triple classification algorithms, helping researchers determine the most appropriate methods for certain knowledge graph situations and applications [131].

4.1.3 Relation prediction through triple classification

The use of triple classification has emerged as a robust method for predicting relationships. The task is framed as a classification challenge, in which a model is trained to differentiate between valid and invalid triples in a knowledge graph (KG). A valid triple denotes a verifiable assertion that is accurate within the realm of knowledge, while an invalid triple does not. By correctly categorizing triples, the model can learn the fundamental relational patterns between things and forecast missing relationships. The procedure progresses via a sequence of clearly defined steps:

1. The first step is to represent entities and relations in the knowledge graph (KG) as numerical vectors. By using the technique of embedding, we are able to harness the capabilities of machine learning for the purpose of accomplishing categorization tasks. Multiple embedding models are available, and *TransE* is a notable one. *TransE* maps entities and relations onto a common vector space, where the proximity of their embeddings represents the similarity between them. This allows the model to acquire a compact representation that captures the semantic connections inside the knowledge graph.
2. Triple Representation: After embedding entities and relations, it is necessary to provide a single representation for each full triple. One can do this by using a variety of methods, such as adding or combining the entity and connection vectors. The selected approach seeks to capture the semantic association between the head, relation, and tail items in the triple. For example, a summation technique implies that the connection on the head item has a straightforward additive impact on the tail entity, but concatenation allows for a more intricate interaction between the components.
3. The process of generating negative samples and training the classification model becomes more complex at this stage. In order to train a classification model, it is necessary to have a

dataset that has an equal number of valid and invalid triples. Nevertheless, Knowledge Graphs intrinsically suffer from a significant shortage of clearly labeled invalid triples. In order to tackle this issue, we use a method known as negative sampling. Negative sampling is the process of creating artificial invalid triples by randomly substituting the head or tail object of a known valid triple with another entity from the knowledge graph (KG). It is crucial to acknowledge that the sampling technique must be meticulously planned to prevent the introduction of bias. Ideally, the negative samples produced should have believable but ultimately inaccurate claims within the realm of knowledge. After obtaining a dataset including numerical vectors representing both valid and negative triples, we may proceed to train a classification model. Typical options include logistic regression, support vector machines (SVMs), and deep neural networks (DNNs). The model learns by analyzing this merged information, allowing it to discern the distinguishing features that separate meaningful relationship patterns from illogical combinations. The training procedure aims to refine the parameters of the model in order to decrease classification mistakes, resulting in improved prediction capabilities that are resistant to variations.

4. Relation Prediction: After training the classification model, we can use it to predict relationships. The model receives the vector representations of a pair of items (head and tail) as input and generates a probability distribution that encompasses all potential relations in the knowledge graph (KG). The link with the greatest anticipated probability is regarded as the most probable missing relationship. This technique utilizes the acquired relational patterns from the training data to deduce the most likely connection that fills in the incomplete triple.

4.1.4 Advantages of Triple Classification

Triple classification addresses the core issue of establishing the validity of a given triple in a knowledge graph. This apparently simple process has various benefits [130]:

- **Data Quality Control and Incompleteness Detection:** Triple categorization is a highly beneficial tool in the realm of data quality control. Frequently, knowledge graphs are constructed from a variety of sources, which can result in errors, inconsistencies, and incompleteness. It is possible to train triple classification models to differentiate between valid triples, which faithfully depict relationships in the real world, and invalid triples, which comprise defects or illogical connections. This functionality facilitates the detection of potentially inaccurate data elements within the knowledge graph, thereby enabling focused efforts to cleanse and improve the data. Furthermore, by examining the distribution of triples that have been accurately and inaccurately classified, scholars can gain valuable knowledge about the specific categories of errors or omissions that are prevalent within the knowledge graph. Subsequently, this information can serve as a roadmap for data entry and enhancement procedures, culminating in a knowledge repository that is more exhaustive and dependable.
- **Efficiency Enhancement:** Automated triple classification algorithms improve the efficiency of knowledge graph construction and updating by streamlining the processing of large-scale knowledge graphs. These algorithms reduce the computational effort required to process queries by efficiently assessing the pertinence of triples, thereby improving the system's responsiveness. As a result, triple classification facilitates resource allocation optimization, improving knowledge graph administration and overall system performance.

- **Resource Optimization:** Triple classification can achieve resource optimization, especially in terms of storage and computational resources. By eliminating superfluous or irrelevant data, triple classification reduces operational expenses and preserves valuable resources, thereby diminishing storage demands. Furthermore, triple classification improves scalability through computational resource optimization, ensuring that knowledge graphs can effectively handle expanding quantities of data while maintaining optimal performance.
- **Semantic Enrichment:** Triple classification enhances semantic enrichment by identifying implicit relationships and deducing absent information in the knowledge graph. Methodically categorizing triples reveals latent relationships among entities, thereby enhancing the semantic expressiveness of the graph. Moreover, triple classification ensures that the knowledge graph is thorough and precisely represents the foundational domain knowledge by identifying and categorizing pertinent triples.
- **Enhanced Usability:** A well-organized knowledge graph improves efficacy by providing users with precise and pertinent data in response to inquiries or applications. Triple classification enhanced usability and user satisfaction by incorporating semantically enriched, high-quality data into the knowledge graph. Consequently, this facilitates the efficient implementation of the knowledge graph in various domains and applications, thereby harnessing its complete capabilities as a valuable asset for retrieving and analyzing information.
- **More general model:** One more notable benefit of employing the triple classification model is that it can generate predictions for relationships, subjects, and objects using the same model without requiring retraining. The intrinsic quality of triple classification, which effectively captures the semantic connections between entities in the knowledge graph, results in its efficacy as described above. Using this framework, knowledge graphs can easily improve their prediction skills to include not only the presence of connections but also the possible subjects and objects that are involved. This makes better use of computing resources and speeds up the prediction process. The above-mentioned adaptability ensures a more unified and integrated approach to analyzing knowledge graphs, allowing smooth transitions between different prediction tasks without affecting the model's performance or needing more training sessions.

4.2 Tools used

The Python programming language, the RDFLib library, the GraphDB database, the Tensorflow framework for the development of all the deep learning models used, and the Protégé tool were among the main tools that helped develop the ideas for the current thesis. Each of these tools contributed in its own unique way to the present work.

4.2.1 Python

Renowned for its simplicity, clarity, and adaptability, Python is a highly advanced programming language. Originally developed under Guido van Rossum in the late 1980s, Python has become very popular in many different disciplines like data analysis, web development, scientific computing, and artificial intelligence. Its design approach gives expressive syntax use and code readability a great deal of weight. This attracts programmers of various abilities, including beginners [135], [136], [137].

Python's simple and basic syntax is well-known for helping to facilitate quick development and straightforward comprehension. It stands out among other programming languages because it defines code chunks with whitespace indentation. This tool guarantees a neat and consistent coding style and helps code readability. It also encourages adherence to ideal code layout ideas and removes the need for explicit block delimiters.

Python employs dynamic typing, which, when creating variables, removes the requirement for explicit type announcements. This flexibility allows for quick prototyping and iteration, which simplifies code maintenance and promotes reusability. Python also uses the duck typing concept, which bases an object's fit for a given operation on its behavior rather than its stated type. This paradigm supports the potential to cooperate and grow across multiple libraries and frameworks by encouraging a development approach that is more flexible and powerful.

Beyond its specific system, Python's dynamic nature encompasses a wide range of language features that streamline the execution and analysis of code dynamically. Reflecting and applying metaprogramming helps developers change code structures during runtime, enabling the use of advanced techniques such as dynamic class creation, decorators, and context managers. These qualities help programmers properly communicate their ideas without sacrificing efficiency or readability, thereby enabling the development of complex solutions to challenging problems.

Furthermore, in line with Python's "batteries included," Python's extensive standard library provides a complete set of modules and tools for numerous uses, like file I/O, networking, mathematical computation, and text processing. The large number of pre-installed modules reduces the need for external dependencies, streamlining process development and increasing the capacity to migrate code across platforms.

Furthermore, the Python community, known for its vitality and inclusivity, fosters teamwork, information sharing, and continuous development. For instance, the Python Package Index, or PyPI, is an extensive collection of outside-created third-party libraries and frameworks assembled by developers all across the world. It finds use in a wide spectrum of fields and situations. By providing reusable components and proven approaches for overcoming shared challenges, the open-source software ecosystem stimulates innovation and speeds development.

In essence, Python's simplicity, clarity, and flexibility have driven it to the top of modern programming languages, therefore making it a necessary tool for developers in many other disciplines. This programming language's beautiful syntax, dynamic type, large standard library, and active ecosystem allow developers to gracefully and effectively deal with difficult problems. Its supportive community also promotes knowledge exchange and teamwork.

4.2.2 RDFLib

Python developers specifically created RDFLib to manage, parse, and serialize Resource Description Framework (RDF) data. RDFLib offers an extensive range of features for manipulating RDF data, allowing developers to programmatically generate, search, and perform logical operations on RDF graphs [138].

The ability of RDFLib to express RDF data in a machine-understandable manner is its fundamental aspect. Triples, consisting of a subject, a predicate, and an object, make up RDF data. These triples constitute a directed graph structure. In RDFLib, classes like `rdflib.URIRef`, `rdflib.BNode`, and `rdflib` encapsulate the graph-based representation. We use "literal" to represent resources, "blank nodes" to represent blank nodes, and "literal values" to represent literal values. Using these classes, developers can easily create and manage RDF graphs while adhering to the semantic limitations set by the RDF data format.

Using RDF/XML, Turtle, N-Triples, and JSON-LD, among other forms, RDFLib streamlines the smooth import and export of RDF data. It also enables developers to easily upload RDF data to other locations and obtain RDF data from external sources. Interoperability enables different systems to seamlessly integrate and exchange data, thereby encouraging the acceptance and spread of RDF-based data formats.

RDFLib is renowned for its comprehensive support of SPARQL, a powerful query language specifically designed for RDF data. Using the integrated SPARQL query engine in RDFLib, users may run SPARQL searches on RDF graphs kept either in memory or outside data repositories. This capability lets developers run semantic searches on large-scale databases, extract exact data from RDF graphs, and do sophisticated graph pattern matching. Moreover, the simplicity of RDFLib's SPARQL support helps to integrate RDF data into data processing pipelines and applications, thereby enhancing the compatibility and value of RDF-based solutions.

It also offers limited support for semantic reasoning, allowing for basic inference capabilities on RDF graphs. Developers can make RDF data more semantically expressive by using RDFLib's built-in rule engines and inferencing rules to figure out new RDF assertions from existing ones. Although RDFLib's inferencing capabilities are not as extensive as those of specialized semantic web frameworks such as Apache Jena or the OWL API, they nonetheless enable developers to carry out basic deductive reasoning activities in their Python-based RDF applications.

RDFLib thrives due to a dynamic community of developers and users who actively contribute to its advancement, documentation, and ecosystem. GitHub hosts the RDFLib project, enabling users to report bugs, request features, and contribute code upgrades through pull requests. In addition, RDFLib effortlessly interacts with many Python libraries and frameworks, including Flask, Django, and Pandas. This allows developers to smoothly include RDF data processing into their current Python-based workflows and applications.

4.2.3 GraphDB

Designed specifically to store, manage, and search connected data expressed as graphs, GraphDB, shown in Figure 4.2, is a specialized database management system. Unlike conventional relational databases, graph databases express complex relationships between entities using graph topologies, including nodes, edges, and characteristics, rather than storing data in tables with predefined schemas [139], [140].

Graph models—which organize data as a collection of nodes and edges—form the basic framework of a GraphDB. Nodes are objects or things within the given domain; edges show relationships or interactions between nodes. Every node and edge might have associated characteristics that provide

extra information or metadata about the entities and relationships they show. Graph databases provide a flexible and expressive presentation of complicated and associated data structures, making them ideal for circumstances requiring highly linked or hierarchical data.

Depending on their underlying data models, we can primarily classify graph databases as either RDF (Resource Description Framework) databases or property graph databases. Labeled nodes, typed connections, and key-value characteristics connect nodes and edges in graph databases. Conversely, RDF databases employ triples to characterize data and rigorously follow the W3C RDF standard. Subject-predicate-object tuples—where every member is either a URI or a literal value—make up these triples. Despite the distinct advantages of both models and their suitability for various uses, property graph databases typically prevail due to their simplicity and ease of implementation, especially in sectors like social networks, recommendation systems, and knowledge graphs.

Customized query languages created especially for the graph model enable developers and users to interact with and access data from graph databases. Developers specifically created the popular declarative query language Cypher to describe graph patterns and traversals in property graph databases. It provides a simple vocabulary for specifying graph patterns, filtering criteria, and projections, thereby enabling understanding for users of various skill levels. Conversely, RDF databases often rely on SPARQL as the uniform query language. This permits strong graph pattern matching and aggregation across RDF databases. These query languages help users focus on clearly and effectively expressing their data retrieval and manipulation demands, thereby simplifying the complexities of graph traversal and manipulation.

Particularly for large-scale implementations with complex data structures, a graph database system depends critically on effective indexing and query optimization. Graph databases speed data retrieval and graph traversal operations using a variety of indexing techniques, including property indexes, full-text indexes, and spatial indexes. To further reduce query response times and resource use, they further use query optimization strategies, including query rewriting, caching, and parallel query execution. These optimization techniques help them efficiently handle complex graph searches, extending them to serve ever-growing datasets and user workloads.

Many disciplines, including social networks, recommendation systems, network analysis, semantic web, and information management, use graph databases. Graph databases enable social networks to replicate their members' social interactions. Features like friend recommendations, community discovery, and impact analysis are therefore possible. In recommendation systems, graph databases record user-item interactions and preferences, enabling tailored recommendations using cooperative filtering and graph-based algorithms. In knowledge management, they provide the foundation for encoding and querying structured and connected knowledge graphs, therefore supporting semantic search, data integration, and inference.

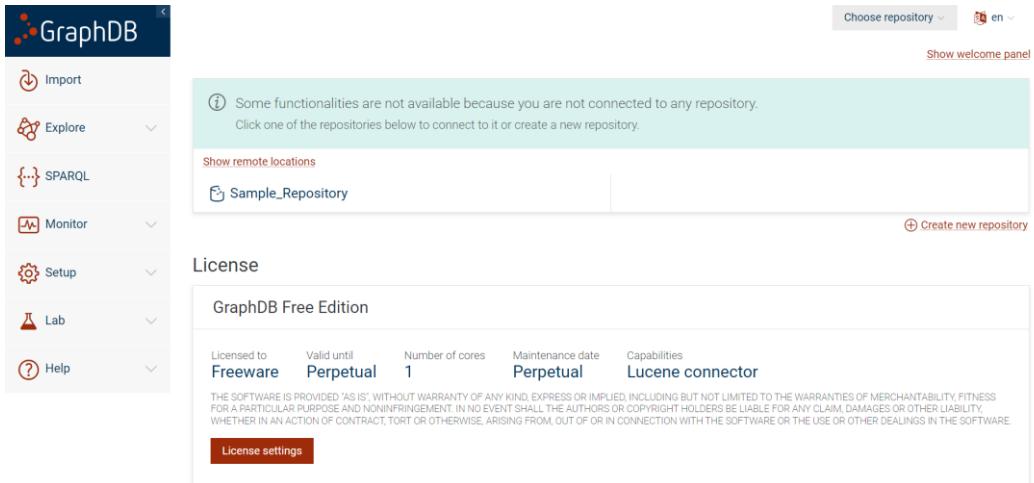


Figure 4.2: GraphDB home page [141]

4.2.4 TensorFlow

The Google Brain Team developed and maintained the leading deep learning system, TensorFlow. It provides a whole platform for building and running large-scale, mostly neural network-based machine learning models. Architecturally, TensorFlow is based on computational graphs—mathematical operations are nodes, and the data flowing between them is on edges. On CPUs and GPUs alike, this approach enables rapid parallel execution and optimization [142], [143].

TensorFlow's computational graph abstraction—which lets users designate complex mathematical processes as a directed network of nodes—is its fundamental tool. The graph views each node as an operation, with the edges illustrating the transmission of tensor-based data between these operations. This declarative approach for building models enables automated differentiation, which quickly computes gradients for training deep neural networks via backpropagation.

TensorFlow allows developers the freedom and ability to maximize speed by allowing them to choose between eager execution and stationary graph modes. By enabling the eager execution mode, one can immediately calculate and, as a result, create dynamic graphs, making debugging easier. Conversely, especially for production models, the stationary graph mode prepares and optimizes computational graphs before execution, hence increasing speed and efficiency.

It also provides a wide spectrum of abstraction layers and APIs made especially for different degrees of expertise and usage situations. The TensorFlow Core API allows users to create unique neural network topologies and training strategies with exact control over model building and optimization. Moreover, TensorFlow provides more sophisticated application programming interfaces (APIs), such as Keras, a user-friendly and simple interface for building and training deep learning models with minimal needless code. Academics, machine learning practitioners, software developers, and data scientists are among the many consumers these abstraction layers satisfy.

Apart from its adaptability and simplicity of use, TensorFlow shines in helping distributed computing reach scalability. TensorFlow's distributed execution architecture, which uses parallelism to speed model training and inference, allows users to easily distribute training and inference loadings across several devices and CPUs. Furthermore, TensorFlow easily integrates with distributed computing

systems such as Apache Spark and Apache Hadoop to enable large-scale data processing and machine learning activities in far-off environments.

Finally, it provides consistent tools and structures for implementing models in production environments. TensorFlow is especially useful for providing machine learning models in manufacturing settings. TensorFlow Serving presents a high-performance serving mechanism. It allows low-latency and scalable inference across many deployment setups. Moreover, TensorFlow Lite helps to create effective, simplified models on edge devices such as IoT devices and smartphones, thereby increasing the availability of machine learning tools in settings with limited resources.

4.2.5 Protégé

In ontology engineering and knowledge representation, a basic tool is Protege software. Protege was initially developed by the Stanford Center for Biomedical Informatics Research; now, the Protege team at Stanford University manages it and offers a strong and flexible framework for building, changing, and supervising ontologies, knowledge graphs, and semantic models. Because of its many features, flexible structure, and user-friendly interface, the tool is indispensable for academics, domain experts, and practitioners in many disciplines, including biomedical informatics, healthcare, finance, and engineering [144], [145].

Protege primarily supports ontology engineering and knowledge representation. It provides consumers with a broad spectrum of tools to specify and arrange conceptual structures, features, and links within a given domain. Using formal ontology languages like OWL and RDF lets users depict complex domains. This lets users provide in a machine-understandable format domain-specific ideas, taxonomies, restrictions, and inference rules.

The Protege program's user interface is simple and understandable. It caters to people with varying degrees of expertise and experience in disciplines such as ontology engineering and knowledge representation. Using the drag-and-drop function, the graphical ontology editor lets users quickly create and alter ontology elements, including classes, attributes, and instances. Protege also offers strong visualization tools like class hierarchies, property hierarchies, and instance graphs, which let users easily and clearly explore difficult ontologies.

Protege shines in helping to enable inference and reasoning across ontological knowledge bases. It can automate testing consistency, categorization, and inference using reasoners such as Pellet, Hermit, and FaCT++. This guarantees that users' ontologies are coherent and correct by helping them to see inconsistencies, repetitions, and entailments.

It promotes collaboration and interoperability by allowing the use of shared ontology exchange forms like OWL, RDF/XML, and Turtle. This makes simple integration with other ontologies, tools, and frameworks possible. Protege's plugin architecture lets users extend and customize its capabilities according to their particular needs, including extra tools like version control, import/export capability, and ontology alignment. Moreover, Protege has shared repositories, version control systems, and collaborative editing environments to help create ontologies in groups. These features enable successful and quick cooperation among far-off groups working on ontology projects.

Users, developers, and contributors that actively contribute to Protege's development, documentation, and distribution make up its vibrant and involved community. Tutorials, user manuals, video demonstrations, and community forums, among other materials, are available on the Protege website. These resources let users maximize Protege's features for ontology engineering and knowledge representation and start using it right away. To further the flow of knowledge, collaboration, and innovation within the Protege community, the team also routinely organizes conferences, hackathons, and workshops. This advances ontology study and application in a variety of disciplines.

4.3 Dataset

The knowledge graph used in this thesis was developed in the first half of 2022 [1]. The architecture for creating the Knowledge Graph consists of three parts: the Virtuoso OpenLink server², a set of applications that leverage the Knowledge Graph, and the Python applications. The Virtuoso database stores the content data and the Eurostat Knowledge Graph, while the Python applications retrieve, process, and store data. Natural language processing (NLP), statistical methods, and SPARQL CONSTRUCT queries enrich the content and knowledge bases by linking articles to Eurostat and OECD topics. The user interacts with the Knowledge Graph by querying the graph database, either through applications or directly via the SPARQL endpoint.

Data extracted from the Eurostat³ and OECD⁴ websites form the basis of the graph. The relational content database stores 66 tables containing information on categorizations, topics, terminologies, entity names, links between entities, glossaries, and statistical articles and datasets. The retrieved data were divided into two major categories: "Statistics Explained Articles" and "Glossary Articles." Statistics Explained Articles are official Eurostat articles presented on the Eurostat website and include statistical topics in an easily understandable way.

The "Glossary Articles" cover all statistical and general terms that need definition or explanation in the "Statistics Explained Articles." There are currently 892 "Statistics Explained Articles" and 1314 "Glossary Articles." The database also includes information about the datasets, such as taxonomies, titles, and URLs pointing to the data. Four main classes comprise the Eurostat ontology schema: the GlossaryTerm class, shown in Figure 4.3; the Content class, shown in Figure 4.4; the Reference class, shown in Figure 4.5; and the Classification class, shown in Figure 4.6. The GlossaryTerm class is utilized to access the diverse glossaries available on the websites of Eurostat and the OECD. Instances of GlossaryTerm are linked via the property relatedTerm, primarily because of their shared theme. Furthermore, the connectedness property hasGlossaryTerm links the remaining classes to the superproperty GlossaryTerm. The following subproperties make up GlossaryTerm: hasCode, hasCODEDTerm, hasOECDTerm, and hasFrequentTerm.

² <http://lod.csd.auth.gr:8890/conductor/>

³ <https://ec.europa.eu/eurostat>

⁴ <https://www.oecd.org/>

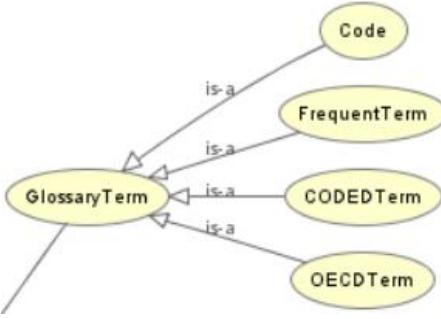


Figure 4.3: The GlossaryTerm class

The class Reference is additionally examined in relation to the classes SEAReference and GARerefence, which contain references to Statistics Explained Articles and Glossary Articles, respectively. Every citation may be classified as either internal or external. Internal references indicate datasets or articles contained within Eurostat or the OECD. External references, on the other hand, link to sources outside the organization, such as Wikipedia⁵. The Content class holds significant importance within the ontology due to its representation of knowledge pertaining to Glossary Articles and Statistics Explained. Furthermore, it signifies expertise pertaining to the datasets. The following information was provided for each article: title, abstract, content, URL from the Eurostat website, internal or external relationships, and creation and revision dates. Regarding the Statistics Explained Articles, we have also indicated whether or not the article is regarded as background material by Eurostat and the knowledge that pertains to their paragraphs.

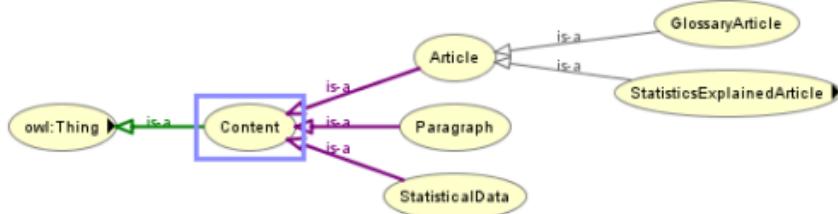


Figure 4.4: The Content class

⁵ https://en.wikipedia.org/wiki/Main_Page

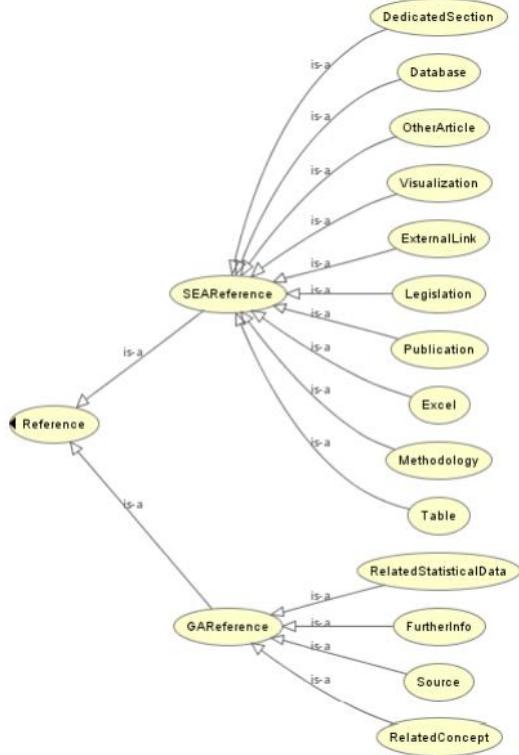


Figure 4.5: The Reference class

The Classification class mostly contains information about the several categories that Eurostat allocates to its entities. The class Category reflects the classification information provided by Eurostat for its articles. The class "Topic" represents information on the subjects covered in the articles. The Type class includes data on the kind or category of information included in an article. For instance, examining whether it includes text or equations can determine whether it is lexical or not. The Theme class provides data on the topics of Eurostat and OECD. The themes serve as a categorization that defines the articles. The main distinction is that the themes of the OECD are associated with themes from Eurostat, which is accomplished via the use of the relatedTheme property.

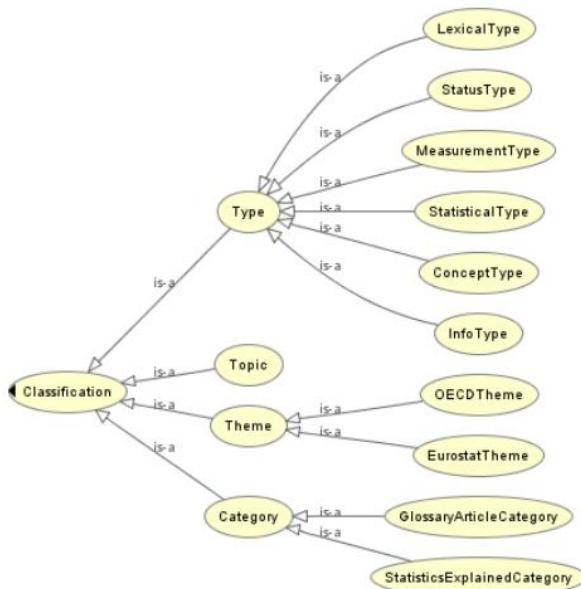


Figure 4.6: The Classification class

The Tables 4.1 and 4.2 below display all of the properties and descriptions that make up the graph.

Table 4.1: Descriptions of the object properties

Property	Usage
hasClassification	A property that shows if an article from the Editorial Content has any Classification
hasCategory	A property that relates an article from the Editorial Content with its Categories
hasCategoryOfGlossaryArticle	A property that relates an article from the glossary category with its category(-ies)
hasCategoryOfStatisticExplainedArticle	A property that relates an article from the statistics explained category with its category(-ies)
hasTopic	A property that relates an article from the Editorial Content with its Topics
hasType	Some articles from the Editorial Content may have some information about their lexical type, status type, and others
hasGlossaryTerm	A property that shows if an article from the Editorial Content has any Glossary Term
hasCode	A property that relates a dataset to its code(-s). Basically, are the labels of the Statistical Data
hasFrequenTerm	A property that relates an article from the Editorial Content with the named entities that it contains
hasCODEDTerm	A property that relates entities from the Editorial Content with the CODED Terms
hasOECDTerm	A property that relates entities from the Editorial Content with the OECD Terms
hasTheme	A property that relates the Editorial Content entities with their Theme(-s)
hasReference	A property that relates articles or terms with references material (e.g. databases, articles, legislation, etc.)
relatedTerm	A property that relates glossary terms to another glossary term
hasSubTheme	A property that relates a OECD Entity with the EuroStat sub Theme(-s)
relatedTheme	A property that relates a OECD Entity with the EuroStat Theme(-s)
hasParagraph	A property that relates the Statistics Explained Articles with its paragraphs
hasOECDTheme	A property that relates the OECD themes with the OECD terms
hasURI	A property that relates referenced resources with an entity from the content class

Table 4.2: Descriptions of the datatype properties

Property	Usage
content	This property relates the articles from the editorial content with their Content (i.e., their Abstract or a small Description)
context	A property that relates the Articles from the Editorial Content with their context (a small description of the context that the article is applicable to)
dataSource	A property that connects the articles from the Editorial Content with their Data Source(-s) (which are strings)
dateCreated	A property that relates a glossary entity with its creation date
dateUpdated	A property that relates a glossary entity with the date that was updated
databasePath	A property that relates the datasets with the path of its names
definition	A property that relates a glossary entity with its definition
fileDescription	A property that relates a dataset with its file description
id	A property that indicates the ID of the glossary or the content entity
keyword	A property that relates an instance from the class Topic with its keyword(-s)
term	A property that indicates the label of the dataset and the label of the Vocabulary entity
level	A property that indicates the depth that the dataset is in the Statistical Data tree
paragraph	A property that relates a paragraph from the paragraph class with its context
remark	A property that relates a glossary entity with its remark
sourcePublication	A property that relates the OECD entities with their source Publication
title	This property relates the articles from the editorial content with their Title
fileLink	A property that indicates the link of a dataset
hasURL	A property that contains the URL(s) of a referenced resource

4.3.1 Statistics

This section provides fundamental statistical information about the graph using the SPARQL language. The GraphDB environment executed all the queries.

SPARQL Query 1

Figure 4.7 displays this query, which counts all explicit triples in the knowledge graph. In Figure 4.8, we can see the results from the query.

```
1  SELECT (COUNT(*) AS ?tripleCount)
2  WHERE {
3      ?subject ?predicate ?object.
4  }
5
```

Figure 4.7: SPARQL Query 1

	tripleCount
1	"307444"^^xsd:integer

Figure 4.8: Results of SPARQL Query 1

SPARQL Query 2

Figure 4.9 displays this query, which computes the total number of unique entities across all categories in the Knowledge Graph, such as articles, glossary words, and themes. The query is designed to count every distinct entity, where an entity is defined as any resource that appears either as a subject or an object within the RDF triples of the graph. In Figure 4.10, we can see the results from the query.

```
1  SELECT (COUNT(DISTINCT ?entity) AS ?entityCount)
2  WHERE {
3      {
4          ?entity ?p ?o .
5      }
6      UNION
7      {
8          ?s ?p ?entity .
9      }
10 }
```

Figure 4.9: SPARQL Query 2

	totalEntities
1	"171050"^^xsd:integer

Figure 4.10: Results of SPARQL Query 2

SPARQL Query 3

Figure 4.11 displays this query, which counts the total number of explicit properties used to connect entities within the KG. In Figure 4.12, we can see the results from the query.

```

1  SELECT (COUNT(DISTINCT ?property) AS ?totalProperties)
2  WHERE {
3    ?entity ?property ?object .
4  }
5

```

Figure 4.11: SPARQL Query 3

	totalProperties
1	"43"^^xsd:integer

Figure 4.12: Results of SPARQL Query 3

SPARQL Query 4

To separately quantify the number of distinct subjects and objects within the knowledge graph, a combined SPARQL query using UNION and GROUP BY with descriptive text labels was formulated and executed. This query independently counts the unique subjects and objects, providing a clear view of the distribution of entities in different positions within the RDF triples.

Figure 4.13 displays this query, which uses two subqueries to collect entities, assigning descriptive labels to differentiate between subjects and objects. The first subquery identifies all unique subjects by selecting entities appearing in the subject position and labels them as "subject". The second subquery identifies all unique objects by selecting entities in the object position and labels them as "object". The UNION operator combines these subqueries. We then group the results, as shown in Figure 4.14, by the descriptive labels to count the distinct subjects and objects separately.

```

1  SELECT ?type (COUNT(DISTINCT ?entity) AS ?count)
2  WHERE {
3    {
4      SELECT ?entity ("subject" AS ?type)
5      WHERE {
6        ?entity ?p ?o .
7      }
8    }
9    UNION
10   {
11     SELECT ?entity ("object" AS ?type)
12     WHERE {
13       ?s ?p ?entity .
14     }
15   }
16 }
17 GROUP BY ?type

```

Figure 4.13: SPARQL Query 4

	type	count
1	"subject"	"59735"^^xsd:integer
2	"object"	"150588"^^xsd:integer

Figure 4.14: Results of SPARQL Query 4

SPARQL Query 5

Figure 4.15 displays this query, which retrieves the count of non-blank entities grouped by their types, limiting the result to the top 15 types with the highest count. Then, using the same query, we find the 15 types with the lowest count, changing the ORDER BY DESC to ORDER BY ASC. Figures 4.16 and 4.17 display the query's results.

```

1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2
3 SELECT ?class (COUNT(?entity) AS ?numEntities)
4 WHERE {
5   ?entity rdf:type ?class .
6   FILTER(!isBlank(?class))
7 }
8 GROUP BY ?class
9 ORDER BY DESC(?numEntities)
10 LIMIT 15

```

Figure 4.15: SPARQL Query 5

	class	numEntities
1	:CODEDTerm	"11220"^^xsd:integer
2	:Code	"8645"^^xsd:integer
3	:OECDTerm	"6926"^^xsd:integer
4	:Paragraph	"4749"^^xsd:integer
5	:OtherArticle	"2880"^^xsd:integer
6	:RelatedConcept	"2685"^^xsd:integer
7	:ExternalLink	"2112"^^xsd:integer
8	:Publication	"2070"^^xsd:integer
9	owl:Class	"1884"^^xsd:integer
10	:StatisticalData	"1828"^^xsd:integer
11	:FrequentTerm	"1827"^^xsd:integer
12	:Methodology	"1804"^^xsd:integer
13	:RelatedStatisticalData	"1804"^^xsd:integer
14	:GlossaryArticle	"1314"^^xsd:integer
15	:Legislation	"1219"^^xsd:integer

Figure 4.16: Results of SPARQL Query 5 (top 15 count)

	class	numEntities
1	:ei_cp	"1"^^xsd:integer
2	:ei_hp	"1"^^xsd:integer
3	:ei_qna	"1"^^xsd:integer
4	:reg	"1"^^xsd:integer
5	:reg_ef_h	"1"^^xsd:integer
6	:reg_agr	"1"^^xsd:integer
7	:cens_11rstr	"1"^^xsd:integer
8	:cens_01rstr	"1"^^xsd:integer
9	:sbs_cre_reg	"1"^^xsd:integer
10	:reg_crim	"1"^^xsd:integer
11	:met	"1"^^xsd:integer
12	:urt	"1"^^xsd:integer
13	:du_isoc_bde15e	"1"^^xsd:integer
14	:urb	"1"^^xsd:integer
15	:enps_nama_a10	"1"^^xsd:integer

Figure 4.17: Results of SPARQL Query 5 (lowest 15 count)

SPARQL Query 6

The query in Figure 4.18 retrieves the most frequently used properties in the graph. Figure 4.19 displays the query's results.

```

1  SELECT ?property (COUNT(?property) AS ?frequency)
2  WHERE {
3      ?entity ?property ?object .
4  }
5  GROUP BY ?property
6  ORDER BY DESC(?frequency)
7  LIMIT 15

```

Figure 4.18: SPARQL Query 6

	property	frequency
1	rdf:type	"67895" ^a /xsd:integer
2	:hasURL	"28874" ^a /xsd:integer
3	:term	"28615" ^a /xsd:integer
4	:title	"27073" ^a /xsd:integer
5	:hasReference	"19735" ^a /xsd:integer
6	:definition	"18151" ^a /xsd:integer
7	:id	"10966" ^a /xsd:integer
8	:databasePath	"10624" ^a /xsd:integer
9	:level	"10026" ^a /xsd:integer
10	rdfs:label	"8727" ^a /xsd:integer
11	:hasCode	"8645" ^a /xsd:integer
12	:hasOECDTheme	"8271" ^a /xsd:integer
13	:fileLink	"6817" ^a /xsd:integer
14	:hasURI	"6550" ^a /xsd:integer
15	:sourcePublication	"6065" ^a /xsd:integer

Figure 4.19: Results of SPARQL Query 6

SPARQL Query 7

Figure 4.20 displays this query, which computes the average number of relationships an entity has with other entities in the KG. This indicates the overall density of connections within the knowledge graph. Figure 4.21 displays the query's results.

```

1  SELECT (ROUND(AVG(?numConnections) * 100) / 100
2    AS ?averageConnections)
3  WHERE {
4    {
5      SELECT (COUNT(?object) AS ?
6        numConnections)
7      WHERE {
8        ?entity ?property ?object .
9      }
10     GROUP BY ?entity
11   }
12 }
```

Figure 4.20: SPARQL Query 7

	averageConnections
1	"5.15" ^a /xsd:decimal

Figure 4.21: Results of SPARQL Query 7

SPARQL Query 8

Figure 4.22 displays this query, which retrieves entities from a dataset and counts how many connections each entity has. This can reveal both highly connected and sparsely connected entities. We opted to present the top 15 connections with the most connectivity. Figure 4.23 displays the query's results.

```
1  SELECT ?entity (COUNT(?object) AS ?numConnections)
2  WHERE {
3      ?entity ?property ?object .
4  }
5  GROUP BY ?entity
6  ORDER BY DESC(?numConnections)
7  LIMIT 15
```

Figure 4.22: SPARQL Query 8

	entity	num_connections
1	estatdata:SEArticle1657	"95"^^xsd:integer
2	estatdata:SEArticle546	"88"^^xsd:integer
3	estatdata:SEArticle4764	"79"^^xsd:integer
4	estatdata:SEArticle7960	"77"^^xsd:integer
5	estatdata:SEArticle95	"76"^^xsd:integer
6	estatdata:SEArticle8560	"74"^^xsd:integer
7	estatdata:SEArticle397	"72"^^xsd:integer
8	estatdata:SEArticle4789	"70"^^xsd:integer
9	estatdata:SEArticle4796	"70"^^xsd:integer
10	estatdata:SEArticle7133	"67"^^xsd:integer
11	estatdata:SEArticle7122	"66"^^xsd:integer
12	estatdata:SEArticle2012	"65"^^xsd:integer
13	estatdata:SEArticle3474	"65"^^xsd:integer
14	estatdata:SEArticle4620	"65"^^xsd:integer
15	estatdata:SEArticle5629	"65"^^xsd:integer

Figure 4.23: Results of SPARQL Query 8

SPARQL Query 9

Figure 4.24 displays this query, which aims to count "orphan" entities in a knowledge graph, which are entities lacking connections to other entities as either subjects or objects in RDF triples. The query begins by defining a prefix for the RDF namespace to identify RDF syntax elements. It then selects all distinct entities that appear as subjects in any triple. This list of potential subjects is then filtered using two MINUS clauses: the first excludes entities that appear as objects, ensuring that entities with incoming connections are removed; the second excludes entities that appear as subjects, removing those with outgoing connections.

By applying these filters, the query isolates entities that are completely disconnected from the rest of the graph. Finally, the outer query counts these isolated entities, returning the total number of orphan entities. This method is effective for identifying and analyzing disconnected or potentially irrelevant entities within a large and complex knowledge graph. Figure 4.25 displays the results, revealing the absence of any "orphan" entities in the graph. It is important to highlight the execution time for this particular query, which is almost one hour.

```

1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2
3 SELECT (COUNT(?entity) AS ?orphanCount)
4 WHERE {
5   {
6     SELECT DISTINCT ?entity
7     WHERE {
8       ?entity ?p ?o .
9     }
10   }
11   MINUS {
12     ?s ?p ?entity .
13   }
14   MINUS {
15     ?entity ?p ?o .
16   }
17 }
```

Figure 4.24: SPARQL Query 9

Filter query results		⚠ Showing results from 1 to 1 of 1. Query took 59m 19s, today at 10:19.
		orphanCount
1		"0"^^xsd:integer

Figure 4.25: Results of SPARQL Query 9

The aforementioned queries offer fundamental statistics for various applications. Python and libraries such as NetworkX can also be used to derive statistics.

Chapter 5

Description of the implementation and the experiments

This chapter will provide a comprehensive description of the challenges encountered. We will provide a comprehensive analysis of each pipeline phase. All of the preprocessing steps, models, and evaluation code are on GitHub⁶.

5.1 Pipeline

As explained in 4.1.3, the main goal of this thesis is to use triple classification to predict relationships and automatically test the newly suggested triples. Figure 5.1 illustrates the pipeline that addresses the problem. Initially, the KG was stored in turtle format. Following this is the preparatory phase, which consists of two tasks: generating negative triples and preparing the data for the model. We then send the triples to the machine-learning model. For the models, TensorFlow was utilized. The evaluation phase follows, where we assess the model's performance on the test set and generate graphical representations. Subsequently, we initiate an additional, one-time procedure. During this stage, we deliberately choose pairings of nodes from the graph that do not share a direct connection through a relationship. For each experiment, we employ these pairings in order to forecast the relationship using each model. We attempt to perform an automated evaluation of the new triples in the final phase.

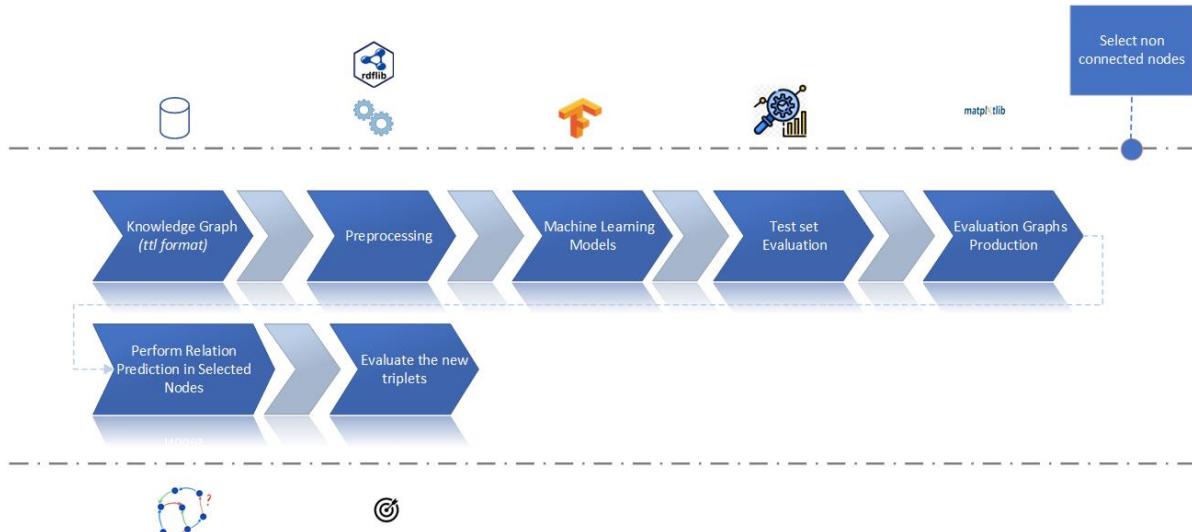


Figure 5.1: Pipeline followed

⁶ <https://github.com/sskaperdas/Master-Thesis>

5.2 Preprocessing

The function `preprocess_data` performs the preprocessing, transforming a triple-representation graph into a structured format for computational analysis, as shown in Figure 5.2. This transformation includes the extraction and indexing of unique entities and relations, the creation of mapping dictionaries, and the conversion of the original triples into indexed form.

Initially, the method detects and isolates all distinct entities present in the graph. The method retrieves entities from the triples' subject and object locations. We accomplish this by first establishing a collection of themes and a collection of objects, then merging these collections together. Simultaneously, the function retrieves all distinct relationships from the graph, identified based on the predicate position of the triples.

Subsequently, the function creates many mapping dictionaries to simplify the conversion process between entities and indices. More precisely, by iterating over the list of unique entities, it generates a dictionary named `entity2idx` that assigns a distinct integer index to each distinct entity. Furthermore, it generates a reverse mapping (`idx2entity`) to link each integer index to its corresponding entity, a valuable tool for decoding or comprehending the results of computational procedures. We build a dictionary named `relation2idx` to establish a one-to-one correspondence between each unique relation and a unique integer index. Furthermore, we establish a reverse mapping known as `idx2relation` to link each integer index to its corresponding relation.

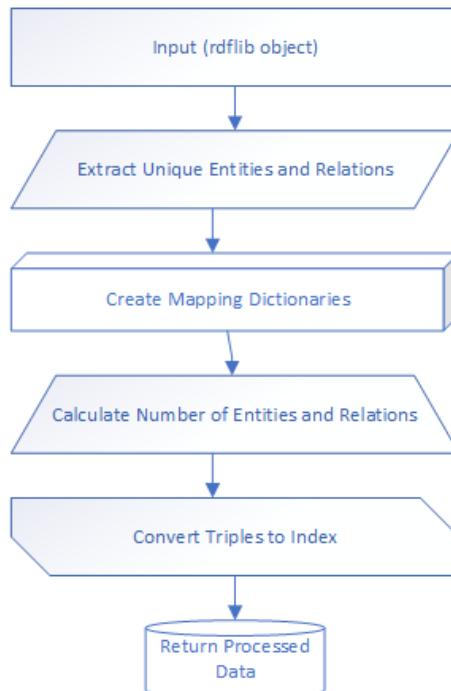


Figure 5.2: Preprocessing function

The function creates the mappings and then transforms the original graph triples into a different format by substituting each item and relation with its associated index. The outcome is a collection of triplets, with each member being a tuple of numbers that represents the indices of the subject, predicate, and object. The indexed format is well-suited for a range of computational approaches, including machine learning algorithms.

In the end, the function provides the following information: the number of unique entities (*num_entities*) and relations (*num_relations*), the set of indexed triples (*triples*), and the mappings between entities and their corresponding indices (*entity2idx* and *idx2entity*) and relations and their corresponding indices (*relation2idx* and *idx2relation*). These outputs make sure that the graph data is in the consistent number format needed for algorithmic processing. This enables the effective and efficient use of graph data in computational models.

Technically, the data is now ready to feed into the model. However, the triples now only contain positive values. If we train the model this way, it will predict a very high probability for every triple it sees. To address this, we created some negative triples. We use the following methods to create these negative triples:

Method 1: Altering the object

Most experiments have applied this simple yet effective method. It involves the following steps:

1. **Iterate Over Original Triples:** The function goes over every triple (s, p, o) in the collection of initial triples. Every triple consists of an object o , a subject s , and a predication p .
2. **Random Object Replacement:** For every triple, we disturb the object o to produce a corrupted triple. This is accomplished by adding to the original object index o a random number between 1 and $numentities - 1$. The addition guarantees corruption because it ensures that the new index does not match the old object.
3. **Modulo Operation:** The addition then produces a modulo *numentities* value that guarantees the new object index falls inside the appropriate range of entity indices. This stage is critical in preventing index overflow and ensuring consistency in the entity index space.
4. **Formation of Negative Triples:** After that, we compile the corrupted triples (s, p, o') into a list where o' stands for the new object index at random. Assuming their absence in the original knowledge graph, we regard these triples as "negative" because of their artificial production.

The approach's advantages

1. **Simple Implementation:** The technique for producing negative triples is computationally efficient and easy to use. It uses computationally cheap operations—that is, fundamental arithmetic and random number generation.
2. **Control Over Corruption:** The method preserves some degree of control over the produced negatives by just altering the object and leaving the subject and predication unaltered. This concentrated corruption teaches the model that while subjects and predicates are still legitimate, the object may not always be accurate.

Disadvantages of the Approach

1. **Randomness Limitation:** Sometimes the randomization in object replacement generates negative triples that unintentionally exist in the original graph, thereby not significantly advancing the learning process. Throughout training, this unintentional creation of false negatives may be deceptive.

2. **Uniform Corruption:** The technique employs a uniform random corruption without regard for the graph's structural or semantic properties. This homogeneity could not always represent complex, incorrect connections, so less effective training might result.
3. **Potential Redundancy:** While the modulo operation guarantees that indices fall within range, improper handling of it may cause negative generation to show patterns. This could reduce the number of negative samples, thereby affecting the training's efficacy.

Method 2: Diverse negative sampling

To enhance the training of machine learning models for knowledge graph embeddings, we define the *diverse_negative_sampling* function for this method. This function adds variety to the negative samples by corrupting only certain parts (subject, predicate, and object) of the original triples based on the corruption strength that was given. It consists of the following steps:

1. **Initialization:**
 - We initialize an empty list of *negative_triples* to store the generated negative triples.
 - To ensure the uniqueness of negative triples, we convert the original triples into a set named *triple_set* for efficient lookup.
2. **Batch Processing:**
 - We process the triples in batches of *batch_size* to efficiently manage large datasets. For each batch:
 - We create a sublist *batch_triples* that contains the triples for the current batch.
 - We initialize an empty list, *batch_negatives*, to store the negative triples generated for the current batch.
3. **Corruption Based on Strength:**
 - For each triple (s, p, o) in the batch, different corruption strategies are applied based on the specified *corruption_strength*:
 - Low Corruption:
 - Only the subject s is replaced with a randomly selected entity index s_{new} .
 - A while-loop ensures that the newly generated triple (s_{new}, p, o) does not exist in the original triples.
 - Medium Corruption:
 - Both the subject s and the object o are replaced with randomly selected entity indices s_{new} and o_{new} , respectively.
 - Two negative triples (s_{new}, p, o) and (s, p, o_{new}) are generated.
 - While-loops ensure that neither of these new triples exists in the original triples.
 - High Corruption:
 - The subject s , predicate p , and object o are all replaced with randomly selected indices s_{new} , p_{new} , and o_{new} , respectively.
 - Three negative triples (s_{new}, p, o) , (s, p_{new}, o) , and (s, p, o_{new}) are generated.
 - While-loops ensure that none of these new triples exist in the original triples.
4. **Collecting Negative Triples:**
 - We append the generated negative triples for the current batch to the *negative_triples* list.
 - We repeat this process for all batches until we have processed all original triples.

5. Return Negative Triples:

- The function returns the complete list of generated negative triples.

The approach's advantages

1. **Diversity in Negative Samples:** By changing the corruption strength, the function generates a variety of negative triples, allowing the model to learn more precisely to distinguish between valid and invalid triples.
2. **Controlled Corruption:** The ability to indicate the degree of corruption lets one optimize the training process. Early training phases could benefit from less corruption; yet, as the model becomes stronger, we can include more corruption.
3. **Efficiency in Handling Large Datasets:** Processing triples in batches makes the function appropriate for big-scale datasets as it helps control memory use and computing demand.
4. **Avoidance of Redundant Negatives:** Using while-loops to search for freshly produced triples guarantees that the negative samples are indeed unique and do not unintentionally match positive triples.

Disadvantages of the Approach

1. **Computational Overhead:** Particularly for big graphs with numerous entities and interactions, the while-loops employed to search for triples might impose processing cost. This might halt the negative sampling process.
2. **Potential for Infinite Loops:** Under high entity density and poor graph variety, the while-loops may take a considerable amount of time to identify a valid negative triple, or in extreme circumstances, they may cause endless loops.
3. **Uniform Randomness:** The random selection of new indices disregards the structural or semantic aspects of the network, making negative triples less useful for training.
4. **Batch Size Dependency:** The function's performance and efficiency may change depending on the batch size option. A wrong batch size could cause extended processing delays or ineffective memory use.

Method 3: Popular based sampling

For this method, we define the *popularity_based_sampling* function, which generates negative triples by leveraging the frequency of entities and relations in the original dataset. This method mimics the graph's popularity distribution by giving more frequent entities and relations a higher chance of selection. It involves the following steps:

1. Counting Entity and Relation Frequencies:

- The function initializes two counters: *entity_counter* and *relation_counter*.
- It iterates over the original triples to count the occurrences of each entity in the subject position and each relation.

2. Ranking Entities and Relations:

- We rank entities and relations based on their frequency. The function uses *rankdata* to assign ranks, where the most frequent entities and relations get the lowest ranks (indicating higher popularity).

3. Probability Distribution Calculation:

- We calculate a probability distribution based on the ranks. The probability of selecting an entity or relation is inversely proportional to its rank plus one. This means that higher ranked (and more popular) entities and relationships have higher probabilities.
- We normalize the raw probabilities to ensure their sum equals one. This normalization ensures that the resulting values form a valid probability distribution.

4. Sampling Based on Probability Distribution:

- Entities and relations are sampled according to the calculated probability distributions. The function uses `np.random.choice` to randomly select entities and relations based on their respective probabilities.
- The sample size is determined by `batch_size`, ensuring that a fixed number of negative triples are generated in each batch.

5. Formation of Negative Triples:

- We pair the sampled entities and relations to form negative triples. We construct each triple by combining a sampled entity, a sampled relation, and another instance of the same entity as the object.
- We return the resulting list of negative triples.

The approach's advantages

1. **Realistic Negative Samples:** Realism selects entities and links based on their popularity, thereby improving the production of negative triples. Actual knowledge graphs typically display asymmetrical popularity distributions, aligning with this distribution.
2. **Concentrate on Frequent Entities and Relations:** Negative triples are more likely to include more commonly occurring entities and linkages. Using this approach ensures that the model gains the capacity to control regular events, which are usually more relevant in terms of performance in practical situations.
3. **Probability-based Sampling:** Using a probability distribution ensures that negative samples are generated in a regulated, statistically sound manner. This habit reduces the likelihood of creating unimportant or unenlightening negative triples.
4. **Efficiency:** Processing large amounts of information in chunks allows the function to properly control memory use and computational load.

Disadvantages of the Approach

1. **Favoring Prominent Entities and Relationships:** While focusing on well-known things and interactions has benefits, this strategy may unintentionally lead to the neglect of less frequently occurring but equally important entities and relationships.
2. **Unbiased Object Selection:** For both the subject and object locations in negative triples, the current approach uses the same entity samples. This strategy may not introduce sufficient variety. This could restrict the positive value of the negative samples.
3. **Restricted Semantic Context:** Advocates of frequency alone ignore the semantic background of objects and relationships. A lack of context may result in negative triples that are less instructive and fail to sufficiently push the model through the training process.

Method 4: Bernoulli sampling

This method implements Bernoulli-negative sampling. First, we initialize an empty list named *negative_triples* to store the generated negative triples. Next, we iterate through each triple in the list of positive triples, *triples*, which consists of subject (*s*), predicate (*p*), and object (*o*). For each triple, we perform a Bernoulli trial by generating a random integer (0 or 1) using *np.random.randint(2)*. This simulates a coin toss, deciding whether to corrupt the subject or the object of the triple.

- *Random.choice(num_entities)* randomly selects a negative subject (*s_neg*) from the range of available entities if the coin toss result is 0. We then form a new triple by replacing the original subject *s* with this negative subject *s_neg*, while maintaining the predicate *p* and the object *o* unchanged. We append this new triple to the *negative_triples* list.
- Using *np.random.choice(num_entities)*, we randomly select a negative object (*o_neg*) from the range of available entities if the coin toss result is 1. We form a new triple by replacing the original object with this negative object, *o_neg*, while maintaining the subject and predicate unchanged. We append this new triple to the *negative_triples* list.

By the end of the iteration, the *negative_triples* list contains a set of negative samples, each of which is a corrupted version of an original positive triple. This negative sampling approach randomly corrupts both subjects and objects, resulting in a balanced set of negative examples.

5.3 Evaluation

There are two phases to the model evaluation process. Like most machine learning models, there is a direct phase on the test set, followed by a relation prediction task that occurs after the model has undergone training and evaluation on the test set.

Test set Evaluation

We make the evaluation using five metrics suitable for binary classification problems. We use *precision*, *recall*, *F1 score*, *roc_auc score*, *pr_auc score*, and *Matthews Correlation Coefficient* (MCC). We use the random predicting method as the baseline method, which provides a basis for comparing the model results. In this method, we randomly choose negative or positive instances, so we get all metric values of 0.5. Next, we obtain various graphs that offer insights into the model's performance. For every model, we take the learning curve plot, the precision recall curve, the confusion matrix (with a threshold > 0.5), the calibration plot for the probabilities, and some plots that interpret the model. For the interpretation, we use the LIME (Local Interpretable Model-Agnostic Explanations) framework for the predictions.

Specifically, it explains how individual features (subject, predicate, and object) contribute to the model's predictions for a subset of instances from the test dataset. For these instances, we also visualize the mean feature importance. We select instances from the test set for analysis. The exact number of instances we explain—usually 300 to 1200—varies according to how much time the model needs to predict each instance. For some models, we also show the exact impact of the contribution of subject, predicate, and object for five randomly selected triples, as shown in Table 5.1. The evaluation mentioned above pertains to the test set and focuses on the triple classification task.

Table 5.1: The five randomly selected triples

<i>Triples 1</i>	<i>glossaryArticle118</i>	<i>ns1:hasReference</i>	<i>referenceSource59</i>
<i>Triples 2</i>	<i>hlth_ehis_aw1u</i>	<i>ns1:term</i>	<i>"hlth_ehis_aw1u"</i>
<i>Triples 3</i>	<i>paragraph9574_3455</i>	<i>a</i>	<i>ns1:Paragraph</i>
<i>Triples 4</i>	<i>ns1:ei_qna</i>	<i>a</i>	<i>ns1:StatisticalData</i>
<i>Triples 5</i>	<i>ns1:fats_08</i>	<i>ns1:level</i>	<i>"4"</i>

Relation Prediction and Evaluation

After this evaluation, we perform relation prediction using the trained model for the triple classification task. To perform relation prediction, we require a pair of entities for which we will predict a single relation from the total number of relations present in the graph. We generate these pairs using the following method, as shown in Figure 5.3: We use random walks to sample entities from the graph and generate pairs of non-connected nodes, saving the results to a file. The *random_walk* function is defined to perform a random walk on the graph, starting with a given entity. It accepts three arguments: *graph*, *entity*, and *walk_length*. The function initializes a walk, starting from the provided entity, and appends it to a list named *walk*. It then iterates up to the specified *walk_length*, selecting a random neighbor of the current entity at each step. If the current entity has no neighbors, indicating a dead end, the walk terminates early. We then return the walk and the resulting list, which includes the order of entities visited during the random walk.

Next, the *sample_entities* function is defined to sample a specified number of unique entities (*num_samples*) from the graph using random walks. This function initializes an empty set named *entities* to store the unique entities encountered during the walks. Until it collects the desired number of unique entities, the function repeatedly selects a triple from the graph and randomly selects either the subject or object of the triple as the starting entity. The function then performs a random walk from the selected entity, updating the set of entities with those encountered during the walk. We convert the set to a list once it contains enough entities and return the first *num_samples* entities.

Then we proceed to generate pairs of non-connected nodes. The variable *num_pairs* determine the quantity of non-connected pairs for generation. To ensure enough diversity in the sampled entities, the *sample_size* is set to twice the number of pairs. We sample entities using the *sample_entities* function with a walk length of five. We initialize an empty list named *non_connected_pairs* to store the generated pairs. We enter a loop to create pairs of non-connected nodes by randomly selecting two distinct entities from the sampled entities. If the two entities differ, they form a pair and add it to the *non_connected_pairs* list. We continue this process until we obtain the required number of pairs (*num_pairs*). Finally, the pairs of non-connected nodes are saved to a file named *non_connected_pairs.txt*. The file is opened in write mode, and each pair is written in the format *entity1 --- entity2*.

In this case we choose to create five files with non-connected pairs containing 200, 200, 300, 300 and 500 pairs of nodes respectively. Now to perform the relation prediction we make the following. Initially, the *RDFLib* library is used to load the RDF graph from a Turtle file. The graph is parsed, and a dictionary is created to store domain and range information for each relation by iterating through the

triples in the graph. If a triple specifies a domain or range, the corresponding relation and entity are recorded in the *relation_domain* and *relation_range* dictionaries.

Afterwards, we define several functions to handle the relation prediction process, which is shown in Figure 5.4. The *generate_candidate_triples* function creates candidate triples by pairing each node pair with all possible relations, producing a list of candidate triples. The *predict_probabilities* function then uses a provided model to predict the probabilities of these candidate triples, converting the list of triples to a NumPy array and utilizing the model's predict method to obtain probabilities.

The *select_relations* function selects the relation with the highest predicted probability for each node pair by iterating through the candidate triples' probabilities. It keeps track of the highest probability relation for each pair and stores the corresponding relation index. The *filter_candidate_triples* function filters these candidate triples based on domain and range restrictions using the previously constructed dictionaries. It ensures that only valid triples, according to these restrictions, are retained.

After defining these functions, we process multiple files containing pairs of non-connected nodes. For each file, it reads the node pairs and maps non-integer values to unique integer identifiers to facilitate processing. It converts the node pairs to integers and generates candidate triples. We filter these triples based on domain and range restrictions, then use the model to predict their probabilities. We select the highest probability relations for each node pair and write the results to an output file. We write the triples in a format that includes the probability, sorted in descending order of probability.

As the final step in this section, we aim to create an automated evaluation of the new triples we recommend. To achieve this, do the following, as shown in Figure 5.5: We load triples from files line by line, extracting the subject, predicate, object, and probability from each line using regular expressions. These extracted triples are stored in a list. Next, we extract unique entities and relations from the loaded triples, storing them in sets named entities and relations, respectively. Following this, the parameters for the *FastText* model are defined, including the vector size, window size, minimum count, and the number of worker threads. The *FastText* model is then trained on the triples to learn vector representations (embeddings) for the entities and relations.

A function named *get_embedding* is defined to retrieve the embedding for a given word (entity or relation) from the *FastText* model. A zero vector is returned if the term is not detected in the model's vocabulary. This function is employed to acquire and store embeddings for all entities and relations in the dictionaries *entity_embeddings* and *relation_embeddings*. These dictionaries are subsequently amalgamated to form a singular dictionary, *all_embeddings*. Principal Component Analysis (PCA) helps visualize embeddings by reducing their dimensionality to two dimensions. This decrease helps the embeddings to be seen in a 2D graph. The embeddings that have been transformed by PCA are stored in the *pca_embeddings* variable.

We next compute the silhouette score for many values of k ranging from 2 to 10 to ascertain the ideal number of clusters (k) for K-means clustering. Higher scores indicate better-defined clusters; the silhouette score gauges how similar an item is to its own cluster relative to other clusters. The value of k that yields the highest silhouette score is selected as the optimal number of clusters. Using the best value of k, K-means clustering is performed on the original embeddings, and the resulting cluster labels are obtained.

Finally, we compute and prints three clustering quality metrics: the *silhouette score*, the *Davies – Bouldin index*, and the *Calinski – Harabasz index*. The *silhouette score*, as previously mentioned, indicates how well the clusters are defined. The *Davies – Bouldin index* measures the average similarity ratio of each cluster with the cluster that is most like it, with lower values indicating better clustering. The *Calinski – Harabasz index* evaluates the ratio of the sum of between-cluster dispersion to within-cluster dispersion, with higher values indicating better-defined clusters.

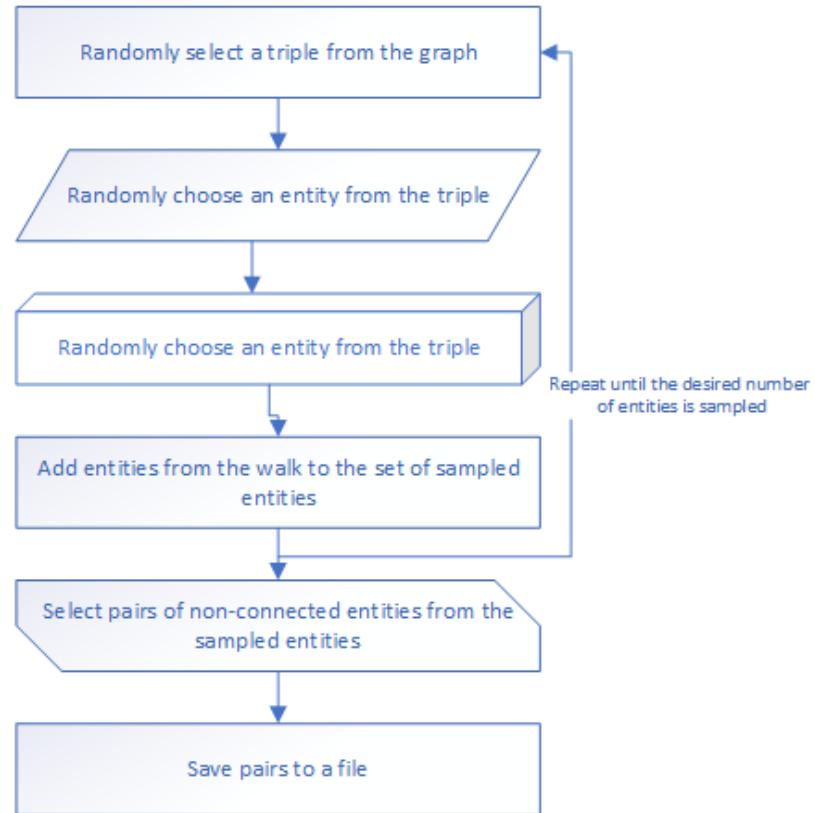


Figure 5.3: Production of the node pairs

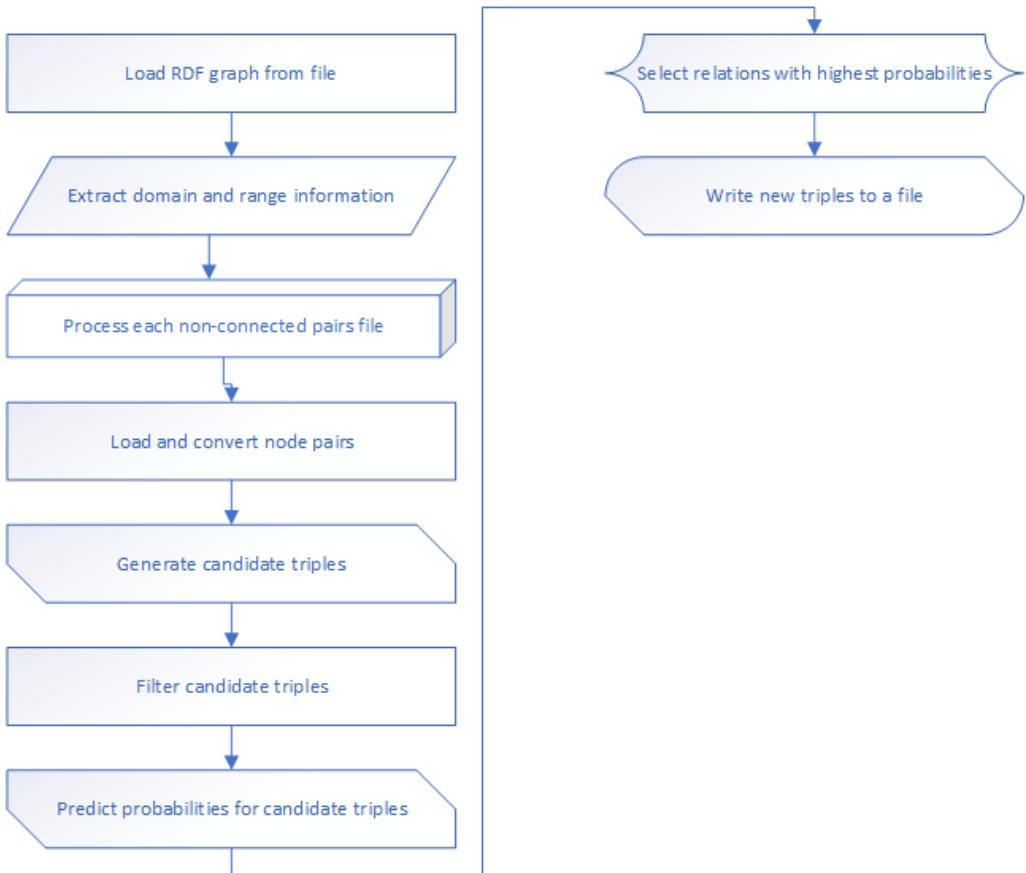


Figure 5.4:Relation prediction process

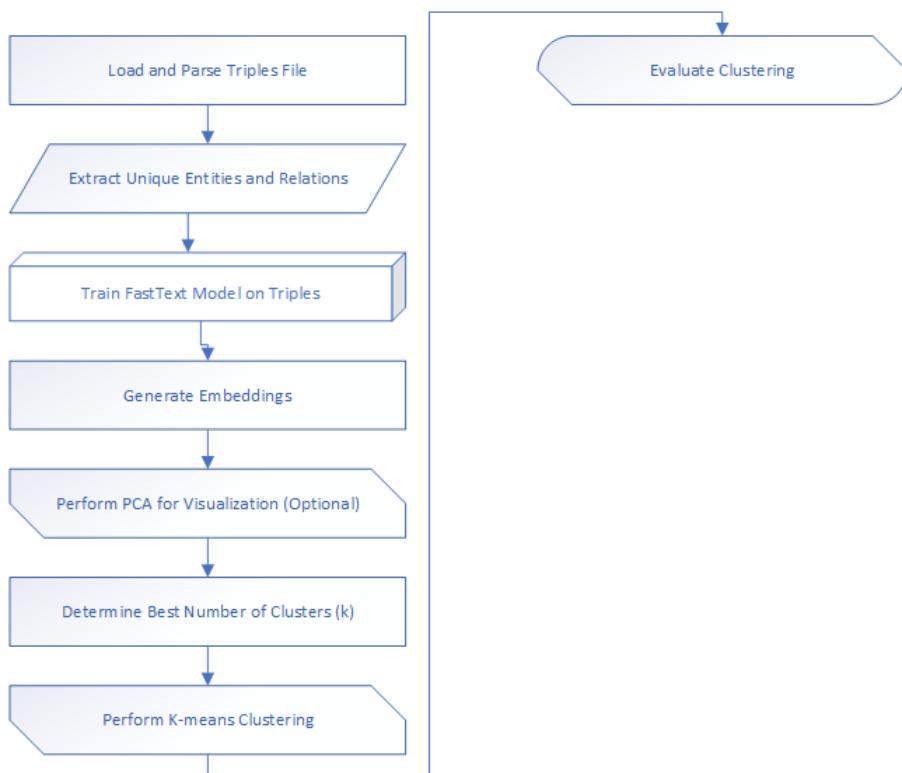


Figure 5.5: Clustering evaluation process

5.4 Models

Below, we outline the machine learning models used for the triple classification task. Two primary architectures are used. These architectures were derived after extensive experimentation, with a view to the efficiency and speed of model training. One architecture mostly utilizes dense layers, whereas the other predominantly employs 1D convolution layers. Subsequently, we use these architectures and add additional components to categorize our experiments based on the architecture and embeddings used, namely *TransE*, *TransH*, *HoIE*, *CompIEx*, *DistMult*, and *RotateE*. Figure 5.6 depicts the initial architectural design. To prevent overfitting, a dropout layer follows each densely connected layer in the model architecture. At first, the architecture consists of a dense layer with 5012 units and a *ReLU* activation function. In order to facilitate the model's acquisition of intricate patterns from the data, the activation function introduces nonlinearity. Following this layer, a dropout layer with a 0.5 rate randomly removes 50% of the nodes during training to avoid overfitting. The second layer—another dense layer with 2048 units and *ReLU* activation—is complemented by a dropout layer with a 0.5 rate. This decrease in the number of units progressively reduces the dimensionality of the feature space.

The third dense layer then comprises 1024 *ReLU*-activated cells adjusted to a factor of 0.01 using L2 regularization. L2 regularization assists in mitigating overfitting by penalizing large weights. This dense layer is followed by a dropout layer with a rate of 0.5. The fourth dense layer has 512 units and also uses *ReLU* activation and L2 regularization (0.01). It is followed by a dropout layer with a rate of 0.5, which keeps using dropouts for regularization. The fifth dense layer has 256 units with *ReLU* activation and L2 regularization (0.01), followed by another dropout layer with a rate of 0.5 to make sure that the network is always regularized.

We flatten the result to produce a single long feature vector from these dense and dropout layers. This step is crucial for converting the multidimensional tensor output from the previous layers into a form suitable for the final dense layer. The final layer consists of a single unit and a tightly linked layer that utilizes a sigmoid activation algorithm. As it generates an output between 0 and 1, the sigmoid activation function is quite helpful for binary classification problems. This result shows the possibility of the positive class.

Figure 5.7 shows the second architecture. The architecture starts with *ReLU* activation from a one-dimensional convolutional layer (Conv1D) with 128 filters and a 3-pixel kernel size. This layer incorporates L2 regularization to assist avoid overfitting by punishing significant weights and employs identical padding to guarantee the output has the same length as the input. The output of this convolutional layer is then sent to a batch normalization layer, which accelerates training, normalizes the activations and enhances stability. Following batch normalization, we randomly set a proportion of input units to zero during training to apply a dropout layer with a dropout rate of 0.5 and thus further reduce overfitting.

The second layer in the sequence is another Conv1D layer, but this time with 256 filters and the same kernel size of 3. It also uses *ReLU* activation, same padding, and L2 regularization. This layer, like the first convolutional block, follows batch normalization and a dropout layer with the same dropout rate, ensuring consistent regularization and normalization across the network.

After the convolutional layers, the model includes a flattening layer that converts the multi-dimensional tensor output from the previous layers into a single, long feature vector. This step is

crucial for preparing the data for the fully connected, dense layers that follow. The first dense layer in this sequence has 512 units with *ReLU* activation and includes L2 regularization. Another batch normalization layer and a dropout layer follow this layer, maintaining the pattern of normalization and dropout for regularization.

Finally, the model includes a dense layer with a single unit and sigmoid activation. This layer also includes L2 regularization to ensure the model's robustness. The sigmoid activation function is suitable for binary classification tasks, as it generates a probability value between 0 and 1 that denotes the probability of the positive class.

Subsequently, we may begin our experimentation. In addition to the embeddings, we also have two more components that are essential to our research. One layer is a graph convolution layer, while the other is a graph attention layer. We implement both layers using the TensorFlow framework. These layers undergo evolution throughout the tests. In this section, we provide a comprehensive analysis of the implementation and specific information about the trials conducted.

5.4.1 Group of Experiments

We classify the experiments based on the model design, the concatenation technique, and the additional components incorporated into the models. We were unable to perform all the trials for some models, mostly because of the extensive training period necessary. Here we provide a comprehensive list of both the group and individual tests conducted. All of the models except those in Group 3 use [Method 1](#) for negative sample generation. Technical difficulties or poor results prevented the completion of some experiments.

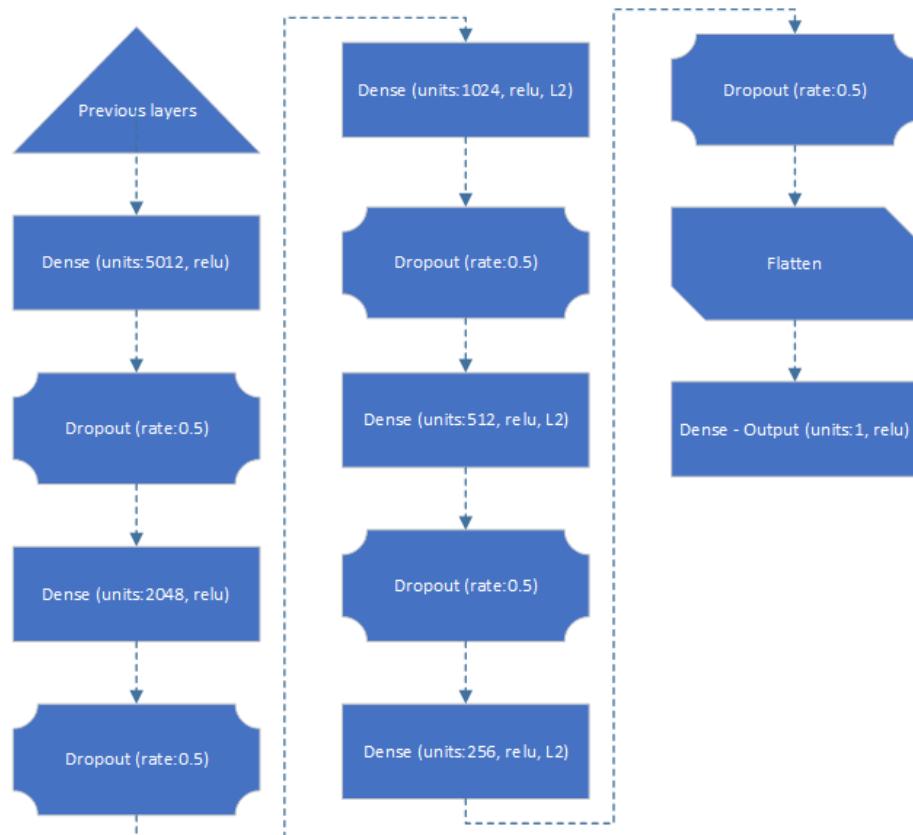


Figure 5.6: Dense architecture

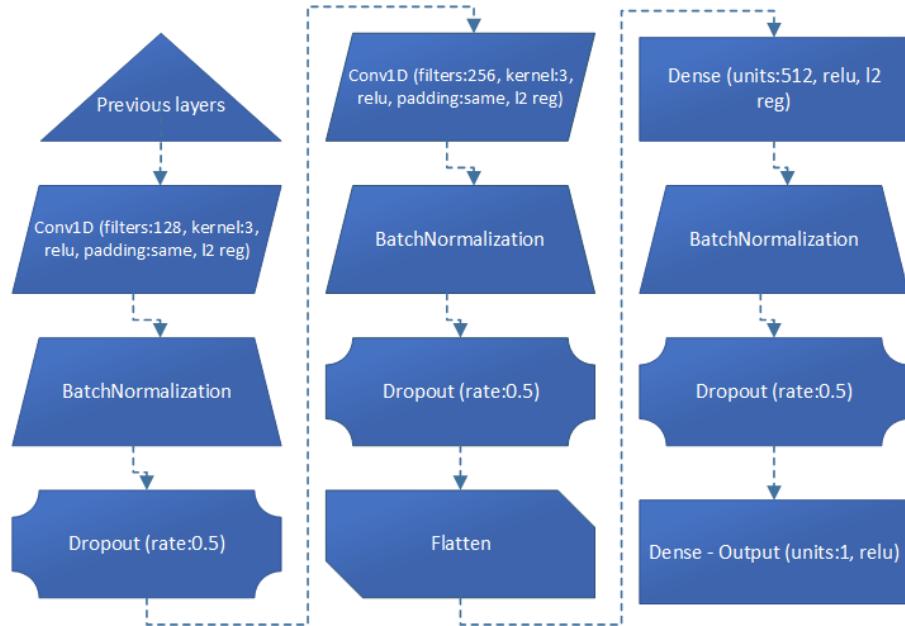


Figure 5.7: Conv1D architecture

Group 1

For this group we used the Dense architecture. Our six models, for this group, are shown in Figure 5.8 – Figure 5.13.

TransE: This model combines the Dense architecture with the *TransE* algorithm, as seen in Figure 5.8. It adds embeddings for entity degree and inverse relation frequency to provide additional information to the model. Loading and [preparing](#) the knowledge graph data from a Turtle (TTL) file starts the process. The basis is graph parsing, entity mapping, and relation mapping to unique indices. This mapping facilitates the conversion of graph triples into a numerical format appropriate for neural network model input.

To ensure repeatability, the script sets the random seeds for TensorFlow, Python's random package, and NumPy to 42. Preprocessing generates relations and entities and converts triples to indexed forms. As shown in [Method 1](#) above, we additionally construct negative instances to provide a balanced dataset including both positive and negative samples. Maintaining a 70/15/15 split ratio, we next shuffle and divide the dataset using Scikit-learn into training, validation, and test sets.

The model's core is a custom *TensorFlow Keras* model that leverages entity and relation embeddings. Initially, `tf.keras.layers.Embedding` generates these embeddings. A dense vector of a specified dimension, `embedding_dim`, represents each entity and encapsulates its features. Similarly, we embed relations into vectors of the same dimensionality. During the training process, the model learns the embeddings for entities and relations, which enables it to capture latent patterns and interactions within the knowledge graph. To further enrich the entity representations, the model incorporates additional embeddings for in-degree, out-degree, and inverse relation frequency. We also create these embeddings using `tf.keras.layers.Embedding`. We create these embedding layers with a dimension of 1. The in-degree embedding captures the number of incoming edges to an entity, the out-degree embedding captures the number of outgoing edges, and the inverse relation frequency embedding captures the frequency of a relation in an inverse manner, emphasizing rare relations.

Using the input triples—subject, predicate, and object—the forward pass obtains the embeddings. For a given triple, we get the embeddings for the subject, predicate, and object. We improve the representations of the subject and the object by adding their in-degree embedding and out-degree embedding, respectively, thereby utilizing structural information from the graph. To change the representations of the subject and object embeddings depending on the frequency of the relation, we include the inverse relation frequency embedding in both of them. A batch normalization layer then normalizes the entity embeddings to ensure a constant scale and thus stabilize the training process. Following the *TransE* model paradigm, the model computes the expected object embedding by merging the subject embedding and the relation embedding. To compute the final representation, the dense architecture concatenates the subject, relation, object, and expected object embeddings. These layers transform the concatenated embeddings into a single prediction value, which represents the likelihood of the supplied triple's validity within the knowledge graph.

Table 5.2 shows the choices we made about how to set up the optimizer, loss function, and training callbacks in the model to get the best performance and avoid overfitting. The Adam optimizer is selected with a learning rate of 0.00001. To suit the learning rate of any parameter, Adam combines the benefits of two different modifications of stochastic gradient descent (*SGD*), namely Adaptive Gradient Algorithm (*AdaGrad*) and Root Mean Square Propagation (*RMSProp*). This makes it ideal for managing typical in knowledge graphs: noisy data and sparse gradients. We use the modest learning rate to provide fine-grained updates to the model parameters, which fosters steady convergence and prevents overshooting of the ideal values during training.

The model compilation uses binary cross-entropy loss. This loss function gauges the performance of a classification model by producing a probability value between 0 and 1. For this task, binary cross-entropy is suitable because it directly relates to the objective of estimating the probability of a given triple being legitimate. The reduction mode *SUM_OVER_BATCH_SIZE* averages the loss over the batch size, ensuring constant gradient updates regardless of the batch size.

To further improve the training process, the model uses a *ReduceLROnPlateau* learning rate scheduler. If the validation loss does not improve for five consecutive epochs, this scheduler tracks it and lowers the learning rate by a factor of 0.6. This dynamic change enables the model to converge more effectively to a reduced validation loss by taking fewer steps as it approaches the ideal answer, thereby fine-tuning the learning rate during training. Set the minimal learning rate to 1e-7 to stop the learning rate from becoming too low and stopping advancement.

Additionally, an early stopping callback is employed to monitor the validation loss. If the validation loss does not improve for 5 consecutive epochs, training is stopped early, and the best model weights observed during training are restored. This prevents overfitting by stopping the training process before the model starts to learn noise in the training data, ensuring that the model maintains its ability to generalize to unseen data.

We evaluate the model's performance on the test set after training using various metrics like precision, *recall*, *F1 score*, *ROC AUC*, *PR AUC*, and *Matthew's correlation coefficient* (MCC). We also generate a confusion matrix to provide a detailed view of the model's classification performance. The model ends by showing the confusion matrix and the training versus validation loss plot. These show how the

model learned and how well it could tell the difference between valid and invalid triples in the knowledge graph.

Table 5.2: Parameters for the TransE Model

Parameter	Value
Embedding Dimension	256
Number of Epochs	20
Batch Size	256
Optimizer	Adam
Learning Rate	0.00001
Loss Function	Binary Crossentropy
Metrics	Accuracy
Learning Rate Scheduler	ReduceLROnPlateau
Monitor	val_loss
Factor	0.6
Patience	5
Minimum Learning Rate	1e-7
Verbose	1
Monitor	val_loss
Patience	5
Restore Best Weights	True

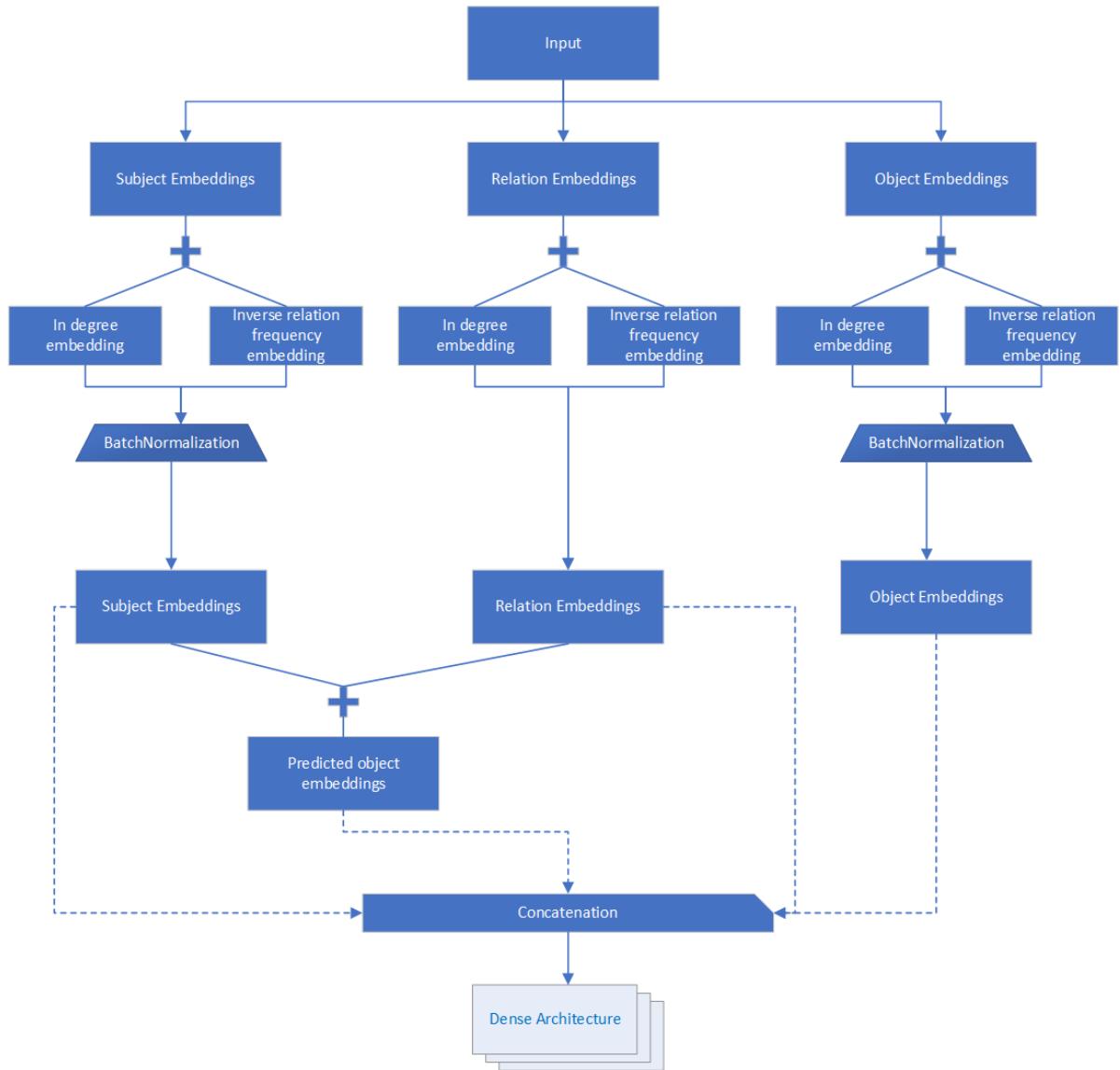


Figure 5.8: TransE Model

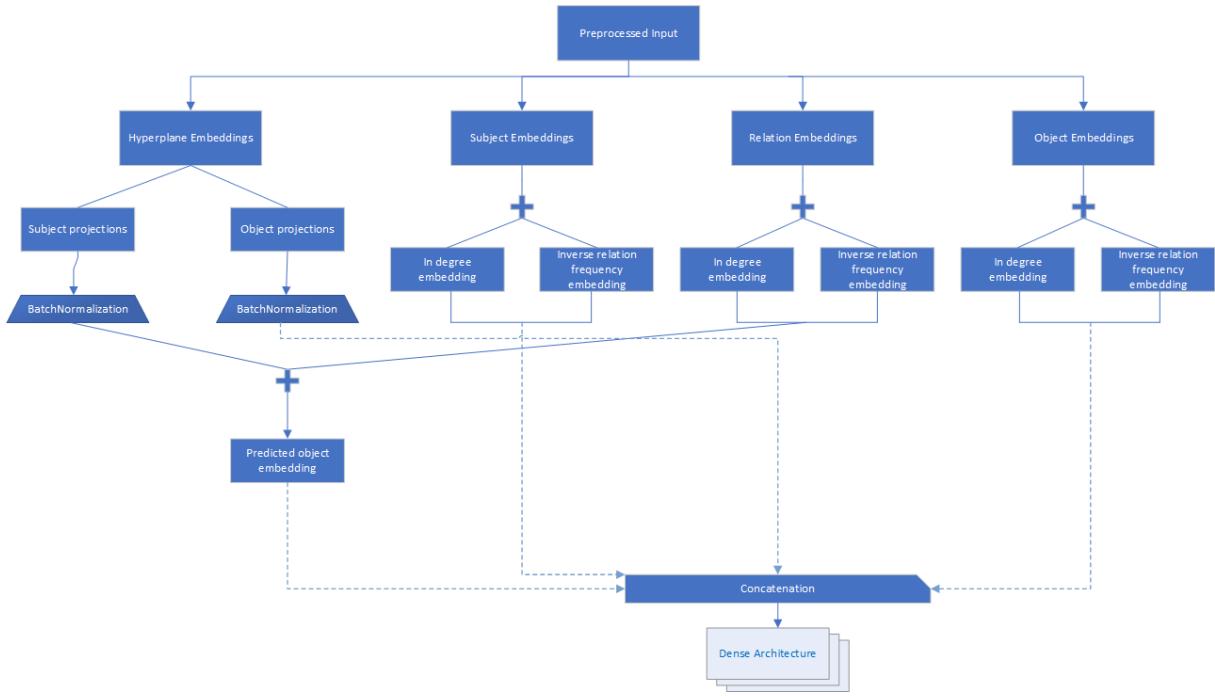


Figure 5.9: TransH Model

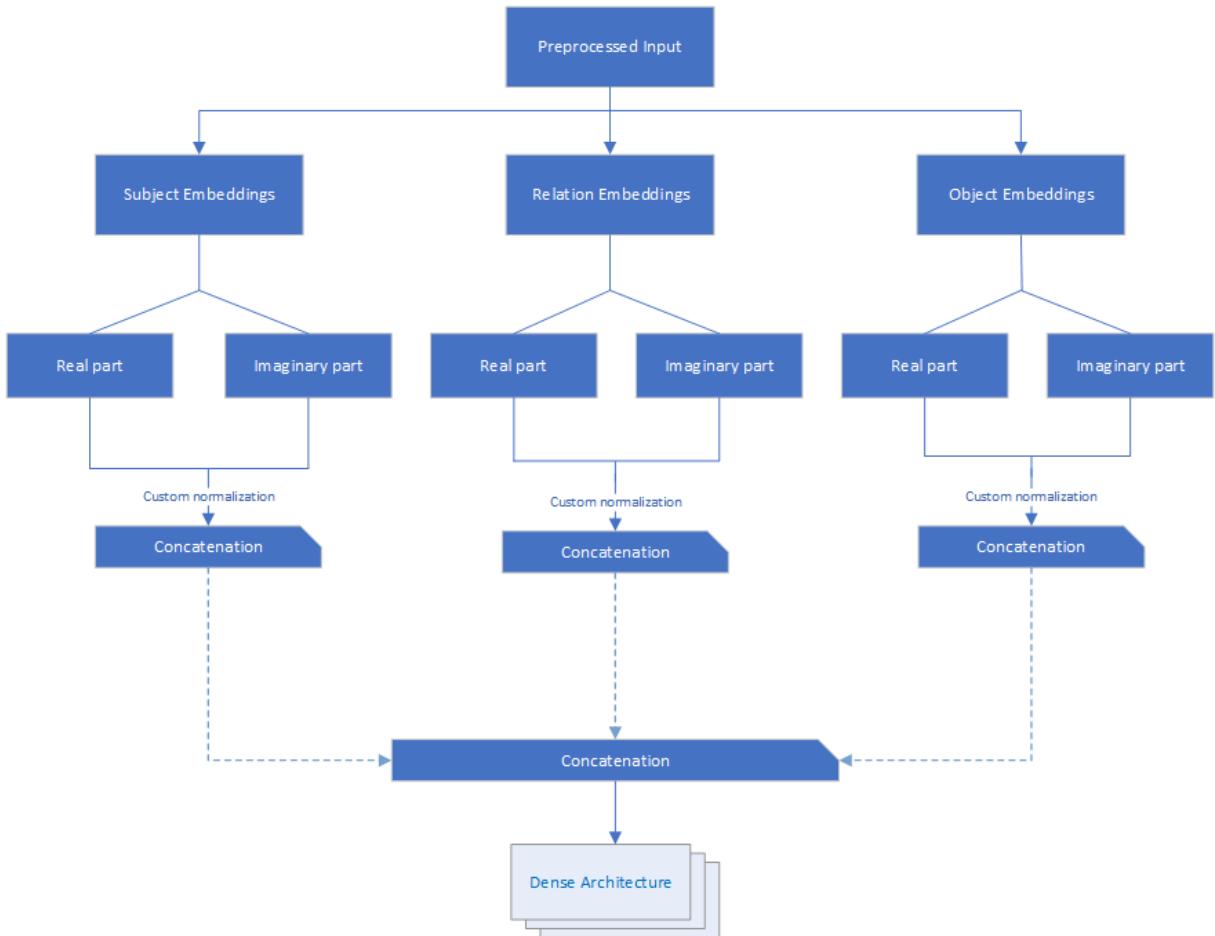


Figure 5.10: RotatE Model

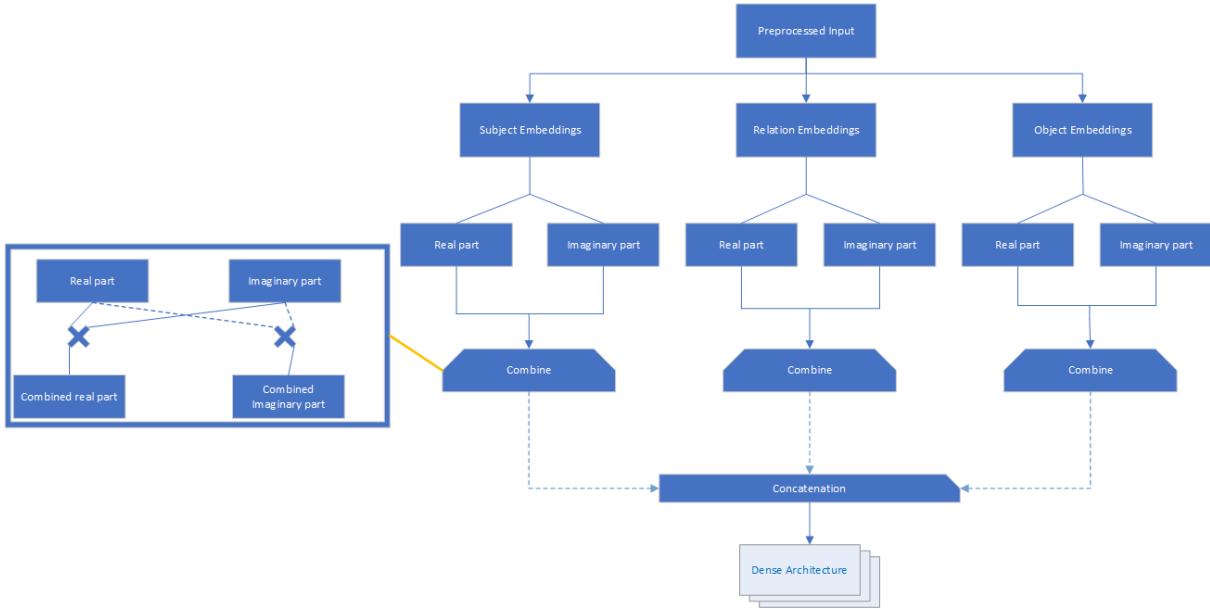


Figure 5.11: HoE Model

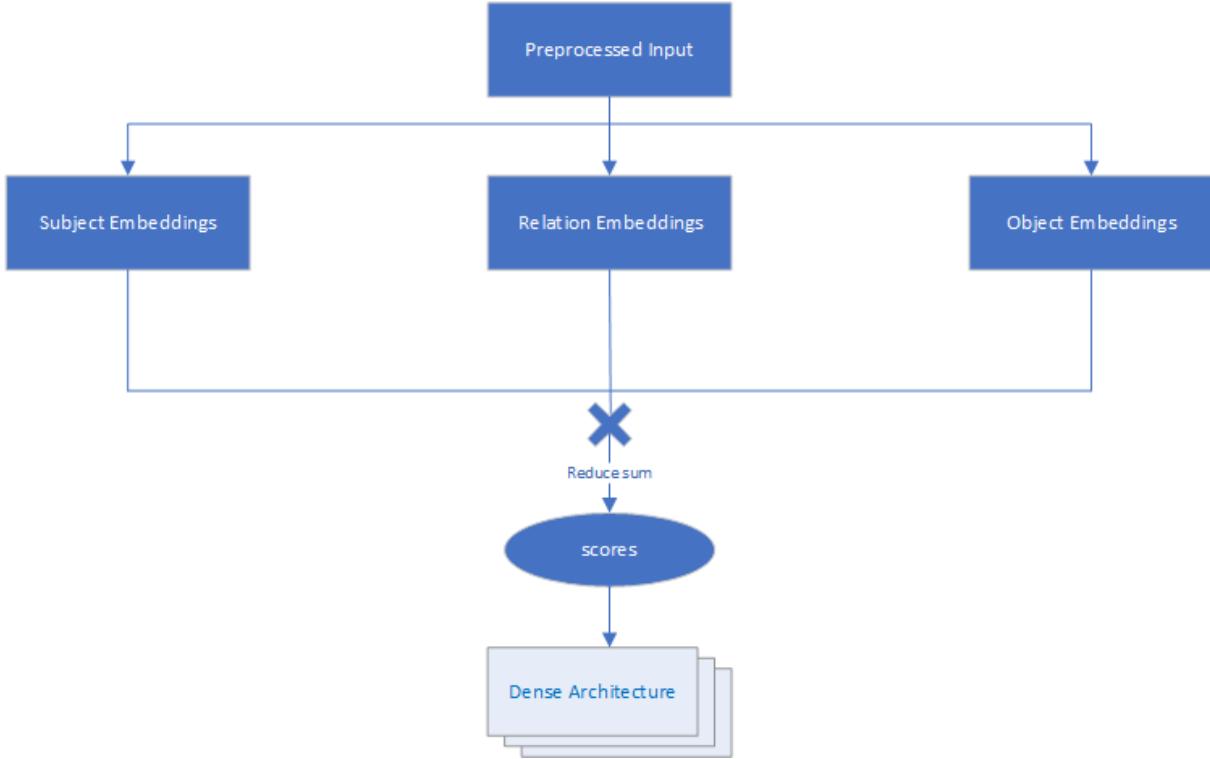


Figure 5.12: DistMult Model

TransH: With the exception of the embeddings and the number of epochs parameter, which we set at 30, the architecture and parameters of this model are identical to those of the previous one. Because we use early stopping, the number of epochs doesn't play a critical role for most of the models. The model employs embeddings for entities and relationships, as well as additional embeddings for in-degree, out-degree, and inverse relation frequency, like in the *TransE* model in this Group. It also incorporates dense layers for hyperplane projections and graph convolution techniques.

The *TransH* approach associates each relation with a hyperplane and projects the entities onto these hyperplanes. We reshape the hyperplane embeddings and use them to perform matrix-vector

multiplication with the entity embeddings, resulting in the projected entity embeddings. After creating a batch of normalization layers, we proceed to normalize these projected embeddings.

Next, we calculate a predicted object embedding by adding the relation embedding to the projected subject embedding. To form a combined embedding, the model concatenates the original subject embedding, the relation embedding, the projected object embedding, and the predicted object embedding. Figure 5.9 displays all the information mentioned above. The dense architecture processes this combined embedding.

We flatten the output from the final dense layer and pass it through a sigmoid-activated dense layer to produce a prediction. This prediction indicates the likelihood of the input triple being valid, allowing the model to perform triple classification tasks on the knowledge graph.

RotateE: This approach uses real and imaginary components in complex embeddings for entities and relations. By rotating entity embeddings in the complex plane, the approach models relational patterns.

We then construct different embeddings for the imaginary and actual aspects of the relationships and entities. These embeddings reflect the subjects, predicates, and objects in the input triples. The technique requires normalizing these embeddings to maintain their magnitudes.

We compute their modulus and divide each component by it to normalize the entity embeddings—real and imaginary portions. We calculate the modulus by taking the square root of the sum of the squares of the real and imaginary embeddings. This procedure ensures that the embeddings coincide with the unit circle in the complex plane, which is very important for *RotateE*'s rotation.

The relationship has also been normalized. We then build combined embeddings for the subject, predicate, and object by concatenating the normalized real and imaginary components of the entity and relation embeddings.

We then concatenate these merged embeddings to provide a consistent triple representation. Figure 5.10 displays all the information mentioned above. We then run the concatenated embedding across the dense architecture. The use of dense layers simulates a simplified graph convolution process, enhancing the model's ability to learn from the graph structure. All the parameters are the same as in the above models.

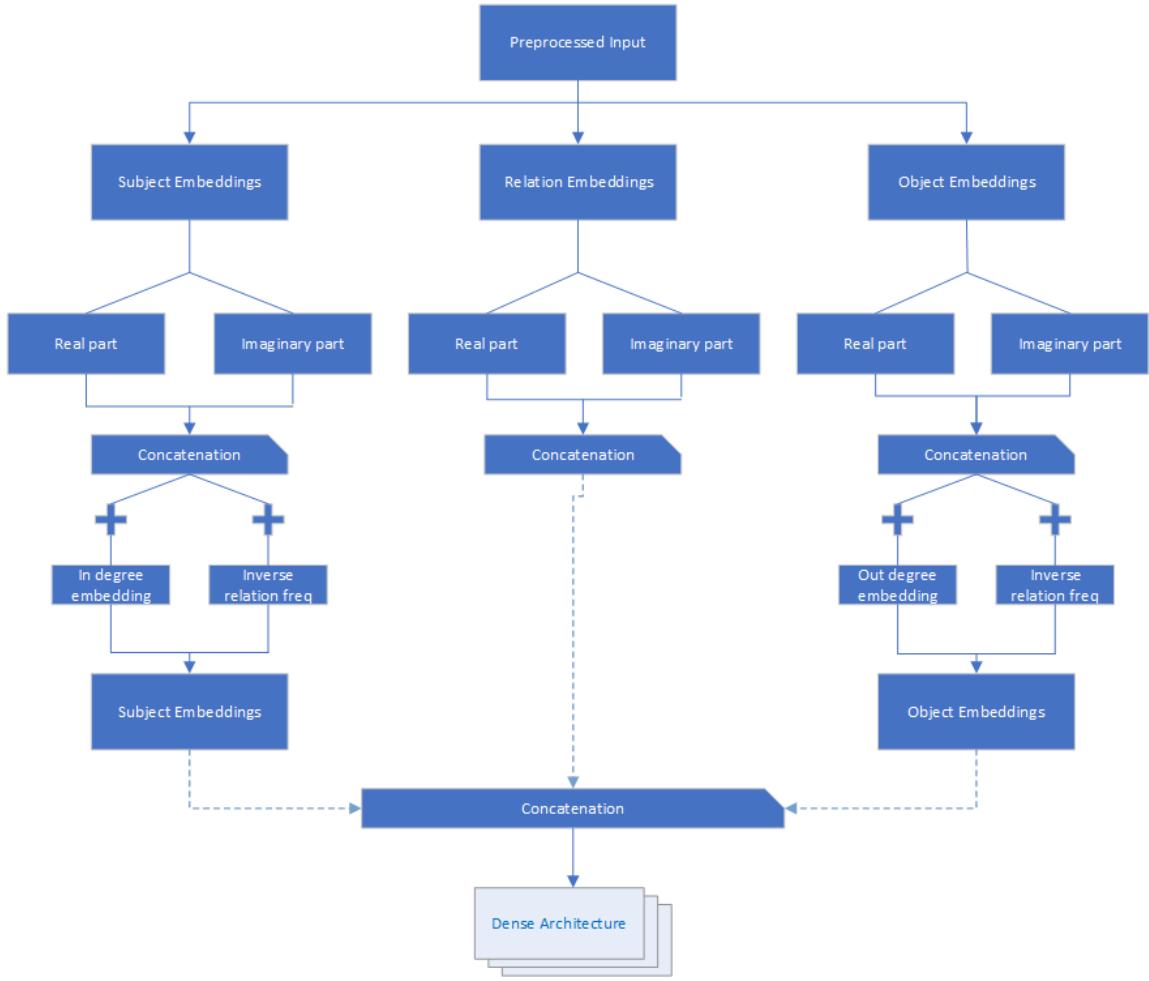


Figure 5.13: ComplEx Model

Hole: The model is based on the Holographic Embeddings (*HoIE*) approach. It employs real-valued embeddings for entities and relations, as well as a combination function for embedding interactions.

Initially, we create embeddings for the real parts of entities and relations. The method uses the circular correlation of these embeddings to show how the entities and relations in the input triples interact with each other.

The function combines the real parts of the embeddings using element-wise multiplication (although circular correlation typically involves a more complex operation, this simplified version uses direct multiplication). This function concatenates the combined real part with itself to form the final embedding representation.

The model's call method fetches the real part embeddings for the subject, predicate, and object. The function then combines these embeddings to create embeddings for the subject, predicate, and object.

Next, we concatenate these combined embeddings to create a unified representation of the input triple. This concatenated embedding is passed through a series of dense layers, each of which is followed by a dropout for regularization.

We flatten the output from the final dense layer and pass it through a sigmoid-activated dense layer to produce a prediction. The remaining parameters remain unchanged from the models mentioned above. Figure 5.11 displays all the information mentioned above.

DistMult: Here, we define the model using the *DistMult* approach. This model employs embeddings for entities and relations and leverages element-wise multiplication to capture the interactions between these entities and relations.

Initially, we create embeddings for entities and relations, which represent the subjects, predicates, and objects in the input triples. The *DistMult* approach involves calculating scores for the triples using element-wise multiplication of these embeddings.

The model's call method fetches the embeddings for the subject, predicate, and object. We compute the scores by performing an element-wise multiplication of the subject, predicate, and object embeddings, then summing the result across the embedding dimensions. This produces a score for each triple, indicating the strength of the relationship between the entities.

Next, we pass the computed scores to the Dense architecture. We flatten the output from the final dense layer and pass it through a sigmoid-activated dense layer to produce a prediction. The remaining parameters remain unchanged from the models mentioned above. Figure 5.12 displays all the information mentioned above.

ComplEx: We define a model using the *ComplEx* approach. This model employs real and imaginary embeddings for entities and relations, incorporating additional information such as in-degree, out-degree, and inverse relation frequency for each entity and relation.

Initially, we create embeddings for the real and imaginary parts of entities and relationships. For entities and relationships, we also use fixed embeddings to capture the in-degree, out-degree, and inverse relation frequencies. The difference between these and the above setups is that we set these embeddings to precomputed values, rendering them non-trainable. For this model, we calculate the in-degree, out-degree, and inverse relation frequencies in the preprocessing function as follows: These are arrays of zeros, each with a length equal to the number of entities, and another array, which has a length equal to the number of relations. We use these arrays to store the count of incoming edges for each entity, outgoing edges for each entity, and occurrences of each relation, respectively.

Next, the code iterates over each triple in the triples list, where s is the subject, p is the predicate (relation), and o is the object. For each triple, it increments the out-degree of the subject entity (s) by 1, the in-degree of the object entity (o) by 1, and the frequency of the relation (e) by 1. This loop effectively counts how many times each entity appears as a subject or object, as well as how many times each relation appears in the triples. Finally, the code calculates the inverse relation frequency by taking the reciprocal of the relation frequency plus one. This addition of one ensures that there is no division by zero in cases where a relation has not appeared in the triples.

The model's call method retrieves the real and imaginary parts of the embeddings for subject, predicate, and object. We concatenate these embeddings to form the complete embeddings for each component of the triple.

We add the in-degree and out-degree embeddings to the subject and object embeddings, respectively. We add the inverse relation frequency embedding to both the subject and object embeddings to

incorporate additional relational context. Next, we concatenate the combined embeddings for the subject, predicate, and object to create a unified representation of the input triple. We pass this concatenated embedding through the dense architecture. Figure 5.13 displays all the information mentioned above.

The model also includes skip connections implemented using linear layers. These skip connections connect the embeddings of the subject and predicate, as well as the object and predicate, to intermediate layers, allowing the model to capture more complex relationships. The implementation of skip connections necessitates the use of additional linear layers. These connections bypass specific layers in the dense network, allowing subject and predicate embeddings, as well as object and predicate embeddings, to be directly added to the intermediate layers of the network. More specifically:

- Skip Connection 1: This connection combines the subject and predicate embeddings. The first dense layer (dense1) adds these embeddings to its output. By incorporating this skip connection, the model preserves and directly feeds information about the subject and predicate into subsequent layers, thereby enhancing the learning process.
- Skip Connection 2: This connection combines the object's embeddings with the predicate. Similar to Skip Connection 1, it adds these embeddings to the output of the second dense layer (dense2). Again, this allows the model to retain information about the object, predict it, and incorporate it into the next layers of the network.

These skip connections serve to address the issue of vanishing gradients and facilitate the flow of gradient information during training. By providing direct paths for information flow, skip connections can help alleviate the problem of information loss in deep networks and enable more effective training.

We flatten the output from the final dense layer and pass it through a sigmoid-activated dense layer to produce a prediction.

Group 2

Here we use again the models from Group 1 with minor changes. The main change that we do in all models for this group is the use of the *StratifiedKFold* method.

TransE: Group 1's *TransE* model takes a simple approach to data splitting. We first arbitrarily split the data into test, validation, and training sets. We divided the dataset into three categories: 15% for testing, 15% for validation, and 70% for training. We split this dataset only once to ensure a fixed division. This approach helps the model assess performance on the test set after training and tweak the training and validation sets, respectively. Although this method is computationally fast and simple to use, if the selected split does not reflect the general data distribution, it might cause either overfitting or underfitting. Furthermore, the lack of variety in the data splits suggests that the performance measures derived from this one test set may not be very adaptable for other unknown data.

By comparison, the *TransE* model of this group makes use of stratified k-fold cross-validation, a more advanced and strong data splitting method. This approach divides the data into k equally sized folds—that is, k = 2. Every fold preserves the percentage of each class by having a balanced mix of positive and negative instances. We then run the model k times, each time using a separate fold as the

validation set and the remaining $k-1$ folds as the training set. This procedure guarantees the use of every dataset instance for both training and validation, offering a comprehensive assessment of the model's performance across numerous data splits.

Stratified k -fold cross-validation has several benefits. Exposing the model to many subsets of the data during training and validation helps first reduce the danger of overfitting to a certain data division. By averaging the data from many folds, it also increases the dependability of the performance measures. This is a more realistic estimate of the model's generalizing capacity. Finally, this approach reveals possible variation in the model's performance across many data splits, providing information on its stability and resilience.

Nonetheless, stratified k -fold cross-validation is computationally more demanding than a single train-test split. Multiple training and validation repetitions increase the computational overhead and the model's development time requirement. Still, the advantages of getting a more consistent and generally applicable assessment usually exceed the extra computational expenses. Furthermore, the embeddings changed. Although it does not explicitly start with precomputed values, the *TransE* model of Group 1 contains inverse relation frequency embeddings. Conversely, we configure the inverse relation frequency embeddings to be non-trainable and start them with precomputed values from the dataset's relation frequencies.

This guarantees that, from the beginning, the embeddings fairly represent the actual distribution of relations in the graph. The in- and out-degree embeddings correspond to those in Group 1's model. We nonetheless include the entity degrees in the triples. First, we determine the degrees of the entities involved in the triples. These degrees serve as counts of the frequency with which every entity occurs in the dataset as a subject or objective. Method 1 computes the entity degrees, then produces negative triples. We then expand each negative triple to include the degrees of its subject and object entities.

We also extend positive triples based on the degrees of their individual components. We aggregate the enhanced positive and negative triples into a single dataset and generate matching labels to show whether every triple is positive (1) or negative (0). This approach, by considering how significant entities are in the graph, provides extra information to the triples (entity degrees), enabling the model to distinguish between valid and invalid triples. The Group 1 model's setup and parameters are also applicable here.

TransH: For this model, we maintain the modifications we previously made for this group's *TransE* model. We set the value of k to 3. We maintain the same parameters and setup as Group 1's model, except that the embedding dimension parameters are set to 512.

RotateE: Again, we keep the changes we made above for this group's *TransE* model. We set the value of k to 3. The only differences from Group 1's corresponding model are that we set the embedding dimension to 512 and concatenate each embedding once to the end, rather than separately for the subjects, relations, and objects.

HoIE: Again, we keep the changes we made above for this group's *TransE* model. We set the value of k to 3. We set the embedding dimension to 512, which differs from the corresponding model of Group 1. The rest of the parameters are the same as in the corresponding Group 1 model.

DistMult: Again, we keep the changes we made above for this group's *TransE* model. We set the value of k to 2. Because we use degree and inverse relation frequency embeddings, like in the *TransE* model of this group, we don't directly pass the sum of the product of the embeddings to the dense architecture, like in the corresponding model of Group 1. First, we concatenate all the entity and relation embeddings, and then we pass the reduced sum of the product of the embeddings to the dense architecture.

ComplEx: Again, we keep the changes we made above for this group's *TransE* model. We set the k value to three. This model does not use skip connections, as the corresponding model in Group 1 does. We set the embedding dimension to 512. One last difference from the corresponding model of Group 1 is that we don't add the in, out, and inverse relation frequency embeddings to the corresponding subject, relation, and object embeddings, but we directly concatenate them to the subject, relation, and object embeddings.

Group 3

For this group of experiments, we use the [Method 4](#) for negative samples generation.

TransE, *TransH*: These models have the same setup and parameters as in Group 1.

RotateE: The parameters of this model are the same as in the corresponding model of Group 1. However, there is a difference in the embeddings. To accommodate the model's requirements, we construct entity embeddings with double the usual dimensionality, dividing them into real and imaginary parts to form complex embeddings. This technique allows the model to utilize the geometric properties of complex numbers, which is crucial for modeling the rotational relationships inherent in certain types of data. The main entity embeddings are more accurate when they include extra embeddings that show the in-degree and out-degree of entities, as well as the inverse relation frequency. This is similar to what the *TransE* model in Group 1 does.

This model applies uniform normalization across the entity embeddings using batch normalization, ensuring the standardization of embeddings to stabilize and improve the training process. The Group 1 model, on the other hand, normalizes the real and imaginary parts of the embeddings based on their modulus separately. It keeps the unit length of the embeddings, but it doesn't normalize them as well as this model does overall.

This model also includes a more complex operation that changes the embeddings into a complex space and uses the Fast Fourier transform (FFT) and its inverse (IFFT) to do circular correlation. This operation enables the model to compute relational patterns in a rotational manner, significantly enhancing the expressiveness of the embeddings. We then integrate the resultant into the dense layers for further processing, ensuring that these complex relational patterns contribute to the final prediction. The rest of the parameters are the same as the corresponding model in Group 1.

HoIE: Here, we try to combine two different embedding methods. We use the exact setup in the embeddings, such as in this group's *TransE* model. Then, in the dense architecture, before we make the final prediction, we use the method *combine_hole_embeddings*, the same as in the corresponding model in Group 1, to combine the subject, predicate, and object embeddings. We then concatenate the output of the flattening layer and the combined embeddings to make the final prediction. The rest of the parameters are the same as the corresponding model in Group 1.

DistMult: In a similar vein, Group 1's *TransE* model uses subject, predicate, object, in degree, out degree, and inverse relation frequency. After normalization, we compute the score, like in the corresponding model from Group 1, and concatenate this score with the subject, predicate, and object embeddings. The rest of the parameters are the same as the corresponding model in Group 1.

ComplEx: This model is simpler than the one in Group 1. It doesn't use skip connections or structural information from the graph (degree and relation inverse frequency embeddings). It just uses real and imaginary embeddings for the subjects, predicates, and objects. After we concatenate these embeddings, we feed them into the dense architecture without normalizing them first. The rest of the parameters are the same as the corresponding model in Group 1.

Group 4

In this group of experiments, we introduce a graph convolution layer named *GCNLayer*, shown in Figure 5.14. We specifically design graph-based neural network models to perform convolution operations over graph-structured data. We initialize it with the number of units (*units*) and an optional activation function (*activation*). The *units* parameter specifies the output space's dimensionality, which determines the number of features in the output vectors for each node in the graph. The *activation* parameter enhances the expressive power of the layer's output by applying a non-linear activation function.

During the initialization phase, the *GCNLayer* sets up its configuration based on the provided parameters. The call to the build method initializes the layer's learnable parameters. The build method specifically creates a weight matrix (*kernel*) with a shape defined by the input feature dimension and the specified number of output units. We initialize this weight matrix using the *Glorot* uniform initializer, which sets the initial values to facilitate efficient training.

In the call method's forward pass, the layer applies the learned transformations to the input data in the forward pass of the call method. The matrix multiplication operation (*tf.matmul*) multiplies the input features by the weight matrix. This operation transforms the input features into the output space, which is defined by the number of units. The transformed features use the activation function, if specified during initialization, to introduce non-linearity into the model. The result is the *GCNLayer*'s final output, which can then be passed to subsequent layers or used for predictions.

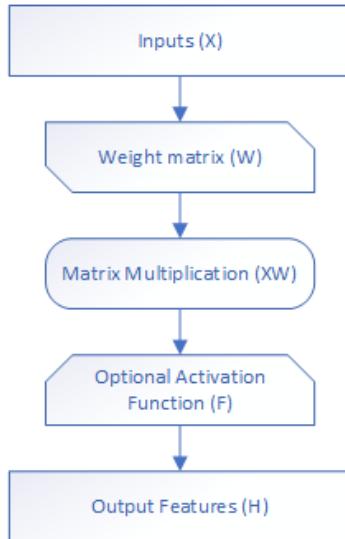


Figure 5.14: *GCNLayer*

TransE: Here, we utilize subject, relation, object in and out degree, and inverse relation frequency embeddings, similar to those found in the corresponding model in Group 1. The model initializes extra embeddings, especially for *TransE* representations of entities and relations, in order to use the *TransE* method. Using `tf.keras.layers.Embedding`, the model first generates these embeddings—whose dimensionality matches that of the main embeddings. Including layers. These embeddings may capture distinct aspects of the relational structure, which the other model embeddings may not fully represent. This implementation directly integrates *TransE*-like embeddings into the model, unlike the traditional *TransE* model, which combines embeddings using vector arithmetic to model relationships. We concatenate the embeddings for subject, predicate, and object with the outputs of the GCN layers; there are two *GCNLayer*s just before the dense architecture with parameters `units = embedding_dim, activation = tf.nn.relu`. This direct concatenation allows the model to leverage the rich relational information embedded in the *TransE*-like vectors without explicitly performing the translational arithmetic.

The model effectively combines the relational features picked up by the embeddings with the structural features learned by the GCN by combining both regular and *TransE*-like embeddings with the outputs of the GCN layers. This hybrid approach allows the model to learn a more comprehensive representation of the graph data, capturing both local neighborhood structure and global relational context. Figure 5.15 displays all the above information. The model maintains the same parameters as Group 1's model, with the exception of setting the learning rate to 0.0001.

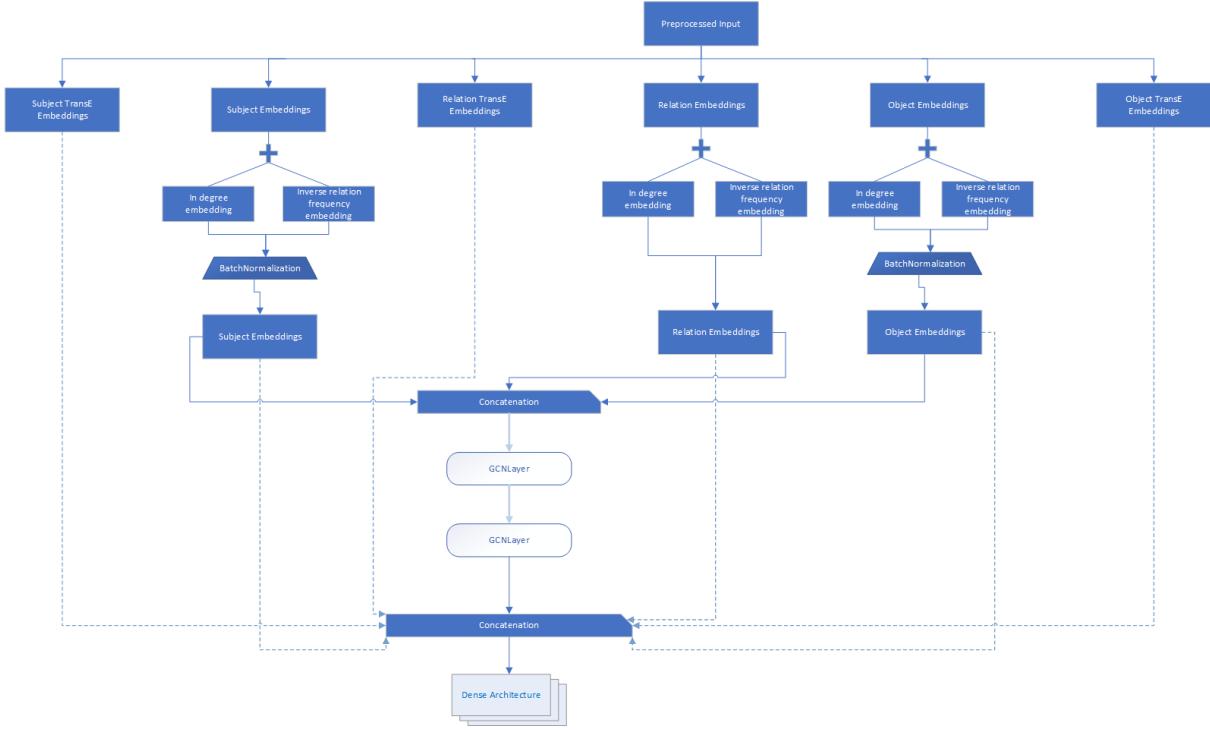


Figure 5.15: TransE Model for Group 4

TransH: This model employs a different embedding strategy by combining a Graph Convolutional Network (GCN) and a more traditional approach from Group 1. Similar to the Group 1 model, it uses standard entity and relation embeddings, as well as scalar embeddings for in-degree, out-degree, and inverse relation frequency. However, instead of using hyperplane embeddings, this model employs a normal vector for each relation to define the hyperplane in the mechanism. We project entity embeddings onto these relation-specific hyperplanes by removing the entity embedding component along the normal vector. As a result, the entity embeddings project onto the hyperplane. The model then uses GCN layers to further refine these embeddings, exactly like in the *TransE* model in this Group. The GCN layers gather data from nearby nodes and directly add relational context to the entity embeddings. We concatenate this aggregated information with the initial entity and relation embeddings, then pass it through the dense architecture for the final prediction. The model maintains the same parameters as Group 1's model, with the exception of setting the learning rate to 0.0001.

HoIE: This model is based on *HoIE* (Higher-order Interaction Embeddings) embeddings. It uses standard entity and relation embeddings and augments them with *HoIE* embeddings, which encode higher-order interactions between entities and relations. We divided the *HoIE* embeddings into two parts: one to augment the subject embedding, and another to augment the object embedding. This transformation captures more complex, higher-order interactions, which are crucial for understanding the multi-faceted nature of relationships in knowledge graphs. Additionally, this model includes embeddings for in-degree, out-degree, and inverse relation frequencies, like the *TransE* model in Group 1. The model then normalizes these enriched embeddings and processes them through GCN layers, like in the *TransE* model of this Group. We join the results of these GCN layers with the first embeddings to make a full feature vector that shows both local and global graph structures. We then pass this feature vector through dense layers to make the final prediction. Group 1's parameters apply to the rest of the model, except for setting the learning rate to 0.0001.

DistMult: Here, we use structural embeddings that are similar to those in Group 1's *TransE* Model. Actually, the first part of the embeddings—until the normalization—is very similar to the *TransE* model of Group 1. After normalizing the subject and object embeddings, we feed the subject, relation, and object embeddings into two GCN layers, like in this group's *TransE* model. We concatenate them and feed them into the dense architecture. Then, in the dense architecture, after the flatten layer, we compute the reduced sum of the product of the flatten layer's output, the predicate, and the object embeddings. At the end, we make the prediction. The model maintains the same parameters as Group 1's model, except for setting the learning rate to 0.0001.

ComplEx: This model is simpler than the corresponding one in Group 1. It doesn't use skip connections. It uses in-and-out degrees and inverse relation frequency embeddings, like in the *TransE* model in Group 1. It uses real and imaginary parts for the subjects, predicates, and objects. Similar to the *TransE* model in Group 1, we join the imaginary and real parts together. We achieve this by incorporating the in and out degrees and inverse relation frequency embeddings into the real part of the embedding. Next, we provide the concatenation of the subjects, predicates, and object embeddings as inputs to the two subsequent GCN layers, like in the *TransE* model in this Group. The dense architecture then follows. The model maintains the same parameters as Group 1's model, with the exception of setting the learning rate to 0.0001.

NoEmbds: This model uses embeddings, but it does so in the simplest possible form, without making any transformations to the embeddings. This model uses in, out, degree, and inverse relation frequency embeddings, as well as subjects, predicates, and object embeddings. It does the same things that the *TransE* model in Group 1 did. Then it normalizes the embeddings again, like in the *TransE* model in Group 1, and gives as input to the two following GCN layers the concatenation of the subject, predicate, and object embeddings. The dense architecture then follows. The model maintains the same parameters as Group 1's *TransE* model, with the exception of setting the learning rate to 0.0001.

Group 5

In this group of experiments, we introduce a graph attention layer named *GATLayer*, shown in Figure 5.16. This layer, a sophisticated neural network layer, leverages the attention mechanism to perform graph-based operations. This layer is characterized by its ability to perform multi-head attention on the input features, enhancing the representation learning capability for graph-structured data. Several hyperparameters initialize the layer: the number of attention heads (*num_heads*), the dimensionality of the output space (*output_dim*), the activation function, and the dropout rate (*dropout_rate*).

In the initialization phase, the *GATLayer* sets up the configuration for the attention heads and defines the activation function using the TensorFlow activation API. At this stage, we also define the dropout rate to prevent overfitting during training.

The call to the build method initializes the layer's learnable parameters. Specifically, it creates a weight matrix (kernel) and a bias vector (bias) for each attention head. The kernel matrices convert the input features into the output space, adding biases to these transformed features. During initialization, we specify both the input feature dimension and the output dimension. To ensure proper weight initialization, we use the *Glorot* uniform initializer for the kernels and initialize the biases to zeros.

During the forward pass, implemented in the call method, the layer applies the learned transformations to the input data. The layer multiplies each attention head's input features by the

corresponding kernel matrix, then adds the corresponding bias. These operations generate multiple transformed feature sets, one for each attention head. We then concatenate the outputs from all heads along the last dimension, effectively combining the multi-head attention results. We pass the concatenated outputs through the specified activation function to introduce non-linearity, thereby enhancing the expressive power of the layer. During training, we apply dropout to the activated outputs to improve generalization, randomly omitting certain neurons with a probability defined by the dropout rate.

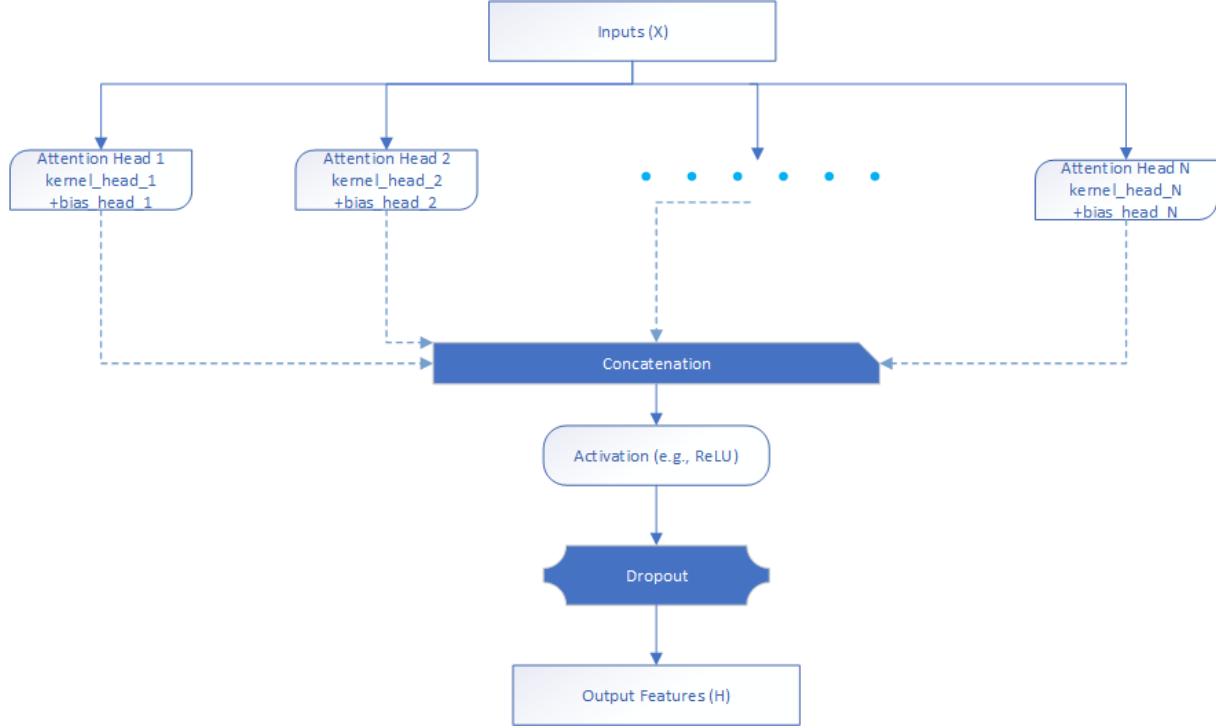


Figure 5.16: GATLayer

For this set of experiments, for the most part, we modify the dense architecture of Figure 5.6 slightly to simplify it. We remove all the dropout layers and the flatten layer from the model, and we apply L2 regularization to all dense layers. We evaluate and utilize the models on the same graph, so allowing the model to slightly overfit during training may not necessarily be detrimental. This is because the model's overfitting implies that it has learned to accurately capture the specific patterns and relationships within the training graph. Therefore, the model's high performance on the training data should theoretically transfer to the test data in the same graph, as it is unlikely to encounter significantly different structures or relationships. However, it is important to acknowledge the potential risks associated with overfitting. Most of the models in this Group use the *MarginRankingLoss*, which has a margin of 0.1.

TransE: For this model, the embeddings are pretty straightforward. We use subject, predicate, and object embeddings. We perform the transformation for object embeddings, similar to the corresponding model in Group 1, then concatenate all the embeddings, apply a dropout (rate 0.5), and then apply one *GATLayer* (*num_heads* = 4, *output_dim* = *embedding_dim*, *dropout_rate* = 0.5). Then the dense architecture follows. We also use a different loss function, *MarginRankingLoss* (margin 0.1), which we use not only here but for many experiments below. Machine learning uses this custom loss function to rank positive samples higher than negative samples by a specified margin.

The `MarginRankingLoss` class, shown in Figure 5.17, is a subclass of `tf.keras.losses`. TensorFlow-based neural network models can seamlessly integrate with Loss. This class initializes with a single parameter, `margin`, that determines the margin for ranking positive samples higher than negative samples. The application's specific requirements can adjust the default value of 0.1 for this margin.

In the call method, the core functionality of the margin ranking loss is defined. This method computes the loss based on the predicted scores (`y_pred`) and the true labels (`y_true`). We assume the true labels to be binary, where 1 signifies positive samples and 0 indicates negative ones. The process begins by separating the predicted scores into positive and negative scores using `tf.boolean_mask`, which filters the predictions based on the true labels.

To make it easier to compute pairwise differences, we reshape the isolated positive and negative scores. We expand positive scores along the rows and negative scores along the columns. This reshaping allows the creation of a margin matrix, where each element represents the difference between a positive and a negative score, adjusted by the specified margin.

We compute the margin matrix by subtracting the positive and negative scores from the specified margin. We then pass this matrix through a `tf.maximum` function, which effectively applies the hinge loss concept by ensuring that only positive differences contribute to the loss. Using `tf.reduce_sum`, we calculate the final loss by summing all positive values in the margin matrix.

This loss function encourages the model to rank positive samples higher than negative ones by at least the specified margin, thus promoting a clear separation between positive and negative predictions. The model maintains the same parameters as Group 1's *TransE* model, with the exception of setting the learning rate to 0.0001 and the embedding dimension to 512.

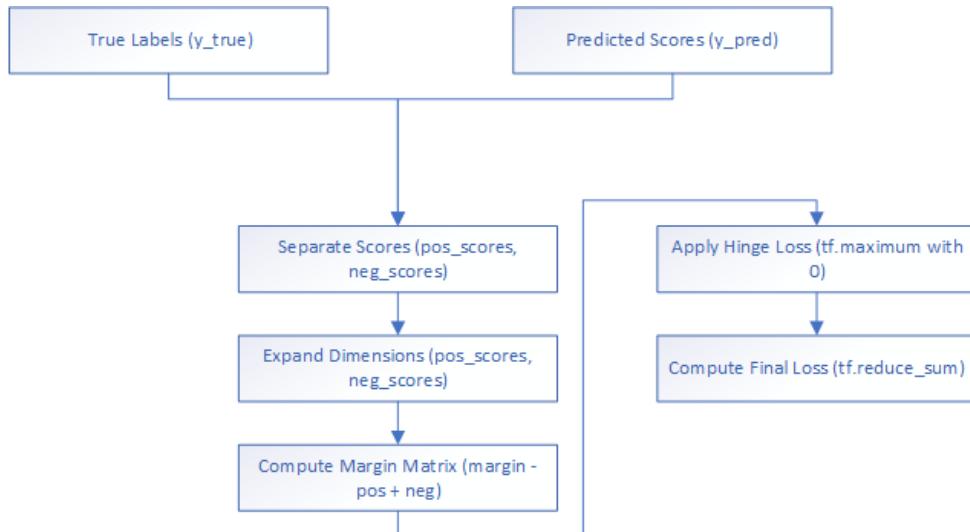


Figure 5.17: `MarginRankingLoss` function

TransH: Group 4's corresponding model employs a simple embedding strategy, initializing entity and relation embeddings using standard embedding layers. We create the entity embeddings using `tf.keras.layers.Embedding`. We create the embedding using parameters for both the number of entities and the embedding dimension. Similarly, we create relation embeddings based on the number of relations. Additionally, another embedding layer creates a normal vector specific to each relation, representing *TransH*-like embeddings. We use this normal vector to project entity embeddings onto

relation-specific hyperplanes, helping to model the multi-faceted nature of relationships in the data. We further refine the embeddings by incorporating degree and relationship information.

In contrast, this model doesn't use degree and relation information and adopts a more complex approach by distinguishing between embeddings in two different spaces: the entity space and the hyperplane space. The model specifically creates separate entity and relation embeddings for the entity space and the hyperplane space. By handling projections and transformations differently in each space, this dual embedding approach enables more sophisticated modeling of the interactions between entities and relations. This model projects entity embeddings onto relation-specific hyperplanes in a way that is similar to Group 4's model, but with the added complexity of embeddings having their own spaces. To avoid overfitting, we concatenate the projected embeddings with the relation embeddings, then process them through a dropout layer. Then follows one GATLayer, like in the *TransE* model in this Group, and the dense architecture. The model maintains the same parameters as Group 5's *TransE* model.

RotateE: This model utilizes both entity and relation embeddings. Using `tf.keras.layers.Embedding`, we initialize the entity embeddings. To accommodate complex-valued representations, the embedding layer has a dimensionality twice that of the embedding layer. Similarly, we initialize the relation embeddings with the same dimensionality. These embeddings are critical for representing the entities and relations within a complex vector space, which enhances the model's ability to capture intricate relational patterns. This model uses a custom layer the *RotateEncoder*, which we use it to encode the entity embeddings.

The *RotateEncoder*, a custom layer that encodes entity embeddings through a rotation-based transformation, is a unique aspect of this model. The *RotateEncoder* takes advantage of complex numbers to *RotateE* entity embeddings in a way that reflects the relational structure. This encoding process is essential for modeling the interactions between entities and relations effectively. The encoder splits the input embeddings into real and imaginary parts, then applies rotational transformations using learned weights. Specifically, it multiplies the real part of the embeddings by the cosine of the learned weights and subtracts the imaginary part multiplied by the sine of the weights to obtain the rotated real part. On the other hand, we obtain the rotated imaginary part by multiplying the real part by the sine of the weights and then adding the multiplied imaginary part by the cosine of the weights. The encoded entity embeddings use the concatenated representation of the rotated real and imaginary parts.

After encoding the entity embeddings, the model calculates the circular correlation between the entity and relation embeddings. The model performs complex multiplication by multiplying and summing the real and imaginary parts of the embeddings, which reflects the intricate interactions between entities and their relations. The resulting complex correlation captures relational semantics in a way that simple real-valued operations cannot.

To prevent overfitting and enhance generalization, the model incorporates a dropout layer with a specified dropout rate, which stochastically sets input units to zero during training. The model applies this dropout to the complex multiplication's output, thereby introducing regularization. Using one GATLayer, like in the *TransE* model in this Group, the model further refines the embeddings. Then the dense architecture follows. The model maintains the same parameters as Group 5's *TransE* model, with the exception that it uses the *BinaryCrossentropy* loss function.

HoIE: This model uses both real and imaginary components for entity and relation embeddings. It combines them to make the most of the *ComplEx* embedding model's features, which handle complex relational patterns well. For the real and imaginary components, we initialize the entity and object embeddings using separate embedding layers. To enhance the representational capacity, the model incorporates higher-order interaction embeddings (*HoIE*). An embedding layer, *hoie_embeddings*, creates these by encoding additional relational information for each entity. We divide the *HoIE* embeddings into real and imaginary components, doubling the embedding dimension to capture more nuanced interactions. A holographic embedding function, *combine_hole_embeddings*, merges these components by multiplying the real and imaginary parts and concatenating them. We apply this process to both subject and object embeddings, yielding *s_hoie* and *o_hoie*, respectively.

The forward pass concatenates the real and imaginary parts of the subject, predicate, and object embeddings to form the full complex embeddings, *s_embed*, *p_embed*, and *o_embed*. These complex embeddings then merge with the holographic embeddings. Then follows one GATLayer, like in the *TransE* model in this Group, and the dense architecture; for this experiment, we use the full architecture with the dropout layers. The remaining layers and model parameters are identical to the other models in this group.

DistMult: For this model, we use subject, predicate, and object embeddings, and the product of these embeddings is used as a score. Then we apply a dropout layer. Then follows one GATLayer, like in the *TransE* model in this Group, and the dense architecture. This model uses the same setup and parameters as the *TransE* model in Group 5.

ComplEx: For this model, we use real and imaginary parts for entity and relation embeddings. We initiate the embeddings using separate embedding layers for the real and imaginary parts of both entities and relations. Additionally, the model includes embeddings for entity degrees and inverse relation frequencies, similar to the *TransE* model in Group 1. The forward pass retrieves the real and imaginary parts of the subject, predicate, and object embeddings. We then concatenate these embeddings to create the combined *ComplEx* embeddings. We then further combine these concatenated embeddings into a single tensor, *concat_embed*, which encapsulates the information. A GATLayer, like in the *TransE* model in this Group, processes the tensor. The dense architecture then follows. This model uses the same setup and parameters as the *ComplEx* model in Group 4.

Group 6

For this group, we use both *GCNLayer* and *GATLayer* in the models. Also, for most experiments, we use the dense architecture without dropout layers and with L2 regularization at all layers, as in Group 5. Most of the models in this Group use the *MarginRankingLoss* with a margin of 0.1.

TransE: Here, we use subject, predicate, and object embeddings, similar to the *TransE* model in Group 1. After the concatenation, we apply a dropout layer (rate 0.5), followed by two *GCNLayers* with parameters *units = embedding_dim*, *activation = 'relu'*, and *GATLayers* with parameters *num_heads = 4*, *output_dim = embedding_dim*, *dropout_rate = 0.5*. The dense architecture then follows. Group 5 contains the remaining parameters and configuration of the *TransE* model.

TransH: Here, we implement the model by projecting the entity embeddings onto the relation hyperplanes. We achieve this projection by subtracting the entity embeddings' parallel component to the relation embedding, which effectively positions the entity embeddings onto the relation

embedding's defined hyperplane. Additionally, the relation projection embeddings transform, normalize, and reshape the relation embeddings appropriately.

Following the projections, we concatenate the projected subject, predicate, and object embeddings into a single representation. Then follows a dropout layer, with a rate of 0.5, two GCN and GAT layers, like in the *TransE* model in this Group, and the dense architecture. The rest of the parameters and configuration are the same as in the corresponding model in Group 5.

RotatE: For this model, we make some changes to the *MarginRankingLoss*, the GAT, and the GCN layers. The *MarginRankingLoss* computes the loss by comparing positive and negative scores directly, whereas the *ComplexMarginRankingLoss* separates the real and imaginary parts of the embeddings, handling them independently before combining the losses. Both the *GATLayer* and the *GCNLayer* work with real-valued inputs. The *GATLayer* uses attention mechanisms to focus on important parts of the graph, while the *GCNLayer* gathers data from nodes that are close by. On the other hand, the *ComplexGATLayer* and *ComplexGCNLayer* cater to complex-valued inputs, separating the real and imaginary components for independent processing before merging them.

Also, this model uses a more complex encoder-decoder mechanism based on complex-valued operations. The *RotatEEncoder* in this model creates separate real and imaginary embeddings for entities and relations. It uses these embeddings to produce complex-valued representations of the subject (*s_embed*), predicate (*p_embed*), and object (*o_embed*). The *RotatEDecoder* then computes the score by reducing the sum of the element-wise product of the complex embeddings. In the model, we use real and imaginary embeddings; we concatenate the real and imaginary embeddings; we transform the object embedding, calculating the product of subject and predicate embeddings; and we concatenate the subject, predicate, and transformed object embeddings. The model then follows two GCN and GAT layers, like in the *TransE* model in this Group. The dense architecture then follows. The rest of the parameters and configuration are the same as in the corresponding model in Group 5.

HoIE: This model retrieves the corresponding real and imaginary embeddings for the subject and object entities, as well as the predicate relation. We then combine the embeddings to create relation-specific transformed embeddings for the object, which we calculate as follows: We derive the real part of the object embedding by subtracting the product of the subject's real and predicate's real embeddings from the product of the subject's imaginary and predicate's imaginary embeddings. We obtain the imaginary part by adding the product of the subject's real and predicate's imaginary embeddings to the product of the subject's imaginary and predicate's real embeddings.

Next, the model concatenates the real and imaginary parts of the subject, transformed objects, and original object embeddings, resulting in a complex embedding that integrates both real and imaginary components. A dropout layer with a rate of 0.5 further processes this combined embedding to prevent overfitting during training. Then follow two GCN and GAT layers, like in the *TransE* model in this Group, and the dense architecture. The rest of the parameters and configuration are the same as in the corresponding model in Group 5.

DistMult: For each triple, the model retrieves the corresponding embeddings for the subject, predicate, and object, like in most of the other models. The model then subjects these embeddings to a dropout layer with a rate of 0.5, randomly setting a fraction of the embedding values to zero during training, thereby preventing overfitting and enhancing generalization. Following dropout, the

embeddings pass through two GCN and GAT layers, like in this Group's *TransE* model. The rest of the parameters and configuration are the same as in the corresponding model in Group 5.

ComplEx: For this model, we use real and imaginary parts for subject, predicate, and object embeddings. The model then performs a relation-specific transformation on the object embeddings. In this transformation, we perform complex multiplication, multiplying the real part of the subject embedding by the real part of the predicate embedding and subtracting it from the product of the imaginary parts. The real part of the predicate embedding multiplies the imaginary part of the subject embedding simultaneously, adding it to the product of the real part of the subject embedding and the imaginary part of the predicate embedding. This results in the transformed real and imaginary parts of the object embedding.

Next, to form a combined embedding, the model concatenates the subject's real and imaginary parts, transformed objects, and object embeddings. We concatenate the real parts together, and similarly, we concatenate the imaginary parts together. We then combine these concatenated real and imaginary embeddings to form the final complex embedding. To avoid overfitting, we subject the concatenated embeddings to a 0.5 dropout rate. Next, the model incorporates two GCN and GAT layers along with a dense architecture, mirroring the other models in this group. The rest of the parameters and configuration are the same as in the corresponding model in Group 5.

Group 7

For this group of experiments, we use the Conv1D architecture (Figure 5.7) combined with the different embedding manipulation methods we use above.

TransE: Here, we utilize subject, predicate, and object embeddings, similar to those found in the *TransE* model in Group 1. In Table 5.3, we can see the parameters of this model. Most models that use the Conv1D architecture will use these parameters.

Table 5.3: Parameters for the TransE model in Group 7

Parameter	Value
Embedding Dimension	400
Number of Epochs	15
Batch Size	256
Optimizer	Adam
Learning Rate	0.00005
Loss Function	Binary Crossentropy
Metrics	Accuracy
Learning Rate Scheduler	ReduceLROnPlateau
Monitor	val_loss
Factor	0.5
Patience	4
Minimum Learning Rate	1e-7
Verbose	1
Monitor	val_loss
Patience	4
Restore Best Weights	True

TransH: The model embeds entities and relationships, resulting in dense vector representations for each entity and relation. The relation projections define a hyperplane onto which we project the subject embeddings. This projection operation involves subtracting the dot product of the subject embedding and the relation projection from the subject embedding itself, scaled by the relation projection. We then translate the projected subject embedding by adding the relation embedding, resulting in the transformed object embedding. Additionally, we compute a *TransE*-like embedding by simply adding the relation embedding to the subject embedding, forming the predicted object embedding.

To form a comprehensive embedding tensor, we concatenate these transformed and predicted embeddings with the original subject and relation embeddings. To prepare for the Conv1D architecture, we expand this concatenated tensor with an additional channel dimension. This model sets patience to 2 in *ReduceLROnPlateau*. The remaining parameters and configuration and setup are identical to those of the *TransE* model in this Group.

RotateE: Initially, we embed entities and relations using `tf.keras.layers.Embedding`. The process of embedding involves representing each entity with a vector twice the length of the specified embedding dimension, and each relation with a vector of the same embedding dimension. In addition

to these standard embeddings, the model uses separate embeddings for the *RotateE* approach and stores them in embedding layers with dimensions scaled appropriately for complex number operations, just like before.

For each input triple, we retrieve the corresponding embeddings for both the standard and *RotateE* embeddings. We then split the *RotateE* embeddings into real and imaginary parts, allowing us to perform complex number operations. We specifically split the subject and object embeddings into their respective real and imaginary components.

The *RotateE* approach's core involves applying a rotation to the embeddings. We achieve this by separately calculating the real and imaginary components of the score. We obtain the real score by summing the product of the relation embedding, the product of the real parts, and the product of the imaginary parts of the subject and object embeddings. We compute the imaginary score similarly, subtracting the product of the real part of the subject and the imaginary part of the object from the product of the imaginary part of the subject and the real part of the object. We then stack and sum these real and imaginary scores to generate a final score. We expand this final score with an additional channel dimension to prepare it for further processing by the Conv1D architecture. The remaining parameters and setup are identical to those in this group's *TransE* model.

NoEmbs: For this model, we simply use subject, predicate, and object embeddings, concatenate them, and feed the result to the Conv1D architecture. The remaining parameters and setup are identical to those of the *TransE* model in this Group.

HoIE: Initially, entities and relations are embedded using `tf.keras.layers.Embedding`. Additionally, an embedding layer stores *HoIE* embeddings with dimensions scaled appropriately for real and imaginary parts.

For each input triple, the system retrieves both the standard and *HoIE* embeddings. We compute the *HoIE* embeddings by combining the real and imaginary parts of the embeddings using a circular correlation operation, similar to the *HoIE* model in Group 1. We then pass the resulting *HoIE* embeddings through two dense layers with *ReLU* activation functions to ensure non-linearity. These dense layers serve to capture complex interactions between entities, enhancing the expressiveness of the model.

The modified concatenation process combines the computed *HoIE* embeddings with the input embeddings. The Conv1D architecture then passes the concatenated embeddings, which include subject, predicate, and object embeddings, along with the computed *HoIE* embeddings for both subject and object. These convolutional layers apply filters to capture patterns in the input embeddings, aiding in feature extraction. The remaining parameters and setup are identical to those of the *TransE* model in this Group.

DistMult: Here, exactly like the corresponding model in Group 1, we compute the *distmult* score of the embeddings and pass it through the Conv1D architecture. The remaining parameters and setup are identical to those of the *TransE* model in this Group.

ComplEx: For this model we use real and imaginary parts for embeddings only for the subject and predicate. Then we concatenate these embeddings and feed them into the Conv1D architecture. The remaining parameters and setup are identical to those of the *TransE* model in this Group.

Group 8

For this Group we use the Conv1D architecture combined with the models in Group 7 and the *GCNLayer*.

TransE: For this model, each entity and relation in the input triples is represented using two types of embeddings: *TransE* embeddings and standard embeddings. These embeddings have the same dimensions and are retrieved from their respective *tf.keras.layers.Embedding* layers, with *TransE* embeddings capturing translational relationships and standard embeddings providing more general representations. These embeddings are then concatenated to form a comprehensive embedding vector that encapsulates both types of representations. Specifically, the *TransE* embeddings of the subject, predicate, and object are concatenated together, followed by concatenation with the standard embeddings. The combined embeddings are then processed through the Conv1D architecture. Two *GCNLayer*s are added to the Conv1D architecture after the second dropout layer in the architecture and before the flatten layer, with parameters *units* = 256, *activation* = 'relu' and *units* = 256, *activation* = 'relu' correspondingly. The remaining parameters and setup are identical to those of the *TransE* model in Group 7.

TransH: This model is the same as the corresponding model in Group 7, adding the *GCNLayer*s as in the *TransE* model of this Group. The remaining parameters and setup are identical to those of the *TransE* model in this Group.

RotateE: For this model, we introduce an improved version of the *GCNLayer* shown in Figure 5.18: Improved *GCNLayer*. The first *GCNLayer* is a simple implementation of a graph convolutional layer. It initializes with a specified number of units and an optional activation function. The layer initializes its weights (kernel) and biases using *Glorot* uniform and zero initializers, respectively. During the forward pass, the kernel multiplies the input and adds a bias. If provided, the output will receive the activation function.

The second *GCNLayer* introduces several enhancements over the first version. It allows for an optional bias term and kernel regularization, providing greater flexibility and control over the layer's behavior. The *use_bias* parameter allows for bias inclusion or exclusion. Any specified regularizer can regularize the kernel, adding a layer of robustness and preventing overfitting. This version also incorporates a residual connection, which adds the original input to the output following the kernel transformation and activation. This link helps train deeper networks by reducing the vanishing gradient issue. This makes the gradient flow better during backpropagation. Furthermore, the kernel initialization is explicitly defined using the *GlorotUniform* initializer.

This model is exactly the same as in the corresponding model in Group 7, adding two improved *GCNLayer*s in the same way as in the *TransE* model in this Group.

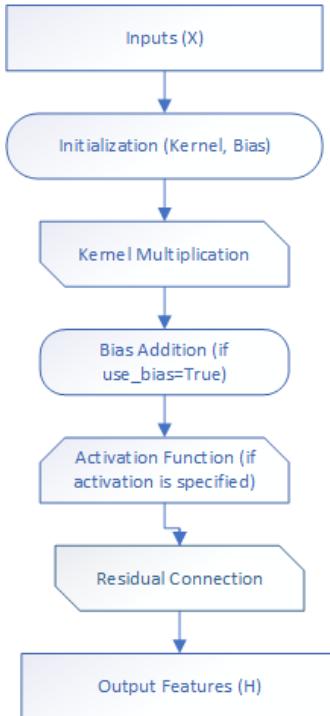


Figure 5.18: Improved *GCNLayer*

NoEmbds: This model is identical to the corresponding one in Group 7. It has two improved at the beginning of the model, before the Conv1D architecture and after the embeddings, with parameters: `units = embedding_dim, activation = 'relu', kernel_regularizer = tf.keras.regularizers.l2(l2_reg)`.

HoIE: This model is identical to the corresponding model in Group 7. It has two improved *GCNLayers* at the start of the model, like in the *NoEmbds* model in this Group, with parameters `units = embedding_dim, activation = 'relu', name = 'gcn_layer1', kernel_regularizer = 'l2', use_bias = True`.

DistMult: We have again, for this model, two improved *GCNLayers* at the start of the model with the same parameters as in the *HoIE* model in this Group. We pass the subject and object embeddings through these two *GCNLayers*, compute the *distmult* score as in the *DistMult* model in Group 1, and then we concatenate and feed the result to the Conv1D architecture. The rest of the setup and parameters are the same as in the corresponding model in Group 7.

Group 9

For this Group we use the Conv1D architecture combined with the models in Group 7 and the *GATLayer*.

TransE: This model is the same as the *TransE* model in Group 8, except for these: instead of having *GCNLayer*, we use one *GATLayer* at the start of the model with parameters `num_heads = 4, output_dim = 64, activation = 'relu', dropout_rate = 0.4`. Additionally, we set the dropout rate to 0.4 for the remaining layers and set the batch size parameter to 128.

TransH: Here, we introduce an improved version of the *GATLayer*. The initial layer initializes each attention head with the specified output dimension (`output_dim`) and concatenates all heads' results.

This means each attention head operates on the full output dimension, resulting in the final output having a dimension of $num_heads * output_dim$. This implementation keeps the output dimension of each head independent of the number of heads, thereby preserving the specified output dimension across all heads.

The improved layer, on the other hand, adjusts the output dimension for each attention head by dividing the specified $output_dim$ by the number of heads (num_heads). This approach ensures that the concatenated output across all heads equals the specified $output_dim$. This layer scales down each head's output dimension to ensure the combined result has the desired dimension. This adjustment ensures more fine-grained control over the attention mechanism by distributing the specified output dimension across multiple heads.

Both layers follow similar steps in their call method, where they compute the outputs for each head, concatenate these outputs, apply an activation function, and then a dropout. However, the critical difference lies in how they handle the output dimension for each attention head, influencing the overall architecture and the distribution of computation across the heads.

At the start of this model, we use one improved *GATLayer* with parameters $num_heads = 8$, $output_dim = 1024$, and $dropout_rate = 0.5$. The rest of the parameters and setup are the same as in the *TransH* model in Group 8.

RotatE: This model is the same as the corresponding model in Group 7, adding to it one improved *GATLayer* after the second dropout layer and before the flatten layer in Conv1D architecture, with parameters $num_heads = 8$, $output_dim = 1024$, $activation = 'relu'$, $dropout_rate = 0.3$.

NoEmbs: This model is the same as the corresponding model in Group 7, adding to it one improved *GATLayer* as in the *RotatE* model in this Group, with parameters $num_heads = 4$, $gat_output_dim = 256$, $activation = 'relu'$, $dropout_rate = 0.3$.

HoIE: This model is the same as the corresponding model in Group 7, adding to it one *GATLayer* before the Conv1D architecture, with parameters $num_heads = 8$, $output_dim = 128$.

DistMult: This model uses subject, predicate, and object embeddings. We concatenate the subject and object embeddings and pass them through an improved *GATLayer*, with parameters $num_heads = 8$, $output_dim = embedding_dim$, $activation = 'relu'$, $dropout_rate = 0.3$. The score is then computed as the reduced sum of the product of the *GATLayer*'s output and the predicate embeddings. Then follows the Conv1D architecture.

Group 10

For this Group we use the Conv1D architecture combined with the models in Group 7, the *GATLayer* and the *GCNLayer*.

TransE: We utilize the parameters and setup from the *TransE* model in Group 9, with the exception of setting the dropout to 0.5. At the start of the model, we use one *GATLayer* with parameters $num_heads = gat_num_heads$, $output_dim = gat_output_dim$, $activation = 'relu'$, $dropout_rate = dropout_rate$, and two improved *GCNLayers* with parameters $units = 256$, $activation = 'relu'$.

RotatE: This model is the same as the corresponding model in Group 7, plus one improved *GCNLayer* and one improved *GATLayer*. The parameters for these layers are *units* = 512, *activation* = 'relu', *use_bias* = True, and *num_heads* = 8, *output_dim* = *embedding_dim* * 2, *activation* = 'relu', *dropout_rate* = 0.5, respectively.

NoEmbs: This model is the same as the corresponding model in Group 7, plus one improved *GCNLayer* and one improved *GATLayer*. The parameters for these layers are *units* = 256, *activation* = 'relu', *kernel_regularizer* = 'l2' and *num_heads* = 4, *output_dim* = 256, *activation* = 'relu', *dropout_rate* = 0.3, respectively.

HoIE: This model is the same as the corresponding model in Group 7, adding to it one improved *GCNLayer* and one *GATLayer* after the second dropout layer and before the flatten of the Conv1D architecture. The parameters for these layers are *units* = 400, *activation* = 'relu' and *num_heads* = 4, *output_dim* = 64, *activation* = 'relu', *dropout_rate* = 0.5.

DistMult: This model is identical to the corresponding model in Group 7, adding one improved *GCNLayer* and one *GATLayer* just before the Conv1D architecture. The parameters for these layers are *units* = *embedding_dim* and *num_heads* = 4, *output_dim* = *embedding_dim*.

Next, we have the individual experiments. We conducted these experiments by selecting models that yielded positive results and experimenting with various methods or components to observe their impact on the models.

5.4.2 Individual Experiments

Experiment 1

For this experiment, we use [Method 2](#) for negative sample generation, with *corruption_strength* set to high, combined with the *DistMult* model from Group 4, without altering the rest of the model at all.

Experiment 2

For this experiment, we use [Method 3](#) as a method for negative sample generation, combined with the *DistMult* model from Group 4, without altering the rest of the model at all.

Experiment 3

For this experiment, we use [Method 2](#) for negative sample generation, with *corruption_strength* set to high, combined with the *TransH* model from Group 4, without altering the rest of the model at all.

Experiment 4

For this experiment, we use [Method 2](#) for negative sample generation, with *corruption_strength* set to high, combined with the *TransH* model from Group 6, without altering the rest of the model at all.

Experiment 5

For this experiment, we use [Method 3](#) for negative sample generation, combined with the *TransH* model from Group 6, without altering the rest of the model at all.

Experiment 6

For this experiment, we use [Method 2](#) for negative sample generation, with *corruption_strength* set to high, combined with the *NoEmbs* model from Group 7, without altering the rest of the model at all.

Experiment 7

For this experiment, we use [Method 3](#) for negative sample generation, combined with the *NoEmbs* model from Group 7, without altering the rest of the model at all.

Experiment 8

For this experiment, we use [Method 3](#) for negative sample generation, combined with the *HoIE* model from Group 10, without altering the rest of the model at all.

Experiment 9

For this experiment, we use the *NoEmbs* model from Group 4 combined with a data augmentation method called flip entities. We designed the *flip_entities* function to transform a set of triples by swapping the subject and object entities in each triple. The function begins by initializing an empty list called *flipped_triples*, which will store the transformed triples. It then iterates over the input list of triples. For each triple, the function constructs a new triple by flipping the subject and object entities, resulting in a triple of the form (o, p, s) . The function then appends this new triple to the *flipped_triples* list. After processing all triples, the function transforms the list of flipped triples into a NumPy array and then returns it.

In this model, we derive the negative samples using Method 1, and then we pass all the triples to the *flip_entities* function. Then, we pass all of the triples to the *flipped_triples* function. To assign labels to the augmented triples, we create a set of the initial positive triples for fast membership testing. We check each triple in the augmented dataset against this set, assigning a label of 1 for positive examples and 0 for negative ones.

Experiment 10

For this experiment, we use the model *HoIE* from Group 8, adding and using the *flip_entities* function, just like in Experiment 9.

Experiment 11

For this experiment, we utilize the model from experiment 10, but we use [Method 2](#) for generating negative samples, and we set the *corruption_strength* to high.

Experiment 12

For this experiment, we are using the model from experiment 10, but we are using [Method 3](#) for the generation of negative samples.

Experiment 13

In this experiment, we utilize the model from experiment 10 and add a custom attention layer, as shown in Figure 5.19, before the Conv1D architecture begins, using the parameters `units = embedding_dim`. This layer allows the model to focus on specific parts of the input sequence when making predictions, mimicking the human ability to pay attention to relevant information while disregarding irrelevant details.

Upon initialization, the Attention layer defines three dense (fully connected) sub-layers: W_{query} , W_{key} , and V . Each of these sub-layers has a specified number of units, which determines the dimensionality of the transformation applied to the input data. Upon invoking the call method, the layer initially processes the input query through the W_{query} sub-layer. We use this transformation to extract relevant features from the query to compute attention scores.

We then expand the query vector's dimensions to ensure compatibility with the key vectors, enabling element-wise operations. This step involves reshaping the query vector to facilitate broadcasting across the sequence of values. The attention mechanism computes a score that represents the relevance of each value in the sequence concerning the query. We achieve this by combining the transformed query with the transformed keys, which we obtain by passing the values through W_{key} and then applying a non-linear activation function (`tanh`). The V sub-layer generates the final scores.

The `softmax` function normalizes the computed scores, yielding attention weights. These weights indicate the importance of each value in the sequence relative to the query. We use the attention weights to scale the input values. We multiply each value in the sequence by its corresponding attention weight, then sum the resulting vectors to produce a context vector. This context vector is a weighted sum of the input values, where more relevant values (as determined by the attention weights) contribute more significantly to the context.

The layer then returns the context vector along with the attention weights. The context vector provides a condensed representation of the input sequence, highlighting the most relevant information as dictated by the query. On the other hand, the attention weights provide insight into the parts of the input that the computation deems most important.

Experiment 14

For this experiment, we use the *TransE* model from Group 10, adding dynamic embeddings to it. Dynamic embeddings are a core innovation in this model. We compute them by taking into account the interaction between the subject (`s_embed`) and object (`o_embed`) embeddings. The calculation of dynamic embeddings entails several steps. Firstly, we compute the outer product of the subject and object embeddings, which captures pairwise interactions between their dimensions. An attention layer processes the flattened outer product, allowing the model to focus on the most relevant interactions. We then reshape the attention output back to matrix form and apply a sum operation along the second dimension to aggregate the interactions. Finally, we use a dense layer (`dynamic_dense`) to non-linearly transform the aggregated interactions, resulting in the final dynamic embeddings.

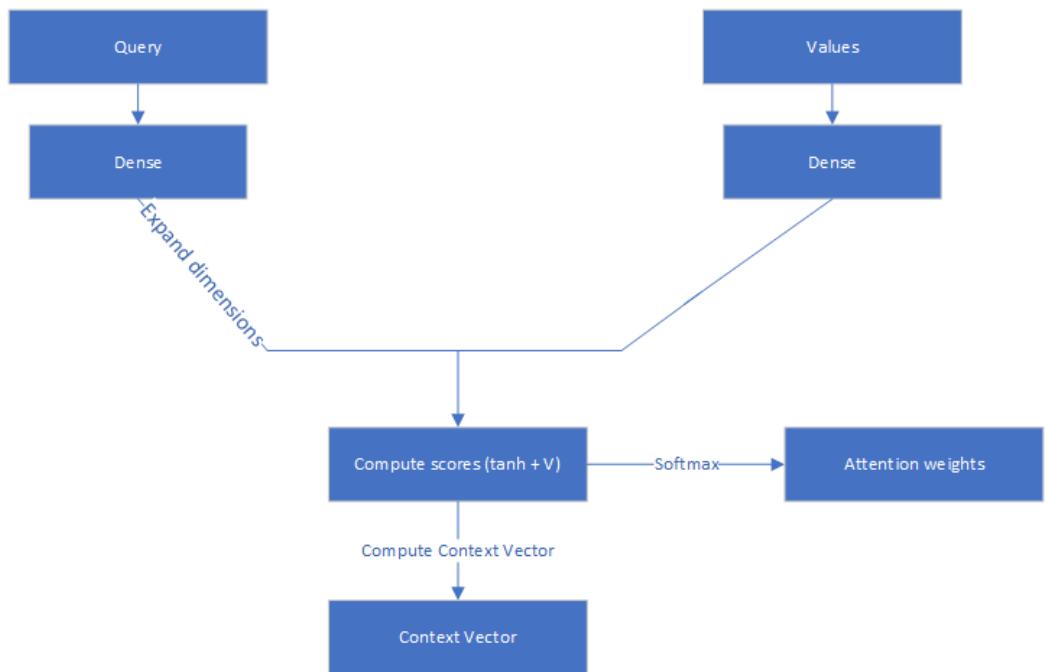


Figure 5.19: Attention Layer for experiment 13

Chapter 6

Results of the experiments

This chapter presents most of the results of all the models from groups and individual experiments. In the current chapter, we present the metrics for each model at the outset, considering three different thresholds (0.5, 0.6, and 0.7), followed by the number of epochs and time required for the model to train. In terms of training time, all experiments were performed on CPUs carried out on a desktop PC with a CPU Core i7-6800k and 32 GB of RAM. The rest of the results are located in [Appendix A](#). The appendix presents the results in the following format: For each experiment group, we start with the metrics for the clustering evaluation. Next, we present the evaluation plots for each model, which include training and validation, precision-recall, probability calibration, confusion matrix, and interpretation plots and tables. Next, we present the statistical data pertaining to the relationships each model predicts. These statistics include both quantitative and qualitative data. Table 6.1 illustrates the quantitative data. The qualitative information includes the top five relations and their frequencies; the relations with a single appearance; and the relations with the highest and lowest average probability.

In [Appendix A](#), the individual experiments start with a clustering evaluation. This is followed by evaluation plots that look like the ones in the groups, then quantitative statistical data about the relationships each model predicts, and finally qualitative statistical data about the relationships each model predicts. For some models, the primary reason for the absence of certain results is their poor performance in saving space within the thesis. For models that performed poorly, we typically include only the training and validation losses, along with a probability calibration plot to illustrate the range of predicted probabilities. However, [Chapter 5's GitHub repository](#) contains all the results and code.

Table 6.1: Abbreviations for columns

<i>Column abbreviation</i>	<i>Column meaning</i>
<i>C1</i>	Total number of distinct relations
<i>C2</i>	Number of triples with probability > 60%
<i>C3</i>	Number of triples with probability > 90%
<i>C4</i>	Number of distinct relations with probability > 90%
<i>C5</i>	Average probability of all triples
<i>C6</i>	Maximum probability
<i>C7</i>	Minimum probability
<i>C8</i>	Standard deviation of probabilities

Group 1

For this set of experiments, the dense architecture shown in Figure 5.6 is used, along with a simple splitting method and the *TransE*, *TransH*, *RotatE*, *HoIE*, *DistMult*, and *ComplEx* embeddings.

Table 6.2: Metrics for threshold 0.5

	Precision	Recall	F1	ROC AUC	PR -AUC	MCC
<i>TransE</i>	0.6307	0.7191	0.6720	0.7639	0.7977	0.3010
<i>TransH</i>	0.6342	0.8619	0.7307	0.8100	0.8365	0.3910
<i>RotatE</i>	0.6459	0.7071	0.6751	0.7837	0.8134	0.3210
<i>HoIE</i>	0.6562	0.6013	0.6276	0.7549	0.7933	0.2874
<i>DistMult</i>	0.5000	1.0000	0.6666	0.5000	0.5000	0.0000
<i>ComplEx</i>	0.7009	0.8418	0.7649	0.8224	0.7732	0.4926

Table 6.3: Metrics for threshold 0.6

	Precision	Recall	F1	ROC -AUC	PR -AUC	MCC
<i>TransE</i>	0.7263	0.5358	0.6166	0.7639	0.7977	0.3460
<i>TransH</i>	0.6579	0.7889	0.7175	0.8100	0.8365	0.3865
<i>RotatE</i>	0.7643	0.5190	0.6182	0.7837	0.8134	0.3790
<i>HoIE</i>	0.7034	0.5384	0.6099	0.7549	0.7933	0.3204
<i>DistMult</i>	0.0000	0.0000	0.0000	0.5000	0.5000	0.0000
<i>ComplEx</i>	0.7344	0.7337	0.7340	0.8224	0.7732	0.4683

Table 6.4: Metrics for threshold 0.7

	Precision	Recall	F1	ROC -AUC	PR - AUC	MCC
<i>TransE</i>	0.8279	0.4493	0.5825	0.7639	0.7977	0.4003
<i>TransH</i>	0.7020	0.6908	0.6964	0.8100	0.8365	0.3977
<i>RotatE</i>	0.8528	0.4707	0.6066	0.7837	0.8134	0.4357
<i>HoIE</i>	0.7659	0.4836	0.5929	0.7549	0.7933	0.3613
<i>DistMult</i>	0.0000	0.0000	0.0000	0.5000	0.5000	0.0000
<i>ComplEx</i>	0.7924	0.5578	0.6547	0.8224	0.7732	0.4311

Table 6.5: Training times and epochs required for each model

	Seconds per epoch	Total epochs
<i>TransE</i>	1300	12
<i>TransH</i>	2300	11
<i>RotateE</i>	2300	12
<i>HoIE</i>	700	11
<i>DistMult</i>	1200	16
<i>ComplEx</i>	1150	20

Group 2

For this set of experiments, the dense architecture shown in Figure 5.6 is used, along with the stratified K-fold splitting method, *TransE*, *TransH*, *RotateE*, *HoIE*, *DistMult*, and *ComplEx* embeddings.

Table 6.6: Metrics for threshold 0.5

	Precision	Recall	F1	ROC AUC	PR -AUC	MCC
<i>TransE</i>	0.6184	0.7405	0.6740	0.7496	0.7881	0.2892
<i>TransH</i>	0.6541	0.8122	0.7247	0.8121	0.8382	0.3945
<i>RotateE</i>	0.6421	0.6400	0.6411	0.7597	0.7974	0.2833
<i>HoIE</i>	0.6467	0.6300	0.6383	0.7500	0.7898	0.2860
<i>DistMult</i>	0.5000	1.0000	0.6667	0.5000	0.5000	0
<i>ComplEx</i>	0.6443	0.6352	0.6397	0.7596	0.7969	0.2845

Table 6.7: Metrics for threshold 0.6

	Precision	Recall	F1	ROC -AUC	PR -AUC	MCC
<i>TransE</i>	0.6365	0.6819	0.6584	0.7496	0.7881	0.2933
<i>TransH</i>	0.6724	0.7707	0.7182	0.8121	0.8382	0.3995
<i>RotateE</i>	0.6678	0.5909	0.6270	0.7597	0.7974	0.2990
<i>HoIE</i>	0.6737	0.5764	0.6213	0.7500	0.7898	0.3004
<i>DistMult</i>	0	0	0	0.5000	0.5000	0
<i>ComplEx</i>	0.6681	0.5851	0.6238	0.7596	0.7969	0.2967

Table 6.8: Metrics for threshold 0.7

	Precision	Recall	F1	ROC -AUC	PR - AUC	MCC
<i>TransE</i>	0.6614	0.6207	0.6404	0.7496	0.7881	0.3036
<i>TransH</i>	0.6944	0.7209	0.7074	0.8121	0.8382	0.4039
<i>RotateE</i>	0.7030	0.5430	0.6127	0.7597	0.7974	0.3221
<i>HoIE</i>	0.7125	0.5241	0.6039	0.7500	0.7898	0.3241
<i>DistMult</i>	0	0	0	0.5000	0.5000	0
<i>ComplEx</i>	0.7015	0.5367	0.6081	0.7596	0.7969	0.3172

Table 6.9: Training times and epochs required for each model

	Seconds per epoch	Total epochs	Total folds
<i>TransE</i>	450	16	2
<i>TransH</i>	1300	12	3
<i>RotateE</i>	1550	12	3
<i>HoIE</i>	1100	12	3
<i>DistMult</i>	750	20	2
<i>ComplEx</i>	1400	14	3

Group 3

For this set of experiments, the dense architecture shown in Figure 5.6 is used, along with the Bernouli method for negative sampling generation, a simple splitting method, and the *TransE*, *TransH*, *RotateE*, *HoIE*, *DistMult*, and *ComplEx* embeddings.

Table 6.10: Metrics for threshold 0.5

	Precision	Recall	F1	ROC- AUC	PR - AUC	MCC
<i>TransE</i>	0.5000	1.0000	0.6666	0.5000	0.5000	0
<i>TransH</i>	0.5317	0.8767	0.6619	0.6017	0.5881	0.1376
<i>RotateE</i>	0.5303	0.8460	0.6520	0.5962	0.5913	0.1206
<i>HoIE</i>	0.5394	0.7512	0.6279	0.6056	0.5932	0.1196
<i>DistMult</i>	0.5363	0.8043	0.6435	0.6029	0.5922	0.1259
<i>ComplEx</i>	0.5000	1.0000	0.6666	0.5000	0.5000	0

Table 6.11: Metrics for threshold 0.6

	Precision	Recall	F1	ROC-AUC	PR -AUC	MCC
<i>TransE</i>	0	0	0	0.5000	0.5000	0
<i>TransH</i>	0.5501	0.5917	0.5701	0.6017	0.5881	0.1081
<i>RotateE</i>	0.6218	0.3493	0.4473	0.5962	0.5913	0.1523
<i>HoIE</i>	0.6542	0.2832	0.3952	0.6056	0.5932	0.1621
<i>DistMult</i>	0.6009	0.3900	0.4730	0.6029	0.5922	0.1399
<i>ComplEx</i>	0	0	0	0.5000	0.5000	0

Table 6.12: Metrics for threshold 0.7

	Precision	Recall	F1	ROC-AUC	PR - AUC	MCC
<i>TransE</i>	0	0	0	0.5000	0.5000	0
<i>TransH</i>	0.6459	0.0209	0.0405	0.6017	0.5881	0.0375
<i>RotateE</i>	0.6713	0.1572	0.2547	0.5962	0.5913	0.1248
<i>HoIE</i>	0.6745	0.0785	0.1406	0.6056	0.5932	0.0867
<i>DistMult</i>	0.6797	0.0677	0.1232	0.6029	0.5922	0.0823
<i>ComplEx</i>	0	0	0	0.5000	0.5000	0

Table 6.13: Training times and epochs required for each model

	Seconds per epoch	Total epochs
<i>TransE</i>	800	15
<i>TransH</i>	800	12
<i>RotateE</i>	1400	12
<i>HoIE</i>	750	12
<i>DistMult</i>	900	12
<i>ComplEx</i>	1100	14

Group 4

For this set of experiments, the dense architecture shown in Figure 5.6 is used, along with [Method 1](#) for negative sampling generation, the StratifiedShuffleSplit splitting method, the [GCNLayer](#), and the *TransE*, *TransH*, *NoEmbds*, *HoIE*, *DistMult*, and *ComplEx* embeddings.

Table 6.14: Metrics for threshold 0.5

	Precision	Recall	F1	ROC-AUC	PR -AUC	MCC
<i>TransE</i>	0.6505	0.7725	0.7063	0.7993	0.8255	0.3639
<i>TransH</i>	0.6421	0.8259	0.7225	0.7974	0.8241	0.3815
<i>NoEmbds</i>	0.7668	0.5506	0.6409	0.8062	0.8305	0.3994
<i>HoIE</i>	0.6454	0.8760	0.7432	0.8103	0.8335	0.4225
<i>DistMult</i>	0.7459	0.6286	0.6822	0.8247	0.8450	0.4196
<i>ComplEx</i>	0.6766	0.7791	0.7243	0.8225	0.8442	0.4115

Table 6.15: Metrics for threshold 0.6

	Precision	Recall	F1	ROC-AUC	PR -AUC	MCC
<i>TransE</i>	0.8428	0.4894	0.6192	0.7993	0.8255	0.4385
<i>TransH</i>	0.8378	0.4933	0.6210	0.7974	0.8241	0.4364
<i>NoEmbds</i>	0.8570	0.4858	0.6201	0.8062	0.8305	0.4491
<i>HoIE</i>	0.8685	0.4869	0.6240	0.8103	0.8335	0.4600
<i>DistMult</i>	0.9015	0.4693	0.6172	0.8247	0.8450	0.4763
<i>ComplEx</i>	0.8948	0.4866	0.6304	0.8225	0.8442	0.4825

Table 6.16: Metrics for threshold 0.7

	Precision	Recall	F1	ROC-AUC	PR - AUC	MCC
<i>TransE</i>	0.8873	0.4720	0.6162	0.7993	0.8255	0.4663
<i>TransH</i>	0.8732	0.4760	0.6161	0.7974	0.8241	0.4569
<i>NoEmbds</i>	0.8918	0.4714	0.6168	0.8062	0.8305	0.4697
<i>HoIE</i>	0.8989	0.4716	0.6186	0.8103	0.8335	0.4757
<i>DistMult</i>	0.9663	0.4271	0.5924	0.8247	0.8450	0.4968
<i>ComplEx</i>	0.9244	0.4737	0.6264	0.8225	0.8442	0.4982

Table 6.17: Training times and epochs required for each model

	Seconds per epoch		Total epochs
	TransE	TransH	
<i>TransE</i>	1150		5
<i>TransH</i>	760		5
<i>NoEmbds</i>	800		5
<i>HoIE</i>	1300		5
<i>DistMult</i>	1300		5
<i>ComplEx</i>	1100		5

Group 5

For this set of experiments, the dense architecture shown in Figure 5.6 is used, along with [Method 1](#) for negative sampling generation, the StratifiedShuffleSplit splitting method, the [GATLayer](#), and the *TransE*, *TransH*, *RotatE*, *HoIE*, *DistMult*, and *ComplEx* embeddings.

Table 6.18: Metrics for threshold 0.5

	Precision	Recall	F1	ROC-AUC	PR -AUC	MCC
<i>TransE</i>	0.6643	0.8601	0.7496	0.8265	0.8456	0.4452
<i>TransH</i>	0.6672	0.8937	0.7640	0.8373	0.8550	0.4763
<i>RotatE</i>	0.5000	1.0000	0.6666	0.5000	0.5000	0
<i>HoIE</i>	0.6397	0.9433	0.7624	0.8290	0.8471	0.4682
<i>DistMult</i>	1.0000	0.0221	0.0433	0.5139	0.5329	0.1058
<i>ComplEx</i>	0.6552	0.9017	0.7590	0.8314	0.8500	0.4612

Table 6.19: Metrics for threshold 0.6

	Precision	Recall	F1	ROC-AUC	PR -AUC	MCC
<i>TransE</i>	0.8539	0.5100	0.6386	0.8265	0.8456	0.4619
<i>TransH</i>	0.9315	0.4846	0.6375	0.8373	0.8550	0.5117
<i>RotatE</i>	0	0	0	0.5000	0.5000	0
<i>HoIE</i>	0.8915	0.5013	0.6417	0.8290	0.8471	0.4897
<i>DistMult</i>	1.0000	0.0221	0.0433	0.5139	0.5329	0.1058
<i>ComplEx</i>	0.9388	0.4692	0.6257	0.8314	0.8500	0.5066

Table 6.20: Metrics for threshold 0.7

	Precision	Recall	F1	ROC-AUC	PR - AUC	MCC
<i>TransE</i>	0.9811	0.3999	0.5682	0.8265	0.8456	0.4868
<i>TransH</i>	0.9922	0.3857	0.5554	0.8373	0.8550	0.4835
<i>RotatE</i>	0	0	0	0.5000	0.5000	0
<i>HoIE</i>	0.9614	0.4374	0.6013	0.8290	0.8471	0.5008
<i>DistMult</i>	1.0000	0.0221	0.0433	0.5139	0.5329	0.1058
<i>ComplEx</i>	0.9944	0.3217	0.4861	0.8314	0.8500	0.4343

Table 6.21: Training times and epochs required for each model

	Seconds per epoch	Total epochs
<i>TransE</i>	1450	8
<i>TransH</i>	1450	7
<i>RotateE</i>	2450	7
<i>HoIE</i>	4000	7
<i>DistMult</i>	1450	7
<i>ComplEx</i>	3300	5

Group 6

For this set of experiments, the dense architecture shown in Figure 5.6 is used, along with [Method 1](#) for negative sampling generation, the StratifiedShuffleSplit splitting method, the [GCNLayer](#), the [GATLayer](#), and the *TransE*, *TransH*, *RotatE*, *HoIE*, *DistMult*, and *ComplEx* embeddings.

Table 6.22: Metrics for threshold 0.5

	Precision	Recall	F1	ROC- AUC	PR -AUC	MCC
<i>TransE</i>	0.6500	0.9171	0.7608	0.8279	0.8469	0.4643
<i>TransH</i>	0.6580	0.8845	0.7547	0.8293	0.8481	0.4526
<i>RotateE</i>	0.5264	0.9182	0.6692	0.7060	0.7722	0.1383
<i>HoIE</i>	0.7031	0.7637	0.7321	0.8351	0.8527	0.4429
<i>DistMult</i>	0.5000	1.0000	0.6666	0.7583	0.7809	0
<i>ComplEx</i>	0.5000	1.0000	0.6666	0.6682	0.7560	0

Table 6.23: Metrics for threshold 0.6

	Precision	Recall	F1	ROC-AUC	PR -AUC	MCC
<i>TransE</i>	0.9256	0.4690	0.6226	0.8279	0.8469	0.4958
<i>TransH</i>	0.9801	0.3975	0.5656	0.8293	0.8481	0.4843
<i>RotateE</i>	0.8846	0.4129	0.5630	0.7060	0.7722	0.4245
<i>HoIE</i>	0.9507	0.4517	0.6124	0.8351	0.8527	0.5031
<i>DistMult</i>	0.5475	0.9966	0.7068	0.7583	0.7809	0.3026
<i>ComplEx</i>	0.6005	0.5832	0.5917	0.6682	0.7560	0.1954

Table 6.24: Metrics for threshold 0.7

	Precision	Recall	F1	ROC-AUC	PR - AUC	MCC
<i>TransE</i>	0.9948	0.3453	0.5127	0.8279	0.8469	0.4535
<i>TransH</i>	0.9985	0.2722	0.4278	0.8293	0.8481	0.3961
<i>RotateE</i>	0.9888	0.3397	0.5057	0.7060	0.7722	0.4453
<i>HoIE</i>	0.9976	0.3110	0.4742	0.8351	0.8527	0.4277
<i>DistMult</i>	0.6165	0.8771	0.7240	0.7583	0.7809	0.3658
<i>ComplEx</i>	0.9367	0.4184	0.5784	0.6682	0.7560	0.4684

Table 6.25: Training times and epochs required for each model

	Seconds per epoch	Total epochs
<i>TransE</i>	1370	7
<i>TransH</i>	1350	7
<i>RotateE</i>	2500	6
<i>HoIE</i>	1550	7
<i>DistMult</i>	1670	7
<i>ComplEx</i>	3300	5

Group 7

For this set of experiments, the Conv1D architecture shown in Figure 5.7 is used, along with [Method 1](#) for negative sampling generation, the StratifiedShuffleSplit splitting method, and the *TransE*, *TransH*, *RotatE*, *NoEmbds*, *HoIE*, *DistMult*, and *ComplEx* embeddings.

Table 6.26: Metrics for threshold 0.5

	Precision	Recall	F1	ROC- AUC	PR - AUC	MCC
<i>TransE</i>	0.7416	0.5879	0.6559	0.8096	0.8336	0.3915
<i>TransH</i>	0.5000	1.0000	0.6667	0.5000	0.5000	0
<i>RotateE</i>	0.5046	0.4224	0.4598	0.5089	0.5260	0.0078
<i>NoEmbds</i>	0.7022	0.7329	0.7172	0.8287	0.8490	-
<i>HoIE</i>	0.7153	0.5876	0.6452	0.8035	0.8297	0.3595
<i>DistMult</i>	0.5433	0.3597	0.4329	0.5391	0.5685	0.0610
<i>ComplEx</i>	0.5000	1.0000	0.6667	0.5000	0.5000	0

Table 6.27: Metrics for threshold 0.6

	Precision	Recall	F1	ROC-AUC	PR - AUC	MCC
<i>TransE</i>	0.8770	0.4993	0.6363	0.8096	0.8336	0.4756
<i>TransH</i>	0	0	0	0.5000	0.5000	0
<i>RotateE</i>	0.9990	0.0223	0.0436	0.5089	0.5260	0.1060
<i>NoEmbds</i>	0.7655	0.6056	0.6762	0.8287	0.8490	-
<i>HoIE</i>	0.8623	0.4969	0.6305	0.8035	0.8297	0.4610
<i>DistMult</i>	0.5480	0.3338	0.4148	0.5391	0.5685	0.0635
<i>ComplEx</i>	0	0	0	0.5000	0.5000	0

Table 6.28: Metrics for threshold 0.7

	Precision	Recall	F1	ROC-AUC	PR - AUC	MCC
<i>TransE</i>	0.9329	0.4696	0.6247	0.8096	0.8336	0.5021
<i>TransH</i>	0	0	0	0.5000	0.5000	0
<i>RotateE</i>	1.0000	0.0222	0.0435	0.5089	0.5260	0.1061
<i>NoEmbds</i>	0.8841	0.5024	0.6407	0.8287	0.8490	-
<i>HoIE</i>	0.9312	0.4740	0.6282	0.8035	0.8297	0.5038
<i>DistMult</i>	0.5546	0.3069	0.3951	0.5391	0.5685	0.0675
<i>ComplEx</i>	0	0	0	0.5000	0.5000	0

Table 6.29: Training times and epochs required for each model

	Seconds per epoch	Total epochs
<i>TransE</i>	8800	7
<i>TransH</i>	8300	13
<i>RotateE</i>	2850	5
<i>NoEmbds</i>	8500	6
<i>HoIE</i>	11500	6
<i>DistMult</i>	860	5
<i>ComplEx</i>	8900	11

Group 8

For this set of experiments, the Conv1D architecture shown in Figure 5.7 is used, along with [Method 1](#) for negative sampling generation, the StratifiedShuffleSplit splitting method, the [GCNLayer](#), and the *TransE*, *TransH*, *RotatE*, *NoEmbds*, *HoIE*, *DistMult*, and *ComplEx* embeddings.

Table 6.30: Metrics for threshold 0.5

	Precision	Recall	F1	ROC-AUC	PR -AUC	MCC
<i>TransE</i>	0.6821	0.6085	0.6432	0.7919	0.8182	-
<i>TransH</i>	0	0	0	0.5	0.5	0
<i>RotatE</i>	0.5110	0.6422	0.5691	0.5177	0.5116	0.0286
<i>NoEmbds</i>	0.8300	0.5363	0.6516	0.8282	0.8487	-
<i>HoIE</i>	0.6754	0.6527	0.6639	0.7576	0.7491	0.3392
<i>DistMult</i>	0.6675	0.5880	0.6252	0.7003	0.7523	0.2972
<i>ComplEx</i>	0.5000	1.0000	0.6667	0.5000	0.5000	0

Table 6.31: Metrics for threshold 0.6

	Precision	Recall	F1	ROC-AUC	PR -AUC	MCC
<i>TransE</i>	0.9081	0.4596	0.6103	0.7919	0.8182	-
<i>TransH</i>	0	0	0	0.5	0.5	0
<i>RotatE</i>	0	0	0	0.5177	0.5116	0
<i>NoEmbds</i>	0.8971	0.4941	0.6372	0.8282	0.8487	-
<i>HoIE</i>	0.6923	0.6045	0.6454	0.7576	0.7491	0.3386
<i>DistMult</i>	0.6794	0.5729	0.6216	0.7003	0.7523	0.3063
<i>ComplEx</i>	0	0	0	0.5	0.5	0

Table 6.32: Metrics for threshold 0.7

	Precision	Recall	F1	ROC-AUC	PR - AUC	MCC
<i>TransE</i>	0.9550	0.4349	0.5976	0.7919	0.8182	-
<i>TransH</i>	0	0	0	0.5	0.5	0
<i>RotatE</i>	0	0	0	0.5177	0.5116	0
<i>NoEmbds</i>	0.9326	0.4768	0.6310	0.8282	0.8487	-
<i>HoIE</i>	0.7113	0.5642	0.6293	0.7576	0.7491	0.3427
<i>DistMult</i>	0.6927	0.5577	0.6179	0.7003	0.7523	0.3164
<i>ComplEx</i>	0	0	0	0.5	0.5	0

Table 6.33: Training times and epochs required for each model

	Seconds per epoch	Total epochs
<i>TransE</i>	16000	5
<i>TransH</i>	9000	15
<i>RotateE</i>	1800	5
<i>NoEmbds</i>	8300	5
<i>HoIE</i>	15000	7
<i>DistMult</i>	1550	5

Group 9

For this set of experiments, the Conv1D architecture shown in Figure 5.7 is used, along with [Method 1](#) for negative sampling generation, the StratifiedShuffleSplit splitting method, the [GATLayer](#), and the *TransE*, *TransH*, *RotatE*, *NoEmbds*, *HoIE*, and *DistMult* embeddings.

Table 6.34: Metrics for threshold 0.5

	Precision	Recall	F1	ROC-AUC	PR -AUC	MCC
<i>TransE</i>	0.5017	1.0000	0.6682	0.7031	0.6850	-
<i>TransH</i>	0.5000	1.0000	0.6667	0.5000	0.5000	0
<i>RotateE</i>	0.5218	0.2706	0.3564	0.5156	0.5342	0.0258
<i>NoEmbds</i>	0.7577	0.5316	0.6249	0.7943	0.8226	0.3789
<i>HoIE</i>	0.8870	0.4647	0.6099	0.8159	0.8358	0.4611
<i>DistMult</i>	0.6902	0.6535	0.6713	0.7873	0.8216	0.3607

Table 6.35: Metrics for threshold 0.6

	Precision	Recall	F1	ROC-AUC	PR -AUC	MCC
<i>TransE</i>	0.5022	1.0000	0.6686	0.7031	0.6850	-
<i>TransH</i>	0	0	0	0.5000	0.5000	0
<i>RotateE</i>	0.8947	0.0249	0.0484	0.5156	0.5342	0.0937
<i>NoEmbds</i>	0.9108	0.4674	0.6178	0.7943	0.8226	0.4827
<i>HoIE</i>	0.9582	0.4160	0.5801	0.8159	0.8358	0.4825
<i>DistMult</i>	0.7017	0.6359	0.6672	0.7873	0.8216	0.3672

Table 6.36: Metrics for threshold 0.7

	Precision	Recall	F1	ROC-AUC	PR - AUC	MCC
<i>TransE</i>	0.5028	1.0000	0.6691	0.7031	0.6850	-
	0	0	0	0.5000	0.5000	0
	1.0000	0.0214	0.0420	0.5156	0.5342	0.1041
	0.9542	0.4474	0.6092	0.7943	0.8226	0.5027
	0.9753	0.3844	0.5514	0.8159	0.8358	0.4709
	0.7158	0.6175	0.6630	0.7873	0.8216	0.3758

Table 6.37: Training times and epochs required for each model

	Seconds per epoch	Total epochs
<i>TransE</i>	20500	5
	5500	15
	1900	5
	11000	5
	6600	6
	950	7

Group 10

For this set of experiments, the Conv1D architecture shown in Figure 5.7 is used, along with [Method 1](#) for negative sampling generation, the StratifiedShuffleSplit splitting method, the [GCNLayer](#), the [GATLayer](#), and the *TransE*, *RotatE*, *NoEmbds*, *HoIE*, and *DistMult* embeddings.

Table 6.38: Metrics for threshold 0.5

	Precision	Recall	F1	ROC- AUC	PR -AUC	MCC
<i>TransE</i>	0.6763	0.5814	0.6252	0.7613	0.7923	0.3061
	0.5095	0.8218	0.6291	0.5236	0.5148	0.0390
	0.7898	0.5499	0.6484	0.8168	0.8387	-
	0.6478	0.7789	0.7073	0.7994	0.8276	0.3630
	0.7169	0.5207	0.6033	0.7717	0.7788	0.3276

Table 6.39: Metrics for threshold 0.6

	Precision	Recall	F1	ROC-AUC	PR -AUC	MCC
<i>TransE</i>	0.7266	0.4905	0.5857	0.7613	0.7923	0.3235
<i>RotateE</i>	0.5126	0.7281	0.6016	0.5236	0.5148	0.0395
<i>NoEmbds</i>	0.9301	0.4660	0.6209	0.8168	0.8387	-
<i>Hole</i>	0.7623	0.5407	0.6327	0.7998	0.8267	0.3889
<i>DistMult</i>	0.7431	0.4679	0.5742	0.7717	0.7788	0.3296

Table 6.40: Metrics for threshold 0.7

	Precision	Recall	F1	ROC-AUC	PR - AUC	MCC
<i>TransE</i>	0.7828	0.4559	0.5762	0.7613	0.7923	0.3626
<i>RotateE</i>	0.5172	0.4266	0.4675	0.5236	0.5148	0.0289
<i>NoEmbds</i>	0.9566	0.4504	0.6125	0.8168	0.8387	-
<i>Hole</i>	0.9032	0.4820	0.6285	0.7998	0.8267	0.4865
<i>DistMult</i>	0.7822	0.4075	0.5359	0.7717	0.7788	0.3350

Table 6.41: Training times and epochs required for each model

	Seconds per epoch	Total epochs
<i>TransE</i>	25500	7
<i>RotateE</i>	1800	7
<i>NoEmbds</i>	16500	5
<i>Hole</i>	20500	6
<i>DistMult</i>	5900	7

Experiment 1

For this experiment, we use [Method 2](#) for negative sample generation, with *corruption_strength* set to high, combined with the *DistMult* model from Group 4.

Table 6.42: Metrics for different thresholds

	Precision	Recall	F1	ROC - AUC	PR - AUC	MCC
<i>Threshold > 0.5</i>	0.5077	0.1992	0.2862	0.6774	0.3888	0.1963
<i>Threshold > 0.6</i>	0.5470	0.0173	0.0335	0.6774	0.3888	0.0612
<i>Threshold > 0.7</i>	0	0	0	0.6774	0.3888	-0.0013

Table 6.43: Training time and epoch required for the model

Seconds per epoch	Total epochs
1550	5

Experiment 2

For this experiment, we use [Method 3](#) as a method for negative sample generation, combined with the *DistMult* model from Group 4.

Table 6.44: Metrics for different thresholds

	Precision	Recall	F1	ROC-AUC	PR - AUC	MCC
<i>Threshold > 0.5</i>	0.7865	0.9500	0.8605	0.9667	0.9702	0.7075
<i>Threshold > 0.6</i>	0.8092	0.9297	0.8653	0.9667	0.9702	-
<i>Threshold > 0.7</i>	0.8427	0.9075	0.8739	0.9667	0.9702	-

Table 6.45: Training time and epoch required for the model

Seconds per epoch	Total epochs
850	5

Experiment 3

For this experiment, we use [Method 2](#) for negative sample generation, with *corruption_strength* set to high, combined with the *TransH* model from Group 4.

Table 6.46: Metrics for different thresholds

	Precision	Recall	F1	ROC-AUC	PR - AUC	MCC
<i>Threshold > 0.5</i>	0.5638	0.2256	0.3222	0.7290	0.4412	0.2416
<i>Threshold > 0.6</i>	0	0	0	0.7290	0.4412	0
<i>Threshold > 0.7</i>	0	0	0	0.7290	0.4412	0

Table 6.47: Training time and epoch required for the model

Seconds per epoch	Total epochs
1590	5

Experiment 4

For this experiment, we use [Method 2](#) for negative sample generation, with *corruption_strength* set to high, combined with the *TransH* model from Group 6.

Table 6.48: Metrics for different thresholds

	Precision	Recall	F1	ROC-AUC	PR -AUC	MCC
<i>Threshold > 0.5</i>	0	0	0	0.7220	0.4201	0

Table 6.49: Training time and epoch required for the model

Seconds per epoch	Total epochs
3100	7

Experiment 5

For this experiment, we use [Method 3](#) for negative sample generation, combined with the *TransH* model from Group 6.

Table 6.50: Metrics for different thresholds

	Precision	Recall	F1	ROC-AUC	PR - AUC	MCC
<i>Threshold > 0.5</i>	0.9930	0.9883	0.9906	0.9998	0.9998	0.9813
<i>Threshold > 0.6</i>	0.9987	0.9715	0.9849	0.9998	0.9998	0.9705
<i>Threshold > 0.7</i>	1.0000	0.9499	0.9743	0.9998	0.9998	0.9510

Table 6.51: Training time and epoch required for the model

Seconds per epoch	Total epochs
1500	13

Experiment 6

For this experiment, we use [Method 2](#) for negative sample generation, with *corruption_strength* set to high, combined with the *NoEmbds* model from Group 7.

Table 6.52: Metrics for different thresholds

	Precision	Recall	F1	ROC-AUC	PR - AUC	MCC
<i>Threshold > 0.5</i>	0.6412	0.3733	0.4719	0.8081	0.5925	-
<i>Threshold > 0.6</i>	0.7219	0.2817	0.4053	0.8081	0.5925	-
<i>Threshold > 0.7</i>	0.7882	0.1962	0.3142	0.8081	0.5925	-

Table 6.53: Training time and epoch required for the model

Seconds per epoch	Total epochs
17400	7

Experiment 7

For this experiment, we use [Method 3](#) for negative sample generation, combined with the *NoEmbs* model from Group 7.

Table 6.54: Metrics for different thresholds

	Precision	Recall	F1	ROC - AUC	PR - AUC	MCC
<i>Threshold > 0.5</i>	0.9942	0.9761	0.9851	0.9990	0.9990	0.9706
<i>Threshold > 0.6</i>	0.9949	0.9682	0.9814	0.9990	0.9990	0.9637
<i>Threshold > 0.7</i>	0.9956	0.9576	0.9762	0.9990	0.9990	0.9540

Table 6.55: Training time and epoch required for the model

Seconds per epoch	Total epochs
8400	5

Experiment 8

For this experiment, we use [Method 3](#) for negative sample generation, combined with the *HoIE* model from Group 10.

Table 6.56: Metrics for different thresholds

	Precision	Recall	F1	ROC - AUC	PR - AUC	MCC
<i>Threshold > 0.5</i>	0.9929	0.9794	0.9861	0.9991	0.9991	0.9725
<i>Threshold > 0.6</i>	0.9936	0.9730	0.9832	0.9991	0.9991	0.9669
<i>Threshold > 0.7</i>	0.9943	0.9643	0.9790	0.9991	0.9991	0.9591

Table 6.57: Training time and epoch required for the model

Seconds per epoch	Total epochs
20500	5

Experiment 9

For this experiment, we use the *NoEmbds* model from Group 4 combined with the data augmentation method called [flip entities](#).

Table 6.58: Metrics for different thresholds

	Precision	Recall	F1	ROC - AUC	PR - AUC	MCC
Threshold > 0.5	0.9970	0.2929	0.4528	0.9264	0.8334	0.4839
Threshold > 0.6	0.9994	0.0341	0.0659	0.9264	0.8334	0.1597
Threshold > 0.7	1.0000	0.0031	0.0062	0.9264	0.8334	0.0482

Table 6.59: Training time and epoch required for the model

Seconds per epoch	Total epochs
3150	5

Experiment 10

For this experiment, we use the model *HoIE* from Group 8, adding and using the *flip_entities* function, just like in Experiment 9.

Table 6.60: Metrics for different thresholds

	Precision	Recall	F1	ROC - AUC	PR - AUC	MCC
Threshold > 0.5	0.7297	0.7866	0.7571	0.9441	0.8682	0.6695
Threshold > 0.6	0.8105	0.6472	0.7197	0.9441	0.8682	0.6435
Threshold > 0.7	0.9292	0.5129	0.6609	0.9441	0.8682	0.6253

Table 6.61: Training time and epoch required for the model

Seconds per epoch	Total epochs
7900	5

Experiment 11

For this experiment, we utilize the model from experiment 10, but we use [Method 2](#) for generating negative samples, and we set the *corruption_strength* to high.

Table 6.62: Metrics for different thresholds

	Precision	Recall	F1	ROC - AUC	PR - AUC	MCC
Threshold > 0.5	0.6783	0.4923	0.5705	0.9310	0.6697	0.5268
Threshold > 0.6	0.7540	0.3949	0.5184	0.9310	0.6697	0.5021
Threshold > 0.7	0.8134	0.2891	0.4265	0.9310	0.6697	0.4477

Table 6.63: Training time and epoch required for the model

Seconds per epoch	Total epochs
15000	5

Experiment 12

For this experiment, we are using the model from experiment 10, but we are using [Method 3](#) for the generation of negative samples.

Table 6.64: Metrics for different thresholds

	Precision	Recall	F1	ROC - AUC	PR - AUC	MCC
Threshold > 0.5	0.9863	0.9820	0.9841	0.9979	0.9968	0.9786
Threshold > 0.6	0.9873	0.9806	0.9839	0.9979	0.9968	0.9784
Threshold > 0.7	0.9880	0.9788	0.9834	0.9979	0.9968	0.9776

Table 6.65: Training time and epoch required for the model

Seconds per epoch	Total epochs
7400	13

Experiment 13

In this experiment, we utilize the model from experiment 10 and add a [custom attention layer](#), as shown in Figure 5.19, before the Conv1D architecture begins.

Table 6.66: Metrics for different thresholds

	Precision	Recall	F1	ROC - AUC	PR - AUC	MCC
Threshold > 0.5	0	0	0	0.5000	0.2572	0

Table 6.67: Training time and epoch required for the model

Seconds per epoch	Total epochs
7800	15

Experiment 14

For this experiment, we use the *TransE* model from Group 10, adding dynamic embeddings to it.

Table 6.68: Metrics for different thresholds

	Precision	Recall	F1	ROC - AUC	PR - AUC	MCC
Threshold > 0.5	0.6466	0.6434	0.6450	0.7597	0.7925	0.2917
Threshold > 0.6	0.7018	0.5327	0.6057	0.7597	0.7925	0.3157
Threshold > 0.7	0.7466	0.4755	0.5810	0.7597	0.7925	0.3372

Table 6.69: Training time and epoch required for the model

Seconds per epoch	Total epochs
30500	7

6.1 Discussion

After analyzing the results, we can draw several intriguing conclusions about the models. We can examine how the various components and methods impact the efficiency and training time of the models. Furthermore, we can evaluate how well the models performed in the relation prediction task using our approach, as outlined in [Chapter 5](#). In every set of experiments, we highlight the top-performing model in bold. In certain experiments, there may be multiple models highlighted in bold. This is because certain models may outperform others in certain metrics, while others excel in the rest of the metrics. In order to determine the best model, we examine its performance across the metrics and compare it to the other models. Beginning with the triple classification task, which serves as the primary objective for all models, it is evident that we have successfully accomplished the task. In fact, we have achieved nearly flawless results in certain experiments.

Upon examining the aggregated results, it becomes evident that there is a wide range of both quantitative and qualitative findings. The initial set of models, whose results are shown in Tables 6.2, 6.3, and 6.4, utilized *ComplEx*-like embeddings and incorporated skip connections, resulting in improved and well-balanced performance across all three thresholds. The *TransH*-like embeddings model also achieved a well-balanced performance across the metrics for the 0.7 threshold. It is worth mentioning that all the models in this group performed admirably, with only minor variations in the metrics. Overall, the model using *DistMult*-like embeddings did not perform well. The graphs clearly demonstrate the significant impact of adding dropout layers, normalization, and early stopping on preventing overfitting and creating models suitable for a wide range of scenarios.

When examining the accuracy of the probabilities derived from the calibration plots, it becomes apparent that for most models, particularly for lower probabilities (≤ 0.5), which are typically less significant, the models tend to lack confidence. However, when it comes to probabilities greater than 0.5, the models tend to be overly confident. The *ComplEx* models that excelled in this group demonstrated nearly flawless calibration for probabilities ranging from 0.5 to 0.9, but showed a slight tendency to be overly confident for probabilities exceeding 0.9. The precision-recall curves exhibit similar patterns across all models, with the exception of the *ComplEx* models, which demonstrate a slightly improved curve for higher recall values. By analyzing the interpretation plots, we can determine which component of the triple has the greatest impact on the probability we select. In most models, the relationship has a significant impact on the probability. When analyzing the *TransH* model, it becomes evident that both the object and subject have a noticeable impact, although it is not as significant as the impact of the relation. In the *ComplEx* model, the probability is greatly influenced by both the subject and the object, sometimes even more so than the relationship itself.

By examining the impact of each component of the triplet on the five specific triples, we can verify that the impacts are consistent with those depicted in the interpretation graphs. We can now observe that all of the models provide satisfactory metrics in the relation prediction assignment, with the *RotateE* model performing slightly better. The *TransH* model produces the most diverse results from the statistics of the recommended triplets, predicting 29 distinct relations, 16 of which have a very high probability exceeding 0.9. In this model, the average probability of triplets is high, exceeding 0.5. The *owl#imports* and the *owl#equivalentClass* are the two most frequently predicted relations, with frequencies of 852 and 374, respectively. Furthermore, the *TransE* model contained numerous distinct relationships with a probability greater than 0.9 in comparison to the total number of distinct predicted relations (7 out of 9). It is noteworthy that all of the models for this group produced triples and relations with high probabilities. According to Table 6.5, the average training time and epochs required for the models in this group are 1500 seconds per epoch and 14 epochs, respectively.

For all the different thresholds we used, Group 2's *TransH*-like embeddings model nearly outperformed all other models, as shown in Tables 6.6, 6.7, and 6.8. All models performed similarly across all metrics, except for the *DistMult* model, which again performed poorly in the aggregate. The models' graphs are quite similar to those in Group 1. The *TransH* model exhibits a significantly superior precision-recall curve and calibration plot in comparison to the other models. The calibration approaches the ideal calibration for probabilities above 0.9, which are relevant to us, despite the fact that all models for probabilities over 0.5 are overconfident. For the new recommended triples, the models predict relationships with probabilities exceeding 0.9. However, they do not make divergent predictions, predicting a maximum of four distinct relations.

The *ComplEx* model is an exception, as it outperformed the other models in terms of clustering metrics and predicts 14 distinct relations, six of which have a probability greater than 0.9. The alternative division strategy does not appear to enhance the models' performance. Folds were used in the dividing procedure, resulting in a significant increase in training time. In this group, according to Table 6.9, the average training time for the models is 1200 seconds per epoch and 14 epochs for each fold, respectively.

In general, the results of Group 3, shown in Tables 6.10, 6.11, and 6.12, are inferior to those of the preceding two categories. The *TransE* and the *ComplEx* are two models that demonstrated subpar aggregate performance. The *RotateE* model, which used the Fourier transform, performed better than

the other models. In this group, the results of the *DistMult* model are in stark contrast to those of the previous categories, which outperformed the clustering metrics. The training and validation diagrams are quite similar to those of the previous groups. The precision-recall curves are significantly worse than those of the previous groups. It is important to note that none of the models predict probabilities exceeding 0.9; however, they exhibit relatively excellent calibration.

The interpretation diagrams' conclusion indicates that the relationship continues to significantly influence the probabilities. However, there are cases where the subject and object also significantly influence the triples' probabilities. The predicted triple statistics show that none of the models predict relationships with probabilities greater than 0.9. The *TransH* model predicts 576 out of 1500 predicted triplets, with a probability greater than 0.6 and a maximal probability of 0.86. It predicts 8 distinct relations. The *DistMult* model, which outperformed the other models, predicts 25 distinct relations with 75 triples with a probability greater than 0.6 and a maximal probability of 0.68. The alternative negative sampling generation strategy does not appear to enhance the models' performance. In this group, according to Table 6.13, the average training time for the models is 1100 seconds per epoch and 13 epochs, respectively.

In Group 4, the results of which are shown in the Tables 6.14, 6.15, and 6.16, where we employ the *GCLayer*, the *HoIE* and *DistMult* models outperform the other models in terms of metrics at thresholds of 0.5 and 0.6. Setting the threshold value to 0.7 results in superior performance across all metrics for the *ComplEx* and *DistMult* models. It is evident that the models in this group exhibit superior performance in comparison to those in Groups 2 and 3. When compared to Group 1, the metrics for Group 1 are more evenly distributed, as evidenced by the nearly equal precision and recall. The *ComplEx* models appear to outperform the other models in terms of the clustering metrics. We can observe that the models have a propensity to overfit when we examine the plots, beginning with the training and validation plots. The early halting halts the training process in a timely manner, but the absence of dropout layers is evident.

All the precision-recall curves, with the exception of those in the *TransH* and *NoEmbds* models, are essentially identical. All the models, except for *DistMult*, share a common feature in their calibration plots for probabilities. It's perfectly calibrated for the low probabilities (≤ 0.5), overconfident for the probabilities between 0.5 and 0.8, and approaches perfect calibration for the high probabilities (> 0.8). For the *DistMult* model, the calibration line is nearly perfect for all the probabilities. The interpretation plots clearly show that, in most cases, the relations significantly influence the models, a conclusion further reinforced by the magnitude of impact for each of the five triples. For the *TransE*, *TransH*, and *DistMult* models, there are some instances where the greatest impact comes from the subject of the triples.

Looking at the predicted triple statistics, we can see that the models predict only three to four distinct relations. Many of them also have high probabilities, above 0.9. The *TransH* model has the most triples (50), with a probability over 0.9. In terms of triples, the *ComplEx* model has the highest average probability (0.469) and the lowest standard deviation (0.11). The models predict the following relations: *level*, *hasReference*, *term*, and *22 – rdf – syntax – ns#type*. In this group, according

to Table 6.17, the average training time for the models is 1200 seconds per epoch and 5 epochs, respectively.

In comparison to Group 4, the results in Group 5, shown in Tables 6.18, 6.19, and 6.20, which utilize the *GATLayer*, are marginally inferior. In this context, the *TransH* for thresholds of 0.5 and 0.6 and the *HoIE* for thresholds of 0.7 are the most effective models. In the clustering evaluation, the *DistMult* model outperforms the others. Overall, the *RotatE* model demonstrated subpar performance for this group. We can observe that we have high precision and low recall, or the opposite. The *DistMult*-like embeddings model's metrics were consistent across all three thresholds we used. It has an absolute precision of 1.0, but its recall is extremely low (0.02). All models in this category exhibit a greater tendency to overfit than those in Group 4. The precision-recall curves are largely consistent with the previous ones, with the exception of the *DistMult* and *RotatE*-like embeddings models, which yield suboptimal results for this group. The probability calibration diagrams indicate that the models are overconfident for probabilities up to 0.7, while they are sufficiently close to the precisely calibrated line for higher probabilities. The relation appears to have a significant impact on the interpretation plots in the majority of cases; however, there are instances in which the subject or object had a greater impact.

Upon examining the predicted relation statistics, it is evident that there are no triples or relations with probabilities exceeding 0.9. The *TransE* model has the highest number of triples (140), with a probability at or above 0.6 and a maximal predicted probability of 0.74. The results of the *HoIE* model were also comparable. We can conclude that the *GATLayer* does not appear to enhance the models. In comparison to the preceding groups, this layer also increases the training duration and the number of epochs that the models require to train. According to Table 6.21, 7 epochs and 3200 seconds per epoch are the average training times for the models, respectively.

Within Group 6, we make use of both the *GCNLayer* and the *GATLayer*. After reviewing the metrics, shown in Tables 6.22, 6.23, and 6.24, it is evident that the models exhibit either high accuracy and poor recall, or vice versa for the higher thresholds. Furthermore, using both *GCNLayer* and *GATLayer* together enhances performance compared to using them separately. The *HoIE* and *TransE* embeddings models consistently outperformed other models across all three thresholds. The *HoIE* model exhibited superior performance in the clustering assessment. The *DistMult* model has a very high recall rate across all thresholds, although its accuracy value is lower. The training and validation charts once again demonstrate the indispensability of the dropout layers. The precision-recall curves of the previous groups' models are mostly consistent, except for the *DistMult* and *ComplEx* models, which perform somewhat worse. Upon examination of the probability calibration, it is evident that the *TransE*, *TransH*, and *HoIE* models have similar characteristics in their calibration. Models exhibit overconfidence for probabilities less than 0.5 and underconfidence for probabilities greater than 0.5.

The *DistMult* model exhibits overconfidence over its entire range of probabilities, except for extremely high probabilities when the model demonstrates perfect calibration. The *ComplEx* model only considers probabilities with values equal to or greater than 0.5. The model exhibits overconfidence for probabilities ranging from 0.5 to 0.8 and underconfidence for probabilities beyond this range. The interpretation plots exhibit a high degree of similarity to the preceding ones. Upon examining the statistics for the projected triples, it is evident that once again, the models do not forecast relations with a probability above 0.9. The *TransE* model yields the highest probability, reaching a value of 0.74. This model forecasts 11 unique relationships. The *ComplEx* model predicts

43 unique relations, with a maximum probability of 0.66 and a low standard deviation of 0.02. Using both layers, we reduced training time compared to Group 6. According to Table 6.25, the average training time is 6 epochs, or 2200 seconds per epoch.

In Group 7, we use the Conv1D architecture. For all thresholds, as shown in Tables 6.26, 6.27, and 6.28, the *NoEmbds* model outperforms the other models. This model also performs better in the clustering evaluation. Overall, the *TransH* and *ComplEx* embedding models perform poorly. The training and validation curves indicate that the models don't overfit to an excessive degree. The precision-recall curve from the *TransE* model, which performed well, is better than the other models. The curves from the *RotatE* and *DistMult* models are not excellent at all. For these models, the probability calibration plots also show that for probabilities up to 0.5, the models are overconfident. For probabilities above 0.5, the *DistMult* model is overconfident, and the *RotatE* model approaches perfect calibration. The interpretation plots again indicate that the relationship has a greater impact in most cases.

When examining the statistics from the predicted triples, we can see that all the models except *TransH*, which predicts only one distinct relation, predict a variety of relationships. The *NoEmbds* model predicts 18 distinct relations with 9 of them having a probability above 0.9 and a maximum probability of 0.997. In this architecture, we have better precision roc-auc and pr-auc values across the thresholds, but we take lower recall and F1 scores. The models that used the Conv1D architecture took a much longer time per epoch to train than the models that used the Dense architecture. According to Table 6.29, the average training time is 9000 seconds per epoch, with a total of 7 epochs. Exceptions are the *DistMult* and *RotatE* models, which required much less time—about 1000 seconds per epoch—and had poor results for both of these models.

Within Group 8, we experiment with integrating the *GCLayer* and Conv1D architectures. It is evident, according to Tables 6.30, 6.31, and 6.32, that the accuracy improves but the recall and F1 score decrease. When comparing the models in Group 4, which used the Dense architecture with the same layer, we see significantly improved outcomes for all models and thresholds. It is worth mentioning that, once again, the model that consistently performed better than the others in most of the measures is the one that does not use sophisticated embeddings. Early stopping prevents overfitting by ending the training process at the right moment. The *NoEmbds* model has a superior precision-recall curve, while the *TransE* model also demonstrates commendable performance in terms of metrics, with a similarly impressive precision-recall curve. Once again, the *TransH* and *RotatE* models exhibit unsatisfactory performance. The probability calibration plots demonstrate that the well-performing models exhibit accurately calibrated probabilities over the entire probability spectrum. The calibration becomes almost flawless, particularly for probabilities over 0.8.

By analyzing the statistics of the projected triples, it is evident that both the *HoIE* and *DistMult* models make many predictions of relations with significant probability, above 0.9. However, it is important to note that these models exhibit excessive confidence when it comes to high probabilities. The *TransE* model, known for its accurate probability calibration plot, predicts 11 unique relationships, with two having a probability above 0.9 and the highest probability being 0.952. According to Table 6.33, the average training time is 12000 seconds per epoch, for a total of 7 epochs.

In Group 9, where we try the *GATLayer* combined with the Conv1D architecture, we can see similar impacts to the models as in Group 5. We have observed in Tables 6.34, 6.35, and 6.36 an increase in

precision surpassing that of Group 8, as well as a slight decrease in the remaining metrics. For this group, the *HoIE* model outperforms the rest of the models, with the *NoEmbds* model also performing very well. In the clustering evaluation, these two models performed the best. All of these models' evaluation plots are consistent with the excellent model metrics. We can still see poor results from the *RotateE* and *TransH* models. According to the predicted triple statistics, high-probability relationships exist only in the *DistMult* and *TransE* models. But these models don't have a decent probability calibration plot. The *NoEmbds* model, which performed well, predicts 7 distinct relations with a maximum probability of 0.84. The *HoIE* model, the best in this group, predicts 16 distinct relations with a maximum probability of 0.66. In terms of training time, we can see an increase in all models per epoch when adding *GATLayers* compared with *GCNLayers*. According to Table 6.37, the average training time is 15000 seconds per epoch, with 7 epochs in total.

Finally, in Group 10, we utilize both *GCNLayer* and *GATLayer*. The results shown in Tables 6.38, 6.39, and 6.40 again have high precision and lower recall values. These results don't appear to be significantly superior to using either the *GCNLayer* or the *GATLayer* alone, or when compared to Group 6, which utilizes both layers with Dense architecture. The lower recall values also affect the MCC value and the F1 score, resulting in better precision. The *NoEmbds* and *HoIE* models are superior to the others. The *HoIE* also performs better in the clustering evaluation. Once again, we observe that the *RotateE* model yields inferior results. The models that performed the best in this group also have excellent evaluation plots. The *NoEmbds* model has a better precision-recall curve and almost perfect calibrated probabilities for the whole range of probabilities.

Also, for the majority of instances we examined, the interpretation plots show that the relation has the greatest impact on the model. There are also cases where other triple parts affect probabilities. The *HoIE* model has overconfidently calibrated probabilities for the probabilities between 0.6 and 0.8 and approaches perfect calibration for the rest of the probabilities. In this model, the interpretation plots show that the subject has the greatest impact multiple times. Examining the statistics from the predicted triples, we can see that only the *HoIE*, *NoEmbds*, and *DistMult* models predict relations with high probability, above 0.9. The *NoEmbds* model predicts five distinct relationships, with three having a probability above 0.9, a maximum probability of 0.94, and the lowest standard deviation having a value of 0.21.

The *HoIE* model predicts 3 distinct relations, with one of them having a probability above 0.9, a maximum probability of 0.99, and a 0.26 standard deviation. The *DistMult* model predicts 3 distinct relations, with 2 of them having a probability above 0.9, a maximum probability of 0.99, and a 0.32 standard deviation. Concerning the training time per epoch, the models required increased training time when adding both the *GCNLayer* and *GATLayer*. The average training time, according to Table 6.41, is 20000 seconds per epoch, with 7 epochs in total.

Moving forward to the individual experiments, we can observe in Table 6.42 that the first experiment, using [method 2](#) as a negative sampling generation strategy that multiplies the negative samples by 3, does not significantly improve the results. In contrast, we get poor results both for the triple classification and relation prediction tasks. According to Table 6.43, the training time for experiment 1 is 1550 seconds per epoch, and the required epochs are 5. When using [method 3](#) as the negative sampling generation strategy in experiment 2, we can see in Table 6.44 far better results, and in some cases, even better results, compared with the models in groups 1–10. Precision is the only metric that is lower than the rest. The other metrics remain high across all three thresholds.

The model predicts 25 distinct relations with 21 of them having a probability above 0.9. The average probability is also higher than that of the other models, which typically range from 0.4 to 0.56. The maximum probability is 1, and the minimum is 0. Despite achieving excellent results, the clustering evaluation reveals poorer results compared to other models. This evaluation is merely a suggestion, and it is the sole method available to assess the newly suggested triples. According to Table 6.45, the training time for experiment 2 is 850 seconds per epoch, and the required epochs are 5.

The precision-recall curve is almost perfect; the training and validation plots show that the model doesn't overfit; and the interpretation plots show that the relation has the greatest impact on the probabilities. The probability calibration plot shows perfect calibration for the probabilities below 0.5, overconfidence for the probabilities between 0.5 and 0.9, and nearly perfect calibration for the rest of the probabilities. As shown in Tables 6.46 and 6.48, when we use [method 2](#) as the negative sampling generation strategy in experiments 3 and 4, the models again give very poor results. According to Tables 6.47 and 6.49, the training time for experiment 3 is 1590 seconds per epoch with 5 epochs in total, and for experiment 4, it is 3100 seconds per epoch with 7 epochs in total.

When we use method 3 to obtain the negative samples in experiment 5, the model produces exceptional, nearly flawless results, as shown in Table 6.50. The precision-recall curve is nearly flawless, as are all of the metrics. The calibration plot indicates that the probabilities below 0.5 are overconfident, while the probabilities above 0.5 are underconfident. This was a rare occurrence in the other models. The model predicts 42 out of 43 relationships, with 40 of them having a probability greater than 0.9. The predicted triples have an exceptionally high average probability, with a value of 0.83. The utmost predicted probability is 0.99, and the standard deviation is 0.12. According to Table 6.51, the training time for experiment 5 is 1500 seconds per epoch, with 13 epochs in total.

The interpretation plots show a significant impact on the subject, the relations, and the object. Also noteworthy is that the most frequent relation that this model predicts has a frequency of 134. That means the model doesn't focus too much on a single or a few relations. Despite still being subpar, the results of experiment 6 shown in Table 6.52, which uses [method 2](#) to obtain negative samples, significantly improve over the other models that previously used this method. The model does not predict relationships with a high probability (> 0.9), and it has low recall and F1 score values. The model's probability calibration plot for the entire range of probabilities is also virtually flawless. According to Table 6.53, the training time for experiment 6 is 17400 seconds per epoch, with 7 epochs in total.

In experiments 7 and 8, we achieve exceptional results, as shown in Tables 6.54 and 6.56, by utilizing the Conv1D architecture in conjunction with [method 3](#) to obtain the negative samples. Both investigations' precision-recall curves are perfect. In both experiments, the probability calibration plots are nearly identical, displaying underconfident probabilities for the entire range. In the seventh experiment, the model predicts 11 distinct relations, all of which have probabilities exceeding 0.9. In the eighth experiment, we observe five distinct relations yet again with extremely high probabilities. The average probability for both experiments is high, while the utmost predicted probability is 1. Additionally, both experiments share the most frequent relation, the *hasCode*, and several common relations in the top five. According to Tables 6.55 and 6.57, the training time for experiment 7 is 8400 seconds per epoch with 5 epochs in total, and for experiment 8, it is 20500 seconds per epoch with 5 epochs in total.

We still get poor results in experiment 9, as shown in Table 6.58, where we augment the dataset to ensure that positive triples remain positive and mark the rest as negative. The model forecasts only 4 relations, with a maximum predicted probability of a triple of 0.45. Furthermore, the training time per epoch increases, nearly doubling the time without augmentation. According to Table 6.59, the training time for experiment 9 is 3150 seconds per epoch, with 5 epochs in total. In experiment 10, combining the same augmentation method with the Conv1D architecture and the *GCNLayer* significantly improves the model's performance, as Table 6.60 demonstrates. Although it still doesn't predict relations with probabilities above 0.9. The model has an excellent precision-recall curve and a great probability calibration plot. The interpretation plots demonstrate that the relationship has a greater impact. Notably, despite the augmentation, according to Table 6.61, this model required 7900 seconds for training per epoch and 5 epochs in total, which is both less time and epochs than the model without the augmentation.

In experiment 11, we employ [method 2](#) to obtain negative samples, a method that yielded poor results in previous experiments. However, our results, based on Table 6.62, are superior to those of previous experiments that employed this method. The recall and F1 score values for this experiment, conducted using method 2, are also smaller than the precision. The precision recall is good, and the probability calibration plot shows near-perfect calibration for the entire probability range. The interpretation plots show a significant impact on the probabilities for all parts of the triples. The model predicts 21 relations with only three of them having a probability above 0.6. According to Table 6.63, the training time for experiment 11 is 15000 seconds per epoch, and the required epochs are 5.

In experiment 12, we use the same model as in experiment 11, but with [method 3](#) for getting the negative samples, we can again see perfect results for all the thresholds we use in Table 6.64. The precision-recall curve is perfect, and the probability calibration plot shows very good, calibrated probabilities. The interpretation plots again show a significant impact on the probabilities for all parts of the triples. The model predicts 20 distinct relations with 17 of them having probabilities above 0.9. The predicted triples have an average probability of 0.3, a maximum probability of 1, and a standard deviation of 0.42. In terms of training time per epoch, we can see that in this experiment, it was 7400 seconds compared to 15000 without the augmentation method. However, according to Table 6.65, it required 13 epochs, compared to 7 epochs for the model without the augmentation method.

Table 6.66 demonstrates the unsatisfactory performance of the custom attention layer in experiment 13. The model utilizes all the specified epochs, and although we see a decrease in both training and validation losses, the obtained results are not satisfactory. According to Table 6.67, the training time for experiment 13 is 7800 seconds per epoch, and the required epochs are 15. We concluded experiment 14, where we implemented dynamic embeddings, which produced modest results in terms of metrics, as shown in Table 6.68. The model exhibits a steep and accurate precision-recall curve. The probability calibration figure shows underconfident probabilities below 0.5, overconfident probabilities ranging from 0.5 to 0.9, and a well-calibrated range for the remaining probabilities. The interpretation plots clearly demonstrate that the relationship has a substantial influence on the probability.

The model makes predictions for 8 unique relationships, with an average probability of 0.42 for the triples. The highest probability observed is 0.87, and the standard deviation is 0.25. The most frequently predicted relations by this model have significant differences in frequencies. The relation with the highest frequency is *databasePath*, which occurs 1124 times. The second most common

relationship is *hasFrequentTerm*, which occurs 350 times. The third most frequent relation is *hasCode*, occurring 14 times. According to Table 6.69, the training time for experiment 14 is 30500 seconds per epoch, and the required epochs are 7.

The number of epochs remains constant, but the training time per epoch is more than 5000 seconds. In general, all individual experiments utilizing [method 2](#) for obtaining negative samples resulted in a significantly longer training time per epoch, which was expected given the higher number of negative samples compared to the other negative sampling generation strategies we employed. [Method 3](#) sometimes slightly increases the training time per epoch. The epochs required for training remained almost the same for most experiments compared with the corresponding models from groups 1–10.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis aimed at enriching the Eurostat's knowledge graph utilizing deep learning models. The triple classification task facilitates this achievement. In this task, the model assigns a probability to each triple. As a result, the model is unable to directly predict a relationship for the triples. Thus, we select some non-directly connected nodes from the graph and keep the one with the highest triple probability after trying all possible relations. We chose a small number of not directly connected nodes, 1500 in total, to ensure that no resource issues would arise. Because the number of nodes is small, we can use the model's predict function for all possible triples in every set—or file—of nodes. Otherwise, we would have to use the predict function for each triple separately to determine the impact on the probabilities for the selected instances of the test set. Generally, the training time for each epoch aligns with the time required to execute this function for a single triple. For the dense architecture, which required less time compared to the Conv1D architecture, the time ranges from 400 to 3300 seconds.

For the Conv1D architecture, which required a significantly longer time to train compared to the Dense architecture, the time ranges from 800 to 25500 seconds per epoch for the most demanding time models. The training time posed a challenge, particularly for models with higher time requirements, which frequently necessitated additional time for the remaining stages of the process. Finding the optimal parameters and hyperparameters for the models posed a significant challenge. In many cases, changing even a little parameter (e.g., the unit of the layers, the learning rate) leads to significantly different results and training time demands per epoch. So, in some way, the parameters used constitute a golden mean between the required resources, training time, and allowances. Also, there were increasing demands on RAM memory, especially for the models that used the Conv1D architecture.

Generally, we observe that models and experiments requiring more training time yield superior metrics and evaluation plots in terms of results and metric performance. However, there were times when models that needed less time performed better, such as the *ComplEx* model from Group 1 and most of the models in Group 8. There were also times when a slight improvement in metrics required a much longer training period, such as with experiment 10 and the *ComplEx* model from Group 1. The models achieved evaluation metrics in all possible ranges. The experiments clearly demonstrate the significant impact of the negative samples. In many experiments where we have ensured the high quality of negative samples, we can notice a very substantial increase in model performance. However, in some experiments, such as experiment 1, where we ensure the high quality of negative samples, we still obtain poor results, a fact that calls into question the equal importance of the model and all the parameters used.

With regard to embeddings, we can also see their particularly important effect on the model's performance. There wasn't any one embedding type that consistently outperformed the others, but there were specific embedding types that performed better across some group or individual

experiments. For instance, the *TransH*-like models demonstrated strong performance in the initial group of experiments. However, in subsequent groups utilizing convolutional layers, these models consistently yielded poor results. Additionally, the embeddings significantly influence the training time required per epoch. For example, the *NoEmbds* model in Group 8 requires 8300 seconds per epoch, and the *TransE* model with the same setup in the same group requires 16000 seconds per epoch.

We can also notice that the models in the last groups—the ones that utilized no sophisticated method in the embeddings—performed very well. As a result, we can conclude that it is important to always try the simpler forms in the models because they may perform better in terms of metrics or training time. The minimum time comes from the *TransE* model in Group 2, which requires around 450 seconds per epoch, and the maximum time is from experiment 14, which requires around 30500 seconds per epoch.

It's crucial to consider the potential applications of the recommended models utilized in this thesis. With a single training process, these models can handle a wide range of tasks. For example, one significant application of triple classification models is in entity resolution and disambiguation. Large datasets may present entities in various representations or with ambiguous references. Triple classification models can help clear up these terms by checking which triples make sense. This makes sure that the data accurately shows the different terms. Triple classification models are also pivotal in the construction and maintenance of ontologies. Ontologies, which define the relationships between concepts within a domain, require accurate and plausible triples to be effective. These models assist in verifying the correctness of the relationships defined within an ontology, thereby ensuring its validity and usefulness for tasks such as semantic search and data annotation.

Triple classification models also improve user-item interaction predictions, so recommendation systems benefit. These models can evaluate the likelihood of interactions such as (User, likes, Item) or (User, watched, Movie), thereby refining recommendation algorithms. This results in more personalized and accurate recommendations for users, enhancing their overall experience. Also In scientific research, these models assist in knowledge discovery and hypothesis generation. Triple classification models help researchers identify new connections and insights by validating relationships extracted from scientific literature, thereby advancing knowledge frontiers and facilitating innovative research.

In conclusion, triple classification models extend their utility far beyond simple relation prediction tasks. They are instrumental in entity resolution, data integration, automated reasoning, ontology maintenance, recommendation systems, scientific research, etc. Their ability to evaluate the plausibility of triples makes them invaluable across various domains, ensuring data integrity, enhancing analytical capabilities, and supporting advanced research and applications.

7.2 Future work

Several avenues for future work and enhancements exist, which can further refine and expand the capabilities of the models presented in the current thesis. One promising direction for future work is the integration of more complex and dynamic graph neural networks (GNNs). While the current model employs GCN and GAT layers, exploring other variants such as GraphSAGE, attention-based GNNs, or even more recent developments like transformers for graphs could yield better performance and

scalability. Another potential enhancement involves improving the generation and handling of negative samples. The majority of models use a simple random approach to generate negative triples. Certain advanced methods have demonstrated the importance of the quality of the negative samples. Future work could explore adversarial negative sampling or more sophisticated methods that ensure harder negative samples, leading to better training and more robust embeddings.

Enhancing the scalability of the model is also crucial. As knowledge graphs grow in size, training on large-scale datasets becomes computationally intensive. Techniques such as distributed training, mini-batch training on subgraphs, or leveraging sparse representations can help in scaling the model to handle massive knowledge graphs efficiently. Incorporating richer features for entities and relations is another valuable direction. Currently, the models primarily rely on embeddings derived from the graph structure. Including additional features, such as textual descriptions, ontological information, or multimodal data (images, audio, etc.), can enrich the embeddings and improve the model's performance on various tasks.

It can also be advantageous to investigate transfer learning and domain adaptation. Some common structures or entities may be present in knowledge graphs from various domains. The model can optimize its performance on domain-specific tasks by pre-training it on a large, general-purpose knowledge graph and fine-tuning it on a domain-specific graph, thereby leveraging existing knowledge. Furthermore, the present method for relation prediction entails the iterative selection of the relation with the highest probability from among all potential relations. This brute-force method is computationally expensive and does not inherently account for the semantic plausibility of the generated triples, despite its potential effectiveness. A promising direction for future research is the integration of semantic context to filter and restrict the triples to those that have genuine significance.

One promising avenue is incorporating ontological knowledge and semantic information from external resources like WordNet, ConceptNet, or domain-specific ontologies. By integrating this semantic context, the model can better understand the relationships between entities and the types of relations that are plausible. This can be achieved by mapping entities and relations to their corresponding concepts in these ontologies and using this additional information to guide the prediction process. For instance, if an entity is a 'Person' and another entity is a 'City', the model can be informed that relations like 'lives_in' or 'was_born_in' are semantically meaningful, whereas relations like 'is_made_of' are not.

Furthermore, contextual embeddings are useful for better capturing the semantic context of entities and relations, including those produced by transformer-based models such as BERT. Richer representations able to capture the meaning and use of items in different settings may be produced using these embeddings. Incorporating these embeddings into the model helps it more clearly identify the semantic fit of relations for a given pair of entities.

Implementing a semantic filtering mechanism involves adding a pre-processing step that evaluates the semantic plausibility of each potential triple before passing it to the relation prediction model. This can be achieved using pre-trained language models or rule-based systems that assess the coherence of the entity-relation-entity combinations. Only those triples that pass this semantic filter would be considered for the final prediction, thereby reducing the computational burden and improving the relevance of the predictions.

Furthermore, introducing a semantic regularization term in the loss function can encourage the model to prefer semantically coherent triples. This regularization term can penalize unlikely or nonsensical triples based on external semantic knowledge. For example, if a predicted triple conflicts with known ontological constraints or common-sense knowledge, the regularization term can increase the loss, thereby guiding the model towards more plausible predictions.

However, implementing these enhancements requires careful consideration of several factors, including the availability of ontological resources, the computational overhead introduced by integrating and querying semantic resources, and the increased complexity of the model. Robust validation and testing are necessary to ensure that the benefits of adding semantic context outweigh the potential drawbacks.

References

- [1] A. Vassiliades, N. Bassiliades, G. Meditskos, and K. Spiliopoulos, "Building Eurostat Knowledge Graph," in *Proceedings of the 14th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, SCITEPRESS - Science and Technology Publications, 2022, pp. 128–135. doi: 10.5220/0011527100003335.
- [2] "Home (Eurostat)." Accessed: Jun. 20, 2024. [Online]. Available: <https://ec.europa.eu/eurostat>
- [3] R. E. Neapolitan and X. Jiang, *Artificial intelligence: with an introduction to machine learning*, Second edition. in Chapman & Hall/CRC artificial intelligence and robotics series. Boca Raton: CRC Press, Taylor & Francis Group, 2020.
- [4] E. Alpaydin, *Introduction to machine learning*, Fourth edition. in Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2020.
- [5] Géron and Aurélien, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, Second edition. O'Reilly Media, Inc, 2019.
- [6] A. E. A. Ibrahim, A. A. Elamer, and A. N. Ezat, "The convergence of big data and accounting: innovative research opportunities," *Technol Forecast Soc Change*, vol. 173, p. 121171, Dec. 2021, doi: 10.1016/j.techfore.2021.121171.
- [7] "What is Machine Learning?" Accessed: Apr. 17, 2024. [Online]. Available: <https://www.opentext.com/what-is/machine-learning>
- [8] V. Mayer-Schönberger and K. Cukier, *Big data: a revolution that will transform how we live, work and think*, 1. publ. London: Murray, 2013.
- [9] "A minute on the internet in 2023." Accessed: Apr. 17, 2024. [Online]. Available: <https://www.mickmel.com/a-minute-on-the-internet-in-2023/>
- [10] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. New York, NY: Springer New York, 2009. doi: 10.1007/978-0-387-84858-7.
- [11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. in Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016.
- [12] T. Chen and C. Guestrin, "XGBoost," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA: ACM, Aug. 2016, pp. 785–794. doi: 10.1145/2939672.2939785.
- [13] K. P. Murphy, *Machine learning: a probabilistic perspective*. in Adaptive computation and machine learning series. Cambridge, MA: MIT Press, 2012.

- [14] J. T. Vanderplas, *Python data science handbook: essential tools for working with data*, First edition. Sebastopol, CA: O'Reilly Media, Inc, 2016.
- [15] "What is Feature Engineering?" Accessed: Dec. 14, 2023. [Online]. Available: <https://www.geeksforgeeks.org/what-is-feature-engineering/>
- [16] M. Bowles, *Machine learning with Spark and Python: essential techniques for predictive analytics*, Second edition. Indianapolis, IN: John Wiley & Sons, 2020.
- [17] "Underfitting and Overfitting." Accessed: Dec. 13, 2023. [Online]. Available: <https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/>
- [18] S. Marsland, *Machine learning: an algorithmic perspective*, Second edition. in Chapman & Hall/CRC machine learning & pattern recognition series. Boca Raton: CRC Press, 2015.
- [19] "Underfitting, good fit, overfitting." Accessed: Dec. 13, 2023. [Online]. Available: <https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76>
- [20] "Top 10 Machine Learning Algorithms For Beginners: Supervised, and More." Accessed: Dec. 14, 2023. [Online]. Available: <https://www.simplilearn.com/10-algorithms-machine-learning-engineers-need-to-know-article>
- [21] "Fundamental machine learning algorithms." Accessed: Dec. 13, 2023. [Online]. Available: <https://johnvastola.medium.com/10-must-know-machine-learning-algorithms-for-data-scientists-adbf3272398a>
- [22] "Top 10 Machine Learning Algorithms." Accessed: Dec. 13, 2023. [Online]. Available: <https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/>
- [23] A. C. Müller and S. Guido, *Introduction to machine learning with Python: a guide for data scientists*, First edition. Sebastopol, CA: O'Reilly Media, Inc, 2016.
- [24] "Logistic Regression in R: The Ultimate Tutorial with Examples." Accessed: Dec. 14, 2023. [Online]. Available: <https://www.simplilearn.com/tutorials/data-science-tutorial/logistic-regression-in-r>
- [25] "What Is Principal Component Analysis (PCA)?" Accessed: Feb. 21, 2024. [Online]. Available: <https://www.analyticsvidhya.com/blog/2016/03/pca-practical-guide-principal-component-analysis-python/>
- [26] "Introduction to Deep Learning." Accessed: Dec. 14, 2023. [Online]. Available: <https://www.geeksforgeeks.org/introduction-deep-learning/>
- [27] C. Surianarayanan, J. J. Lawrence, P. R. Chelliah, E. Prakash, and C. Hewage, "A Survey on Optimization Techniques for Edge Artificial Intelligence (AI)," *Sensors*, vol. 23, no. 3, p. 1279, Jan. 2023, doi: 10.3390/s23031279.

- [28] “Deep learning is a subset of machine learning .” Accessed: Dec. 13, 2023. [Online]. Available: <https://flatironschool.com/blog/deep-learning-vs-machine-learning/>
- [29] A. Rosebrock, “Deep Learning for Computer Vision with Python Starter Bundle 1st Edition (1.1.0).”
- [30] C. Albon, *Machine learning with Python cookbook: practical solutions from preprocessing to deep learning*, First edition. Beijing Boston Farnham Sebastopol Tokyo: O’Reilly, 2018.
- [31] “Deep Neural Network: The 3 Popular Types (MLP, CNN and RNN).” Accessed: Dec. 14, 2023. [Online]. Available: <https://viso.ai/deep-learning/deep-neural-network-three-popular-types/>
- [32] F. Chollet, *Deep learning with Python*. Shelter Island, New York: Manning Publications Co, 2018.
- [33] “Training process in Deep Learning algorithms.” Accessed: Apr. 19, 2024. [Online]. Available: https://www.researchgate.net/figure/Training-process-in-Deep-Learning-algorithms_fig1_350361033
- [34] “Machine Learning vs. Deep Learning training features.” Accessed: Dec. 13, 2023. [Online]. Available: <https://jelvix.com/blog/ai-vs-machine-learning-vs-deep-learning>
- [35] J. M. Tomczak, *Deep Generative Modeling*. Springer International Publishing, 2022. doi: 10.1007/978-3-030-93158-2.
- [36] D. Foster and K. J. Friston, *Generative deep learning: teaching machines to paint, write, compose, and play*, Second edition. Beijing ; Boston: O’Reilly, 2023.
- [37] T. N. Kipf and M. Welling, “Semi-Supervised Classification with Graph Convolutional Networks.” arXiv, 2016. doi: 10.48550/ARXIV.1609.02907.
- [38] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A Comprehensive Survey on Graph Neural Networks,” 2019, doi: 10.48550/ARXIV.1901.00596.
- [39] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph Attention Networks.” arXiv, 2017. doi: 10.48550/ARXIV.1710.10903.
- [40] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How Powerful are Graph Neural Networks?” arXiv, 2018. doi: 10.48550/ARXIV.1810.00826.
- [41] J. Zhou *et al.*, “Graph Neural Networks: A Review of Methods and Applications.” arXiv, 2018. doi: 10.48550/ARXIV.1812.08434.
- [42] “Machine Learning Model Evaluation.” Accessed: Dec. 15, 2023. [Online]. Available: <https://www.geeksforgeeks.org/machine-learning-model-evaluation/>
- [43] “What Is Model Training?” Accessed: Dec. 15, 2023. [Online]. Available: <https://oden.io/glossary/model-training/>

- [44] “Data Cleaning and Preprocessing.” Accessed: Dec. 15, 2023. [Online]. Available: <https://medium.com/analytics-vidhya/data-cleaning-and-preprocessing-a4b751f4066f>
- [45] E. Ameisen, *Building machine learning powered applications: going from idea to product*, First edition. Beijing Boston: O'Reilly, 2020.
- [46] J. T. Vanderplas, *Python data science handbook: essential tools for working with data*, First edition. Sebastopol, CA: O'Reilly Media, Inc, 2016.
- [47] “How to Handle Missing Data.” Accessed: Apr. 17, 2024. [Online]. Available: <https://towardsdatascience.com/how-to-handle-missing-data-b557c9e82fa0>
- [48] “Detecting and Treating Outliers | Treating the odd one out!” Accessed: Apr. 17, 2024. [Online]. Available: <https://www.analyticsvidhya.com/blog/2021/05/detecting-and-treating-outliers-treating-the-odd-one-out/>
- [49] “Encoding Categorical Variables: One-hot vs Dummy Encoding.” Accessed: Apr. 17, 2024. [Online]. Available: <https://towardsdatascience.com/encoding-categorical-variables-one-hot-vs-dummy-encoding-6d5b9c46e2db>
- [50] T. M. Mitchell, *Machine Learning*. in McGraw-Hill series in computer science. New York: McGraw-Hill, 1997.
- [51] “Train, test and validation split.” Accessed: Dec. 13, 2023. [Online]. Available: <https://datascience.stackexchange.com/questions/61467/clarification-on-train-test-and-val-and-how-to-use-implement-it>
- [52] S. Simon, N. Kolyada, C. Akiki, M. Potthast, B. Stein, and N. Siegmund, “Exploring Hyperparameter Usage and Tuning in Machine Learning Research,” in *2023 IEEE/ACM 2nd International Conference on AI Engineering – Software Engineering for AI (CAIN)*, IEEE, May 2023, pp. 68–79. doi: 10.1109/CAIN58948.2023.00016.
- [53] “An In-Depth Exploration of Hyperparameter Tuning.” Accessed: Dec. 15, 2023. [Online]. Available: <https://orbofi.medium.com/an-in-depth-exploration-of-hyperparameter-tuning-a63e249fb0>
- [54] “Significance of Hyperparameters.” Accessed: Dec. 13, 2023. [Online]. Available: <https://www.codegigs.app/parameter-and-hyperparameter-in-machine-learning/>
- [55] “Evaluation Metrics in Machine Learning.” Accessed: Dec. 15, 2023. [Online]. Available: <https://www.geeksforgeeks.org/metrics-for-machine-learning-model/>
- [56] “12 Important Model Evaluation Metrics for Machine Learning Everyone Should Know.” Accessed: Dec. 15, 2023. [Online]. Available: <https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/>

- [57] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf Process Manag*, vol. 45, no. 4, pp. 427–437, Jul. 2009, doi: 10.1016/j.ipm.2009.03.002.
- [58] D. J. Hand, "Measuring classifier performance: a coherent alternative to the area under the ROC curve," *Mach Learn*, vol. 77, no. 1, pp. 103–123, Oct. 2009, doi: 10.1007/s10994-009-5119-5.
- [59] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit Lett*, vol. 27, no. 8, pp. 861–874, Jun. 2006, doi: 10.1016/j.patrec.2005.10.010.
- [60] "What is a confusion matrix?" Accessed: Apr. 18, 2024. [Online]. Available: <https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5>
- [61] "A Quick Guide to AUC-ROC in Machine Learning Models." Accessed: Apr. 18, 2024. [Online]. Available: <https://towardsdatascience.com/a-quick-guide-to-auc-roc-in-machine-learning-models-f0aedb78fbad>
- [62] "Evaluation Metrics in Machine Learning." Accessed: Apr. 18, 2024. [Online]. Available: <https://www.geeksforgeeks.org/metrics-for-machine-learning-model/>
- [63] D. L. Davies and D. W. Bouldin, "A Cluster Separation Measure," *IEEE Trans Pattern Anal Mach Intell*, vol. PAMI-1, no. 2, pp. 224–227, Apr. 1979, doi: 10.1109/TPAMI.1979.4766909.
- [64] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *Int J Comput Vis*, vol. 88, no. 2, pp. 303–338, Jun. 2010, doi: 10.1007/s11263-009-0275-4.
- [65] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. New York: Cambridge University Press, 2008.
- [66] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data.," *Biometrics*, vol. 33, no. 1, pp. 159–74, Mar. 1977.
- [67] "Intersection over Union (IoU)." Accessed: Apr. 18, 2024. [Online]. Available: <https://encord.com/glossary/iou-definition/>
- [68] "Metrics for Evaluating Machine Learning Models." Accessed: Dec. 13, 2023. [Online]. Available: <https://www.kdnuggets.com/2018/06/right-metric-evaluating-machine-learning-models-2.html>
- [69] G. Antoniou and G. Antoniou, Eds., *A Semantic Web primer*, 3rd ed. in Cooperative information systems. Cambridge, Mass: MIT Press, 2012.
- [70] M. Kifer, J. de Bruijn, H. Boley, and D. Fensel, "A Realistic Architecture for the Semantic Web," 2005, pp. 17–29. doi: 10.1007/11580072_3.
- [71] "Web for real people." Accessed: Mar. 06, 2024. [Online]. Available: <https://www.w3.org/2005/Talks/0511-keynote-tbl/>

- [72] J. Sequeda and O. Lassila, *Designing and Building Enterprise Knowledge Graphs*. Cham: Springer International Publishing, 2021. doi: 10.1007/978-3-031-01916-6.
- [73] “Resource Description Framework (RDF).” Accessed: Dec. 22, 2023. [Online]. Available: <https://www.w3.org/RDF/>
- [74] “Resource Description Framework (RDF).” Accessed: Dec. 22, 2023. [Online]. Available: <https://www.techtarget.com/searchapparchitecture/definition/Resource-Description-Framework-RDF>
- [75] J. Domingue, D. Fensel, and J. A. Hendler, Eds., *Handbook of Semantic Web Technologies*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-540-92913-0.
- [76] D. Fensel *et al.*, *Enabling Semantic Web Services*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. doi: 10.1007/978-3-540-34520-6.
- [77] N. Konstantinou and D.-E. Spanos, *Materializing the Web of Linked Data*. Cham: Springer International Publishing, 2015. doi: 10.1007/978-3-319-16074-0.
- [78] “Resource Description Framework (RDF) Serialization.” Accessed: Mar. 13, 2024. [Online]. Available: <https://lincsproject.ca/docs/terms/resource-description-framework-serialization>
- [79] D. Allemang and J. A. Hendler, *Semantic Web for the working ontologist: effective modeling in RDFS and OWL*, 2nd ed. Waltham, MA: Morgan Kaufmann/Elsevier, 2011.
- [80] “What is SPARQL?” Accessed: Mar. 13, 2024. [Online]. Available: <https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/>
- [81] S. Sakr, M. Wylot, R. Mutharaju, D. Le Phuoc, and I. Fundulaki, *Linked Data*. Cham: Springer International Publishing, 2018. doi: 10.1007/978-3-319-73515-3.
- [82] S. Bechhofer, “OWL: Web Ontology Language,” in *Encyclopedia of Database Systems*, Boston, MA: Springer US, 2009, pp. 2008–2009. doi: 10.1007/978-0-387-39940-9_1073.
- [83] “OWL Web Ontology Language.” Accessed: Mar. 13, 2024. [Online]. Available: <https://www.w3.org/TR/owl-features/>
- [84] P. Hitzler, M. Krötzsch, and S. Rudolph, *Foundations of Semantic Web technologies*. in Chapman & Hall/CRC textbooks in computing. Boca Raton: CRC Press, 2010.
- [85] *Semantic Web for the Working Ontologist*. Elsevier, 2011. doi: 10.1016/C2010-0-68657-3.
- [86] J. Domingue, D. Fensel, and J. A. Hendler, Eds., *Handbook of Semantic Web Technologies*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. doi: 10.1007/978-3-540-92913-0.

- [87] I. Horrocks, B. Motik, and Z. Wang, “The Hermit OWL Reasoner,” *International Workshop on OWL Reasoner Evaluation*, 2012.
- [88] D. Tsarkov and I. Horrocks, “FaCT++ Description Logic Reasoner: System Description,” 2006, pp. 292–297. doi: 10.1007/11814771_26.
- [89] M. Horridge and S. Bechhofer, “The OWL API: A Java API for OWL ontologies,” *Semant Web*, vol. 2, no. 1, pp. 11–21, 2011, doi: 10.3233/SW-2011-0025.
- [90] S. Bechhofer, R. Volz, and P. Lord, “Cooking the Semantic Web with the OWL API,” no. 2870, D. Fensel, K. Sycara-Cyranski, and J. Mylopoulos, Eds., in Lecture notes in computer science., Berlin ; New York: Springer, 2003, pp. 659–675. doi: 10.1007/978-3-540-39718-2_42.
- [91] “What is knowledge graph?” Accessed: Dec. 24, 2023. [Online]. Available: <https://www.ibm.com/topics/knowledge-graph>
- [92] “What is a Knowledge Graph? A comprehensive Guide.” Accessed: Dec. 24, 2023. [Online]. Available: <https://wordlift.io/blog/en/entity/knowledge-graph/>
- [93] B. Villazón-Terrazas, F. Ortiz-Rodriguez, S. Tiwari, M.-A. Sicilia, and D. Martín-Moncunill, Eds., *Knowledge Graphs and Semantic Web*, vol. 1686. Cham: Springer International Publishing, 2022. doi: 10.1007/978-3-031-21422-6.
- [94] E. F. Codd, “A relational model of data for large shared data banks,” *Commun ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970, doi: 10.1145/362384.362685.
- [95] P. P.-S. Chen, “The entity-relationship model—toward a unified view of data,” *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 9–36, Mar. 1976, doi: 10.1145/320434.320440.
- [96] A. Blumauer and H. Nagy, *The knowledge graph cookbook: recipes that work*, 1st edition. Wien: edition mono/monochrom, 2020.
- [97] “Introduction to knowledge graphs (part 2): History of knowledge graphs.” Accessed: Mar. 13, 2024. [Online]. Available: <https://realkm.com/2023/03/06/introduction-to-knowledge-graphs-part-2-history-of-knowledge-graphs/>
- [98] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, “DBpedia: A Nucleus for a Web of Open Data,” 2007, pp. 722–735. doi: 10.1007/978-3-540-76298-0_52.
- [99] “The Neo4j Graph Platform – The #1 Platform for Connected Data.” Accessed: Mar. 13, 2024. [Online]. Available: <https://neo4j.com/>
- [100] “AllegroGraph (Official website).” Accessed: Mar. 13, 2024. [Online]. Available: <https://allegrograph.com/>
- [101] “Amazon Neptune (Official website).” Accessed: Mar. 13, 2024. [Online]. Available: <https://aws.amazon.com/neptune/>

- [102] “Google Knowledge Graph Search API.” Accessed: Mar. 13, 2024. [Online]. Available: <https://developers.google.com/knowledge-graph>
- [103] “Microsoft Academic Knowledge Graph (MAKG).” Accessed: Mar. 13, 2024. [Online]. Available: <https://makg.org/>
- [104] A. Meroño-Peñuela, “Web Data APIs for Knowledge Graphs Easing Access to Semantic Data for Application Developers Synthesis Lectures on Data, Semantics, and Knowledge.”
- [105] V. Janev, D. Graux, H. Jabeen, and E. Sallinger, Eds., *Knowledge Graphs and Big Data Processing*, vol. 12072. in Lecture Notes in Computer Science, vol. 12072. Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-030-53199-7.
- [106] D. L. Goodhue, M. D. Wybo, and L. J. Kirsch, “The Impact of Data Integration on the Costs and Benefits of Information Systems,” *MIS Quarterly*, vol. 16, no. 3, p. 293, Sep. 1992, doi: 10.2307/249530.
- [107] A. Doan, A. Halevy, and Z. Ives, *Principles of Data Integration*. Saint Louis: Elsevier Science, 2014.
- [108] *Journal on data semantics*. 4, no. 3730. in Lecture notes in computer science. Berlin Heidelberg: Springer, 2005.
- [109] “Triplestore.” Accessed: Mar. 14, 2024. [Online]. Available: <https://en.wikipedia.org/wiki/Triplestore>
- [110] “What is an RDF Triplestore?” Accessed: Mar. 14, 2024. [Online]. Available: <https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf-triplestore/>
- [111] S. Choudhary, T. Luthra, A. Mittal, and R. Singh, “A Survey of Knowledge Graph Embedding and Their Applications,” 2021, doi: 10.48550/ARXIV.2107.07842.
- [112] Q. Wang, Z. Mao, B. Wang, and L. Guo, “Knowledge Graph Embedding: A Survey of Approaches and Applications,” *IEEE Trans Knowl Data Eng*, vol. 29, no. 12, pp. 2724–2743, Dec. 2017, doi: 10.1109/TKDE.2017.2754499.
- [113] “Introduction to Knowledge Graph Embedding.” Accessed: Mar. 14, 2024. [Online]. Available: <https://dglke.dgl.ai/doc/kg.html>
- [114] “Knowledge Extraction.” Accessed: Dec. 24, 2023. [Online]. Available: <https://botpenguin.com/glossary/knowledge-extraction>
- [115] G. Alor-Hernández, J. L. Sánchez-Cervantes, A. Rodríguez-González, and R. Valencia-García, Eds., *Current Trends in Semantic Web Technologies: Theory and Practice*, vol. 815. Cham: Springer International Publishing, 2019. doi: 10.1007/978-3-030-06149-4.
- [116] A. Holzinger, P. Kieseberg, A. M. Tjoa, and E. Weippl, Eds., *Machine Learning and Knowledge Extraction*, vol. 12279. Cham: Springer International Publishing, 2020. doi: 10.1007/978-3-030-57321-8.

- [117] I. Herman, G. Melancon, and M. S. Marshall, “Graph visualization and navigation in information visualization: A survey,” *IEEE Trans Vis Comput Graph*, vol. 6, no. 1, pp. 24–43, 2000, doi: 10.1109/2945.841119.
- [118] J. J. van Wijk, “The Value of Visualization,” in *VIS 05. IEEE Visualization, 2005.*, IEEE, pp. 79–86. doi: 10.1109/VISUAL.2005.1532781.
- [119] “Visualizing Node-Link Graphs.” Accessed: Mar. 16, 2024. [Online]. Available: <https://medium.com/kineviz-blog/visualizing-node-link-graphs-84a40a9b2fcc>
- [120] “Adjacency Matrix & List | Overview, Graphs & Examples.” Accessed: Mar. 16, 2024. [Online]. Available: <https://study.com/academy/lesson/adjacency-representations-of-graphs-in-discrete-math.html>
- [121] M. W. Gonzalez and M. G. Kann, “Chapter 4: Protein Interactions and Disease,” *PLoS Comput Biol*, vol. 8, no. 12, p. e1002819, Dec. 2012, doi: 10.1371/journal.pcbi.1002819.
- [122] G. A. Pavlopoulos, A.-L. Wegener, and R. Schneider, “A survey of visualization tools for biological network analysis,” *BioData Min*, vol. 1, no. 1, p. 12, Dec. 2008, doi: 10.1186/1756-0381-1-12.
- [123] F. Auer, S. Mayer, and F. Kramer, “Data-dependent visualization of biological networks in the web-browser with NDEdit,” *PLoS Comput Biol*, vol. 18, no. 6, p. e1010205, Jun. 2022, doi: 10.1371/journal.pcbi.1010205.
- [124] “What_Are_Knowledge_Graph_Data_Models.” Accessed: Dec. 24, 2023. [Online]. Available: https://github.com/JunyiTao/CS520-Knowledge-Graph-Notes-and-Projects/blob/main/Annotated%20Notes/W2.%20What%20are%20Graph%20Data%20Models__Annotated.pdf
- [125] M. Kejriwal, *Domain-Specific Knowledge Graph Construction*. Cham: Springer International Publishing, 2019. doi: 10.1007/978-3-030-12375-8.
- [126] M. Destandau and J.-D. Fekete, “The missing path: Analysing incompleteness in knowledge graphs,” *Inf Vis*, vol. 20, no. 1, pp. 66–82, Jan. 2021, doi: 10.1177/1473871621991539.
- [127] A. Hogan *et al.*, “Knowledge Graphs,” 2020, doi: 10.48550/ARXIV.2003.02320.
- [128] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*, Fourth Edition. in Pearson Series in Artificial Intelligence. Hoboken, NJ: Pearson, 2021.
- [129] H. Ma and D. Z. Wang, “A Survey On Few-shot Knowledge Graph Completion with Structural and Commonsense Knowledge,” 2023, doi: 10.48550/ARXIV.2301.01172.
- [130] M. Zamini, H. Reza, and M. Rabiei, “A Review of Knowledge Graph Completion,” *Information*, vol. 13, no. 8, p. 396, Aug. 2022, doi: 10.3390/info13080396.

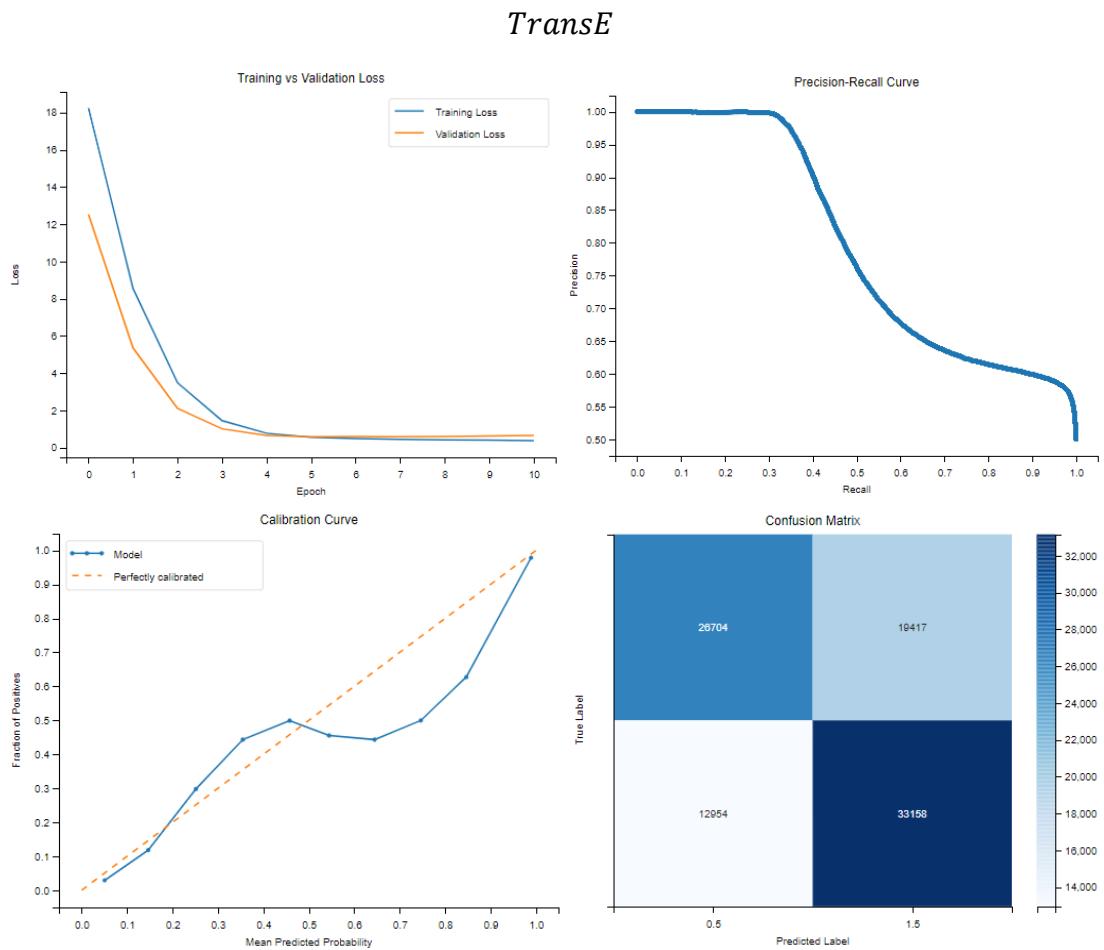
- [131] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classification tasks,” *Inf Process Manag*, vol. 45, no. 4, pp. 427–437, Jul. 2009, doi: 10.1016/j.ipm.2009.03.002.
- [132] “Graph Representation Learning.” Accessed: May 05, 2024. [Online]. Available: <https://wandb.ai/syllogismos/machine-learning-with-graphs/reports/7-Graph-Representation-Learning--VmlldzozNzcwMDk>
- [133] M. Y. Jaradeh, K. Singh, M. Stocker, and S. Auer, “Triple Classification for Scholarly Knowledge Graph Completion,” 2021, doi: 10.48550/ARXIV.2111.11845.
- [134] B. Subagdja, D. Shanthoshigaa, Z. Wang, and A.-H. Tan, “Machine Learning for Refining Knowledge Graphs: A Survey,” *ACM Comput Surv*, vol. 56, no. 6, pp. 1–38, Jun. 2024, doi: 10.1145/3640313.
- [135] “Python 3.12.3 documentation.” Accessed: May 09, 2024. [Online]. Available: <https://docs.python.org/3.12/>
- [136] “Python (programming language).” Accessed: May 09, 2024. [Online]. Available: [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))
- [137] Z. Shaw, *Learn Python 3 the hard way: a very simple introduction to the terrifyingly beautiful world of computers and code*. in Zed Shaw’s hard way series. Boston: Addison-Wesley, 2017.
- [138] L. Jean-Baptiste, *Ontologies with Python: Programming OWL 2.0 Ontologies with Python and Owlready2*. Berkeley, CA: Apress, 2021.
- [139] “GraphDB in Action: Powering State-of-the-Art Research.” Accessed: May 09, 2024. [Online]. Available: <https://ontotext.medium.com/graphdb-in-action-powering-state-of-the-art-research-85f181c26180>
- [140] “About GraphDB.” Accessed: May 09, 2024. [Online]. Available: <https://graphdb.ontotext.com/documentation/10.2/about-graphdb.html>
- [141] “GraphDB.” Accessed: May 07, 2024. [Online]. Available: <https://graphdb.ontotext.com/>
- [142] P. Singh and A. Manure, *Learn TensorFlow 2.0: implement machine learning and deep learning models with Python*. Berkeley, CA: Apress, 2020.
- [143] T. Hope, Y. S. Resheff, and I. Lieder, *Learning TensorFlow: a guide to building deep learning systems*, First edition. Sebastopol, CA: O’Reilly Media, 2017.
- [144] T. Tudorache, C. Nyulas, N. F. Noy, and M. A. Musen, “WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the Web,” *Semant Web*, vol. 4, no. 1, pp. 89–99, 2013, doi: 10.3233/SW-2012-0057.
- [145] M. A. Musen, “The protégé project,” *AI Matters*, vol. 1, no. 4, pp. 4–12, Jun. 2015, doi: 10.1145/2757001.2757003.

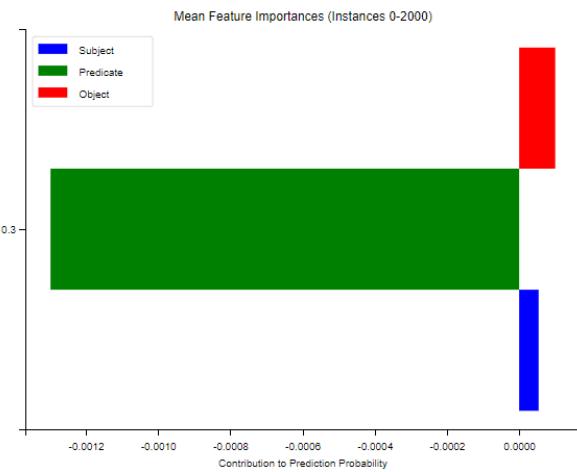
Appendix A

Detailed Experiments Results

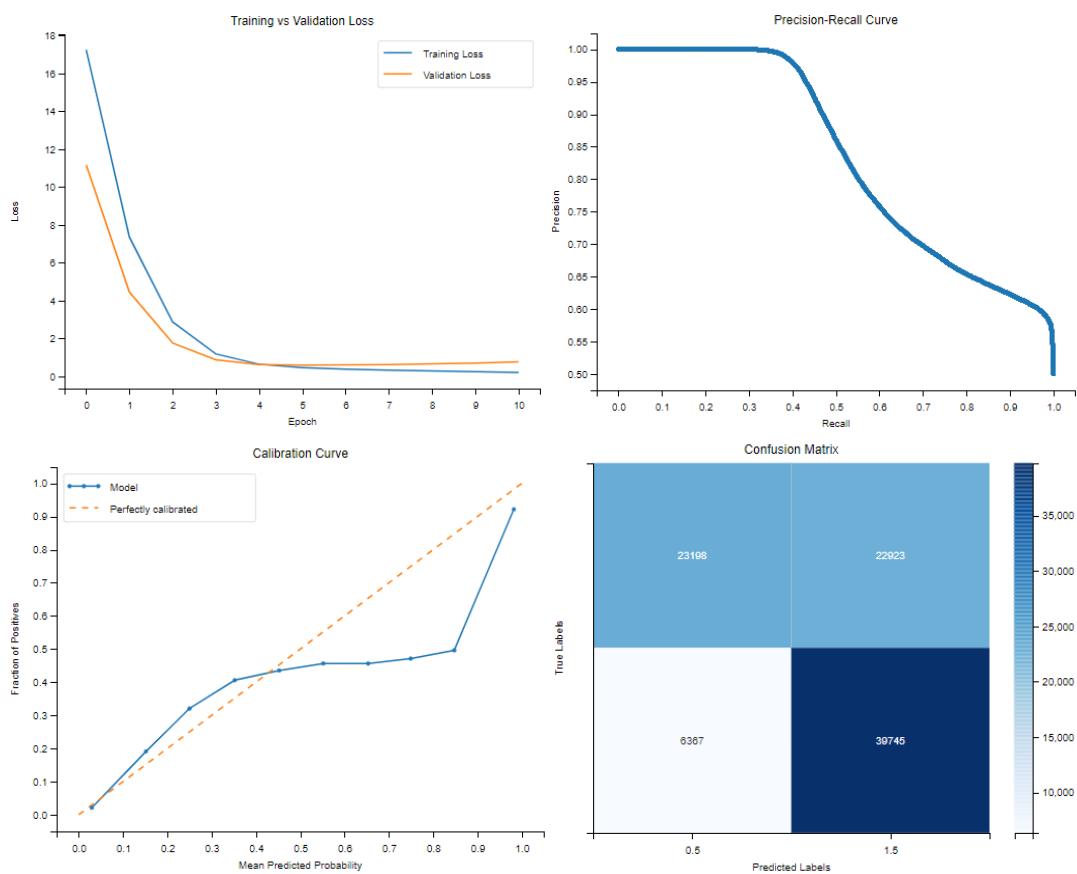
I. Group 1

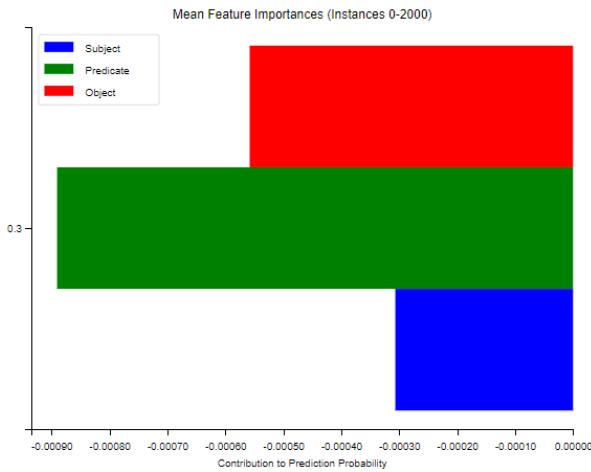
	Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
<i>TransE</i>	0.8020	0.2544	3707.7435
<i>TransH</i>	0.8028	0.2540	3599.7435
<i>RotateE</i>	0.8066	0.2497	4204.4276
<i>HoIE</i>	0.8120	0.2425	3590.5345
<i>DistMult</i>	0.7868	0.2827	3123.9431
<i>ComplEx</i>	0.8059	0.2507	3958.5715



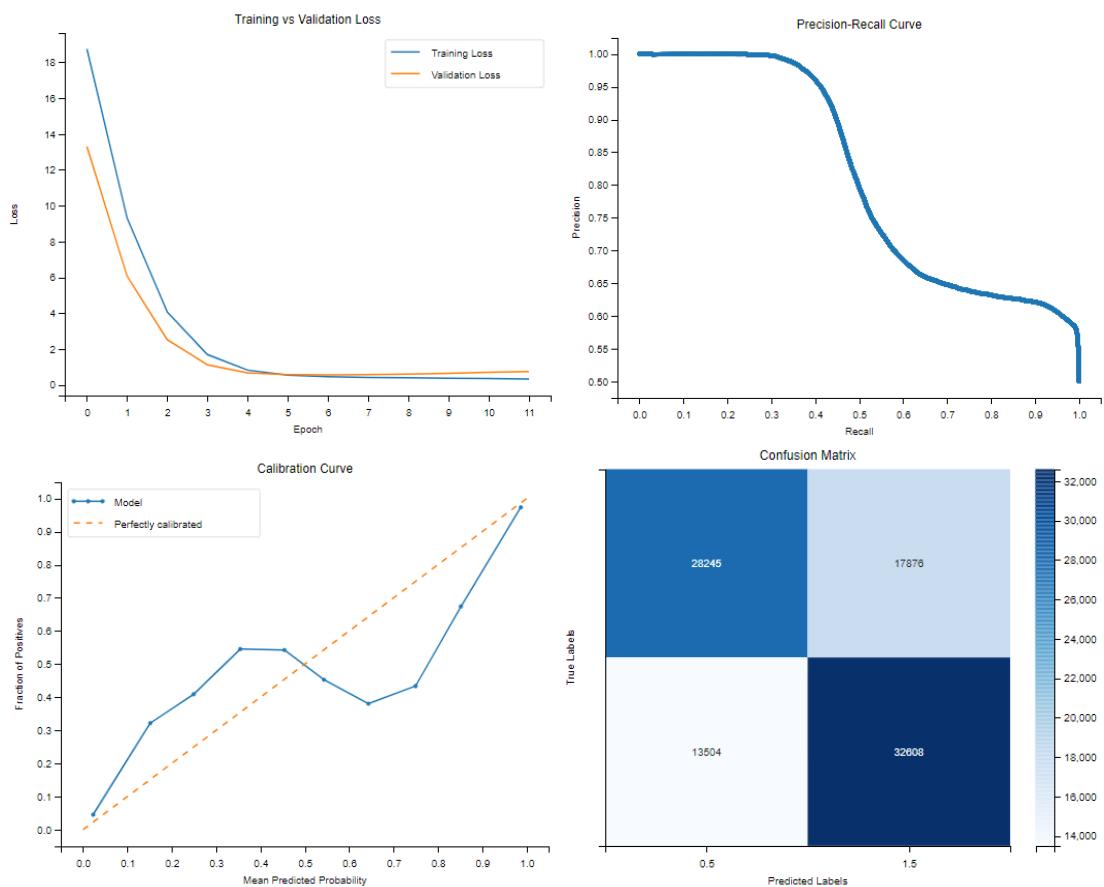


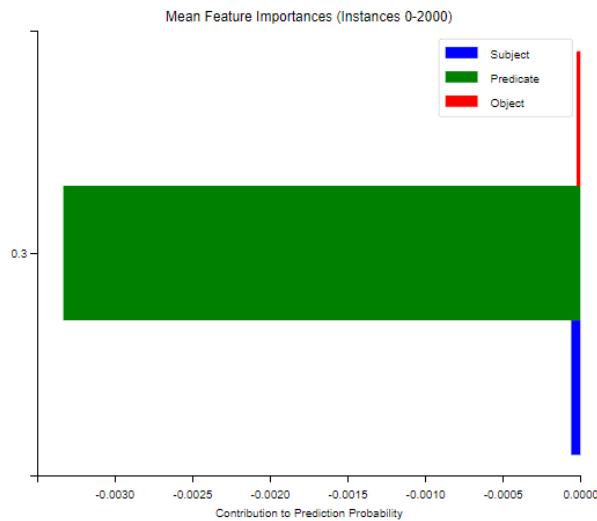
TransH



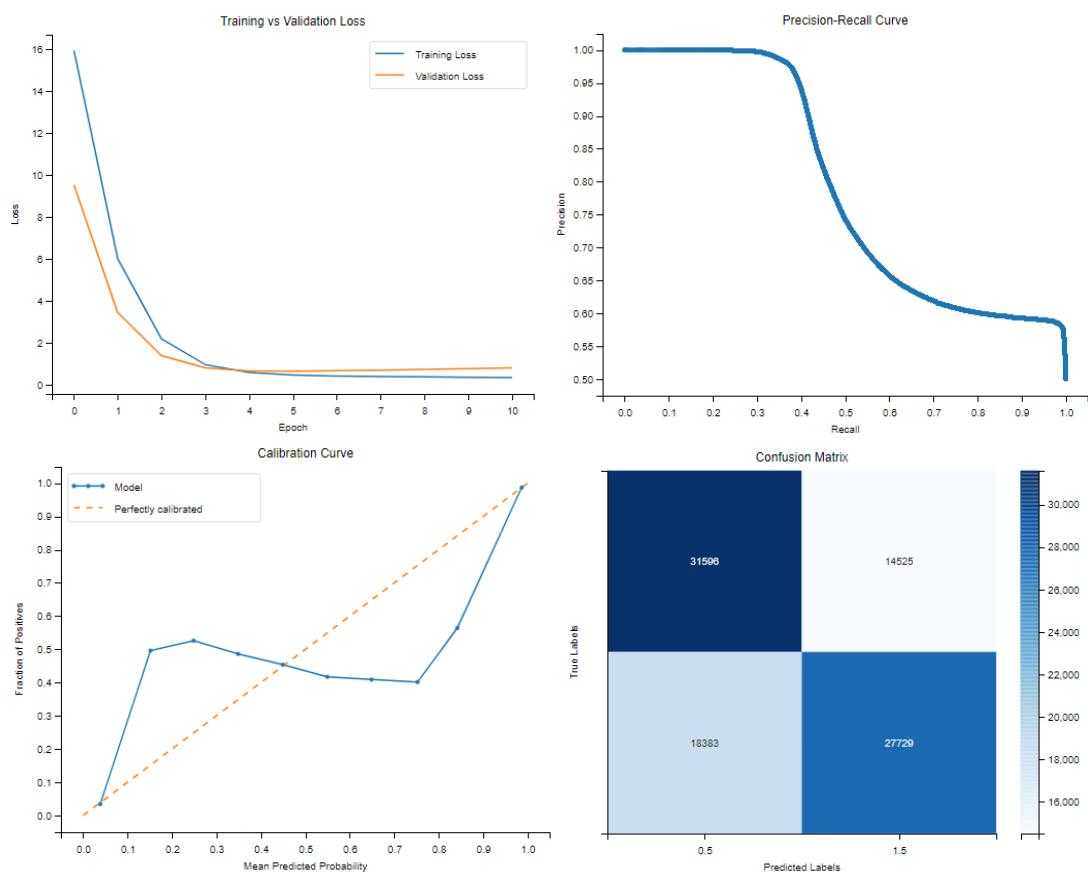


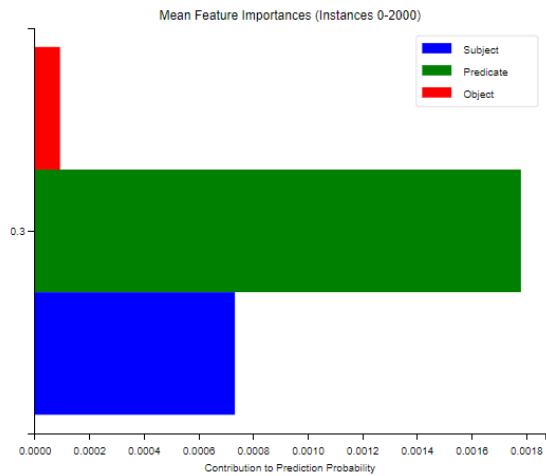
RotateE



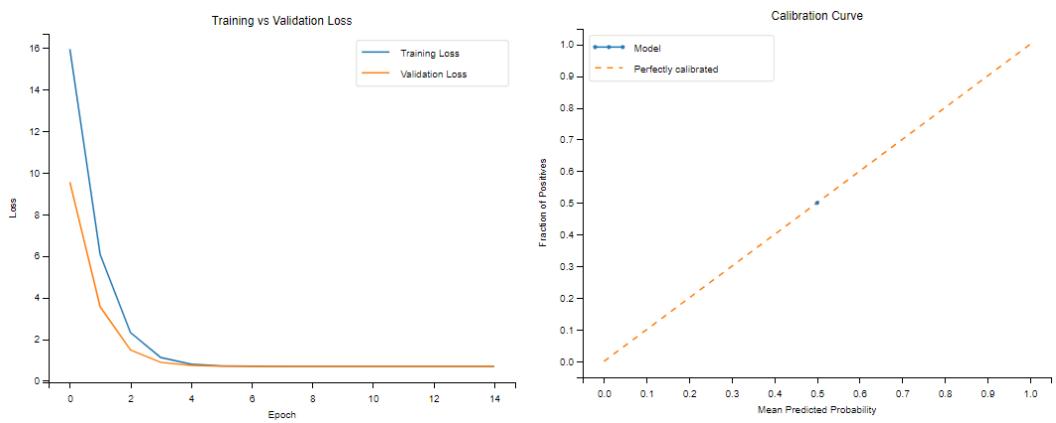


HoIE

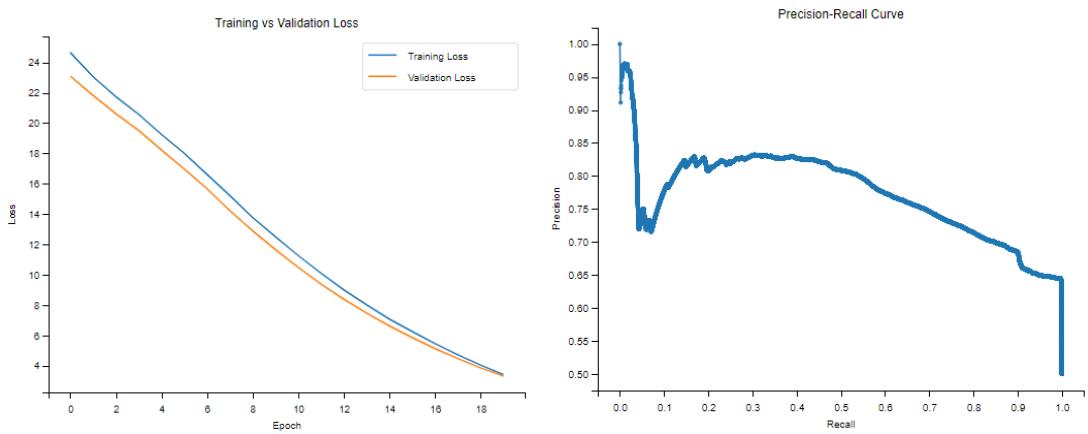


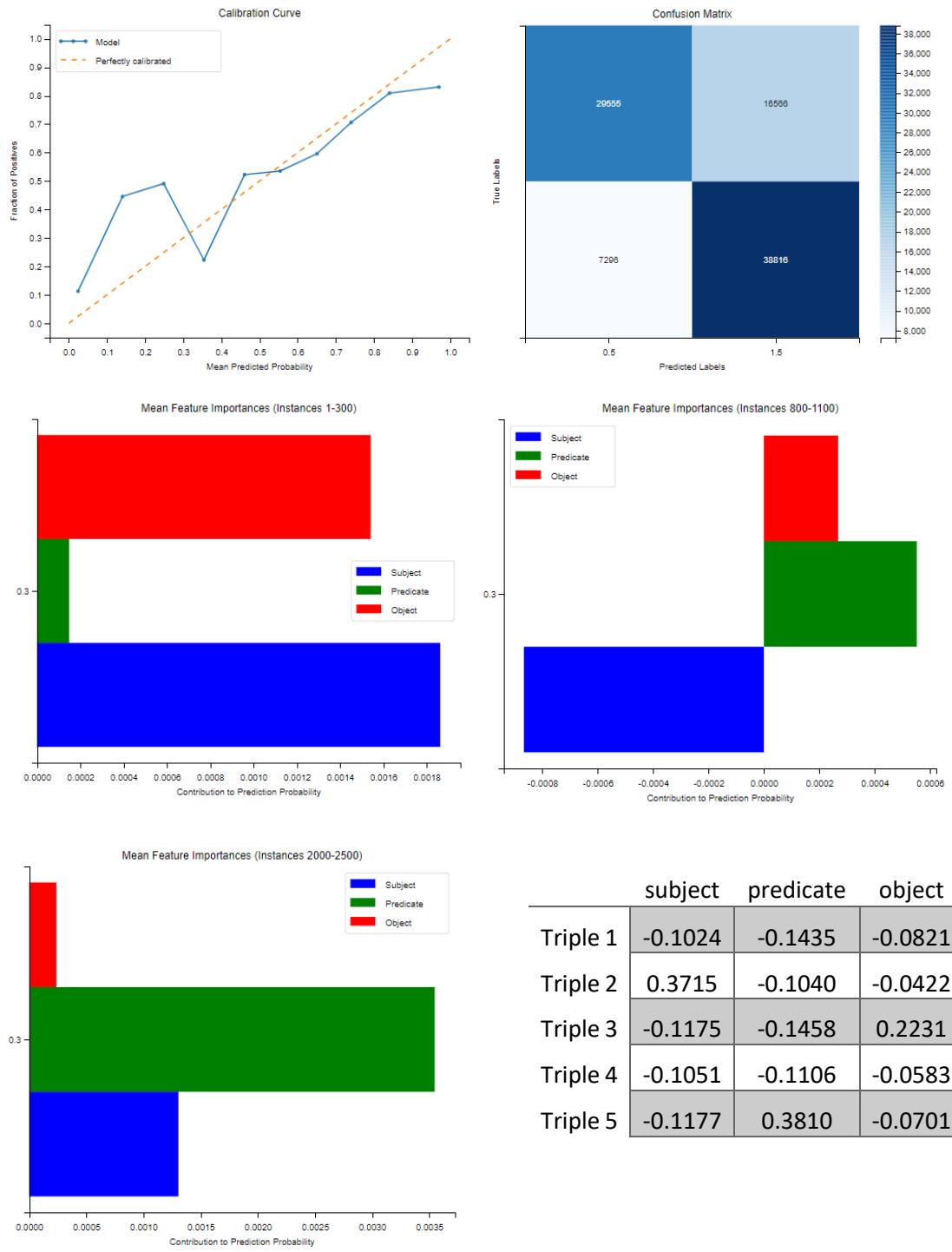


DistMult



ComplEx





	C1	C2	C3	C4	C5	C6	C7	C8
<i>TransE</i>	9	375	61	7	0.4642	0.9959	0.0077	0.2282
<i>TransH</i>	29	698	118	16	0.5148	0.9999	0.0001	0.3045
<i>RotateE</i>	22	291	11	5	0.3690	0.9478	0.0002	0.2509
<i>Hole</i>	5	291	0	0	0.3333	0.8631	0.0028	0.2605
<i>DistMult</i>	1	0	0	0	0.5004	0.5004	0.5004	0
<i>ComplEx</i>	13	281	2	2	0.3154	0.9057	0.0001	0.2514

TransE

1. **Top 5 most frequent relations:**

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 436),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/context', 402),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 374),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/keyword', 121),
 ('http://www.w3.org/1999/02/22-rdf-syntax-ns#rest', 98)]

2. **Relations appearing only once:**

['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference']

3. **Relation with highest average probability:**

('http://www.w3.org/2000/01/rdf-schema#subClassOf', 0.5279035317046302)

4. **Relation with lowest average probability:**

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference',
 0.38319316506385803)

TransH

1. **Top 5 most frequent relations:**

[('http://www.w3.org/2002/07/owl#imports', 852),
 ('http://www.w3.org/2002/07/owl#equivalentClass', 374),
 ('http://www.w3.org/1999/02/22-rdf-syntax-ns#first', 40),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 22),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasSubTheme', 20)]

2. **Relations appearing only once:**

['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode',
 'http://www.w3.org/2002/07/owl#equivalentProperty',
 'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/definition']

3. **Relation with highest average probability:**

('http://www.w3.org/2000/01/rdf-schema#range', 0.7395107001066208)

4. **Relation with lowest average probability:**

('http://www.w3.org/2002/07/owl#equivalentProperty', 0.1863100230693817)

RotateE

1. **Top 5 most frequent relations:**

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 364),

- (<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI>', 192),
 (<http://www.w3.org/2000/01/rdf-schema#subClassOf>', 167),
 (<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/definition>', 128),
 (<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/remark>', 124)]
- 2. Relations appearing only once:**
 [[https://ec.europa.eu/eurostat/NLP4StatRef/ontology\(dataSource](https://ec.europa.eu/eurostat/NLP4StatRef/ontology(dataSource)',
 '<http://www.w3.org/2000/01/rdf-schema#label>']
- 3. Relation with highest average probability:**
 ([https://ec.europa.eu/eurostat/NLP4StatRef/ontology\(dataSource](https://ec.europa.eu/eurostat/NLP4StatRef/ontology(dataSource)',
 0.5118739008903503)
- 4. Relation with lowest average probability:**
 ([https://ec.europa.eu/eurostat/NLP4StatRef/ontology\(keyword](https://ec.europa.eu/eurostat/NLP4StatRef/ontology(keyword)',
 0.24597927373006792)

HoIE

- 1. Top 5 most frequent relations:**
 [<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference>', 1233),
 (<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph>', 226),
 (<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink>', 29),
 (<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/context>', 11),
 (<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>', 1)]
- 2. Relations appearing only once:**
 [<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>']
- 3. Relation with highest average probability:**
 (<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink>',
 0.41237134858965874)
- 4. Relation with lowest average probability:**
 (<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>', 0.1697981059551239)

DistMult

- 1. Top 5 most frequent relations:**
 [<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm>', 1500)]
- 2. Relation with highest average probability:**
 (<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm>',
 0.5004311203956604)
- 3. Relation with lowest average probability:**
 (<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm>',
 0.5004311203956604)

ComplEx

1. Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph', 458),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/content', 427),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink', 244),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/title', 80),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 51)]
```

2. Relations appearing only once:

```
['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasOECDTheme']
```

3. Relation with highest average probability:

```
('http://www.w3.org/2000/01/rdf-schema#label', 0.4366015885025263)
```

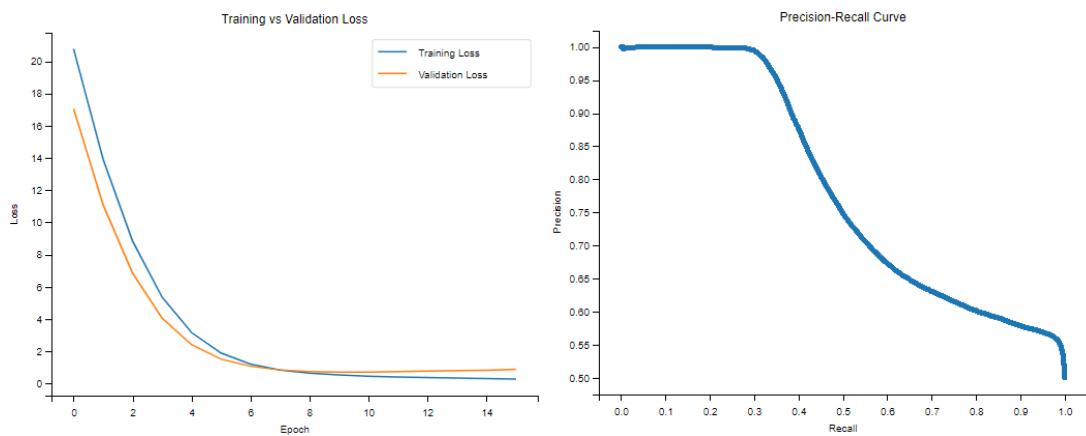
4. Relation with lowest average probability:

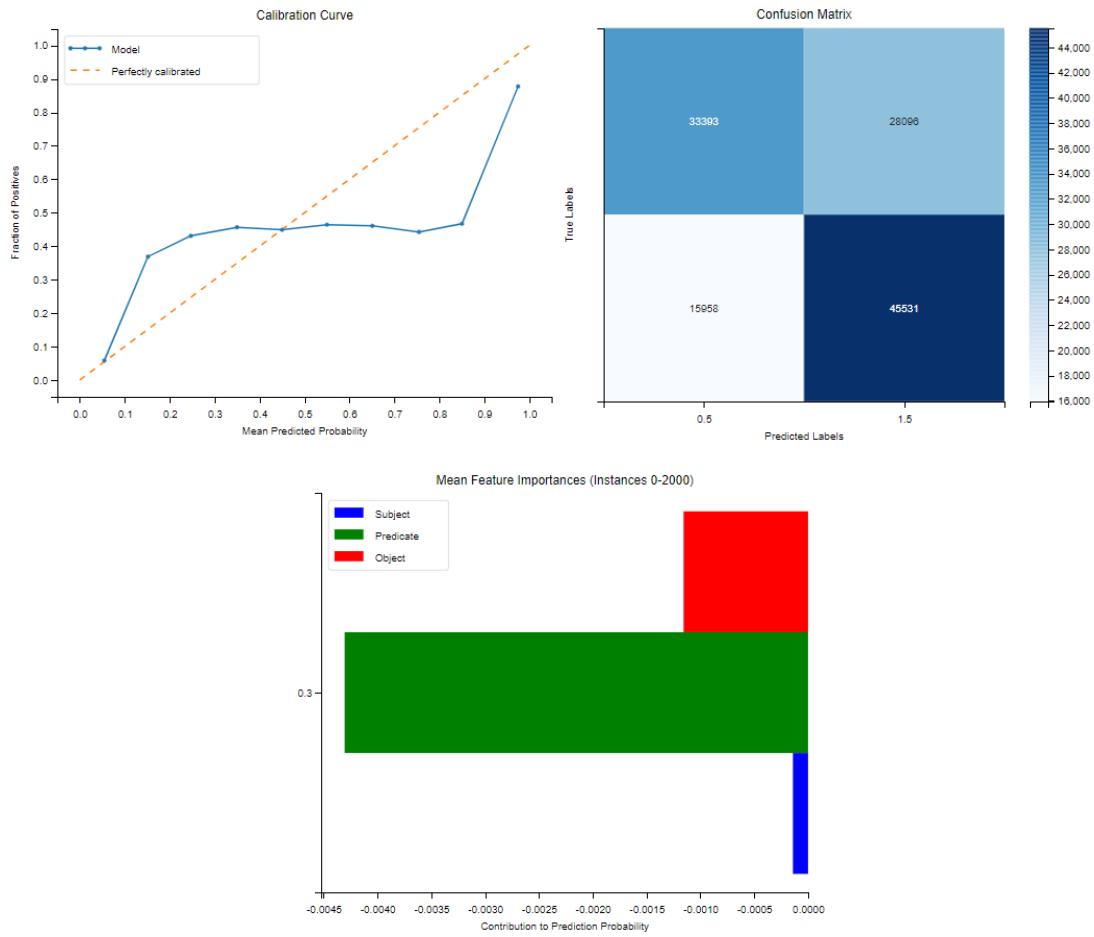
```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasOECDTheme',
 0.10760148614645004)
```

II. Group 2

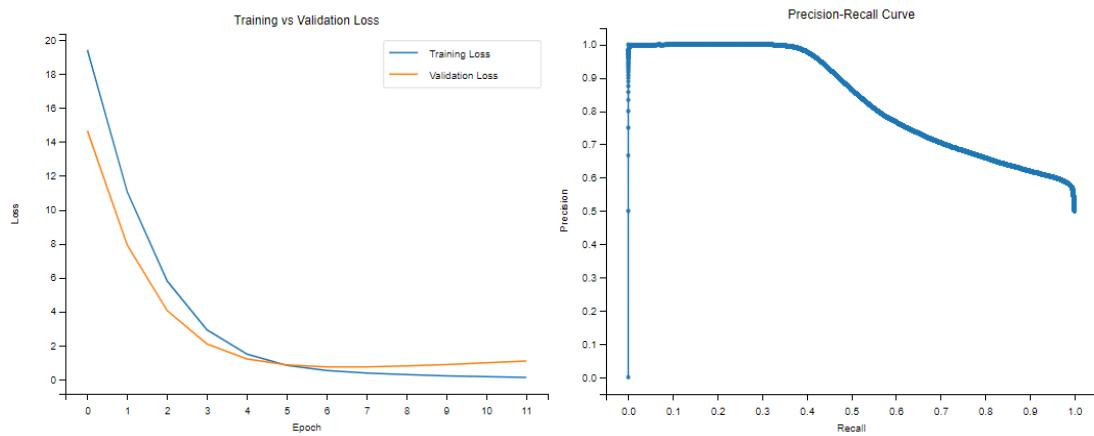
	Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
<i>TransE</i>	0.8095	0.2452	3767.3763
<i>TransH</i>	0.8076	0.2466	3366.1078
<i>RotateE</i>	0.8057	0.2484	3442.7084
<i>HoIE</i>	0.8015	0.2531	3332.1427
<i>DistMult</i>	0.7867	0.2825	3122.6344
<i>ComplEx</i>	0.8137	0.2384	3917.0213

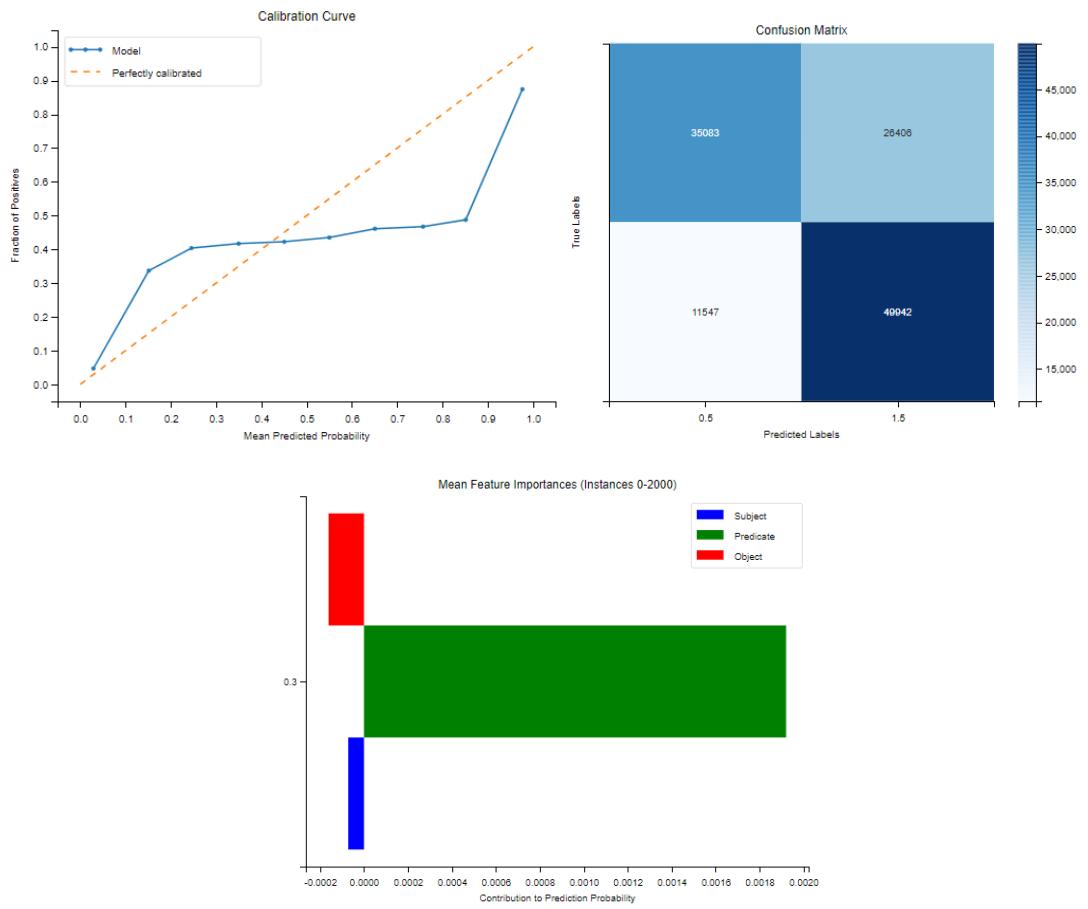
TransE



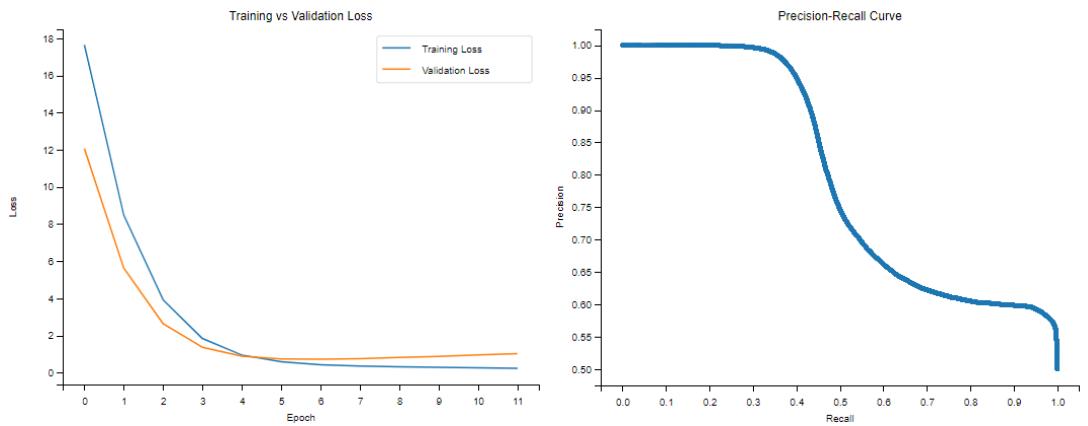


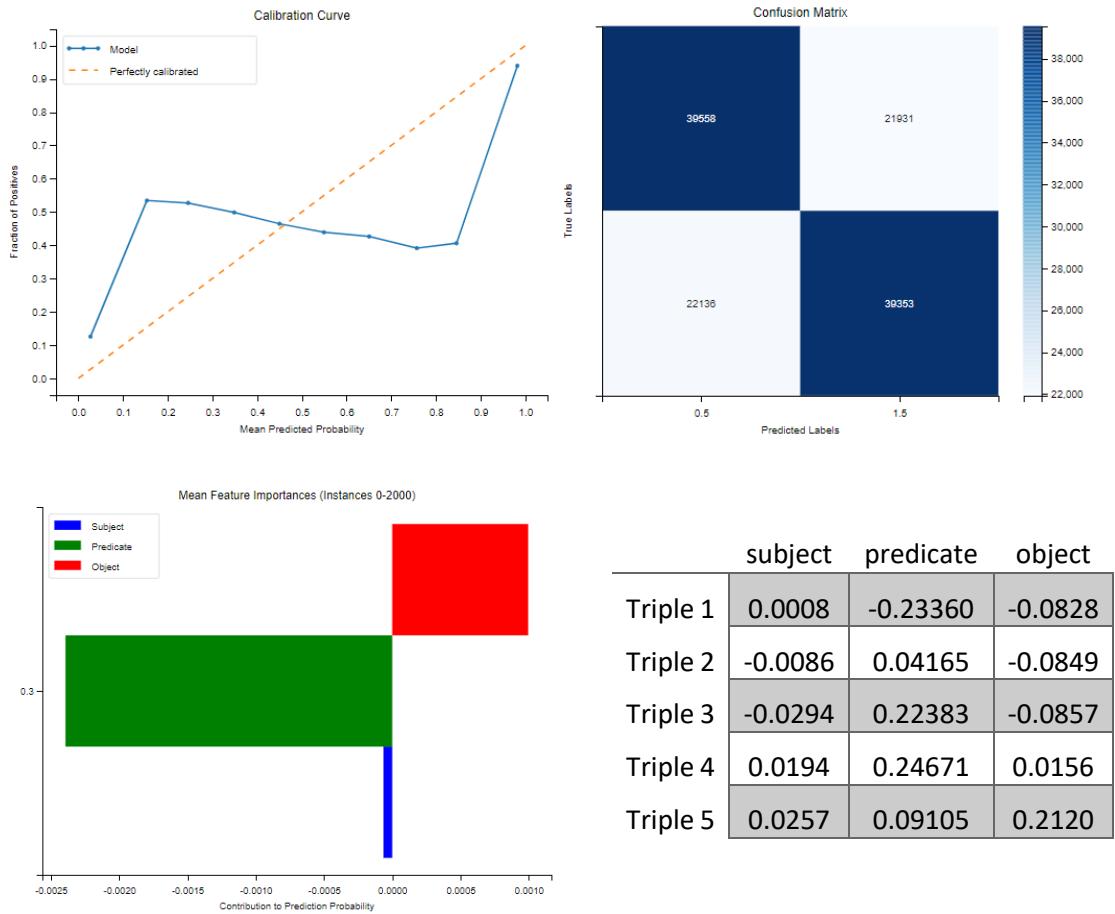
TransH



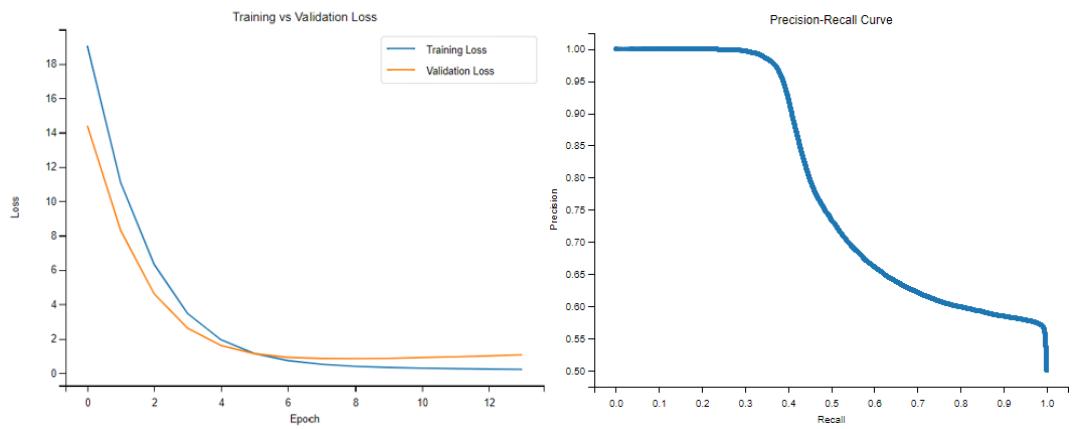


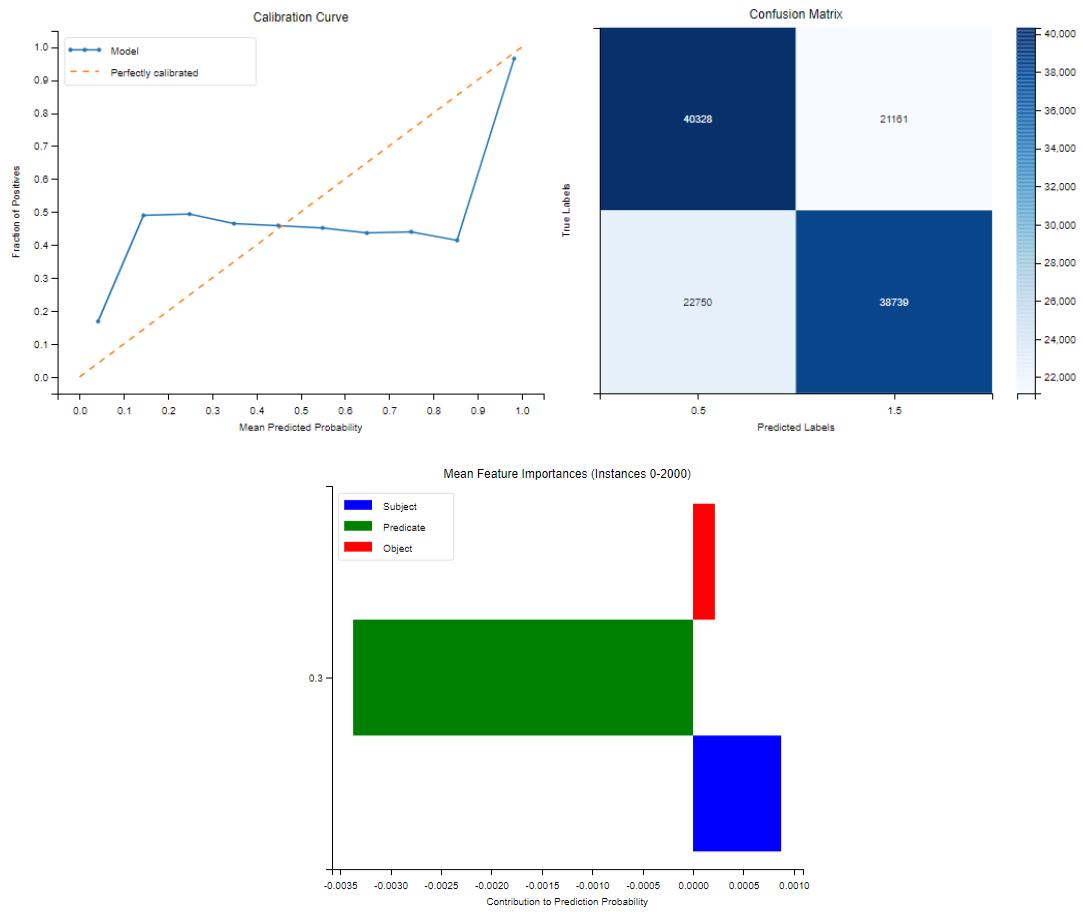
RotateE



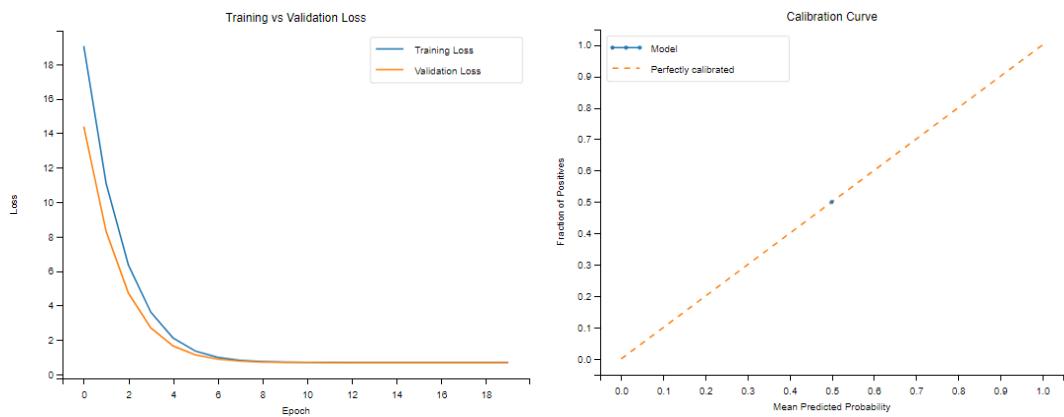


HoIE

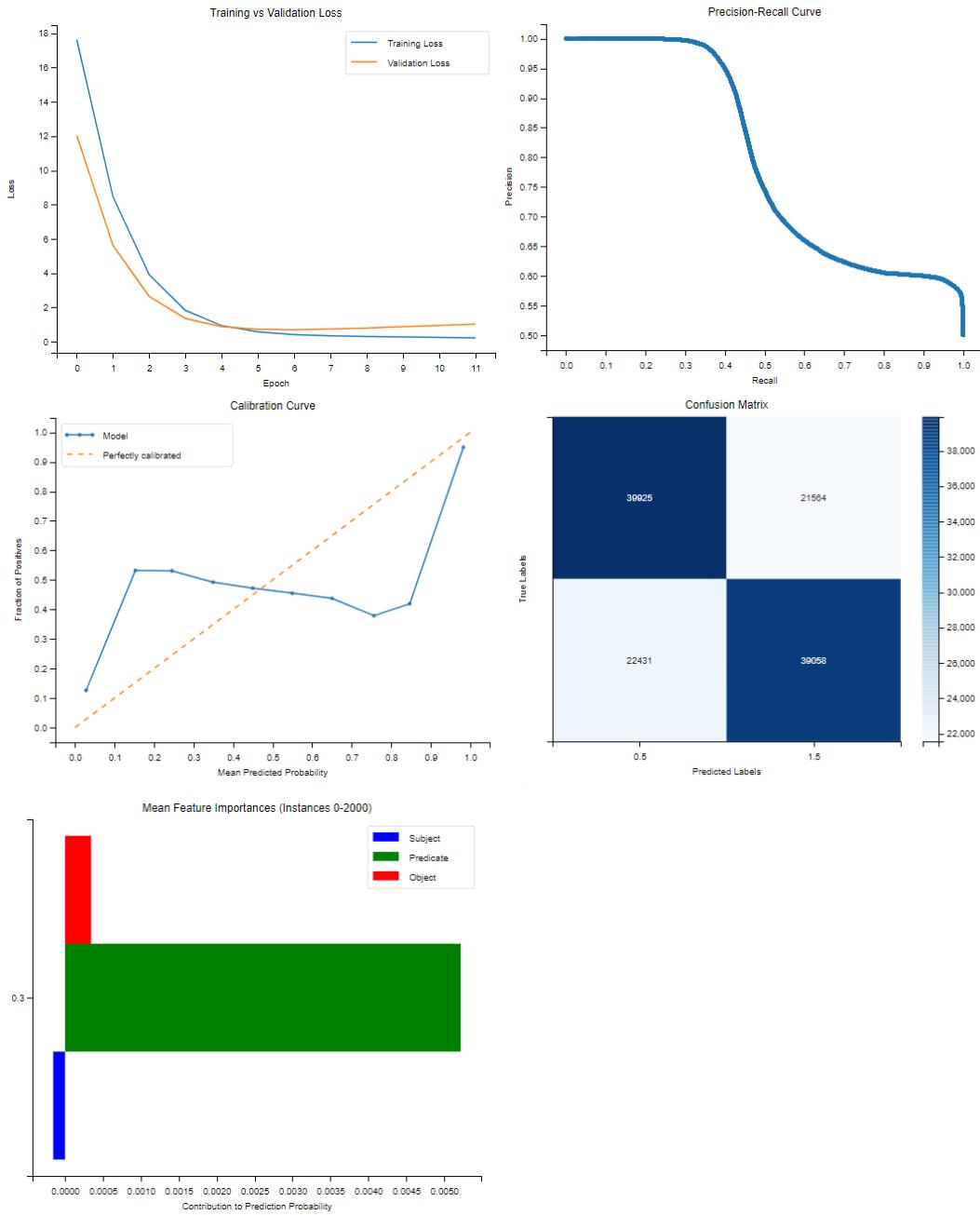




DistMult



ComplEx



	C1	C2	C3	C4	C5	C6	C7	C8
<i>TransE</i>	4	573	42	3	0.4451	0.9463	0.0002	0.3022
<i>TransH</i>	1	638	210	1	0.4824	1.0000	0.0001	0.3560
<i>Rotate</i>	4	298	25	2	0.2966	0.9991	0.0001	0.2928
<i>HoIE</i>	1	355	7	1	0.3524	0.9153	0.0009	0.2804
<i>DistMult</i>	1	0	0	0	0.5	0.5	0.5	0
<i>ComplEx</i>	14	484	48	6	0.4121	0.9856	0.0001	0.2973

TransE

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 796),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph', 674),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/content', 26),
('http://www.w3.org/2002/07/owl#unionOf', 4)]

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 0.447425178314189)

Relation with lowest average probability:

('http://www.w3.org/2002/07/owl#unionOf', 0.4008448729291558)

TransH

Top 5 most frequent relations:

[('http://www.w3.org/2002/07/owl#equivalentClass', 1421)]

Relation with highest average probability:

('http://www.w3.org/2002/07/owl#equivalentClass', 0.4828190933904544)

Relation with lowest average probability:

('http://www.w3.org/2002/07/owl#equivalentClass', 0.4828190933904544)

RotatE

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 1253),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 204),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 3),
('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 3)]

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference',
0.4204809255897999)

Relation with lowest average probability:

('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 0.13876665631930032)

HoIE

Top 5 most frequent relations:

[('http://www.w3.org/2002/07/owl#versionInfo', 1500)]

Relation with highest average probability:

('http://www.w3.org/2002/07/owl#versionInfo', 0.3523507455016952)

Relation with lowest average probability:

('http://www.w3.org/2002/07/owl#versionInfo', 0.3523507455016952)

ComplEx

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/definition', 862),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/title', 216),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 145),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/id', 85),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL', 69)]

Relations appearing only once:

['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/content']

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/content', 0.571976363658905)

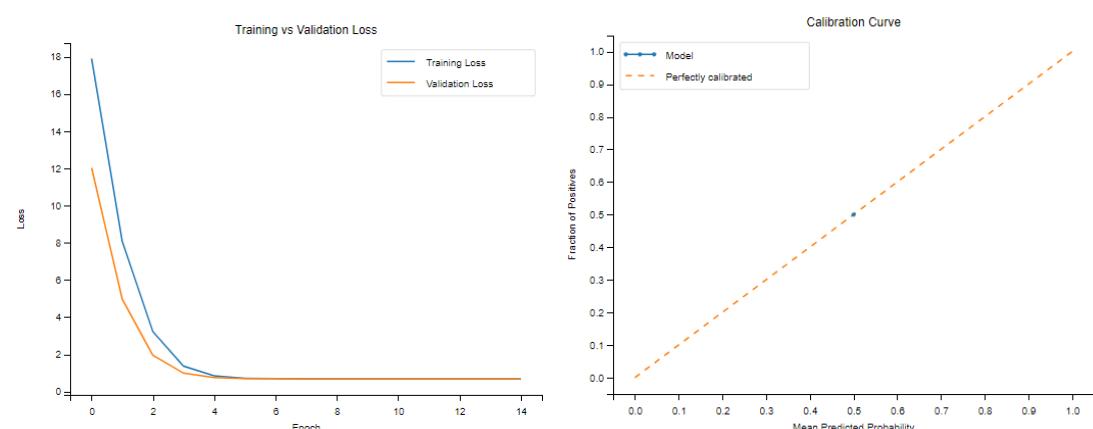
Relation with lowest average probability:

('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 0.2250628693960607)

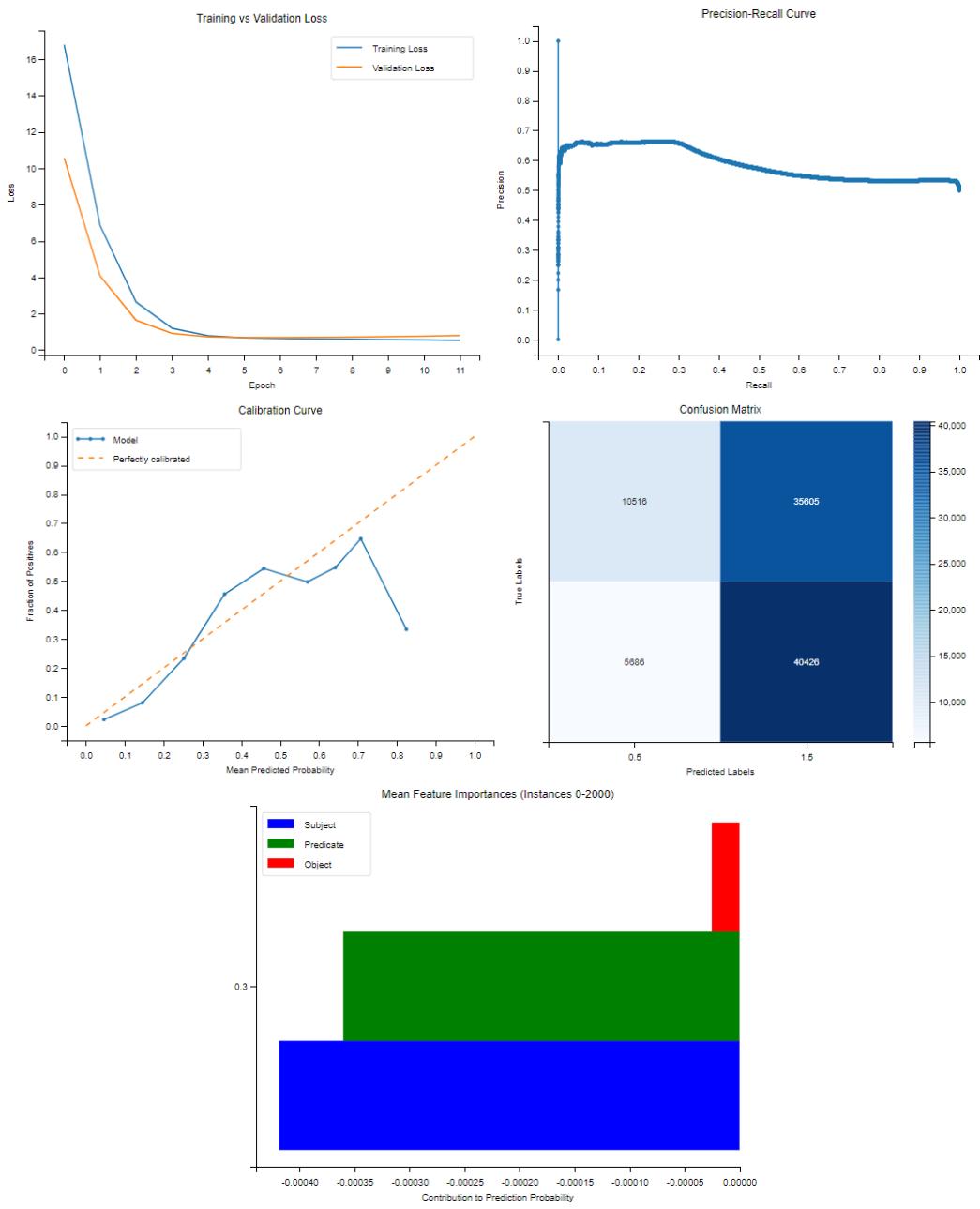
III. Group 3

	Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
<i>TransE</i>	0.7858	0.2831	3103.4511
<i>TransH</i>	0.7958	0.2625	3416.8373
<i>RotateE</i>	0.8122	0.2410	3699.4227
<i>Hole</i>	0.8115	0.2410	4423.7205
<i>DistMult</i>	0.8144	0.2384	4125.0630
<i>ComplEx</i>	0.7833	0.2871	3062.0203

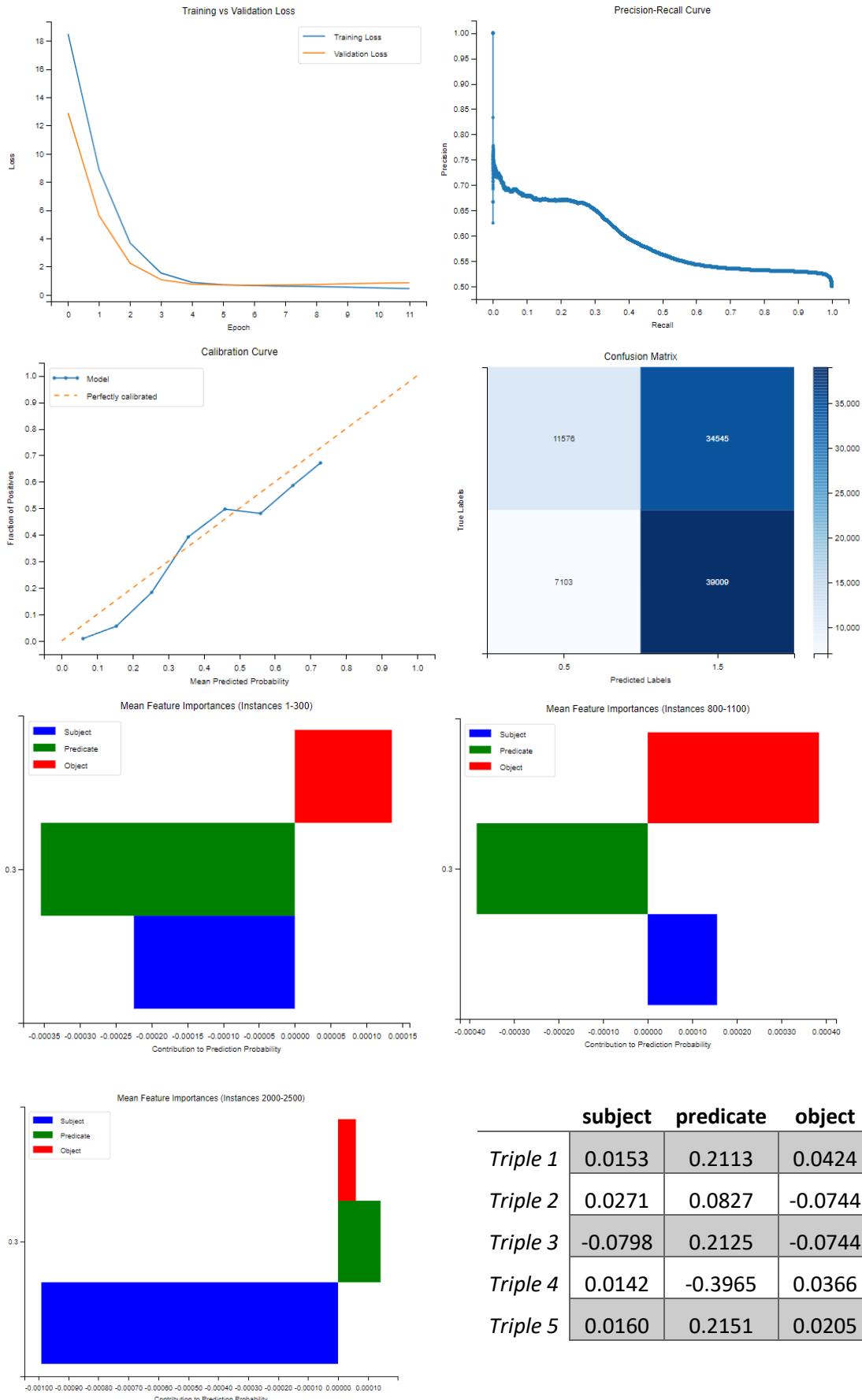
TransE



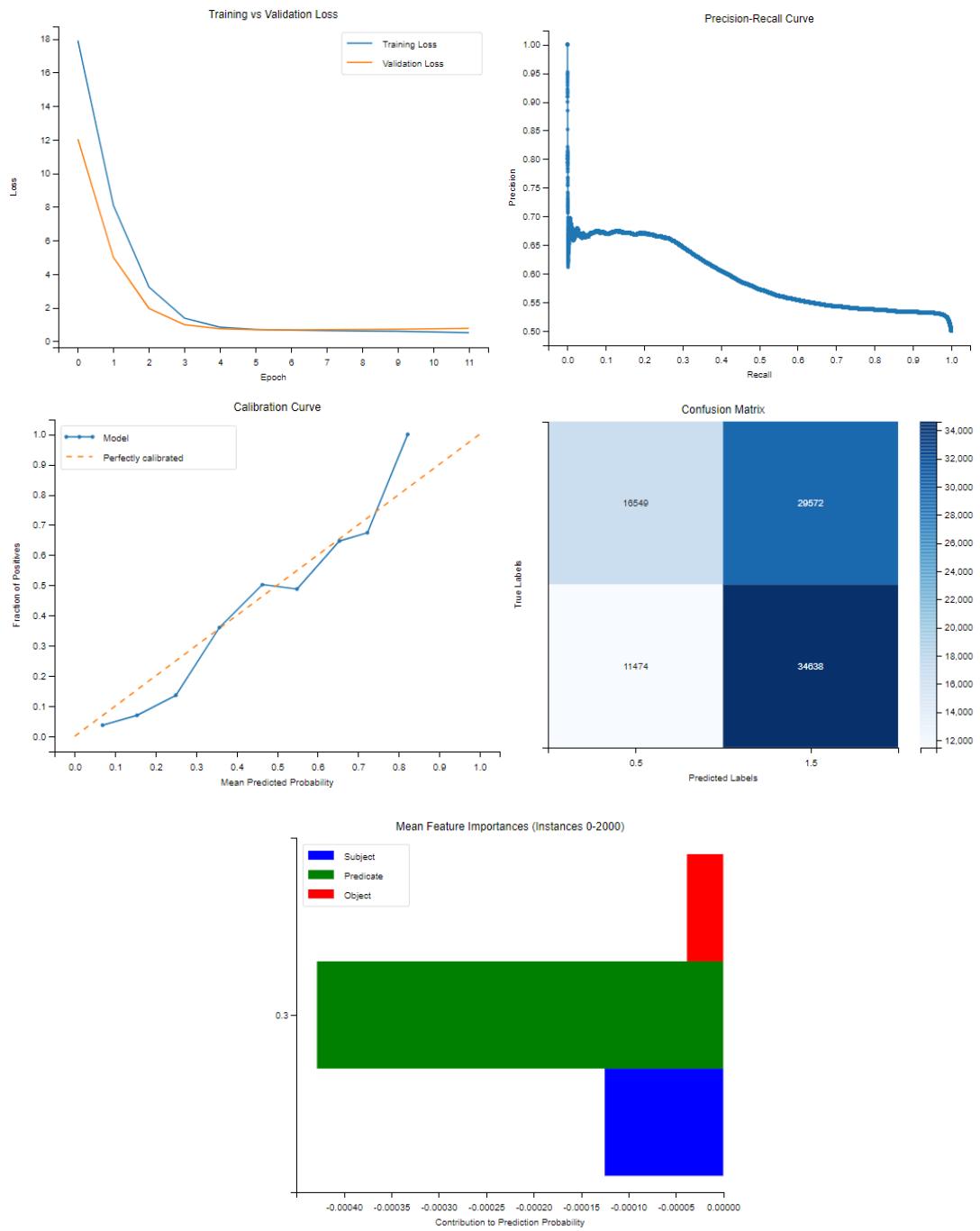
TransH



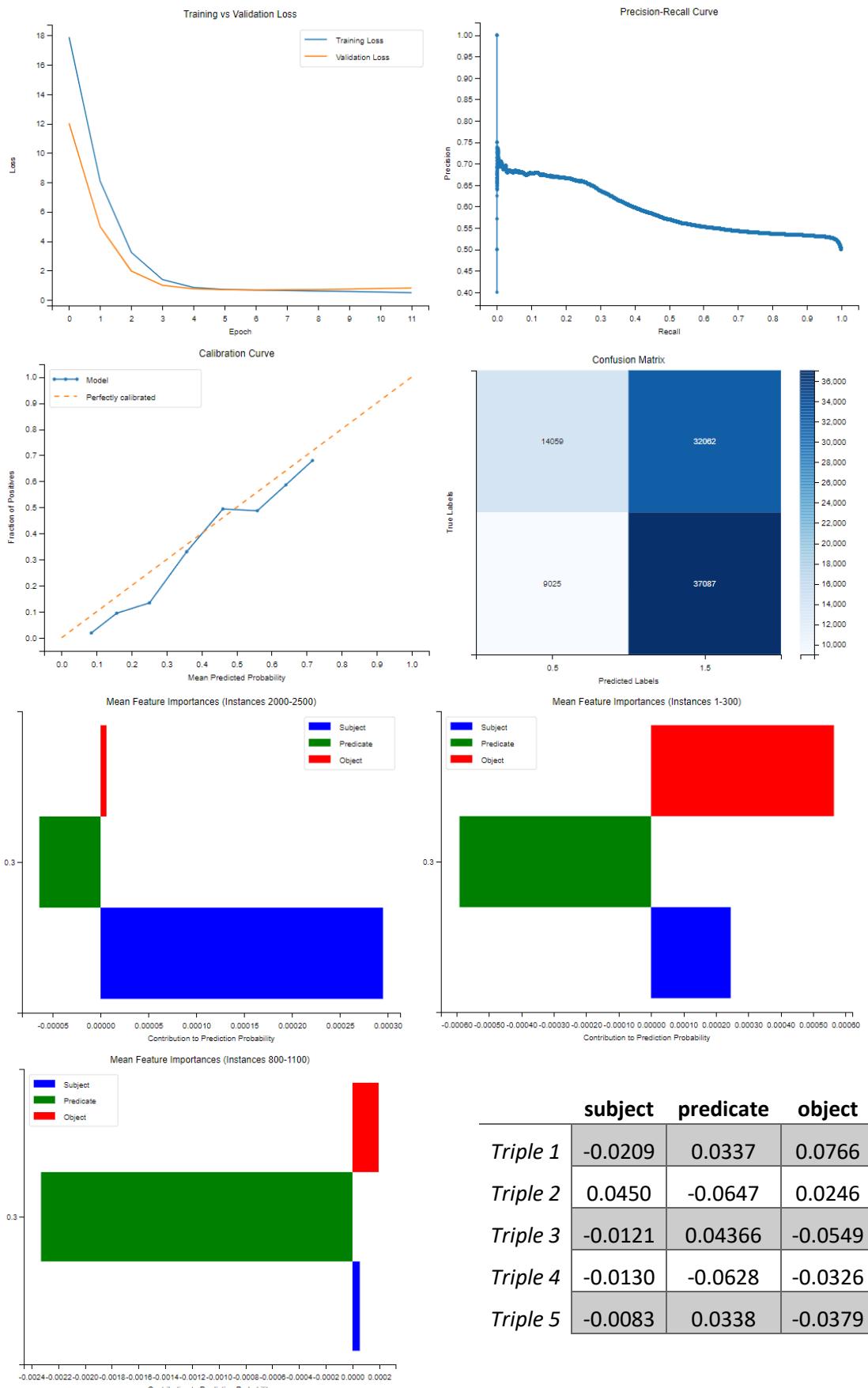
RotateE



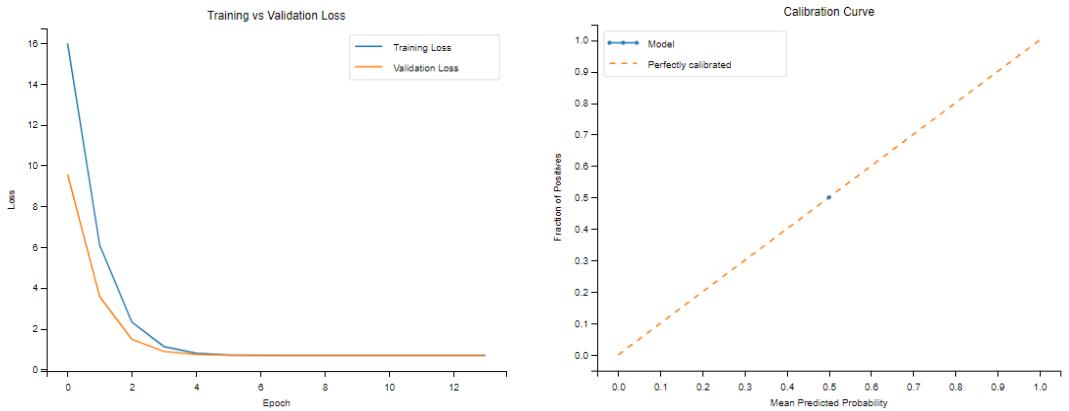
HoIE



DistMult



ComplEx



	C1	C2	C3	C4	C5	C6	C7	C8
<i>TransE</i>	1	0	0	0	0.5003	0.5003	0.5003	0
<i>TransH</i>	8	576	0	0	0.4927	0.8640	0.0001	0.1885
<i>RotatE</i>	4	34	0	0	0.3859	0.6597	0.0044	0.1988
<i>HoIE</i>	39	33	0	0	0.4725	0.6735	0.0319	0.1037
<i>DistMult</i>	25	75	0	0	0.5022	0.6872	0.0480	0.0880
<i>ComplEx</i>	1	0	0	0	0.5003	0.5003	0.5003	0

TransH

Top 5 most frequent relations:

```
[('http://www.w3.org/1999/02/22-rdf-syntax-ns#rest', 816),
 ('http://www.w3.org/1999/02/22-rdf-syntax-ns#first', 620),
 ('http://www.w3.org/2002/07/owl#unionOf', 34),
 ('http://www.w3.org/2002/07/owl#versionInfo', 13),
 ('http://www.w3.org/2000/01/rdf-schema#subPropertyOf', 5)]
```

Relations appearing only once:

```
['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL']
```

Relation with highest average probability:

```
('http://www.w3.org/2000/01/rdf-schema#subPropertyOf', 0.5982378005981446)
```

Relation with lowest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasOECDTheme',
 0.44281700160354376)
```

RotatE

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 1026),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 307),
```

('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 142),
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference>', 25)
Relation with highest average probability:
(<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>', 0.3978518114202249)
Relation with lowest average probability:
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level>', 0.36762830644621475)

HoIE

Top 5 most frequent relations:
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 429),
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI>', 251),
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term>', 148),
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/definition>', 84),
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/context>', 57)]
Relations appearing only once:
['<http://www.w3.org/2000/01/rdf-schema#domain>',
'<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>',
'<http://www.w3.org/2002/07/owl#unionOf>']
Relation with highest average probability:
(<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>', 0.5536947846412659)
Relation with lowest average probability:
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink>', 0.38551637530326843)

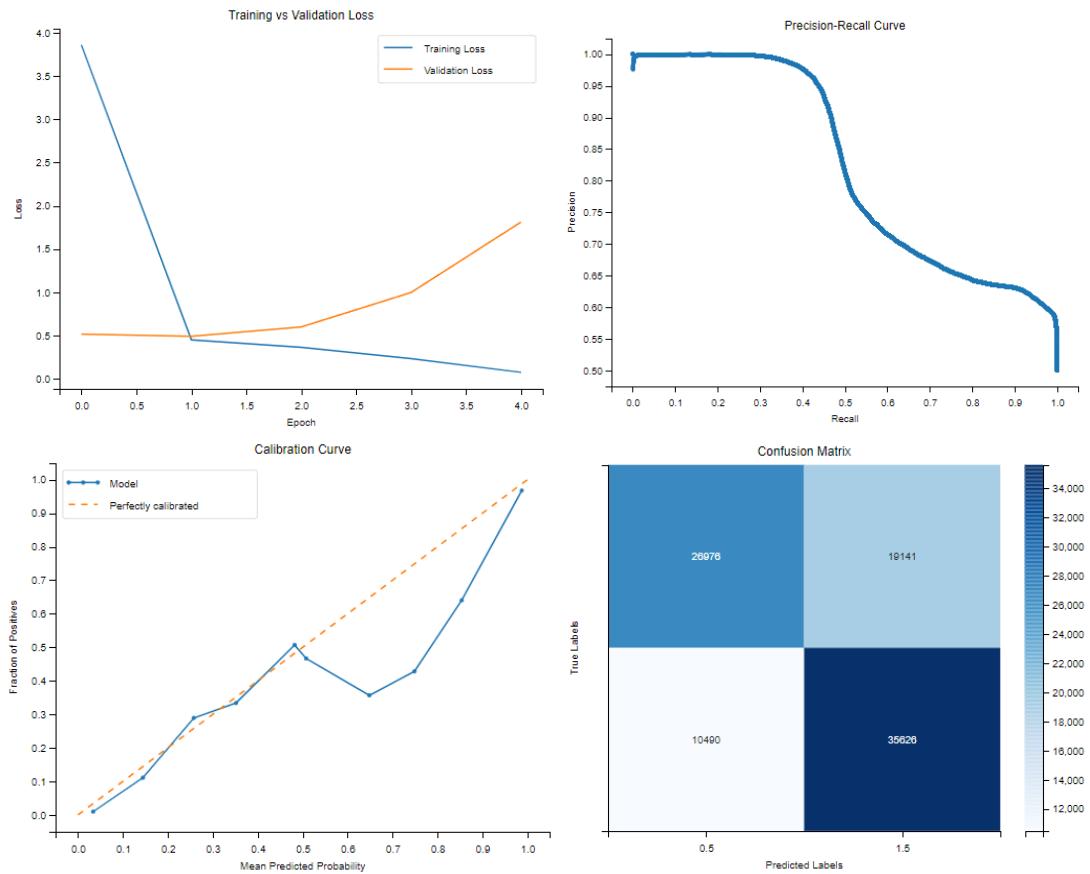
DistMult

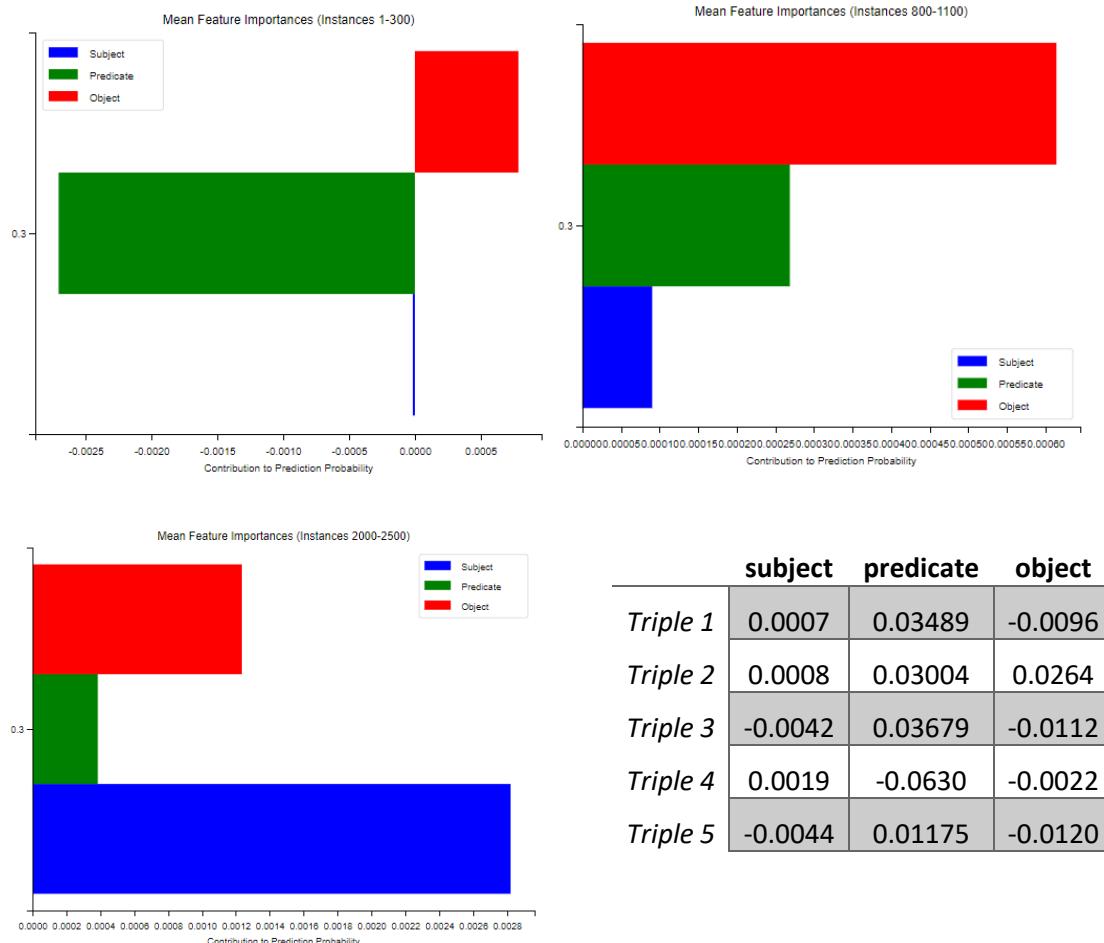
Top 5 most frequent relations:
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 539),
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/context>', 264),
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/title>', 212),
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/keyword>', 86),
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI>', 66)]
Relations appearing only once:
['<http://www.w3.org/2002/07/owl#imports>', '<http://www.w3.org/2000/01/rdf-schema#label>',
'<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCategoryOfGlossaryArticle>',
'<http://www.w3.org/2000/01/rdf-schema#comment>']
Relation with highest average probability:
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasFrequentTerm>',
0.5834585626920065)
Relation with lowest average probability:
(<http://www.w3.org/2002/07/owl#imports>', 0.39941537380218506)

IV. Group 4

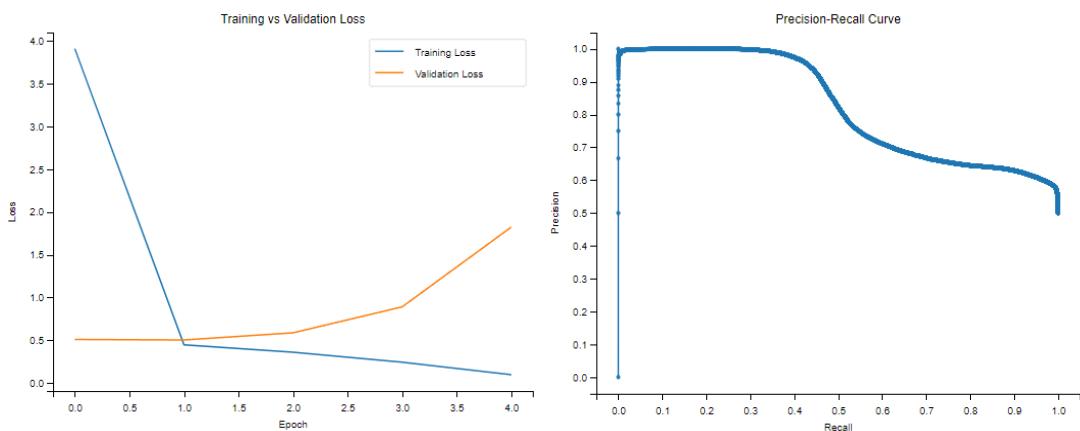
	Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
<i>TransE</i>	0.8104	0.2439	3590.5302
<i>TransH</i>	0.8048	0.2519	3462.0251
<i>NoEmbds</i>	0.7977	0.2641	3273.4176
<i>Hole</i>	0.7958	0.2646	3188.0821
<i>DistMult</i>	0.8024	0.2527	3642.3019
<i>ComplEx</i>	0.8065	0.2480	3854.4008

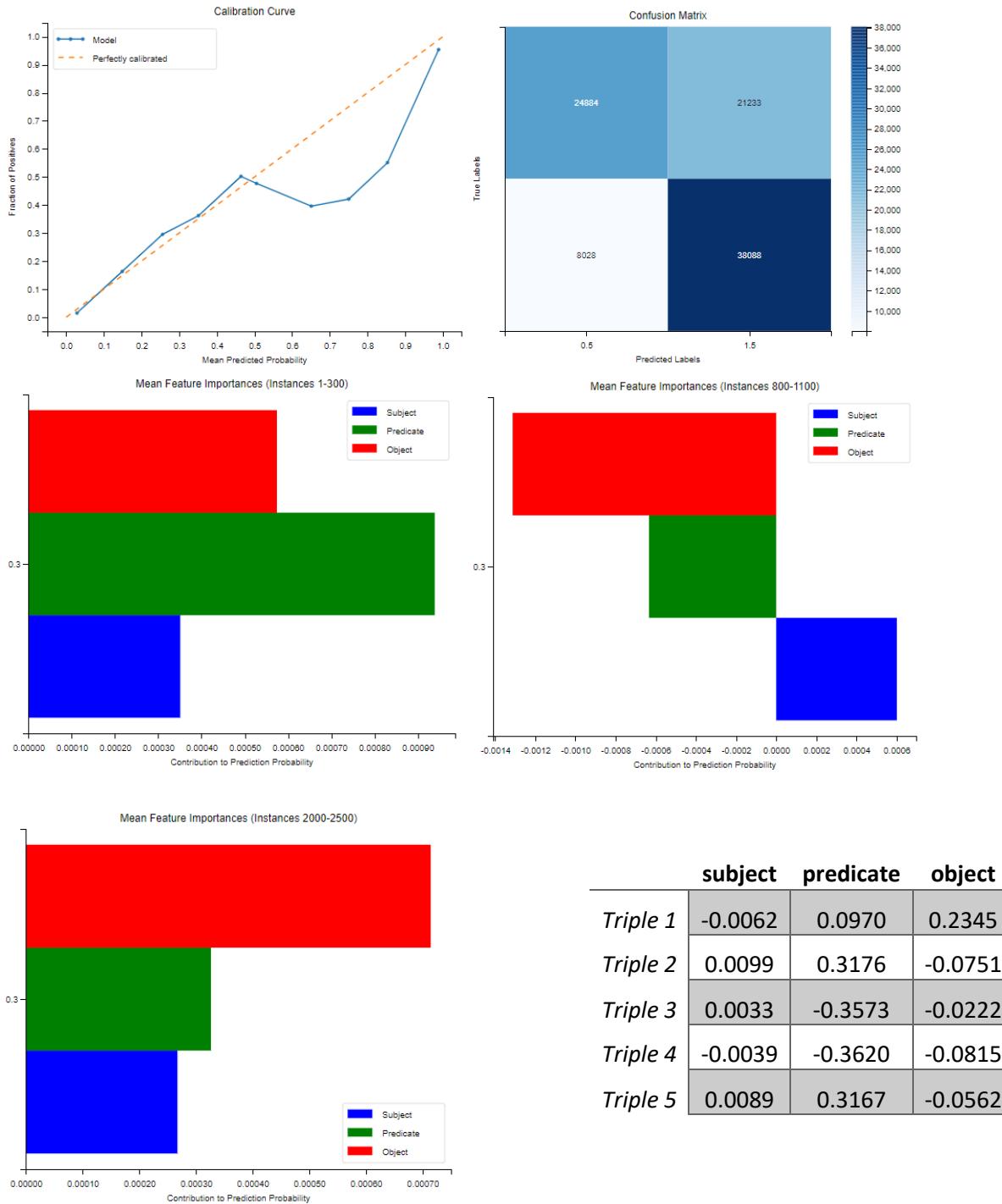
TransE



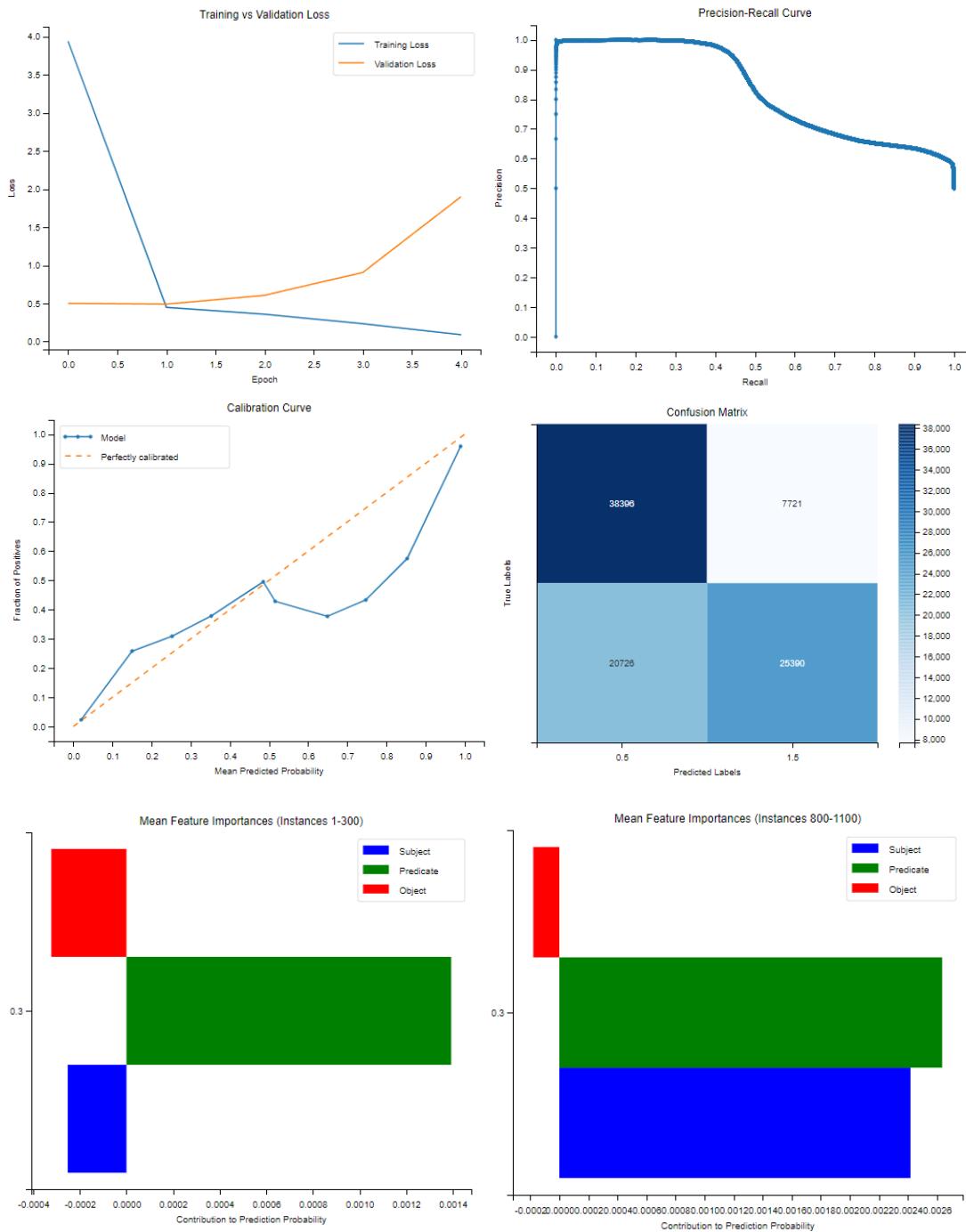


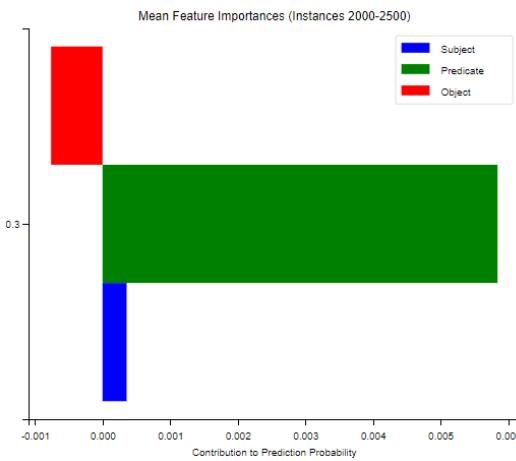
TransH





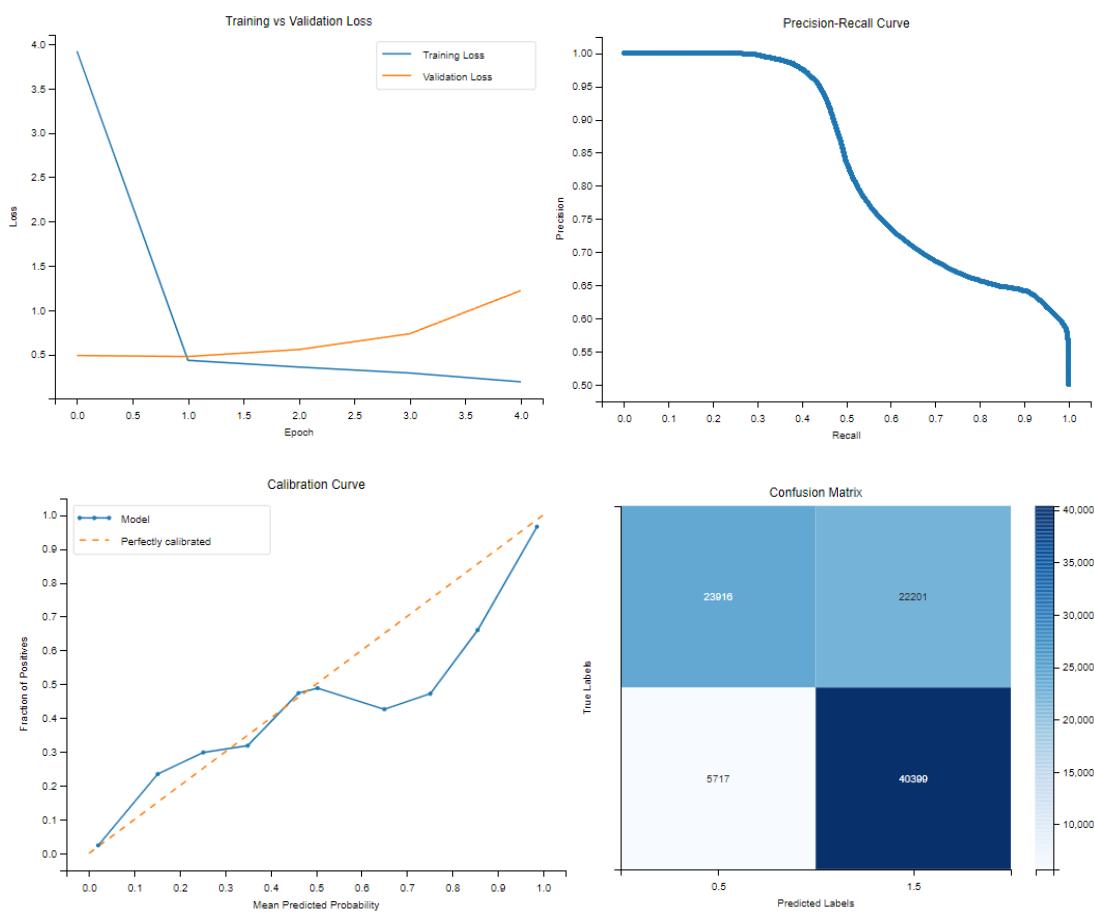
NoEmbds

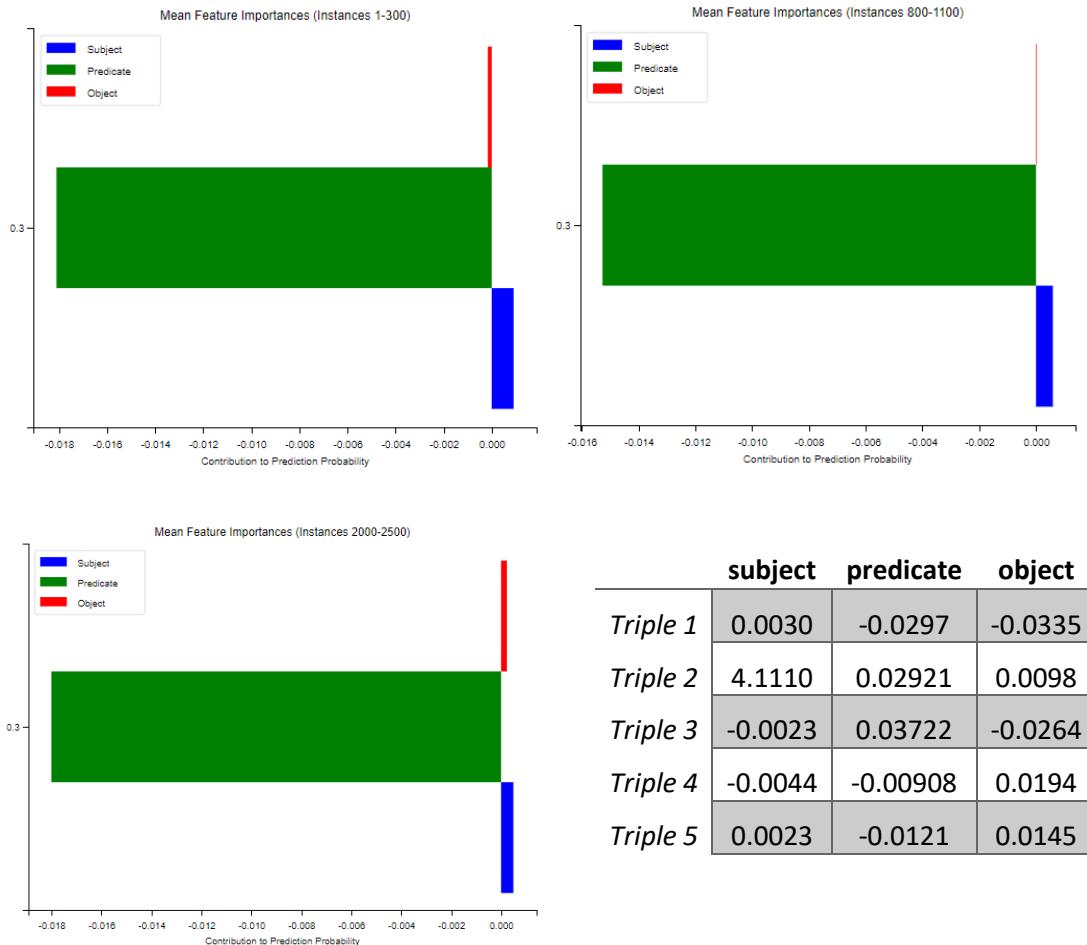




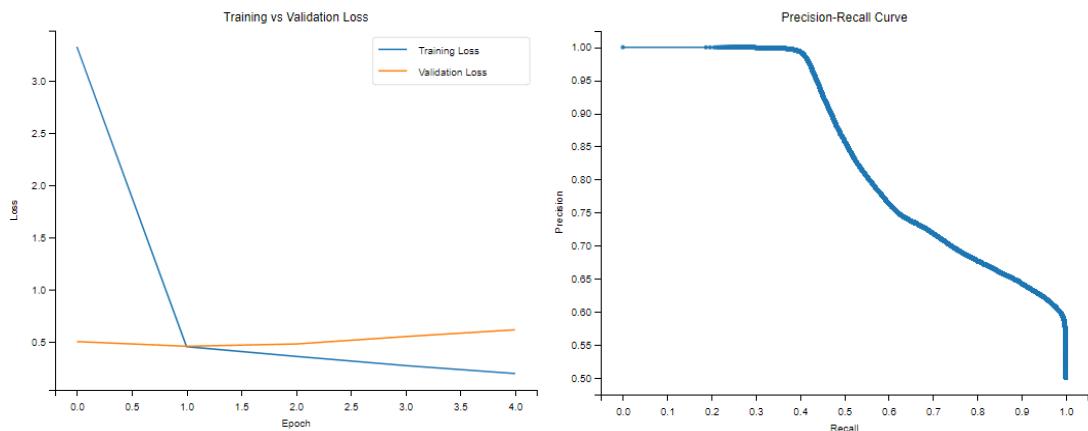
	subject	predicate	object
Triple 1	0.0147	0.1431	0.0033
Triple 2	0.0127	0.18140	-0.0192
Triple 3	-0.0166	0.14519	0.0757
Triple 4	-0.0180	-0.47417	-0.0395
Triple 5	0.0129	0.17572	0.0026

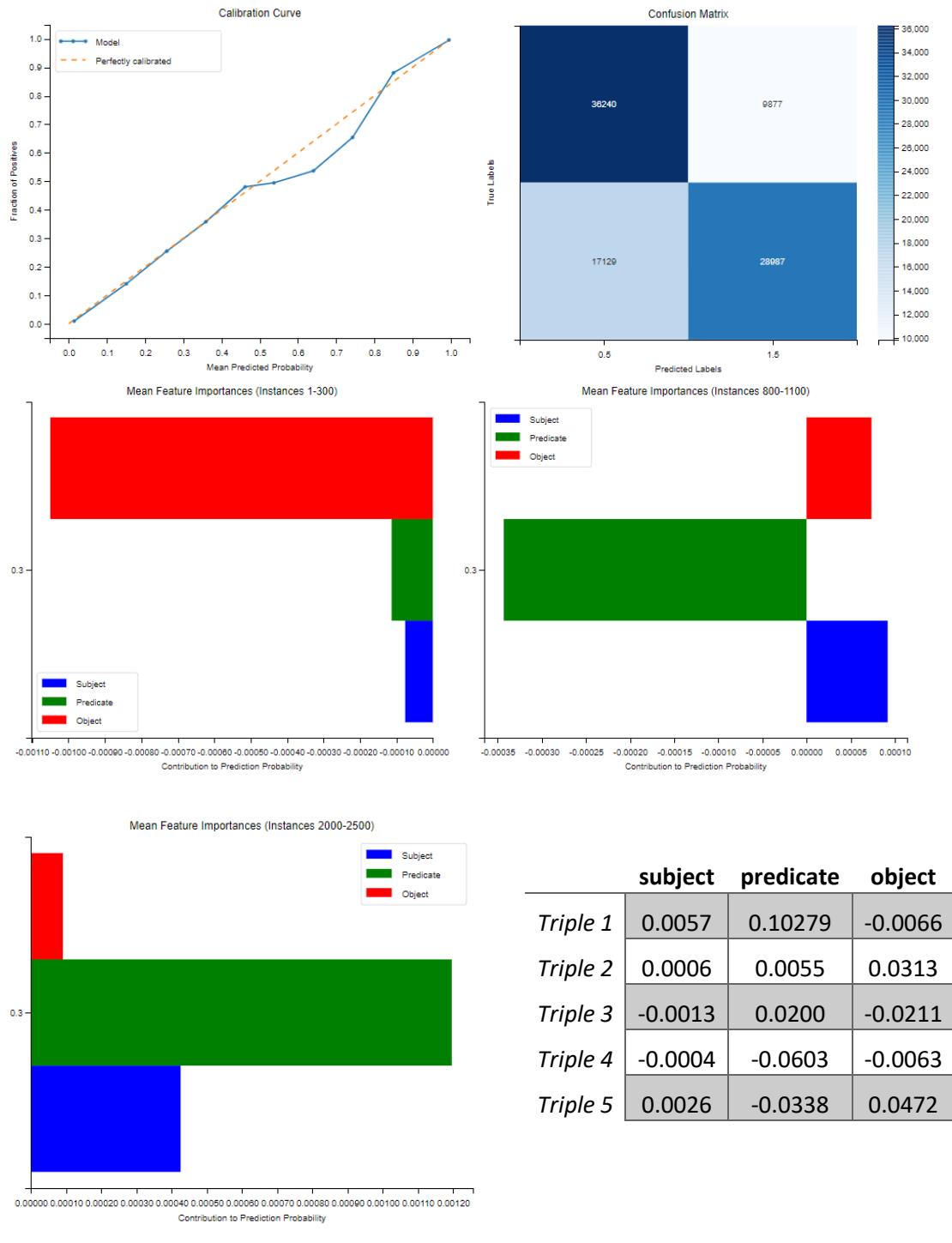
HoIE



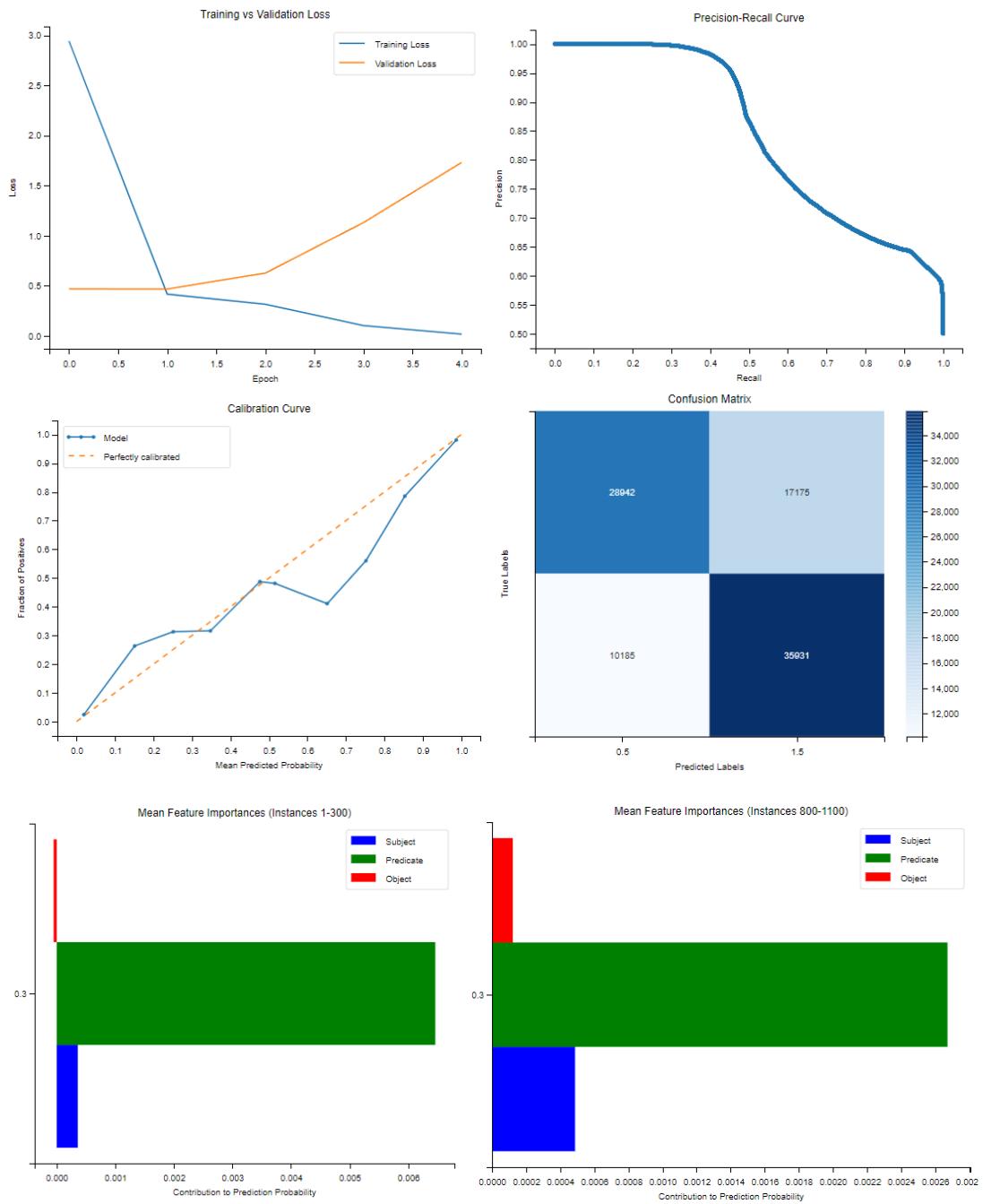


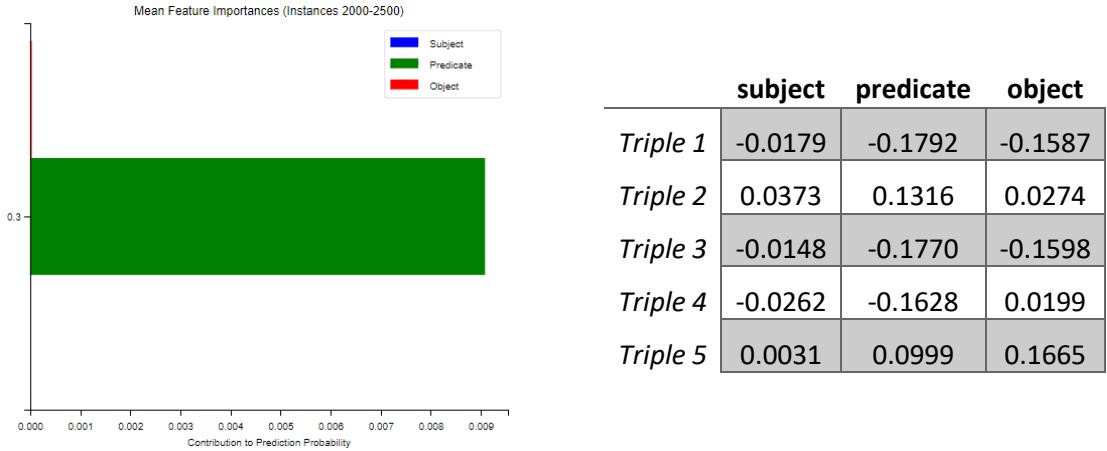
DistMult





ComplEx





	C1	C2	C3	C4	C5	C6	C7	C8
<i>TransE</i>	3	133	15	3	0.3830	0.9862	0.0002	0.2360
<i>TransH</i>	3	193	50	2	0.3646	0.9998	0.0009	0.2445
<i>RotatE</i>	3	90	17	3	0.3648	0.9953	0.0001	0.2239
<i>HoIE</i>	4	94	20	4	0.4152	0.9881	0.0007	0.1975
<i>DistMult</i>	4	118	2	1	0.4340	0.9992	0.0174	0.1414
<i>ComplEx</i>	4	59	2	2	0.4693	0.9691	0.0272	0.1148

TransE

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 822),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 612),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 66)]
```

Relation with highest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 0.42565073893373745)
```

Relation with lowest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference',
 0.37573392293734587)
```

TransH

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 955),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 519),
 ('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 26)]
```

Relations appearing only once:

Relation with highest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 0.3720383586696013)
```

Relation with lowest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 0.35139716155167583)
```

NoEmbds

Top 5 most frequent relations:

```
[('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 662),  
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 527),  
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 207)]
```

Relation with highest average probability:

```
('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 0.3745577901346935)
```

Relation with lowest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 0.35538292570659313)
```

HoIE

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 1216),  
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 184),  
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 99),  
 ('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 1)]
```

Relations appearing only once:

```
['http://www.w3.org/1999/02/22-rdf-syntax-ns#type']
```

Relation with highest average probability:

```
('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 0.9866275191307068)
```

Relation with lowest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference',  
 0.4092006692742296)
```

DistMult

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 630),  
 ('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 536),  
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 237),  
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 97)]
```

Relation with highest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 0.44132194666280633)
```

Relation with lowest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 0.4271914344516736)
```

ComplEx

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 574),  
 ('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 446),  
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 328),  
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 152)]
```

Relation with highest average probability:

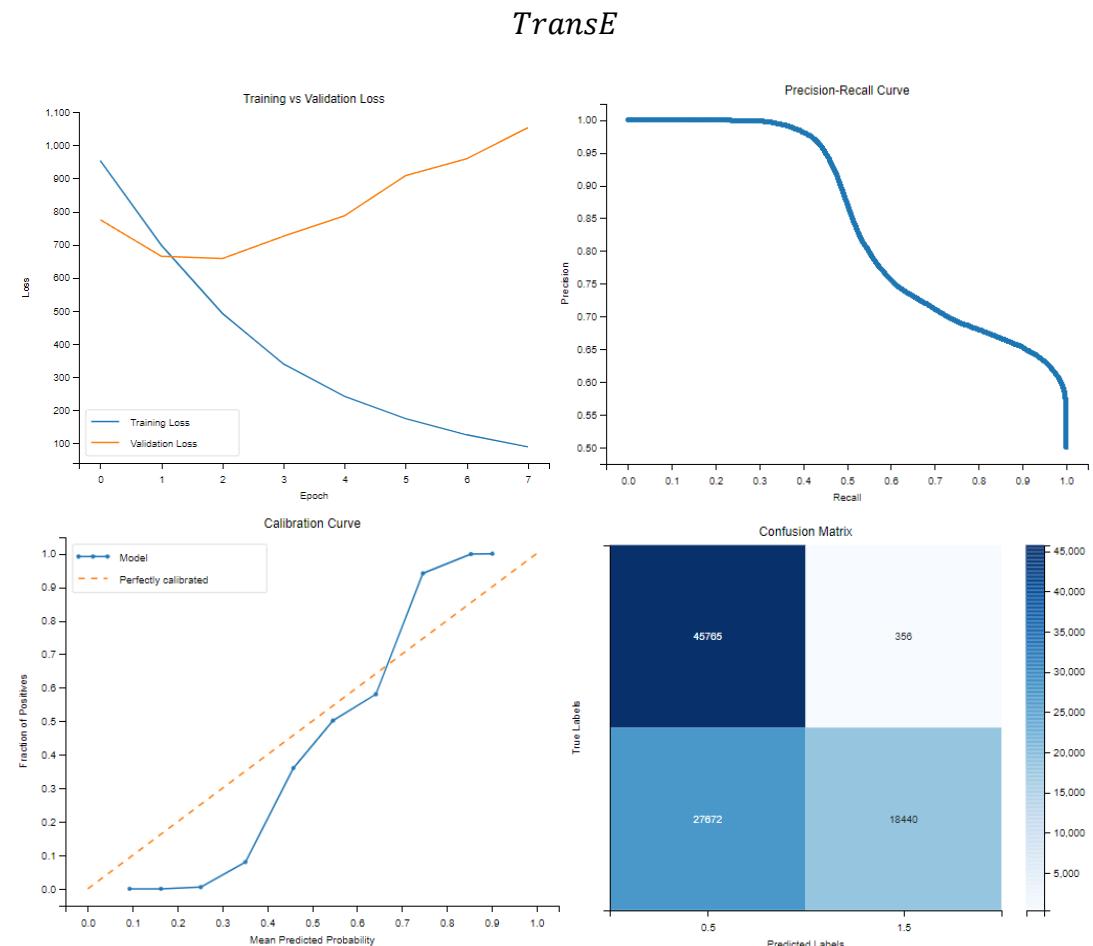
```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 0.48241893212288256)
```

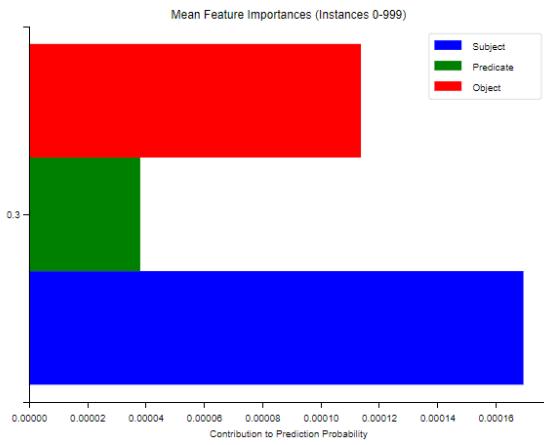
Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference',
0.4608497160339826)

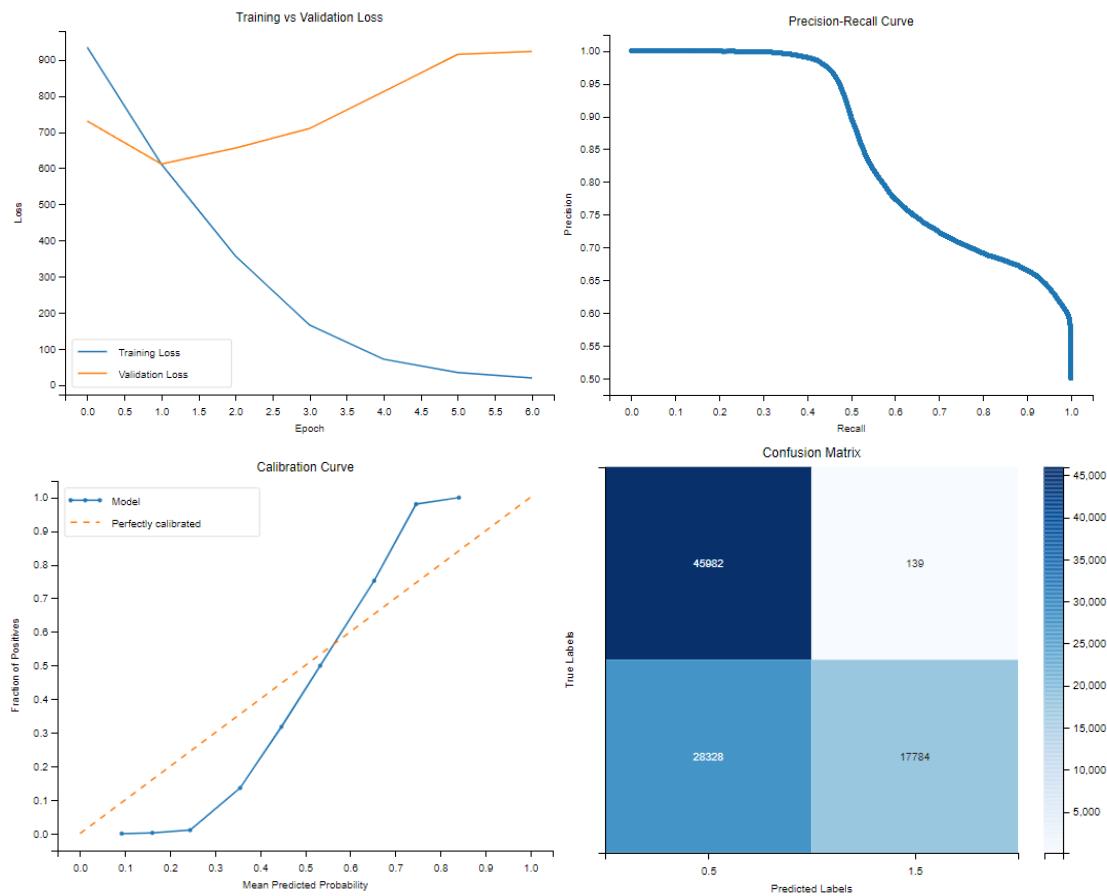
V. Group 5

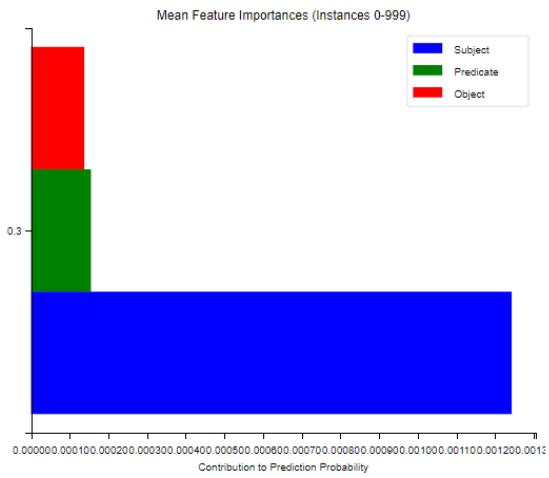
	Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
<i>TransE</i>	0.8121	0.2407	4119.5141
<i>TransH</i>	0.8130	0.2393	4052.4581
<i>RotateE</i>	0.7870	0.2817	3135.0636
<i>HoIE</i>	0.8064	0.2497	3473.7874
<i>DistMult</i>	0.8120	0.2396	4655.1074
<i>ComplEx</i>	0.8031	0.2532	3555.0818



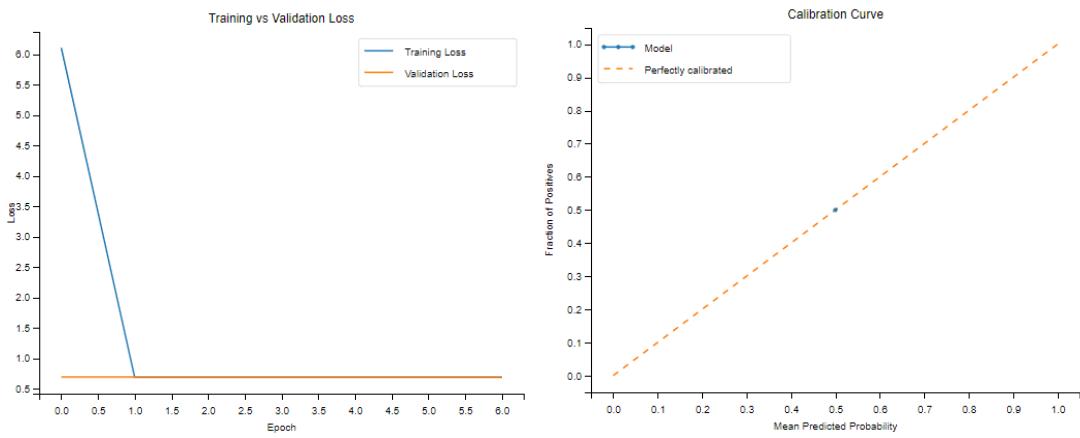


TransH

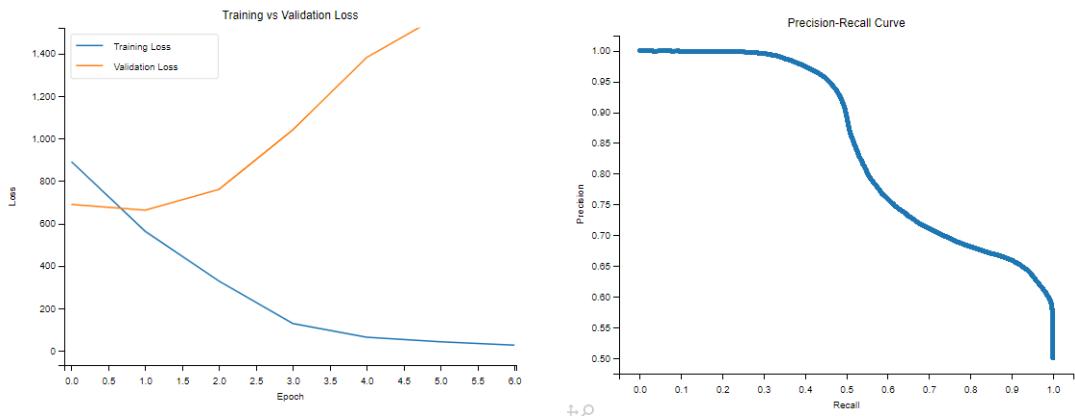


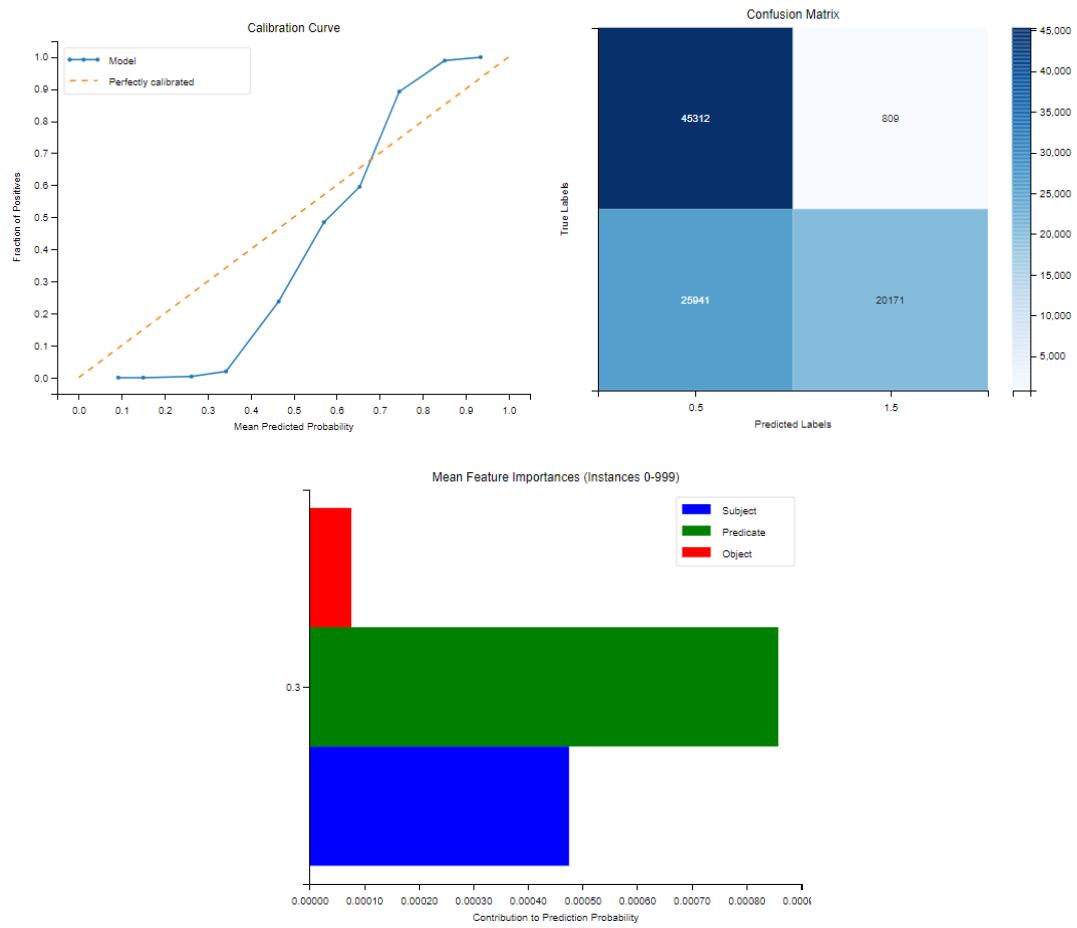


RotateE

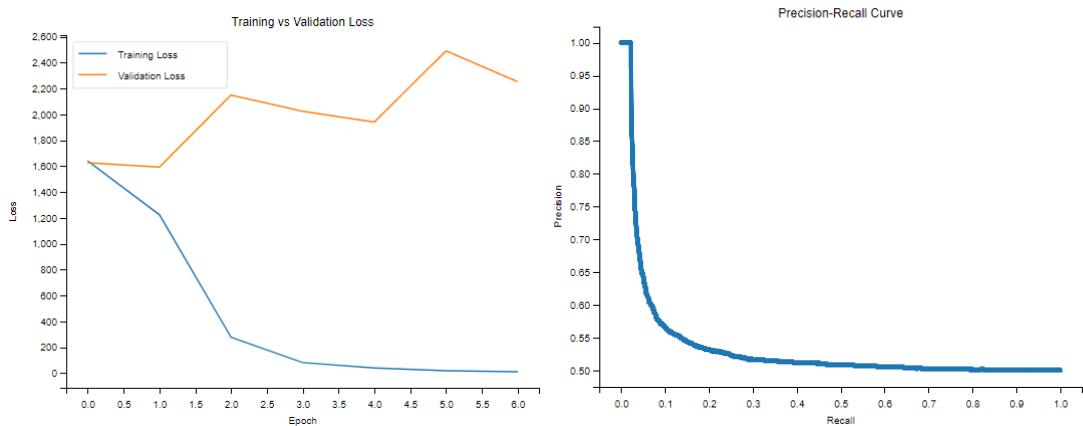


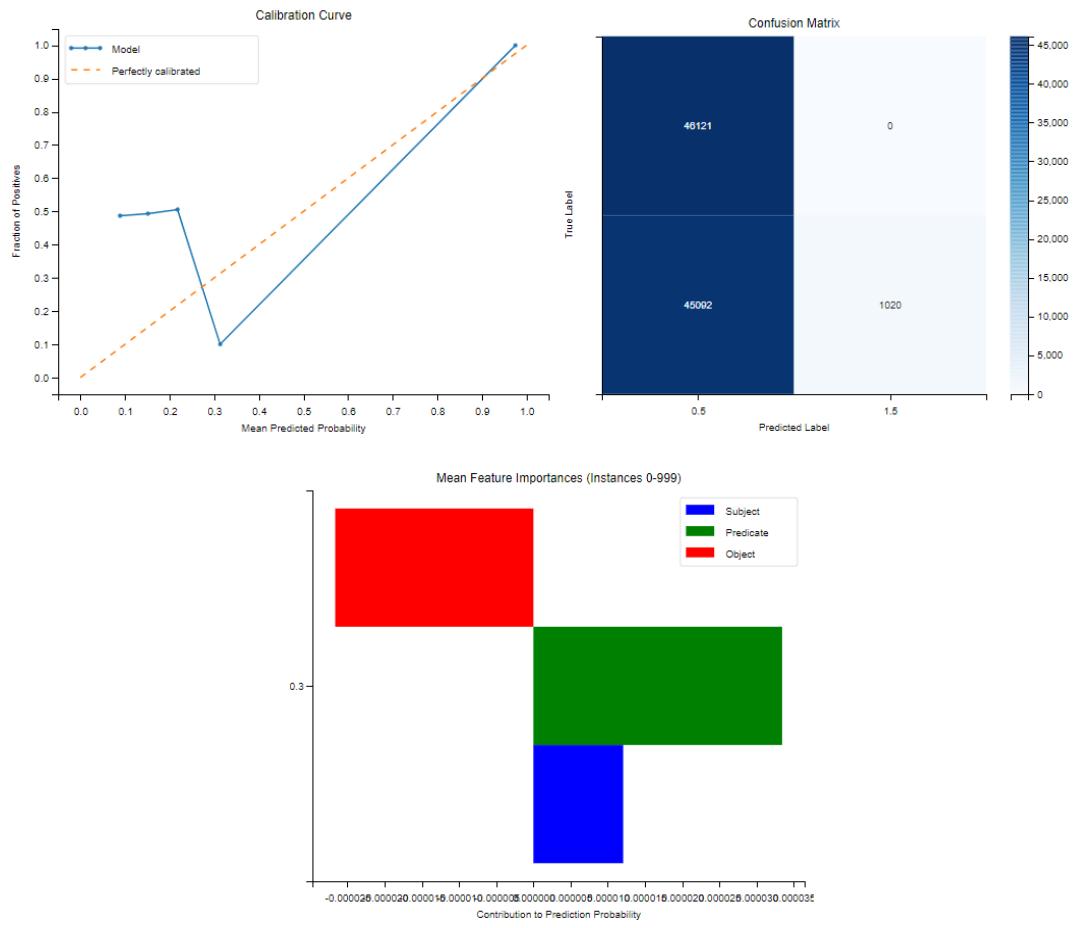
HoIE



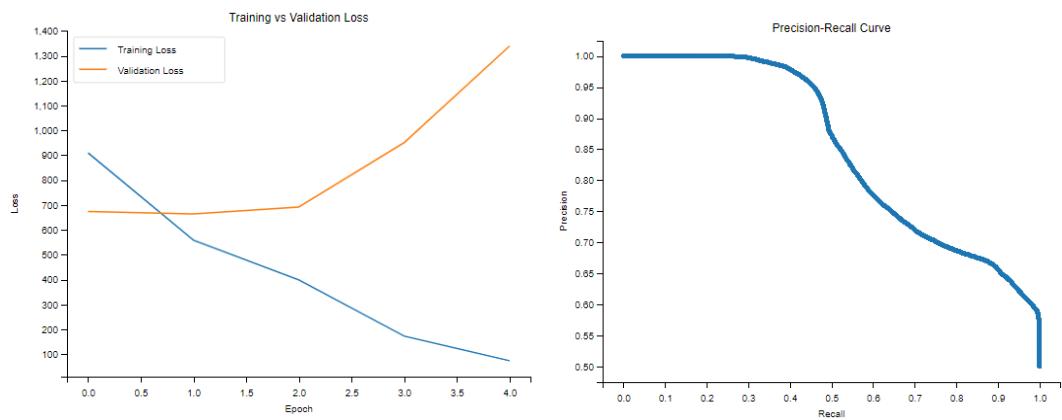


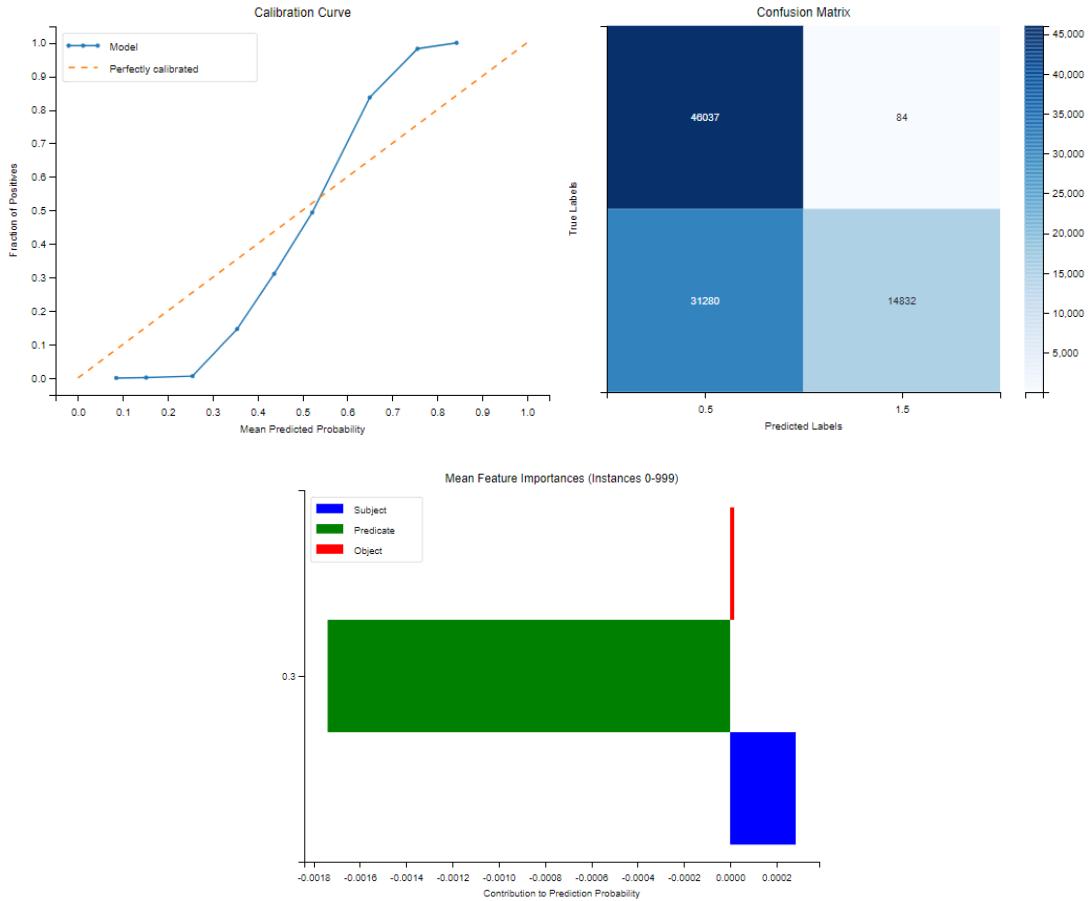
DistMult





ComplEx





	C1	C2	C3	C4	C5	C6	C7	C8
<i>TransE</i>	10	140	0	0	0.4638	0.7433	0.0921	0.1239
<i>TransH</i>	17	51	0	0	0.4401	0.7201	0.0924	0.1235
<i>RotateE</i>	1	0	0	0	0.5000	0.5	0.5	0
<i>HoIE</i>	8	54	0	0	0.4827	0.7362	0.0931	0.1251
<i>DistMult</i>	43	0	0	0	0.1553	0.2747	0.0383	0.0278
<i>ComplEx</i>	8	53	0	0	0.4296	0.6770	0.0585	0.1212

TransE

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/id', 538),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 459),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 226),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/definition', 148),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph', 83)]
```

Relations appearing only once:

```
['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasFrequentTerm']
```

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasFrequentTerm',
0.5402535200119019)

Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/content', 0.44181166047399695)

TransH

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/databasePath', 616),
(<http://www.w3.org/2000/01/rdf-schema#subClassOf>', 138),
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI>', 134),
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/content>', 117),
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term>', 117)]

Relations appearing only once:

[<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph>',
<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/id>',
<http://www.w3.org/2000/01/rdf-schema#label>',
<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode>']

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph',
0.6234550476074219)

Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 0.2667442560195923)

HoIE

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph', 987),
(<http://www.w3.org/2000/01/rdf-schema#subClassOf>', 307),
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode>', 128),
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm>', 44),
(<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL>', 20)]

Relations appearing only once:

[<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink>',
<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasFrequentTerm>']

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasFrequentTerm',
0.627998411655426)

Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink', 0.2694052457809448)

DistMult

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 346),
 ('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 94),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL', 88),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 82),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/title', 76)]

Relations appearing only once:

['http://www.w3.org/2002/07/owl#backwardCompatibleWith']

Relation with highest average probability:

('http://www.w3.org/2002/07/owl#equivalentProperty', 0.17751998826861382)

Relation with lowest average probability:

('http://www.w3.org/2000/01/rdf-schema#range', 0.13417193790276846)

ComplEx

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/content', 707),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 368),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 303),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL', 79),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 30)]

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm', 0.5032630264759064)

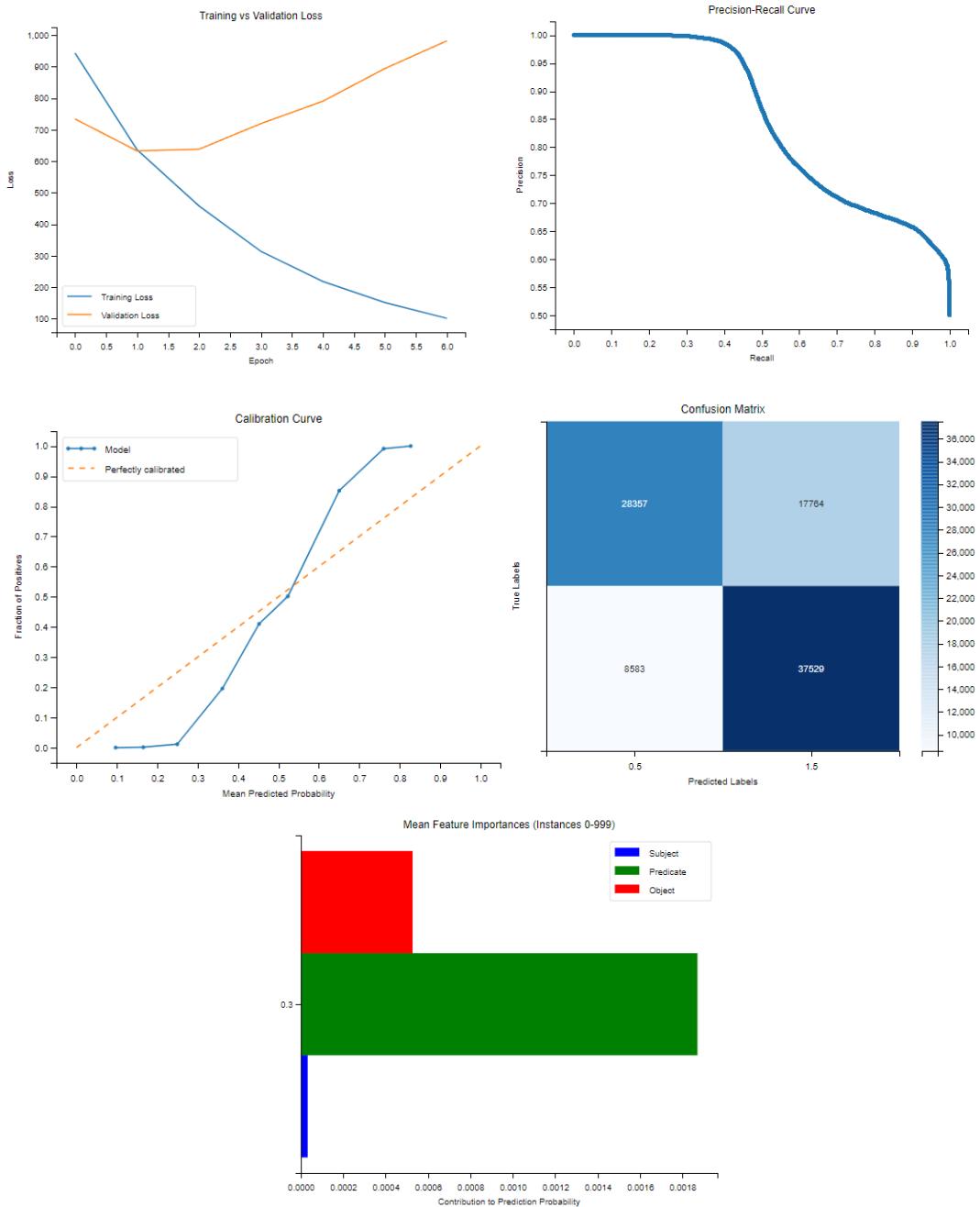
Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 0.41591848321663943)

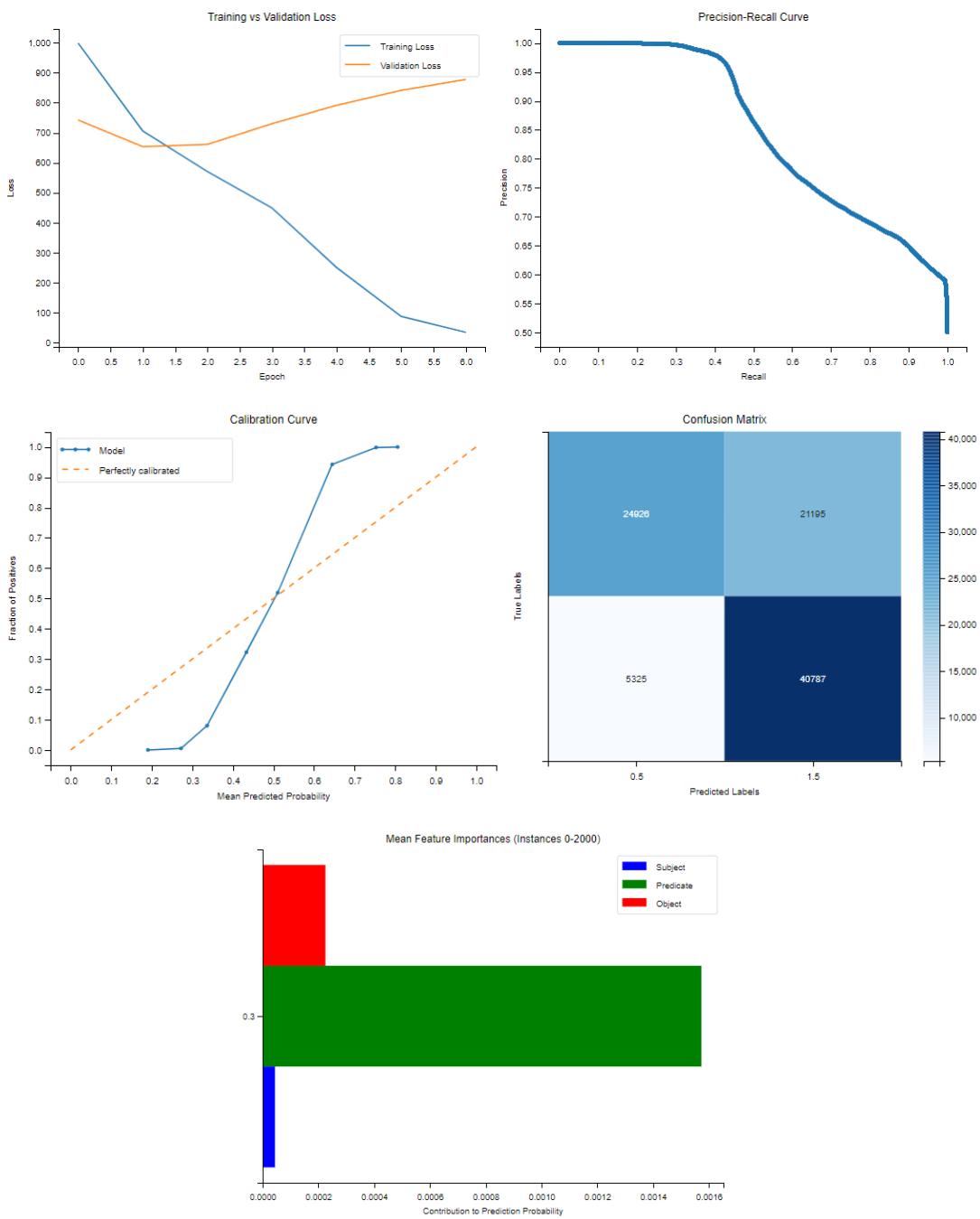
VI. Group 6

	Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
<i>TransE</i>	0.8058	0.2516	3563.2767
<i>TransH</i>	0.8129	0.2398	3744.4443
<i>RotateE</i>	0.7941	0.2700	4132.4151
<i>HoIE</i>	0.8129	0.2402	4226.9172
<i>DistMult</i>	0.8186	0.2322	3862.0030
<i>ComplEx</i>	0.7928	0.2710	4206.7353

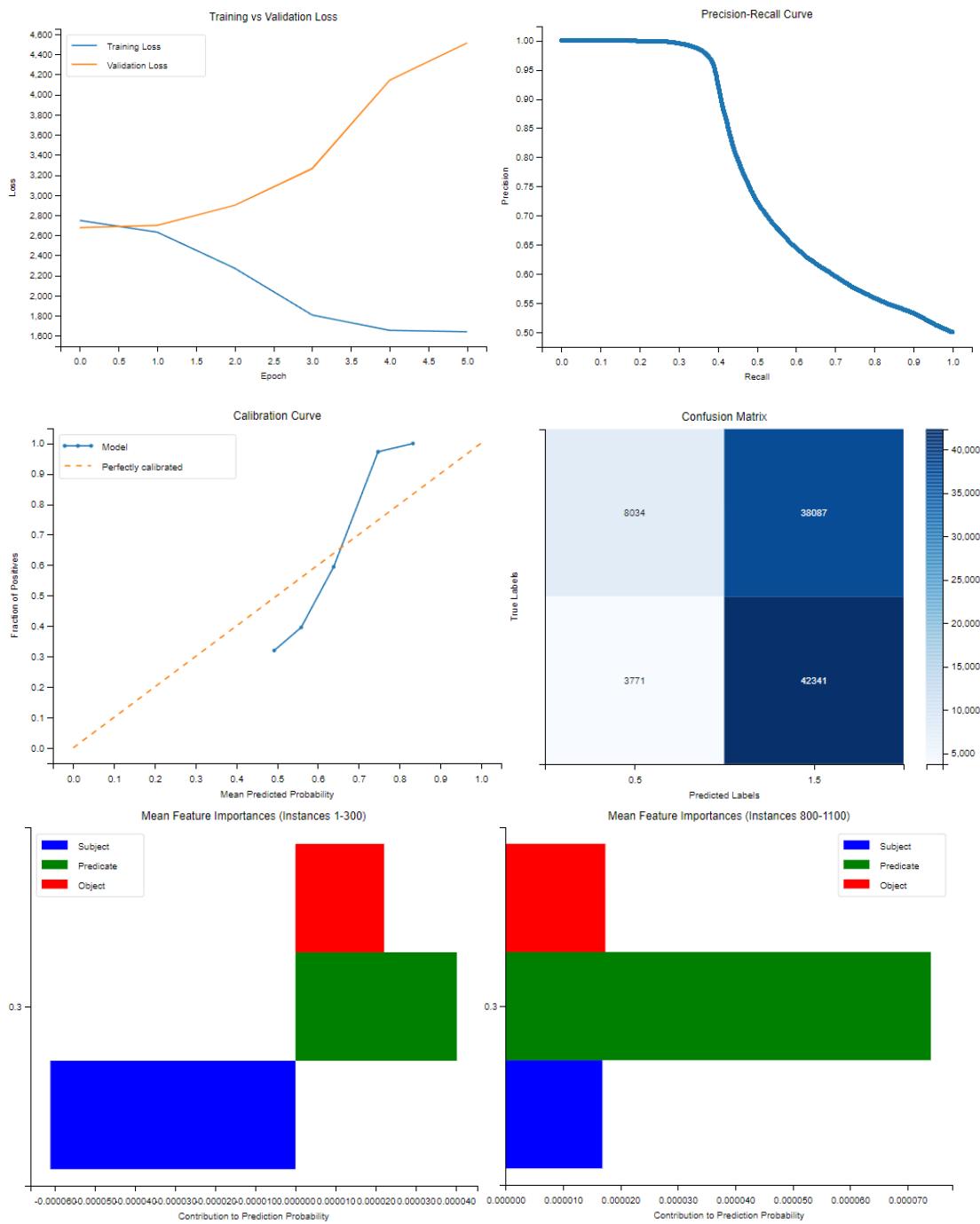
TransE

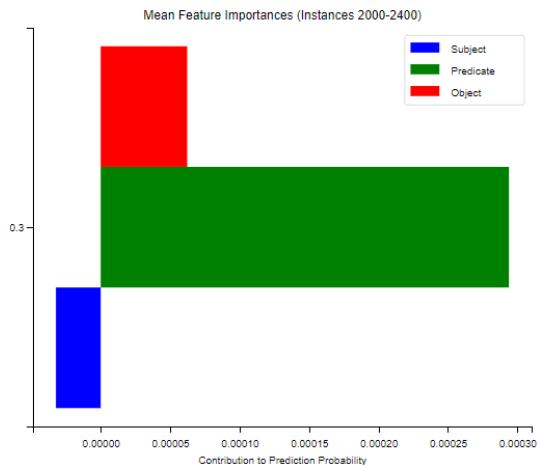


TransH



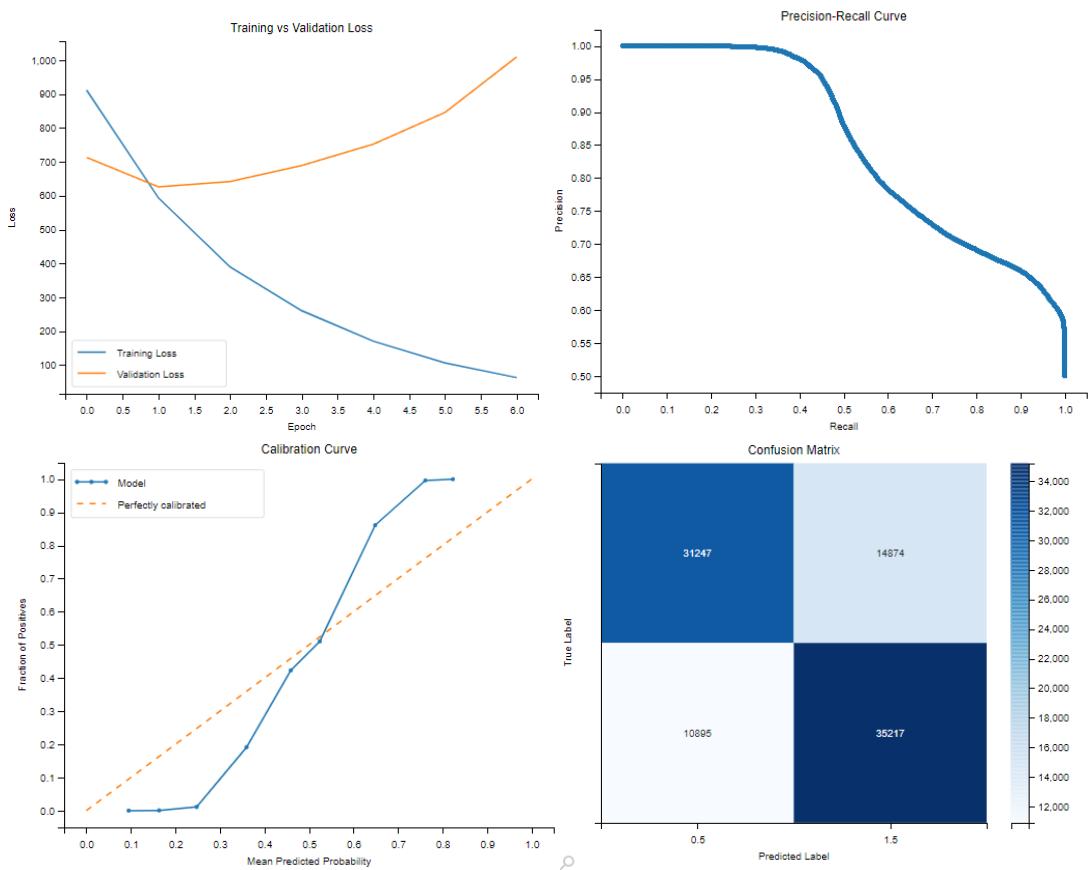
RotateE

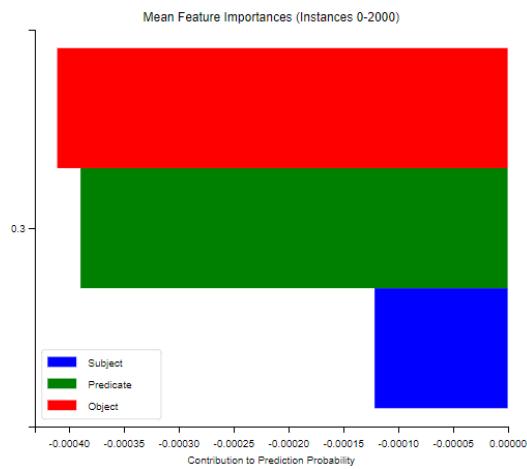




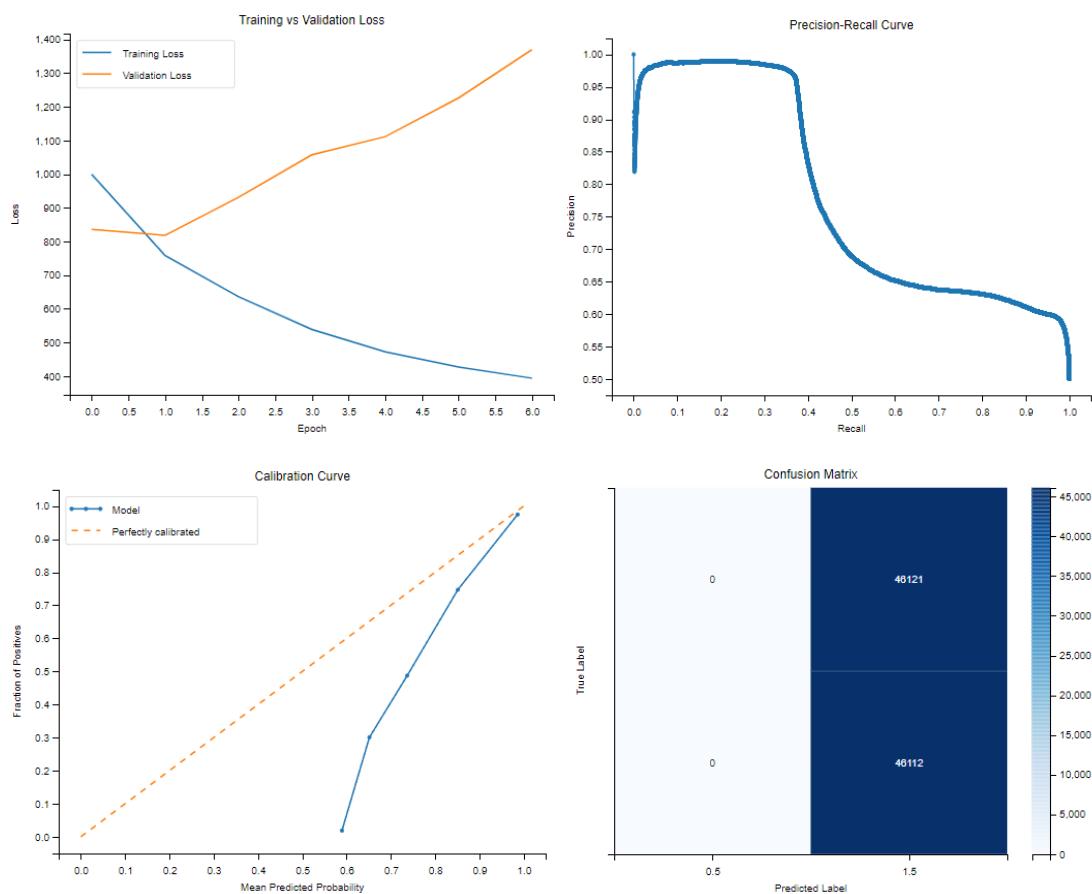
	subject	predicate	object
Triple 1	0.0006	-0.00036	0.0377
Triple 2	-0.0018	0.00010	0.0097
Triple 3	0.0033	-0.00074	0.0106
Triple 4	0.0008	-0.00051	0.0358
Triple 5	-0.0020	-0.00039	-0.0897

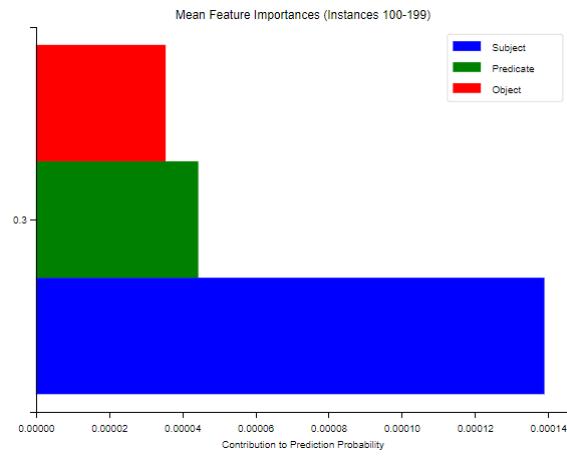
HoIE



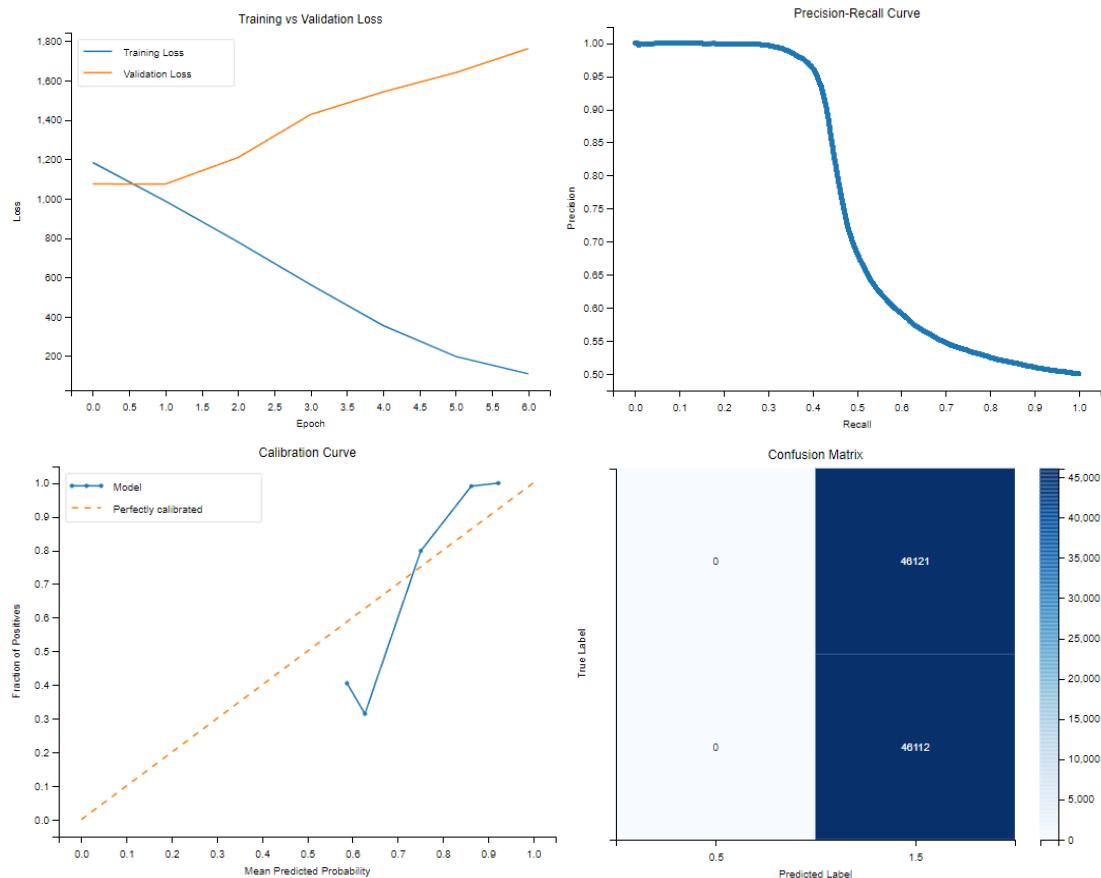


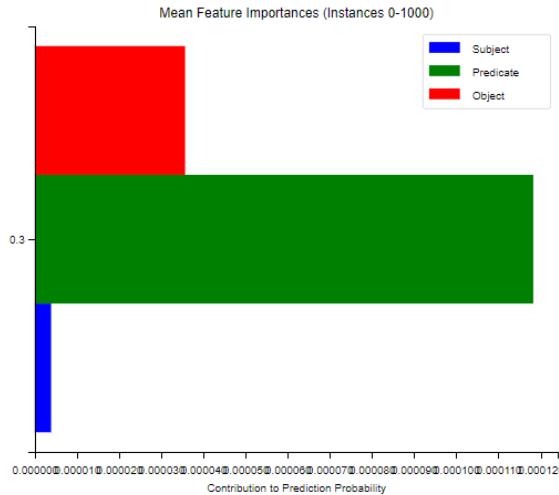
DistMult





ComplEx





	C1	C2	C3	C4	C5	C6	C7	C8
<i>TransE</i>	11	49	0	0	0.4726	0.7412	0.1389	0.1093
<i>TransH</i>	27	0	0	0	0.4295	0.5081	0.1932	0.0821
<i>Rotate</i>	31	0	0	0	0.1157	0.1293	0.1031	0.0071
<i>HoIE</i>	18	18	0	0	0.4174	0.6353	0.1006	0.1089
<i>DistMult</i>	1	1301	0	0	0.6798	0.7938	0.5669	0.0553
<i>ComplEx</i>	43	516	0	0	0.6005	0.6692	0.5706	0.0239

TransE

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 633),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 461),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/content', 229),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/definition', 109),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm', 24)]
```

Relations appearing only once:

```
['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL',
 'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/title',
 'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCategoryOfGlossaryArticle']
```

Relation with highest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL', 0.5593205094337463)
```

Relation with lowest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/title', 0.3952298164367676)
```

TransH

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/definition', 1290),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/title', 27),
```

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/sourcePublication', 27),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 18),
('http://www.w3.org/1999/02/22-rdf-syntax-ns#first', 14)]

Relations appearing only once:

['http://www.w3.org/2000/01/rdf-schema#subPropertyOf',
'http://www.w3.org/2000/01/rdf-schema#domain',
'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/databasePath']

Relation with highest average probability:

('http://www.w3.org/2002/07/owl#equivalentProperty', 0.4814907133579254)

Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasEurostatTheme',
0.35845009982585907)

HoIE

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 423),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/databasePath', 302),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/title', 143),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph', 141),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm', 134)]

Relations appearing only once:

['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term',
'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/remark',
'http://www.w3.org/2002/07/owl#backwardCompatibleWith']

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/remark', 0.48419255018234253)

Relation with lowest average probability:

('http://www.w3.org/2002/07/owl#backwardCompatibleWith', 0.2759597897529602)

DistMult

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink', 1500)]

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink', 0.6797794284025828)

Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink', 0.6797794284025828)

ComplEx

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 70),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasEurostatTheme', 53),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/keyword', 48),

(['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI'](https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI), 47),
 (['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL'](https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL), 47)]

Relation with highest average probability:

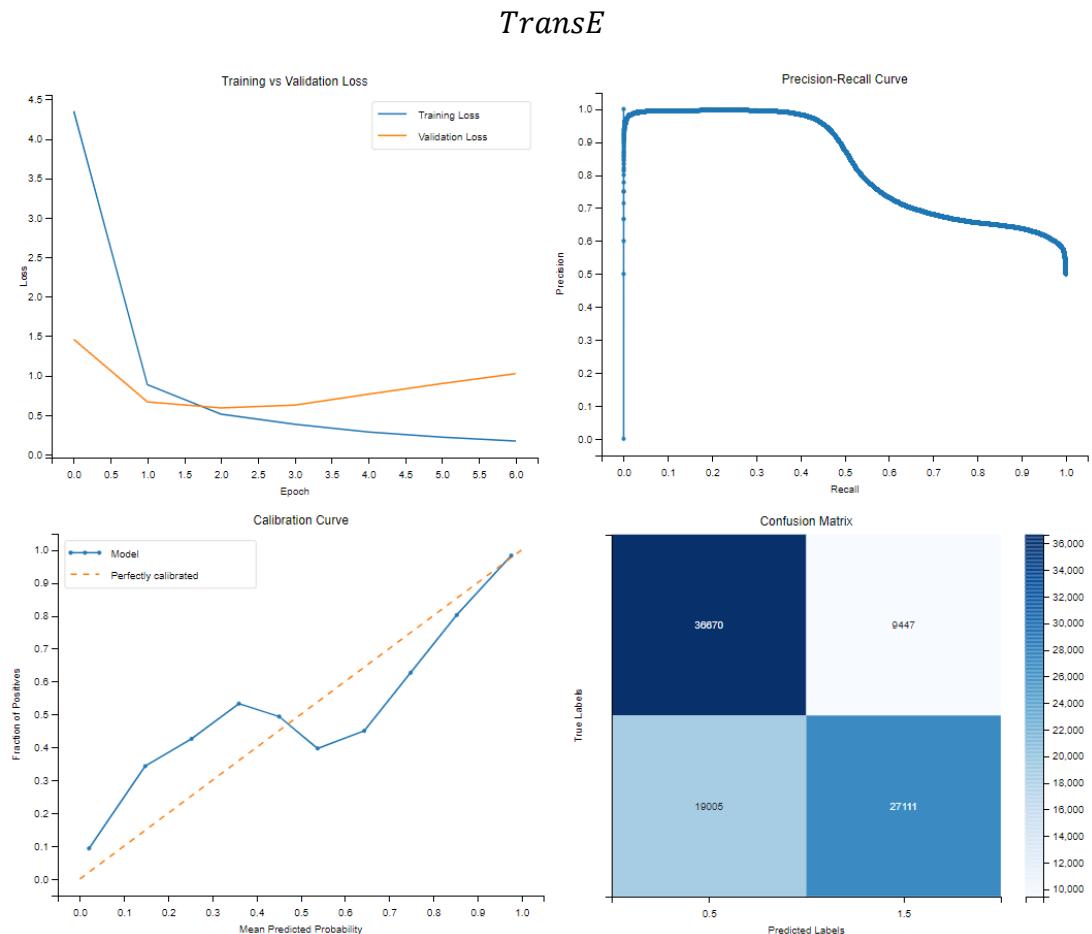
(['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/context'](https://ec.europa.eu/eurostat/NLP4StatRef/ontology/context), 0.6107146433421544)

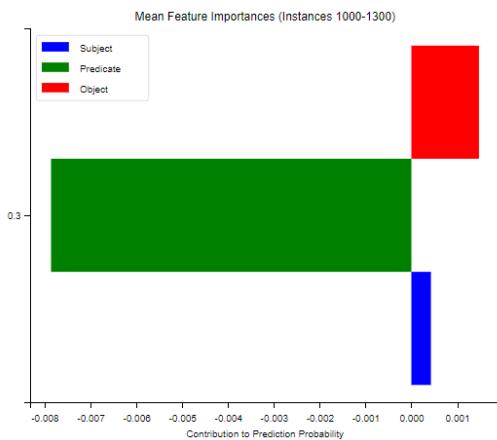
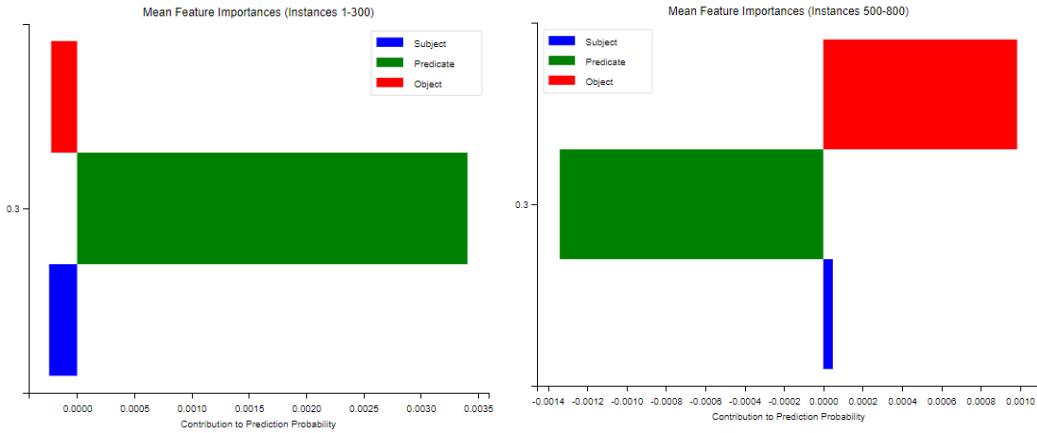
Relation with lowest average probability:

(['http://www.w3.org/1999/02/22-rdf-syntax-ns#first'](http://www.w3.org/1999/02/22-rdf-syntax-ns#first), 0.5913937477504506)

VII. Group 7

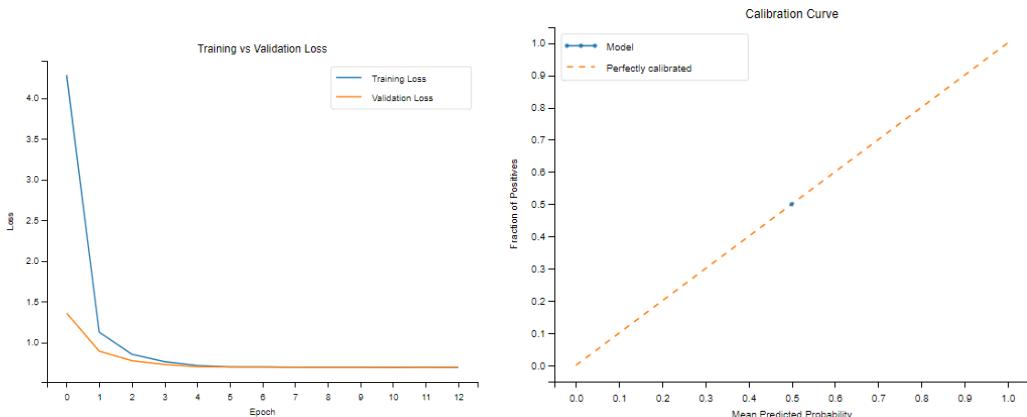
	Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
<i>TransE</i>	0.8061	0.2486	3957.2206
<i>TransH</i>	0.7834	0.2856	3052.8489
<i>RotatE</i>	0.7912	0.2727	4089.8736
<i>NoEmbds</i>	0.8055	0.2481	4085.5100
<i>HoIE</i>	0.8087	0.2464	3820.3877
<i>DistMult</i>	0.7913	0.2738	4053.6681



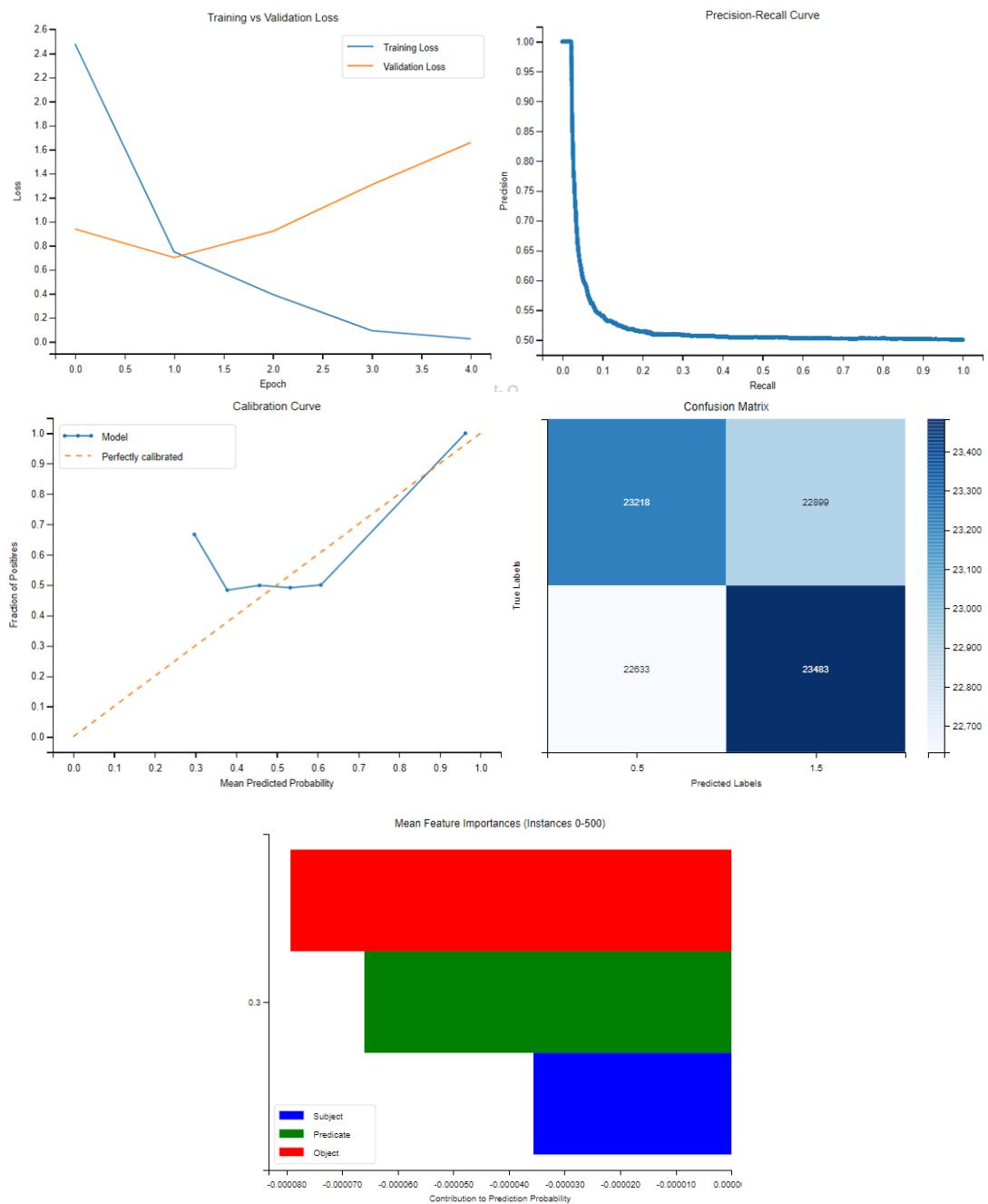


	subject	predicate	object
Triple 1	-0.0022	0.1603	0.0345
Triple 2	0.0372	0.0609	0.0050
Triple 3	0.0011	-0.39708	-0.0489
Triple 4	-0.0317	0.16032	0.0097
Triple 5	-0.0083	0.09255	0.0157

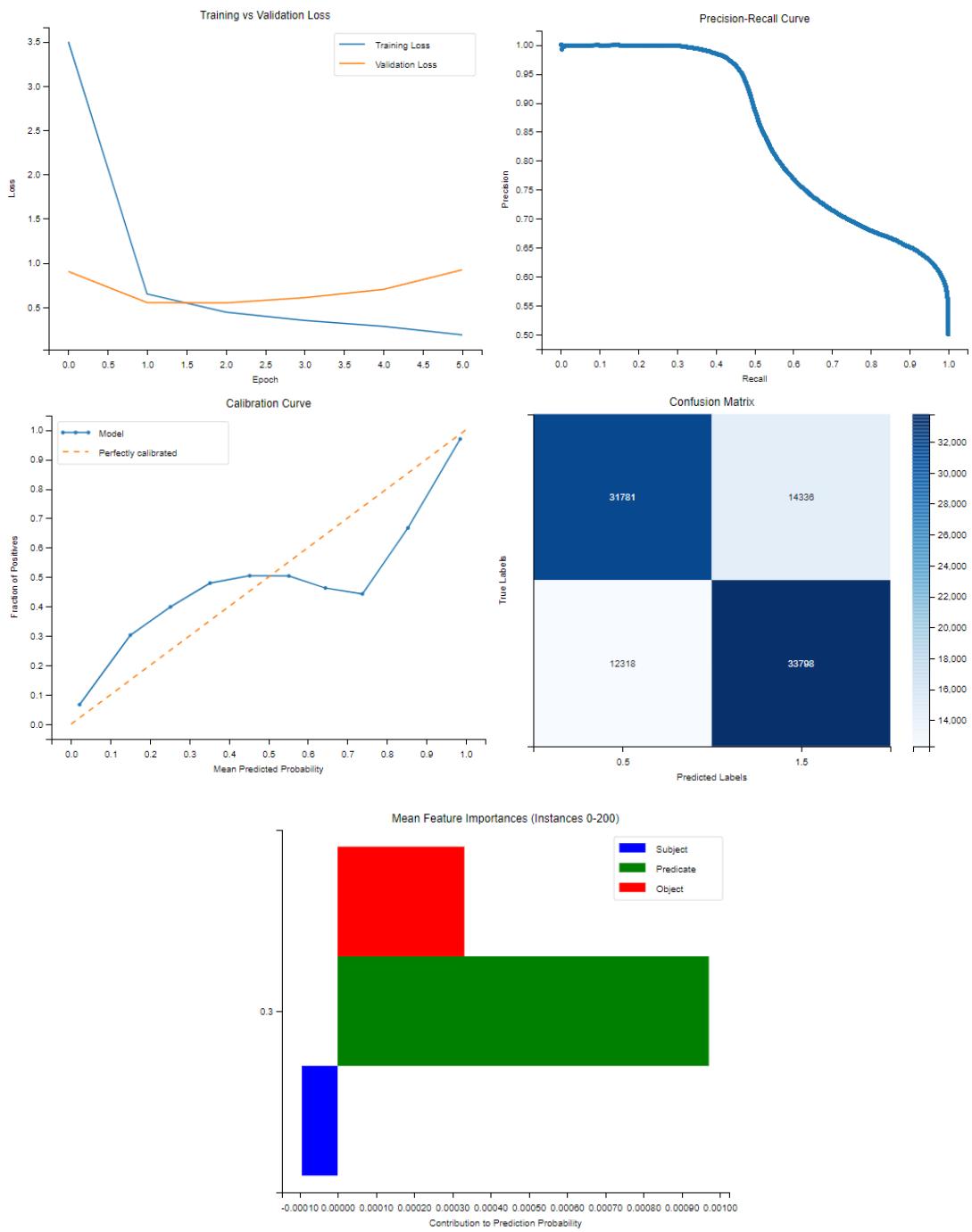
TransH



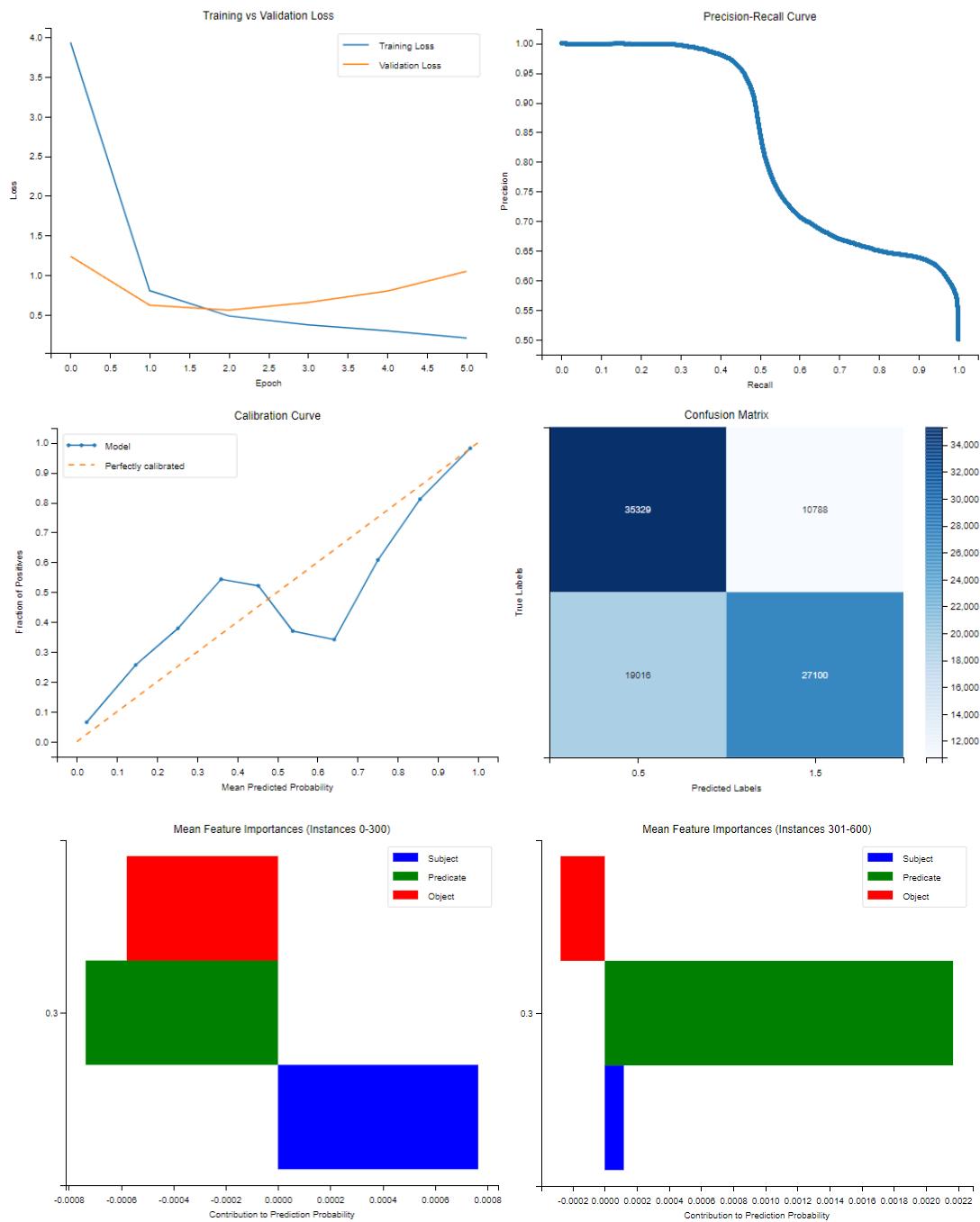
RotateE

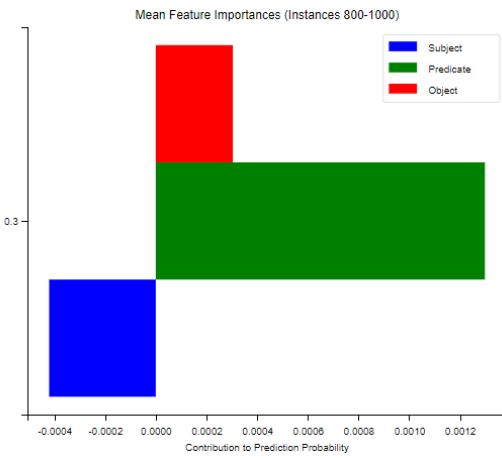


NoEmbeds

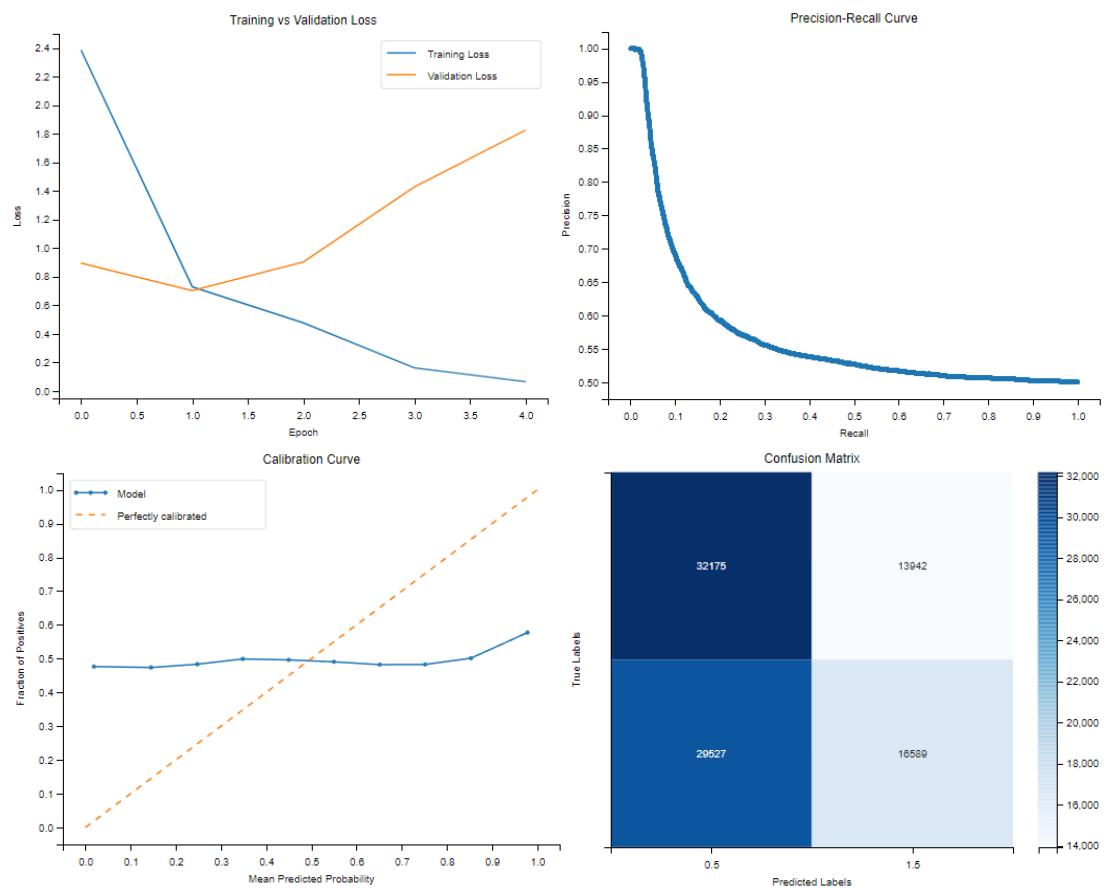


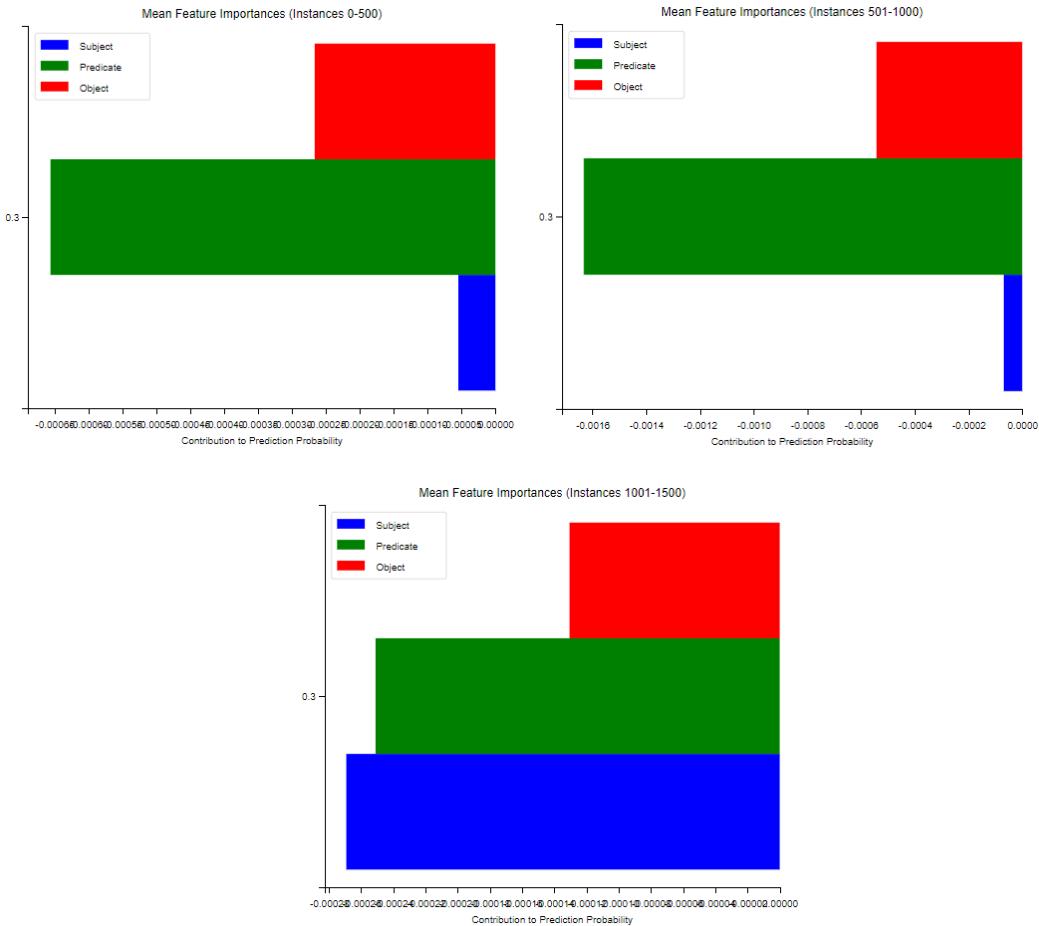
HoIE



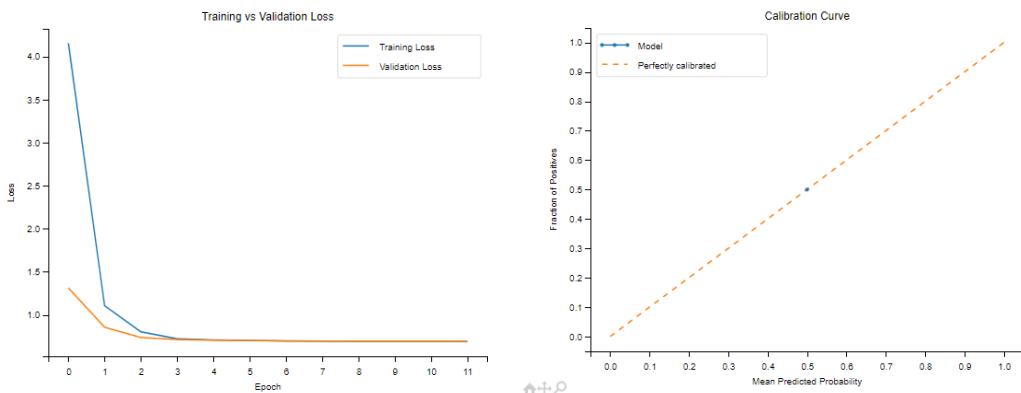


DistMult





ComplEx



	C1	C2	C3	C4	C5	C6	C7	C8
<i>TransE</i>	13	76	6	3	0.3128	0.9611	0.0002	0.2202
<i>TransH</i>	1	0	0	0	0.5000	0.5000	0.5000	0
<i>RotatE</i>	43	0	0	0	0.4872	0.5781	0.2688	0.0648
<i>NoEmbds</i>	18	290	32	9	0.3319	0.9979	0.0002	0.2682
<i>Hole</i>	10	97	0	0	0.2909	0.8395	0.0002	0.2169
<i>DistMult</i>	43	427	256	43	0.3203	0.9996	0.0019	0.3863

TransE

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology(dataSource', 584),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 294),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 232),
('http://www.w3.org/2000/01/rdf-schema#subClassOf', 160),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/context', 133)]
```

Relations appearing only once:

```
['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/title',
'http://www.w3.org/2000/01/rdf-schema#comment',
'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI']
```

Relation with highest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/content', 0.5114988520120581)
```

Relation with lowest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 0.0035236880648881197)
```

RotateE

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 99),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasOECDTheme', 51),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL', 42),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 41),
('http://www.w3.org/2000/01/rdf-schema#range', 41)]
```

Relation with highest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCategoryOfStatisticExplainedArticle', 0.5164193113644918)
```

Relation with lowest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 0.4562055060157069)
```

NoEmbds

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 640),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 158),
('http://www.w3.org/2000/01/rdf-schema#subPropertyOf', 130),
('http://www.w3.org/2002/07/owl#equivalentProperty', 107),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 89)]
```

Relations appearing only once:

```
['http://www.w3.org/2000/01/rdf-schema#range',
'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL']
```

Relation with highest average probability:

```
('http://www.w3.org/2000/01/rdf-schema#range', 0.4767267107963562)
```

Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL', 0.003997777123004198)

*HoIE***Top 5 most frequent relations:**

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/definition', 615),
 ('http://www.w3.org/2000/01/rdf-schema#label', 476),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm', 216),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 88),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/context', 44)]

Relations appearing only once:

['http://www.w3.org/1999/02/22-rdf-syntax-ns#type']

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL', 0.34485906522798665)

Relation with lowest average probability:

('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 0.0012193239526823163)

*DistMult***Top 5 most frequent relations:**

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasTopic', 59),
 ('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 58),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasOECDTheme', 49),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm', 48),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 46)]

Relation with highest average probability:

('http://www.w3.org/2000/01/rdf-schema#comment', 0.5469163116857609)

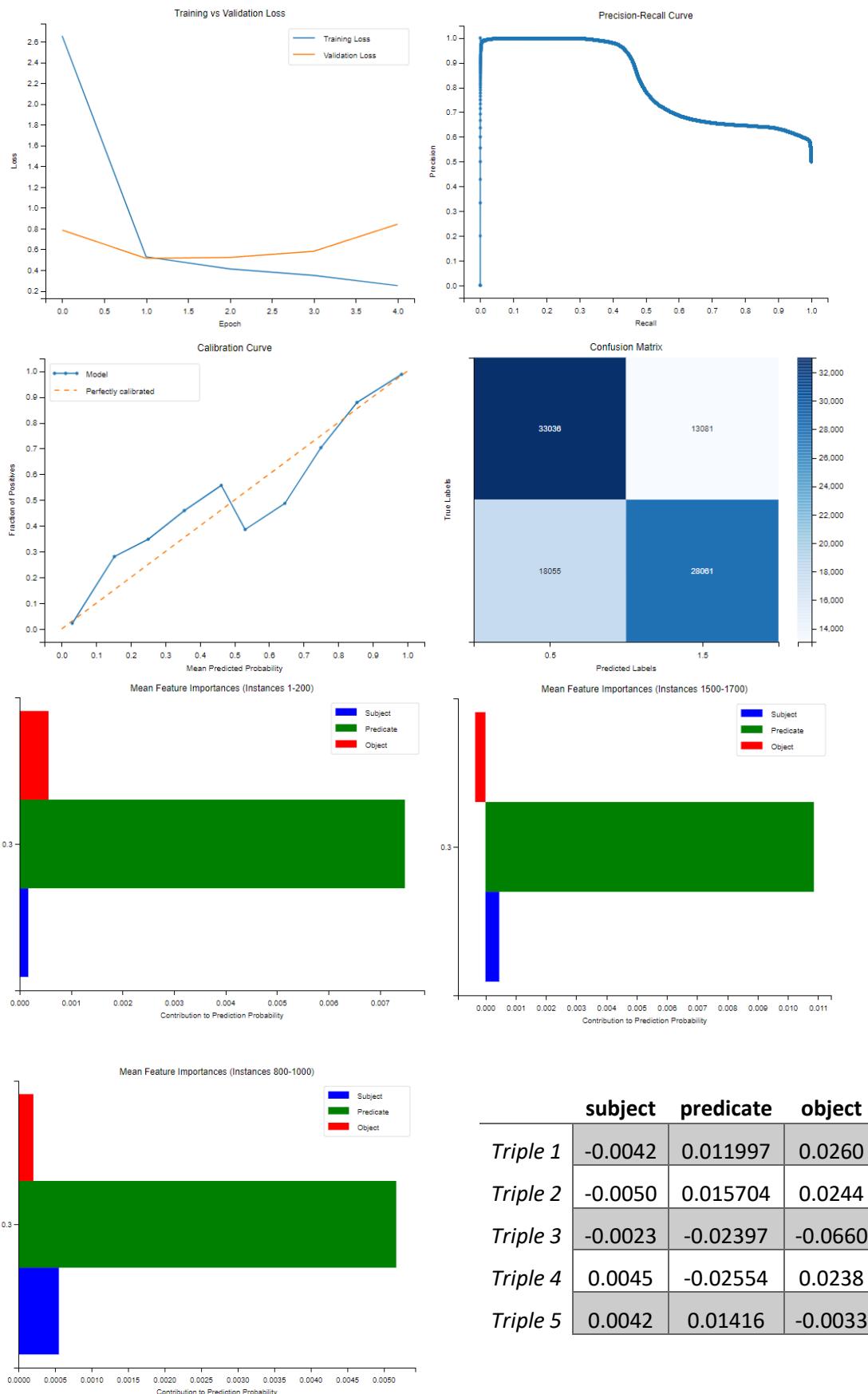
Relation with lowest average probability:

('http://www.w3.org/2002/07/owl#backwardCompatibleWith', 0.16579647843665893)

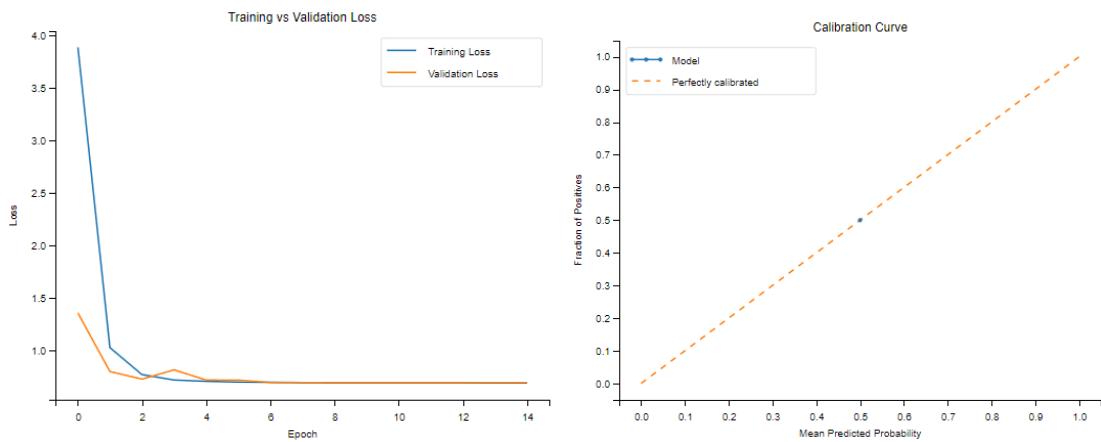
VIII. Group 8

	Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
<i>TransE</i>	0.8091	0.2447	3706.9005
<i>TransH</i>	0.7838	0.2843	3051.8689
<i>RotateE</i>	0.8056	0.2494	3804.2665
<i>NoEmbds</i>	0.8117	0.2402	3664.2924
<i>HoIE</i>	0.7949	0.2663	4074.5816
<i>DistMult</i>	0.7961	0.2659	4306.1863

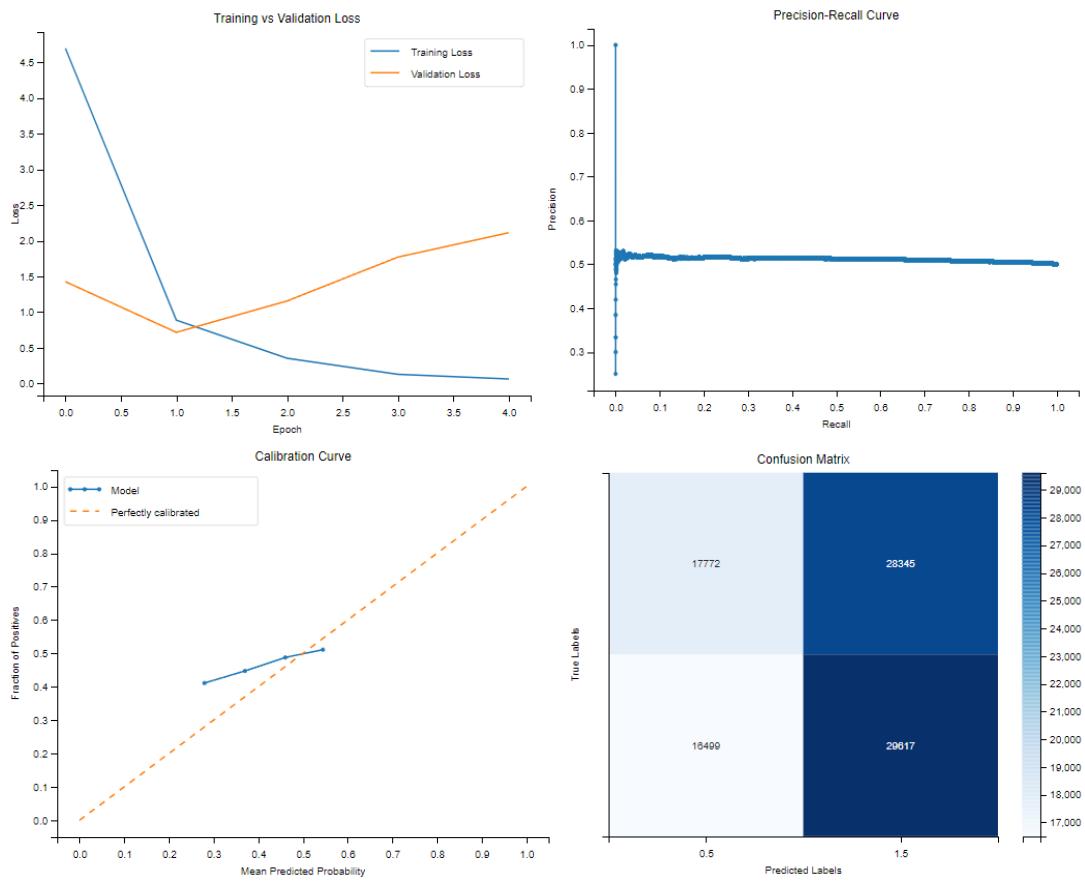
TransE

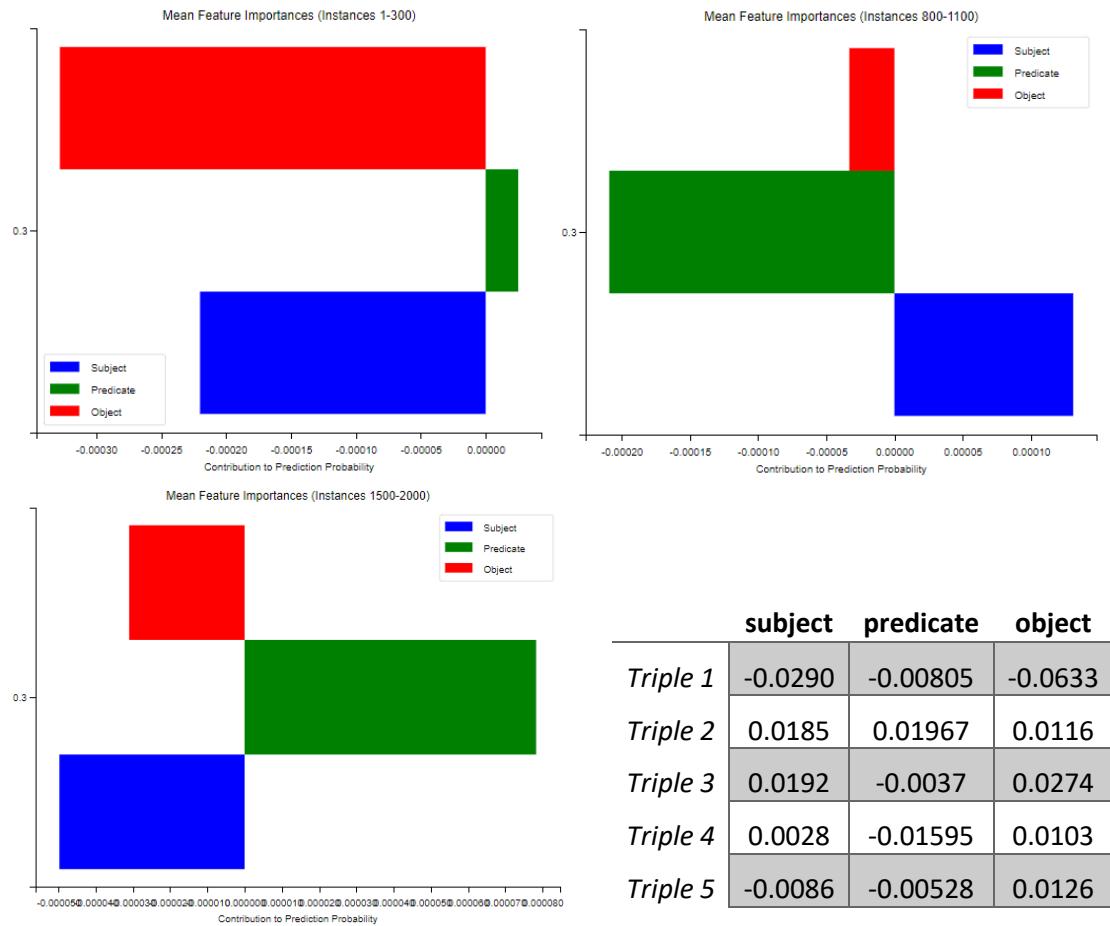


TransH

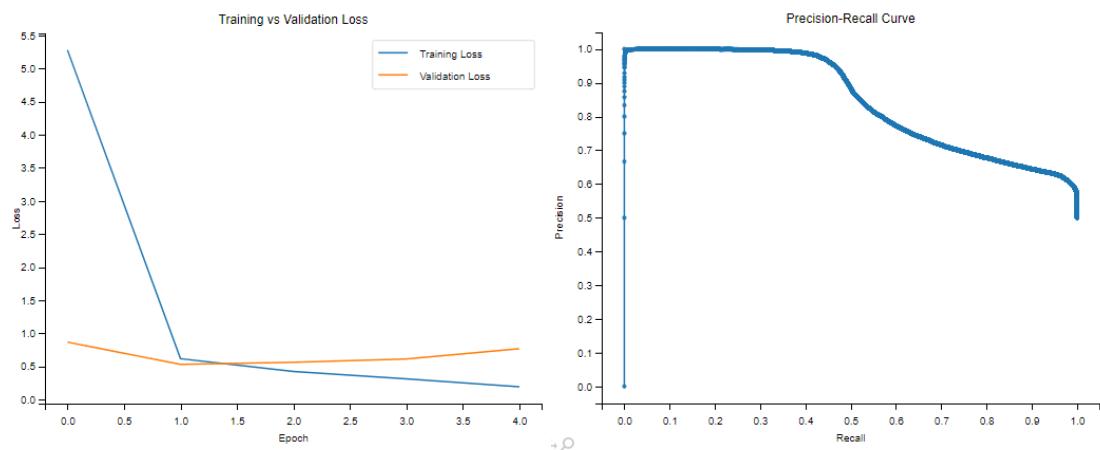


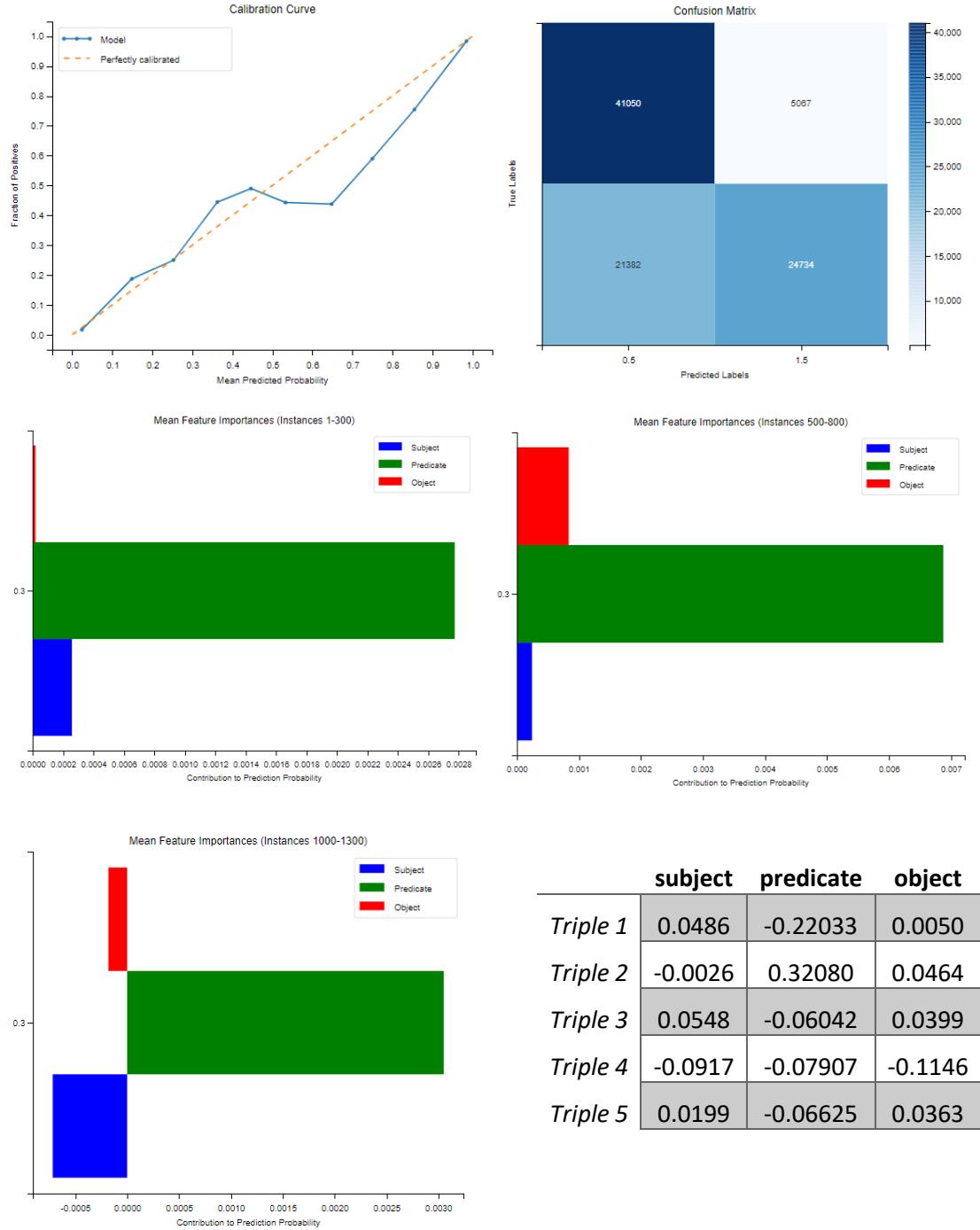
RotateE



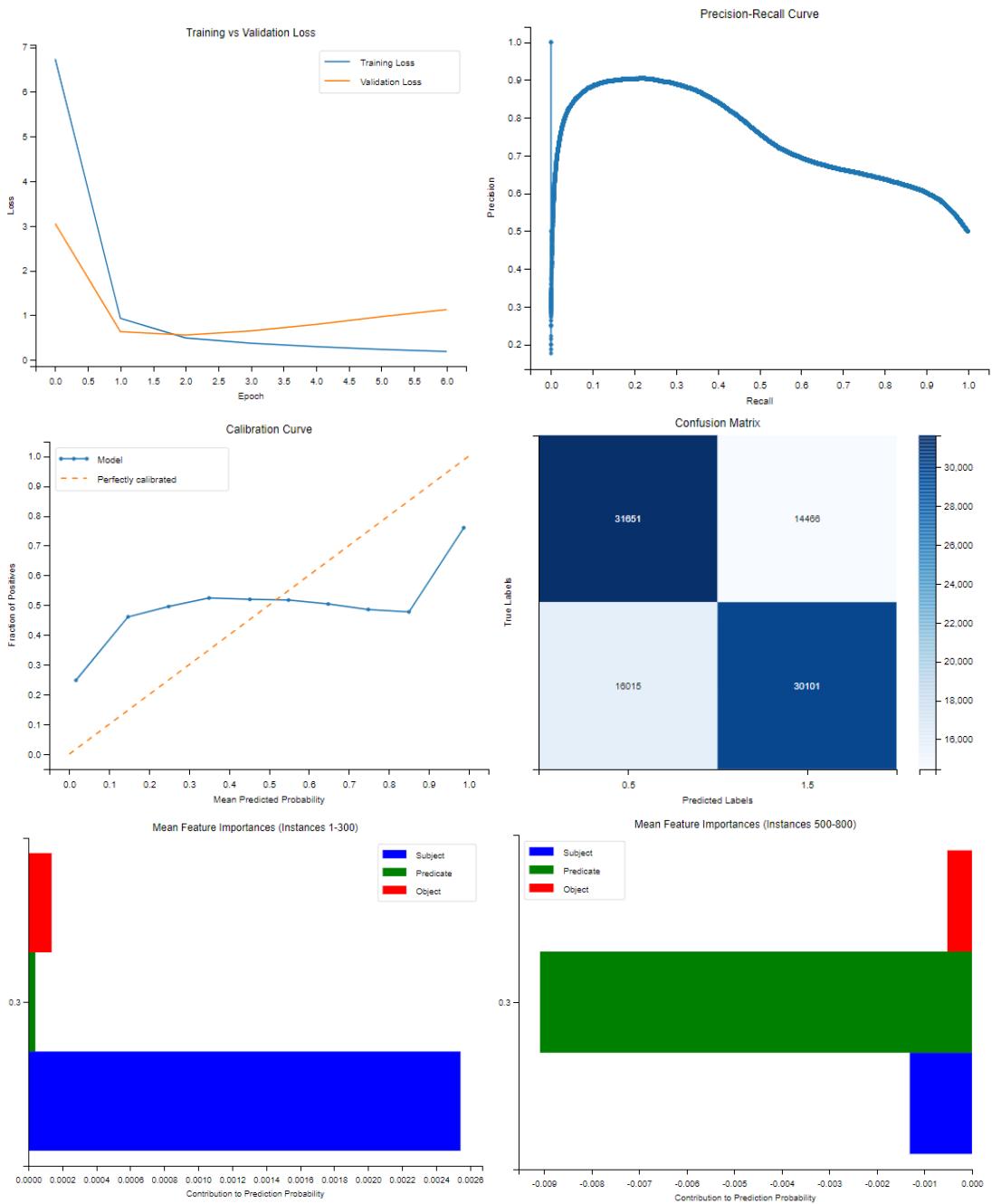


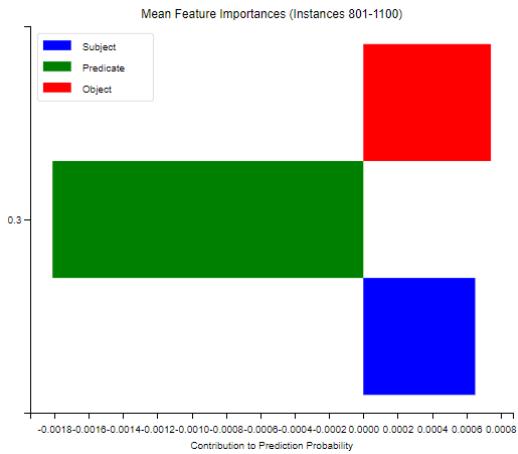
NoEmbds





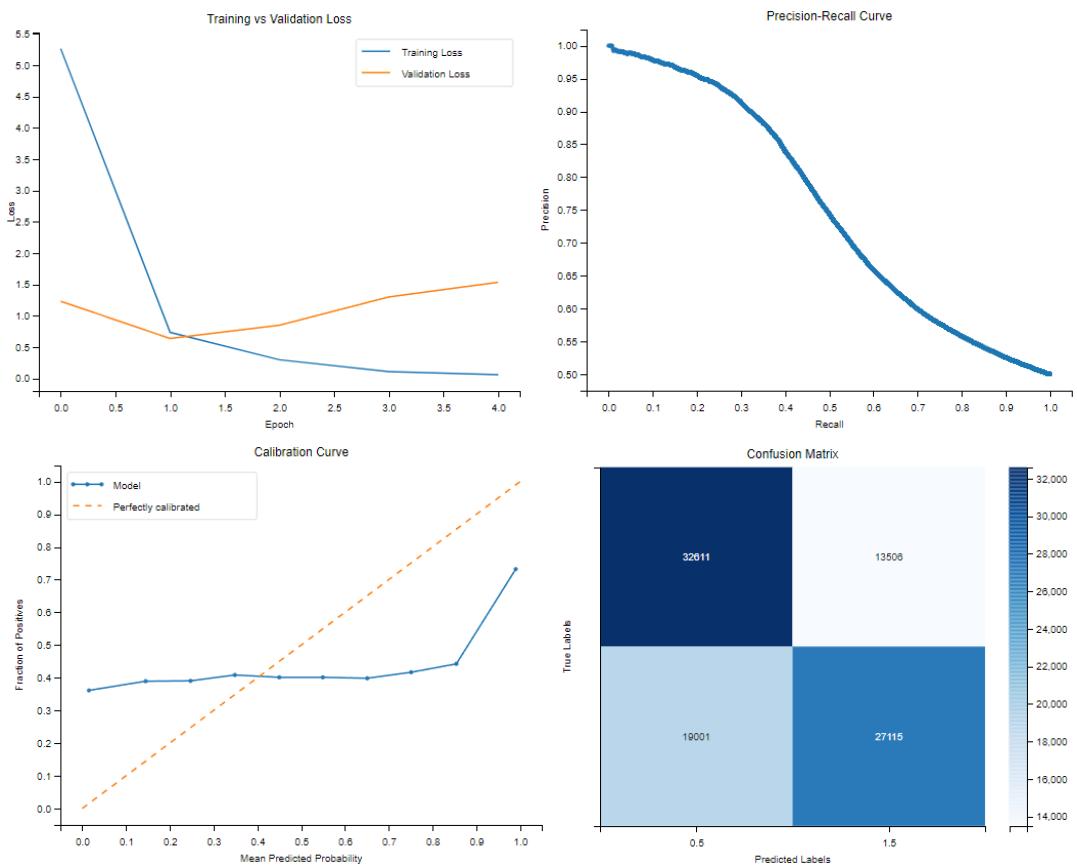
HoIE

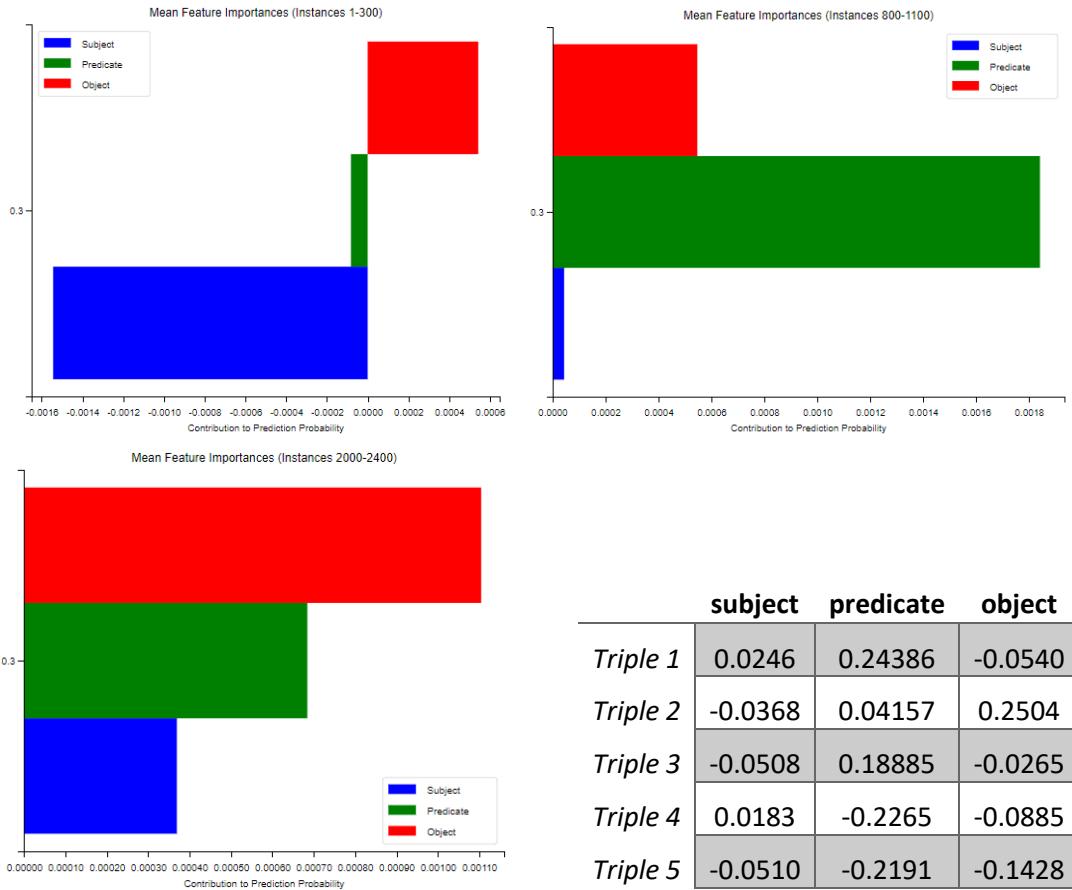




	subject	predicate	object
Triple 1	-0.0254	0.36054	-0.2732
Triple 2	0.0041	-0.48278	0.0560
Triple 3	-0.0383	-0.12850	0.1202
Triple 4	0.0109	0.42805	0.1681
Triple 5	-0.0039	0.01644	0.0384

DistMult





	C1	C2	C3	C4	C5	C6	C7	C8
<i>TransE</i>	11	132	7	2	0.3574	0.9526	0.0015	0.2216
<i>TransH</i>	1	0	0	0	0.5	0.5	0.5	0
<i>RotateE</i>	4	0	0	0	0.5067	0.5729	0.2656	0.0579
<i>NoEmbds</i>	4	193	48	2	0.3518	0.9928	0.0011	0.2427
<i>HoIE</i>	31	501	249	21	0.3970	0.9999	0.0001	0.3726
<i>DistMult</i>	43	360	241	39	0.2813	1.0000	0.0013	0.3796

TransE

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/databasePath', 880),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/id', 255),
 ('http://www.w3.org/2000/01/rdf-schema#comment', 153),
 ('http://www.w3.org/2000/01/rdf-schema#subClassOf', 105),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL', 59)]
```

Relations appearing only once:

```
['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph',
 'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode',
```

'<http://www.w3.org/2000/01/rdf-schema#label>',
'<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm>']

Relation with highest average probability:

('<http://www.w3.org/2000/01/rdf-schema#label>', 0.5816141963005066)

Relation with lowest average probability:

('<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph>',
0.003941344562917948)

RotateE

Top 5 most frequent relations:

[('<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>', 397),
'<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term>', 375),
'<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference>', 366),
'<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level>', 362)

Relation with highest average probability:

('<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term>', 0.5089655706882477)

Relation with lowest average probability:

('<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference>',
0.5016641690105689)

NoEmbds

Top 5 most frequent relations:

[('<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level>', 1040),
'<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term>', 445),
'<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>', 11),
'<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference>', 4)]

Relation with highest average probability:

('<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference>',
0.5648228228092194)

Relation with lowest average probability:

('<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>', 0.2560354122384028)

HoIE

Top 5 most frequent relations:

[('<http://www.w3.org/2002/07/owl#backwardCompatibleWith>', 206),
'<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph>', 175),
'<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/dataSource>', 165),
'<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode>', 139),
'<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink>', 126)]

Relations appearing only once:

['<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/keyword>',
'<http://www.w3.org/2002/07/owl#equivalentProperty>',

'<http://www.w3.org/2000/01/rdf-schema#label>',
 '<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/context>',
 '<http://www.w3.org/2000/01/rdf-schema#comment>',
 '<http://www.w3.org/2002/07/owl#imports>']

Relation with highest average probability:

('<http://www.w3.org/2000/01/rdf-schema#label>', 0.9927085041999817)

Relation with lowest average probability:

('<http://www.w3.org/2002/07/owl#equivalentProperty>', 0.0012489722575992346)

DistMult

Top 5 most frequent relations:

[('<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference>', 90),
 ('<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/databasePath>', 80),
 ('<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/id>', 80),
 ('<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/title>', 79),
 ('<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL>', 78)]

Relation with highest average probability:

('<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/remark>', 0.5627538235328923)

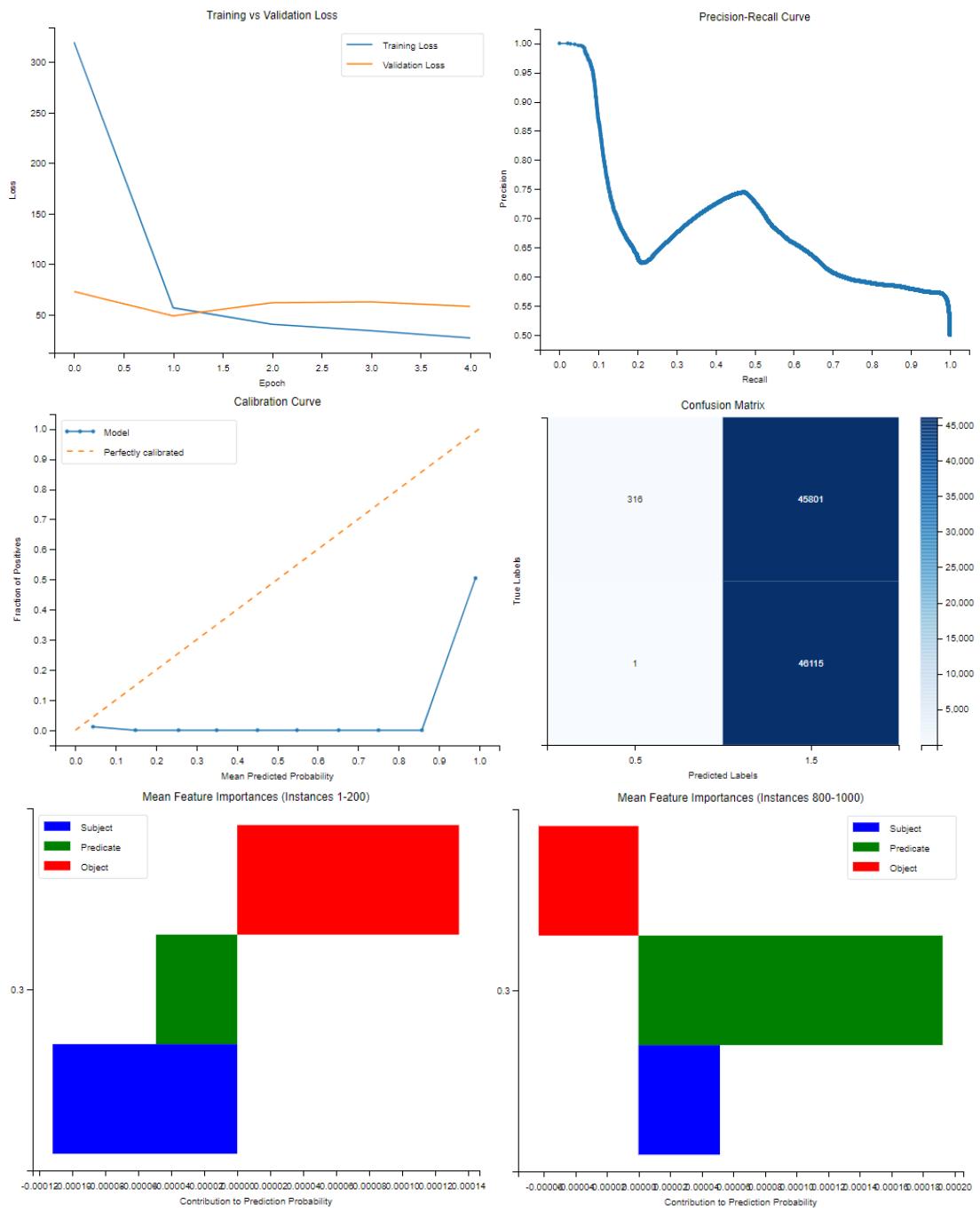
Relation with lowest average probability:

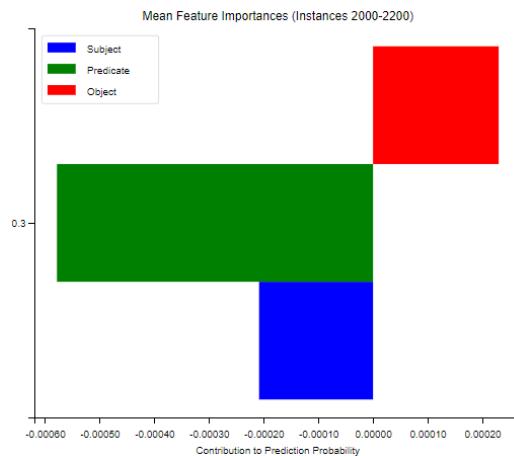
('<https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode>', 0.06724924116861075)

IX. Group 9

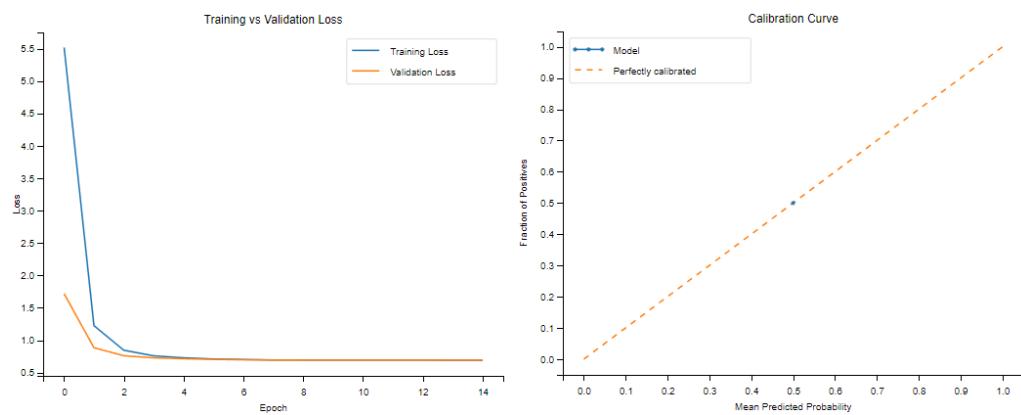
	Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
<i>TransE</i>	0.8156	0.2357	3820.1842
<i>TransH</i>	0.7869	0.2806	3109.6894
<i>RotatE</i>	0.8088	0.2457	3977.7726
<i>NoEmbds</i>	0.8168	0.2351	3893.3948
<i>HoIE</i>	0.8110	0.2450	3699.0573
<i>DistMult</i>	0.7966	0.2621	3555.6123

TransE

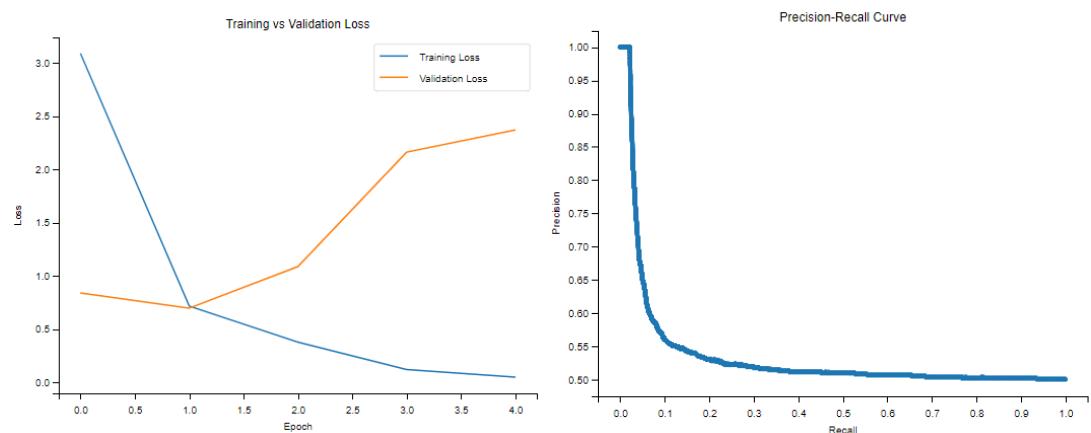


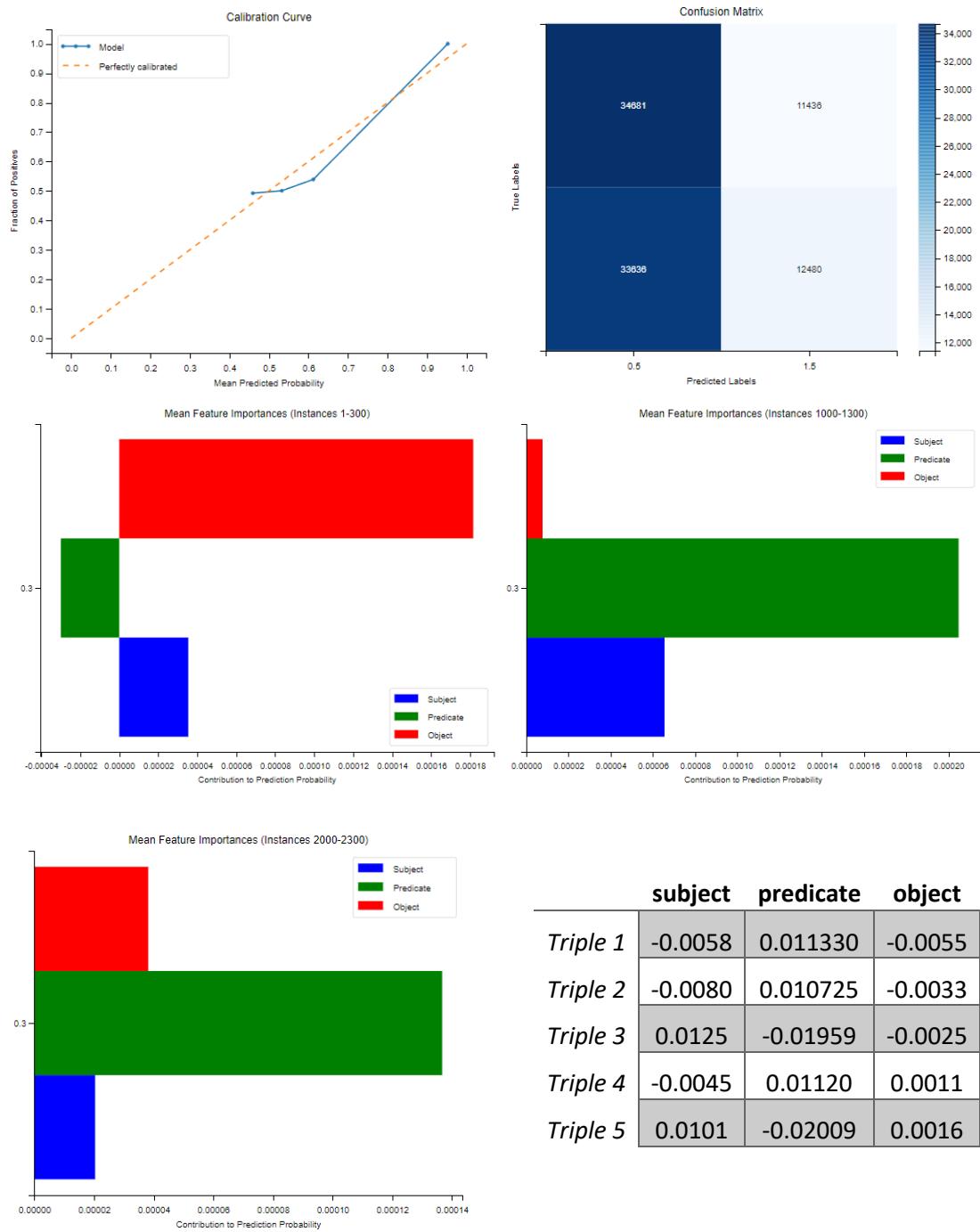


TransH

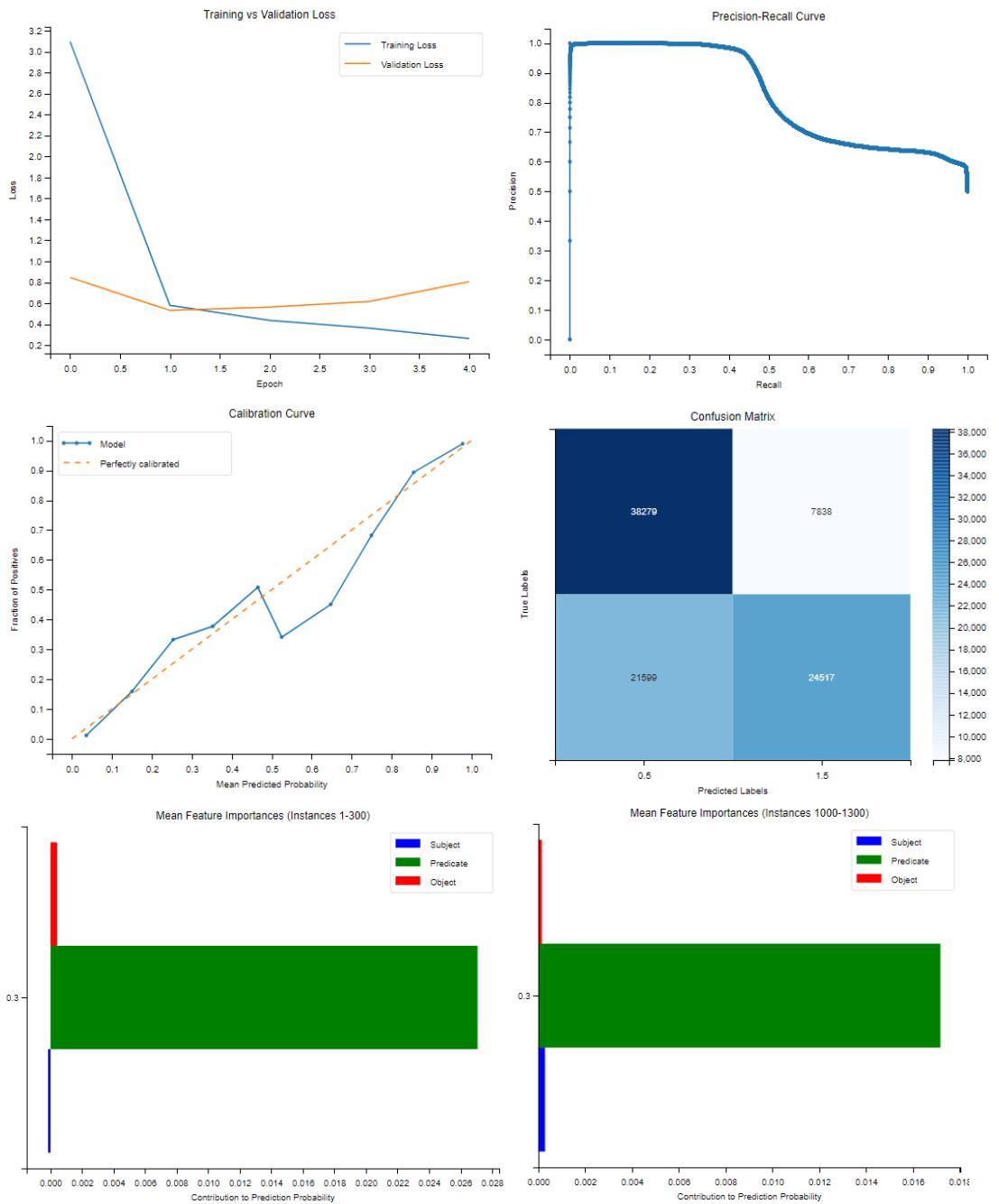


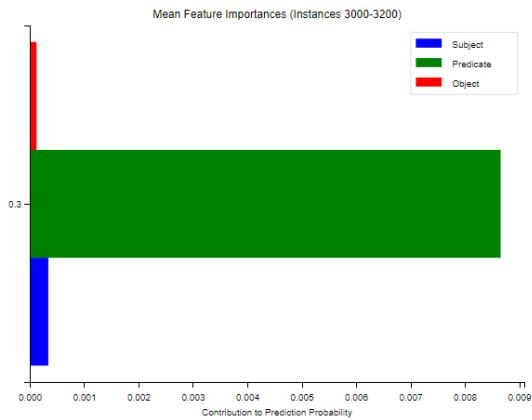
RotateE





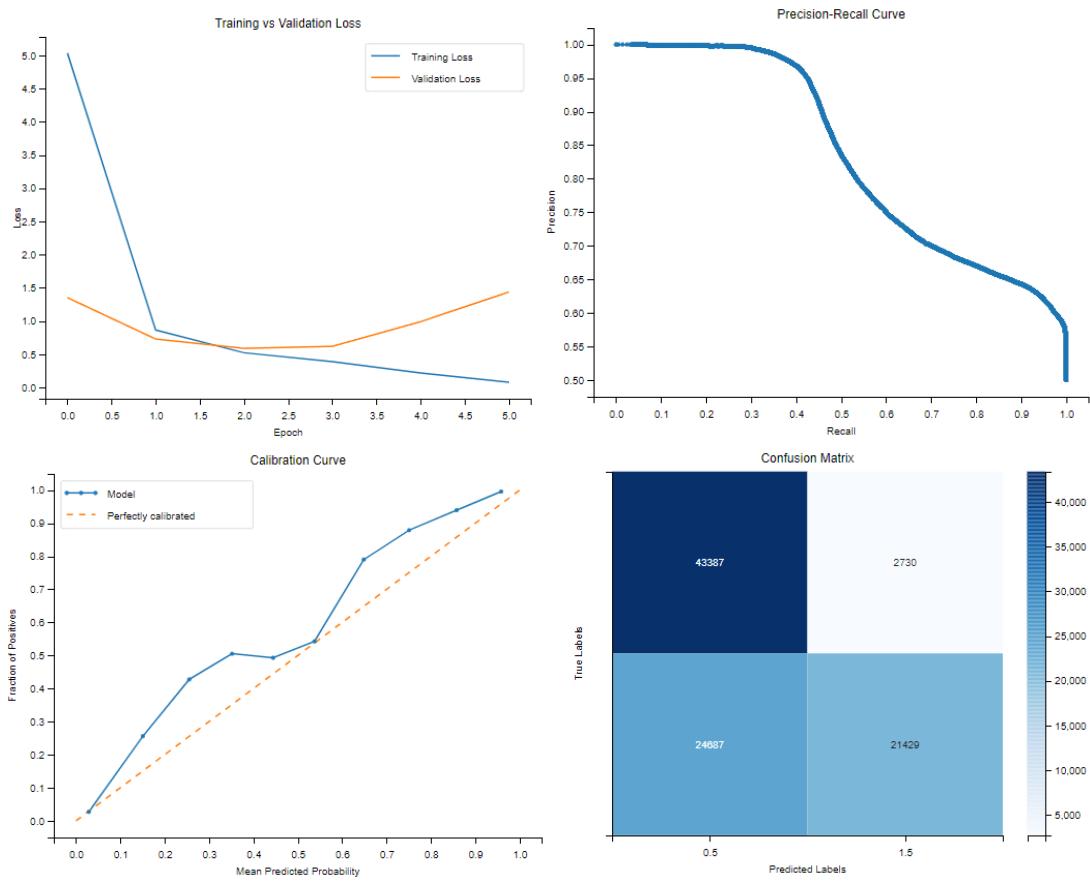
NoEmbds

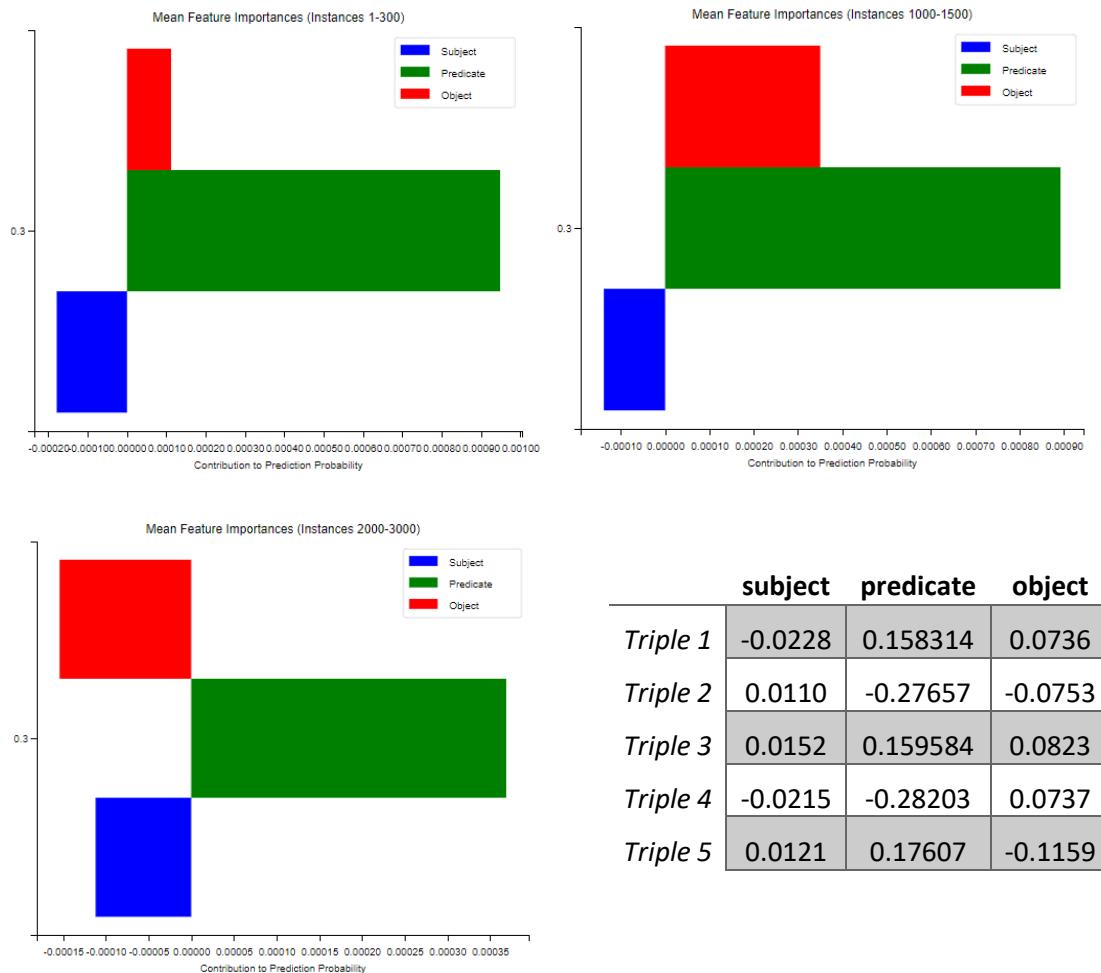




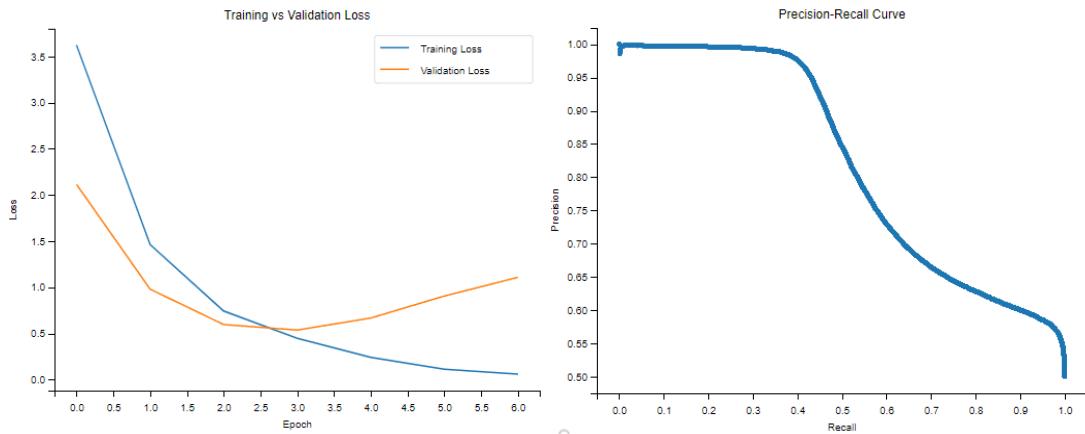
	subject	predicate	object
Triple 1	0.0119	-0.42507	0.0979
Triple 2	-0.0064	-0.41864	-0.0254
Triple 3	0.0016	0.34161	-0.0247
Triple 4	-0.0146	0.28965	-0.0199
Triple 5	0.0124	-0.4198	-0.0332

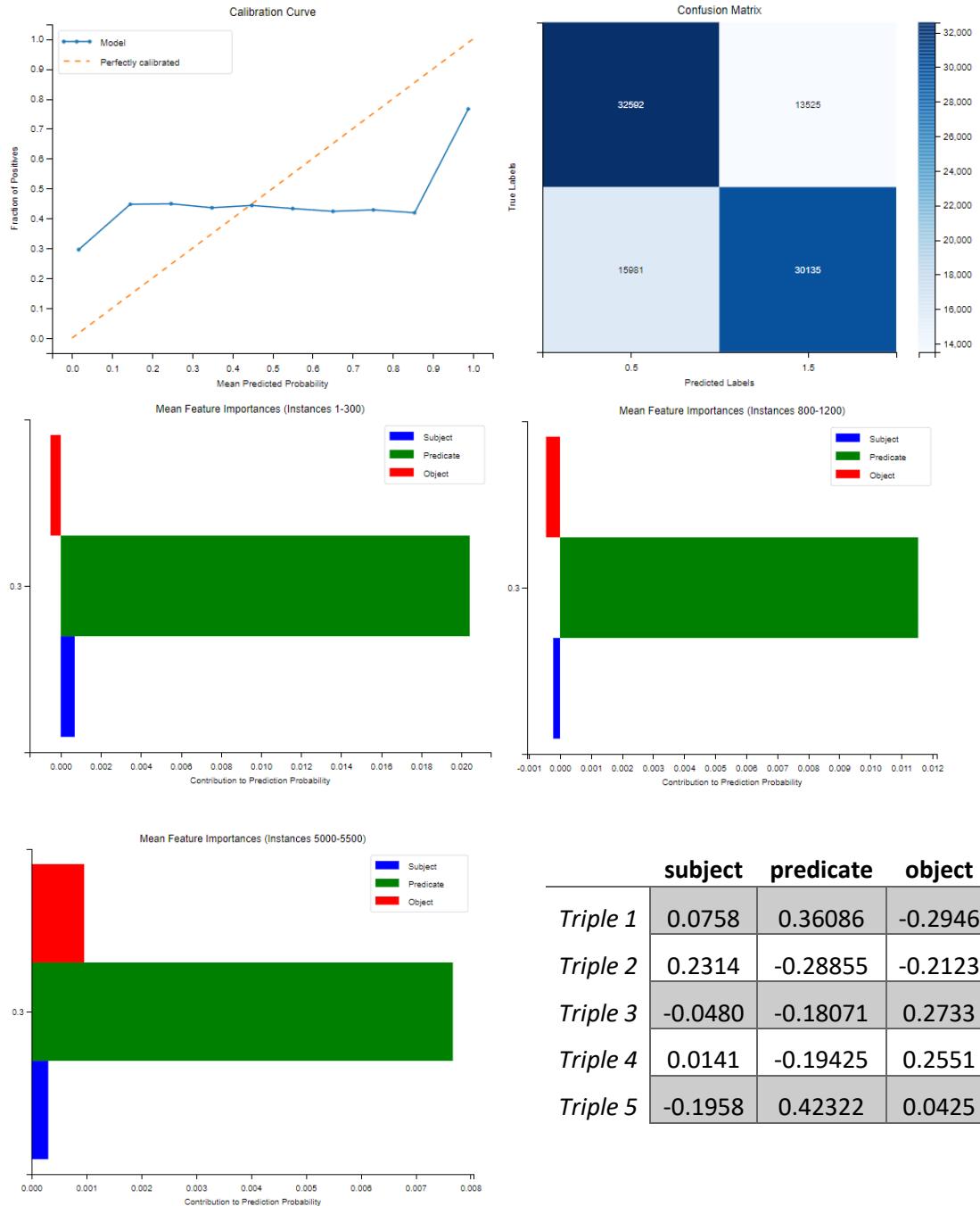
HoIE





DistMult





	C1	C2	C3	C4	C5	C6	C7	C8
<i>TransE</i>	3	1488	1480	3	0.9876	1.0000	0.0125	0.0622
<i>TransH</i>	1	0	0	0	0.5	0.5	0.5	0
<i>RotateE</i>	4	1	0	0	0.4774	0.6251	0.4411	0.0364
<i>NoEmbds</i>	7	48	0	0	0.3528	0.8432	0.0023	0.1859
<i>HoIE</i>	16	4	0	0	0.2376	0.6661	0.0008	0.1391
<i>DistMult</i>	4	405	243	4	0.3173	0.9995	0.0006	0.3777

TransE

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL', 1200),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasTopic', 298),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCategoryOfGlossaryArticle', 2)]

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasTopic', 0.9889348158300323)

Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCategoryOfGlossaryArticle', 0.9263133108615875)

RotatE

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 407),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 377),
('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 358),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 358)]

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 0.47852507566606534)

Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 0.4756814282414954)

NoEmbds

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/databasePath', 968),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 480),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm', 25),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/remark', 13),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/id', 12)]

Relations appearing only once:

['http://www.w3.org/2000/01/rdf-schema#label',
'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph']

Relation with highest average probability:

('http://www.w3.org/2000/01/rdf-schema#label', 0.4436992108821869)

Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/id', 0.3281816483164827)

HoIE

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 788),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/definition', 407),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 64),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm', 63),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph', 51)]
```

Relations appearing only once:

```
['http://www.w3.org/2000/01/rdf-schema#comment']
```

Relation with highest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/remark', 0.353840708732605)
```

Relation with lowest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/keyword', 0.09458667249418795)
```

DistMult

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 570),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 520),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 253),
('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 157)]
```

Relation with highest average probability:

```
('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 0.374340466957741)
```

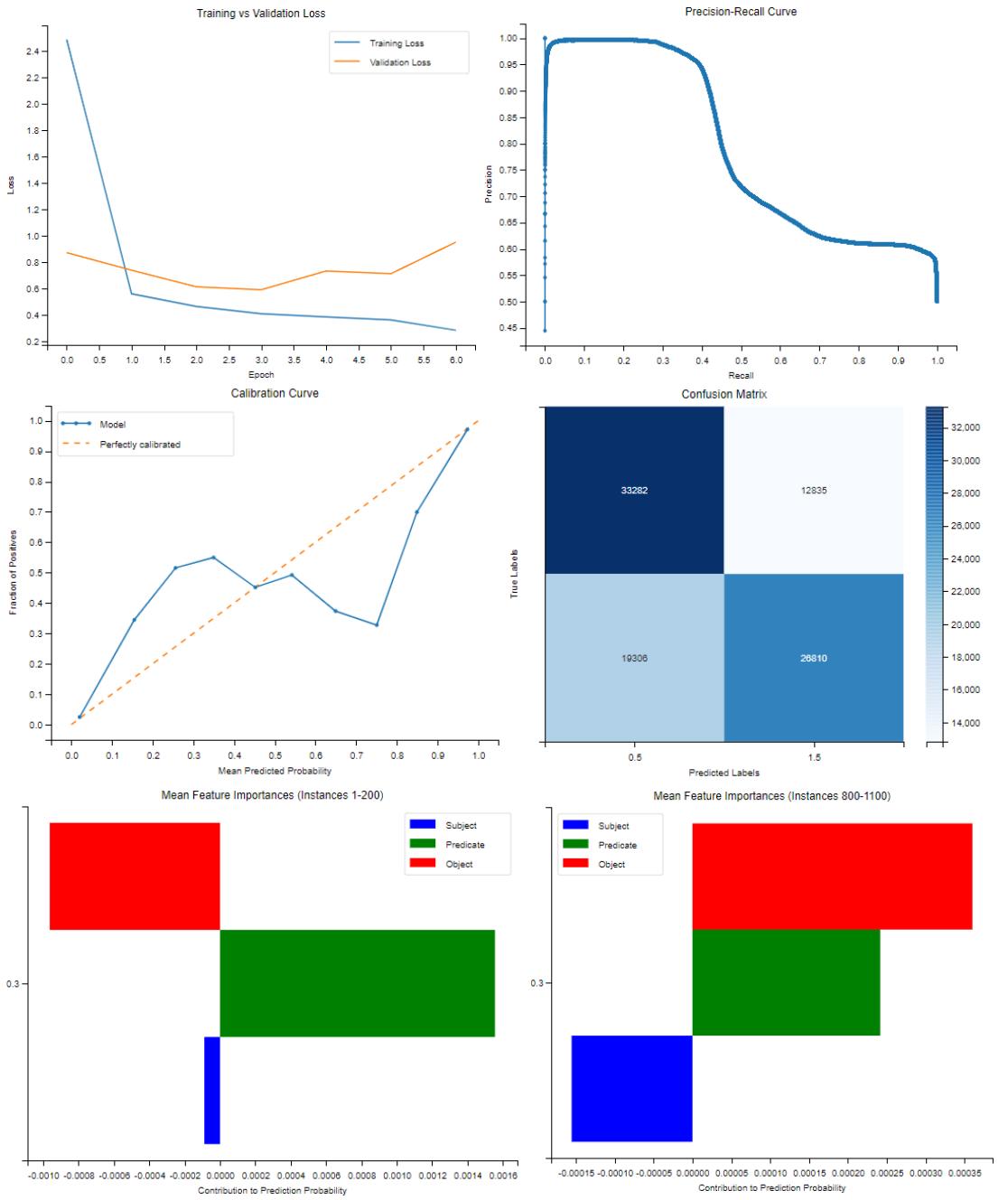
Relation with lowest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 0.28744966788094295)
```

X. Group 10

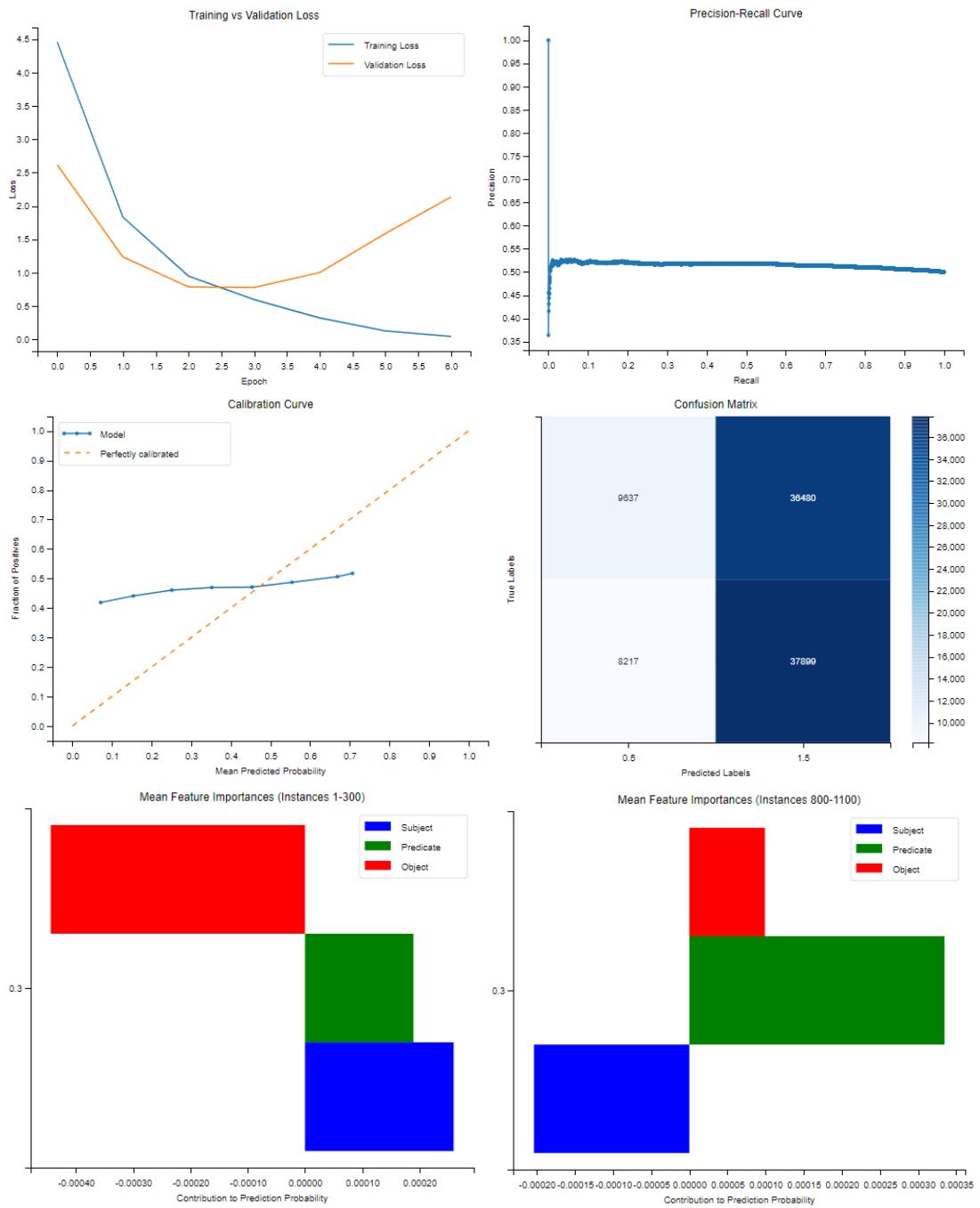
	Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
<i>TransE</i>	0.8033	0.2534	3409.4346
<i>RotateE</i>	0.8038	0.2534	3572.9810
<i>NoEmbds</i>	0.8091	0.2453	3726.2432
<i>HoIE</i>	0.8111	0.2409	3734.2604
<i>DistMult</i>	0.7835	0.2864	2610.8468

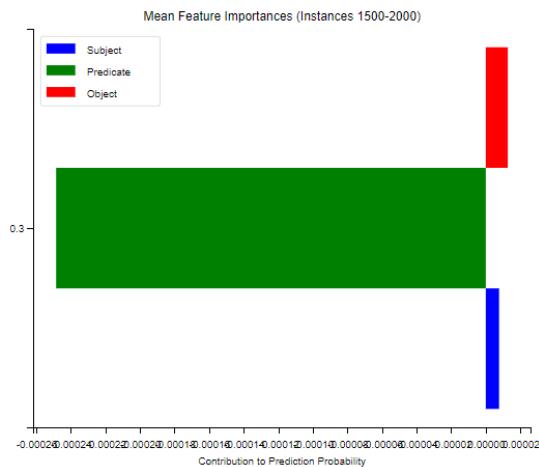
TransE



	subject	predicate	object
<i>Triple 1</i>	0.00238	-0.52889	0.15306
<i>Triple 2</i>	-0.03335	0.30291	-0.10603
<i>Triple 3</i>	-0.01008	0.29046	-0.08580
<i>Triple 4</i>	0.01318	0.30933	-0.19713
<i>Triple 5</i>	0.01228	-0.54333	0.13565

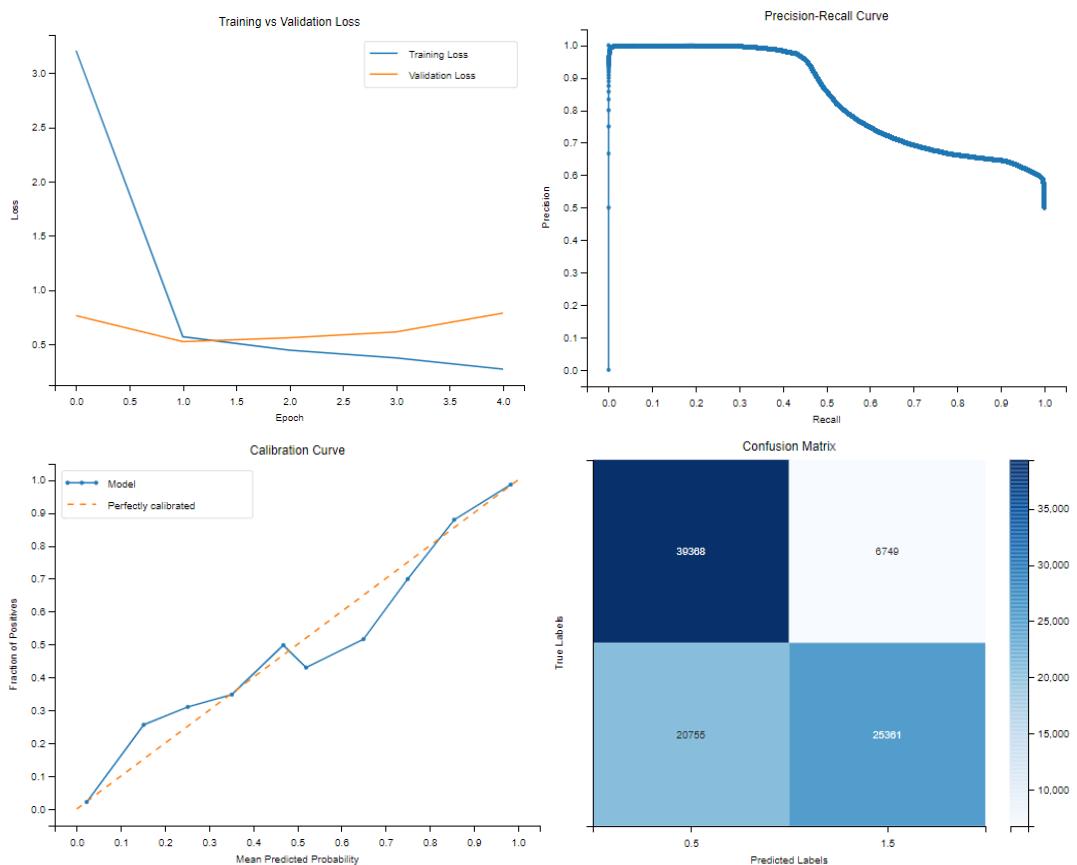
RotateE

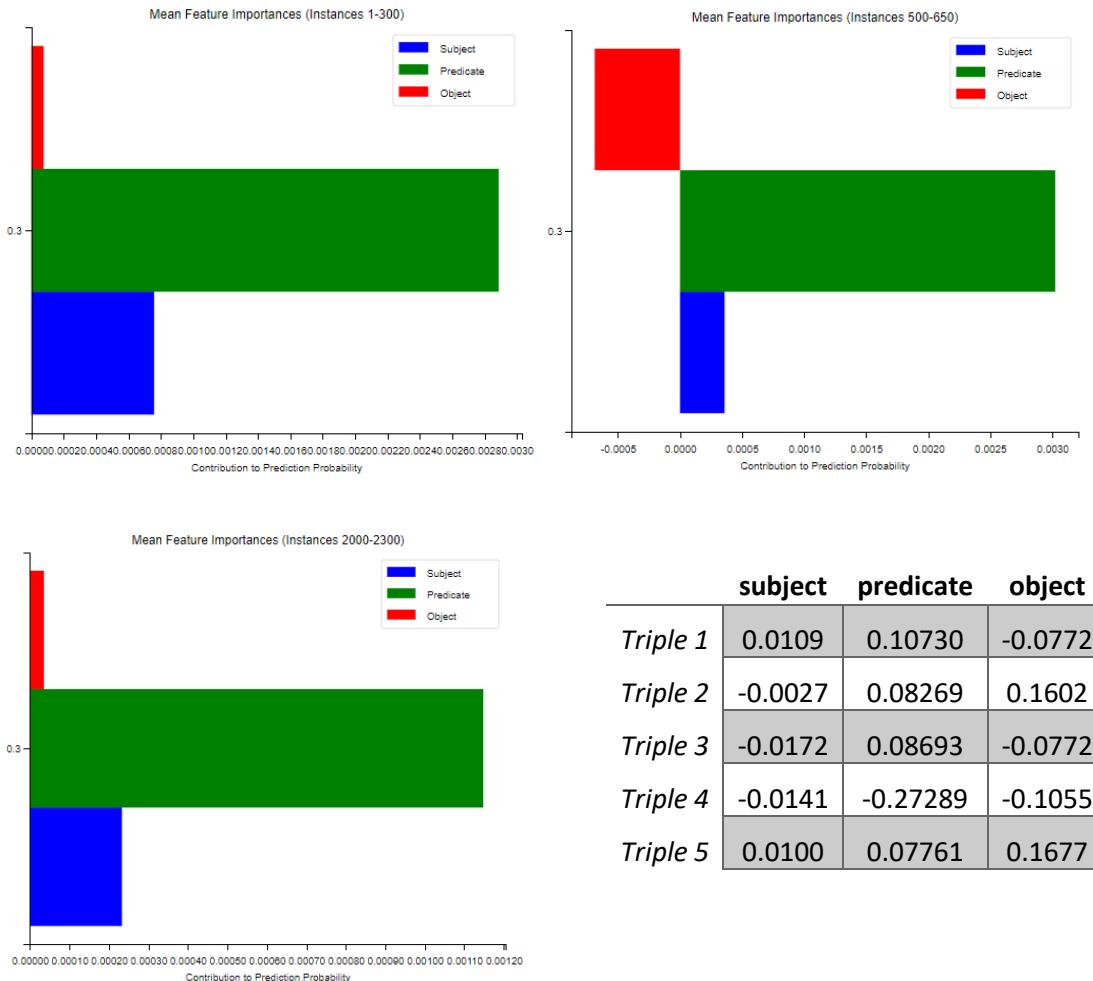




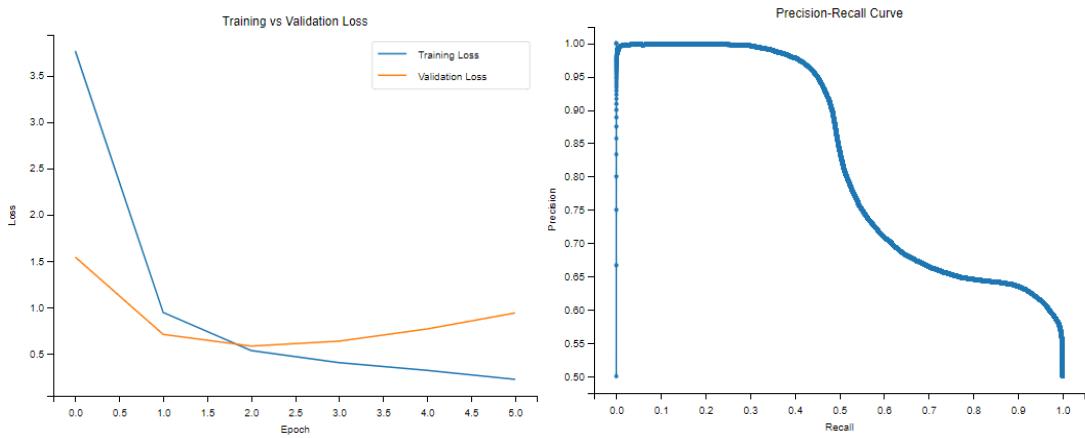
	subject	predicate	object
Triple 1	-0.0218	0.05379	-0.0389
Triple 2	0.0215	-0.03036	-0.0791
Triple 3	-0.0235	-0.05236	0.0226
Triple 4	0.0862	-0.01037	0.0438
Triple 5	-0.0222	0.05824	0.0549

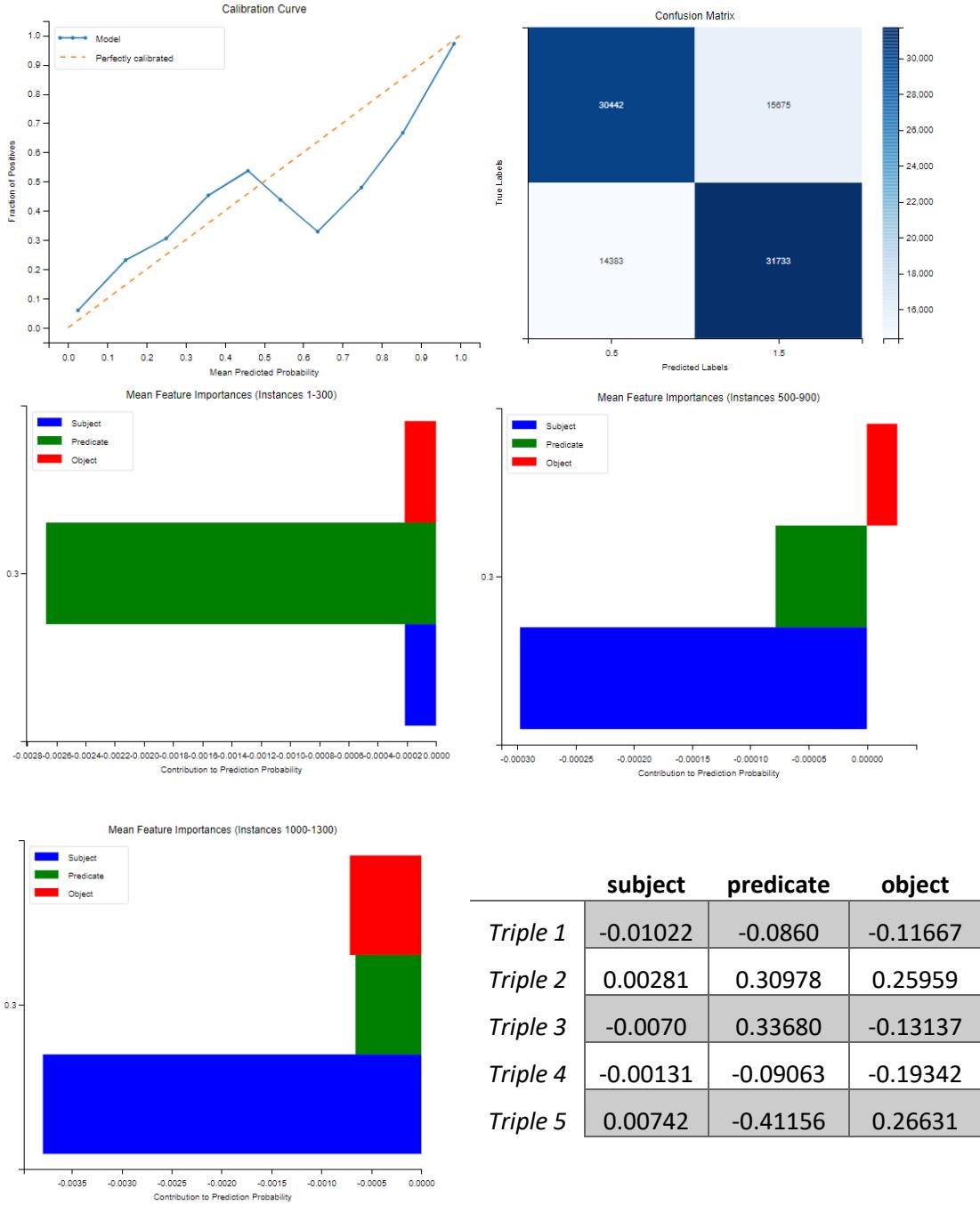
NoEmbds



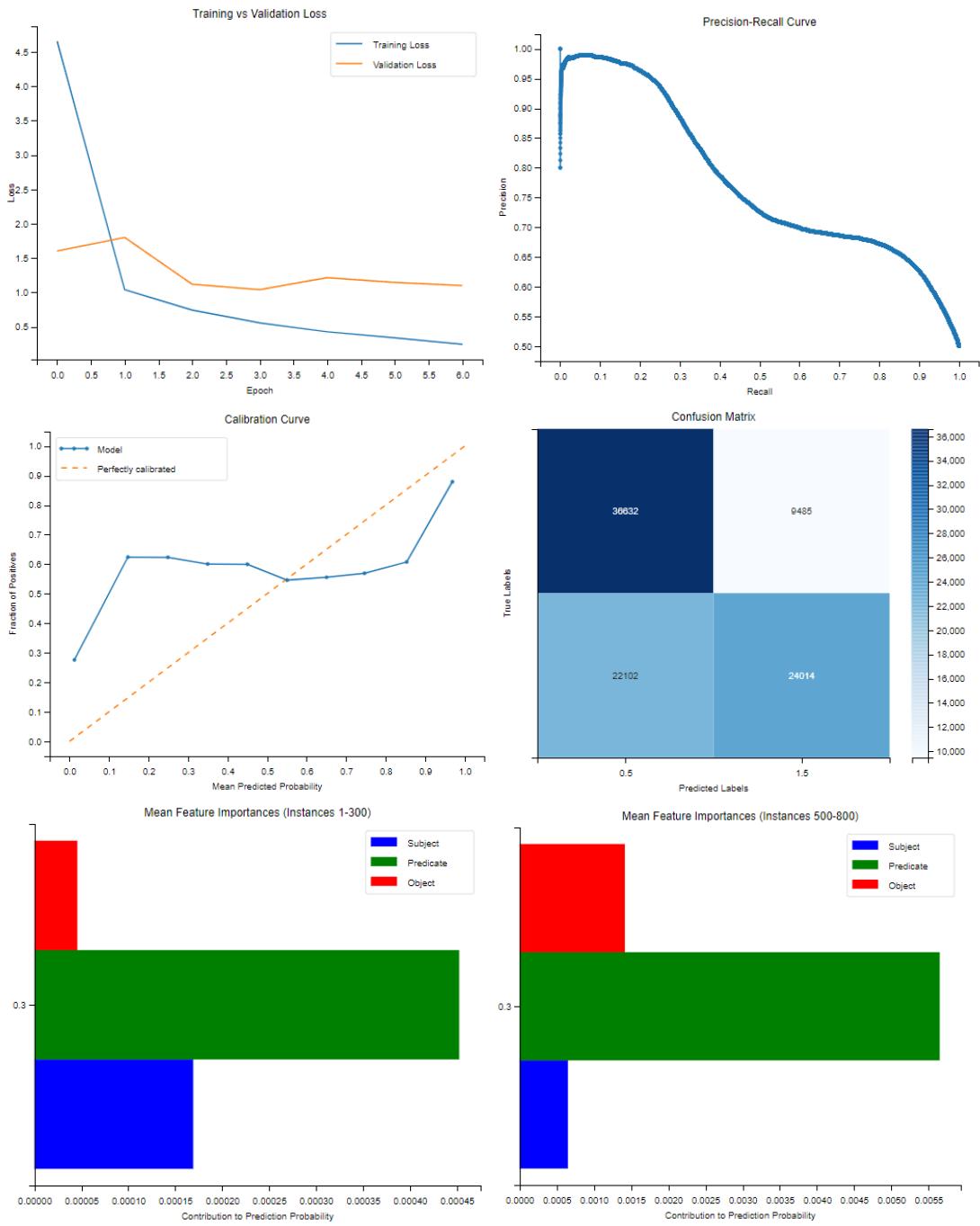


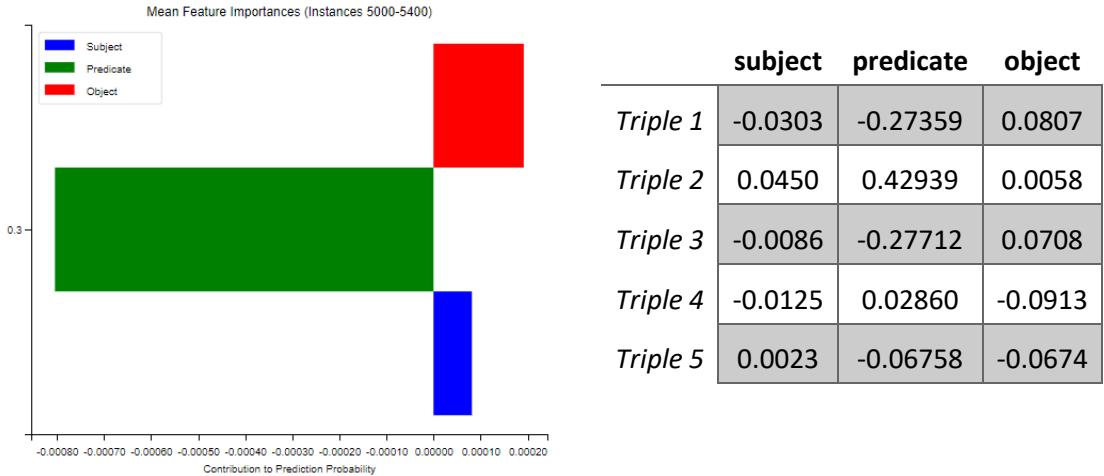
HoIE





DistMult





	C1	C2	C3	C4	C5	C6	C7	C8
<i>TransE</i>	4	200	0	0	0.3217	0.8646	0.0008	0.2288
<i>RotatE</i>	4	1046	0	0	0.6000	0.7093	0.0311	0.1664
<i>NoEmbds</i>	5	70	9	3	0.3306	0.9496	0.0007	0.2144
<i>HoIE</i>	3	241	15	1	0.3399	0.9980	0.0001	0.2610
<i>DistMult</i>	3	250	72	2	0.2341	0.9949	0.0001	0.3235

TransE

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/databasePath', 1193),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/id', 179),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm', 98),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 30)]

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm', 0.3323723630794343)

Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/id', 0.3133362166989366)

RotatE

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 421),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 399),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 359),
 ('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 321)]

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 0.603123583039619)

Relation with lowest average probability:

('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 0.5962778416250446)

NoEmbds

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph', 888),
('http://www.w3.org/2002/07/owl#unionOf', 232),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm', 204),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/id', 165),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/title', 11)]

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/title', 0.3658370489508591)

Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/id', 0.31402607393013593)

HoIE

Top 5 most frequent relations:

[('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 1180),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 306),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 6)]

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 0.35716361697782034)

Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 0.3028070529301961)

DistMult

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 1196),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 35),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 1)]

Relations appearing only once:

['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term']

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 0.23641097045408993)

Relation with lowest average probability:

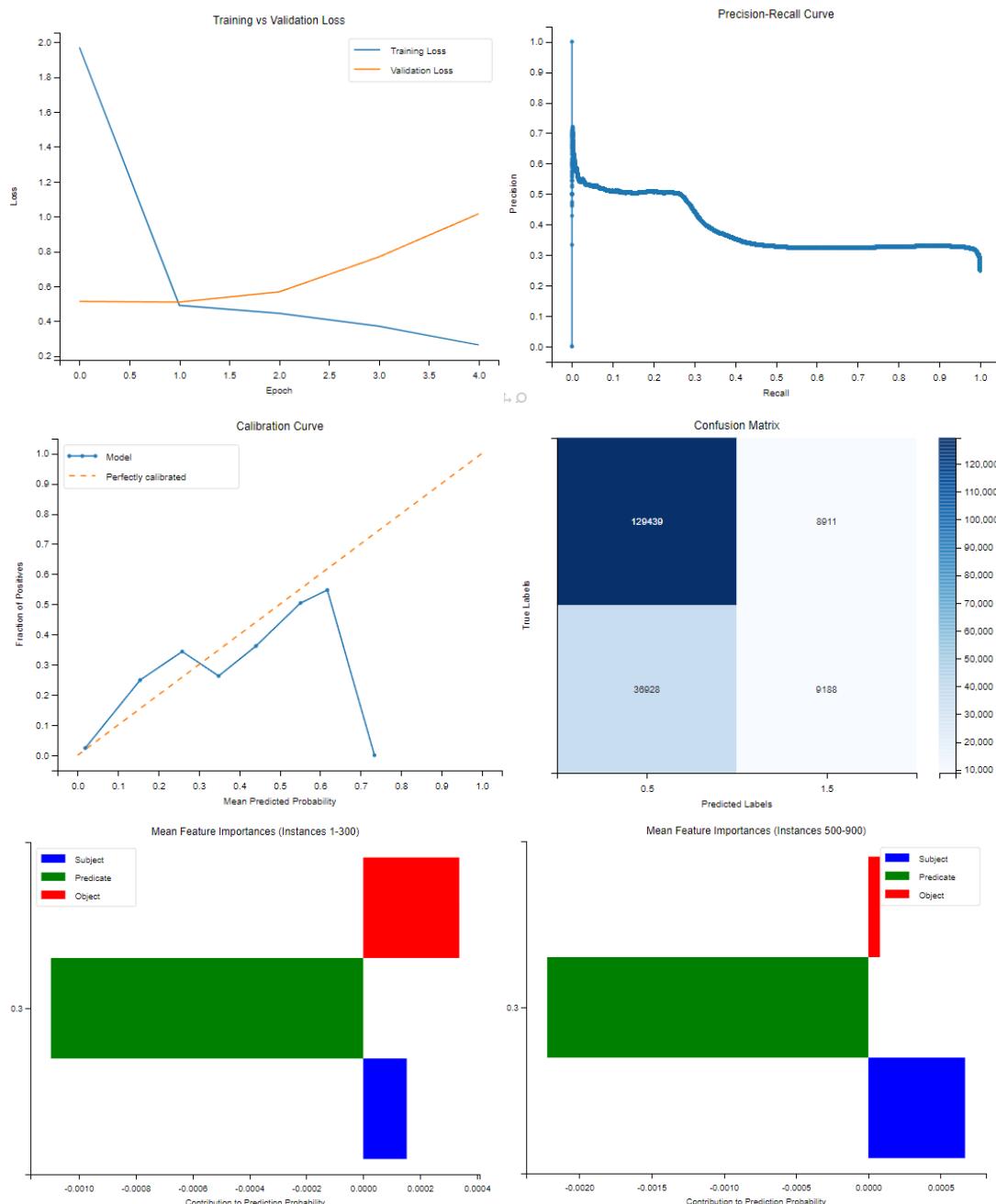
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 0.12646132707595825)

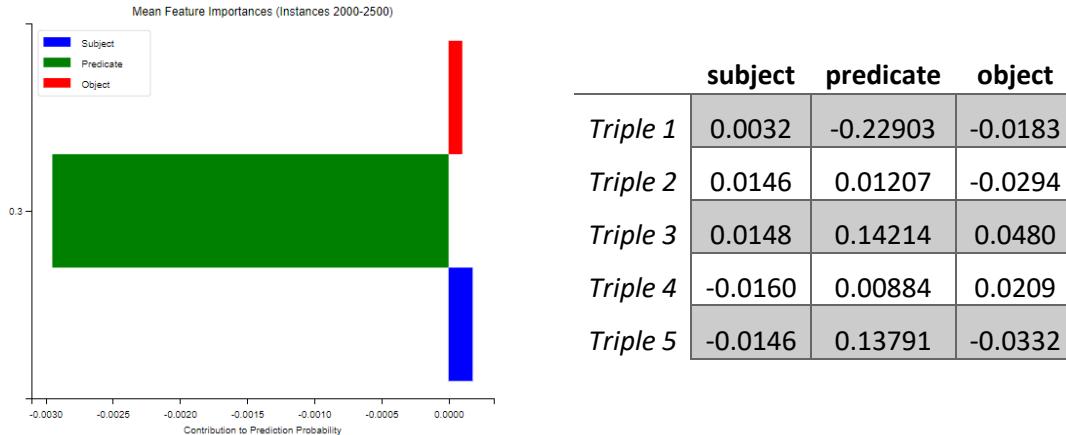
XI. Experiment 1

Mean Silhouette Mean Davies-Bouldin Mean Calinski-Harabasz

0.7622	0.2842	3048.9623
--------	--------	-----------

C1	C2	C3	C4	C5	C6	C7	C8
16	0	0	0	0.1436	0.4584	0.0001	0.4584





Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 359),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/definition', 286),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL', 188),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 77),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/title', 53)]
```

Relations appearing only once:

```
['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/sourcePublication']
```

Relation with highest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/sourcePublication',
 0.36843347549438477)
```

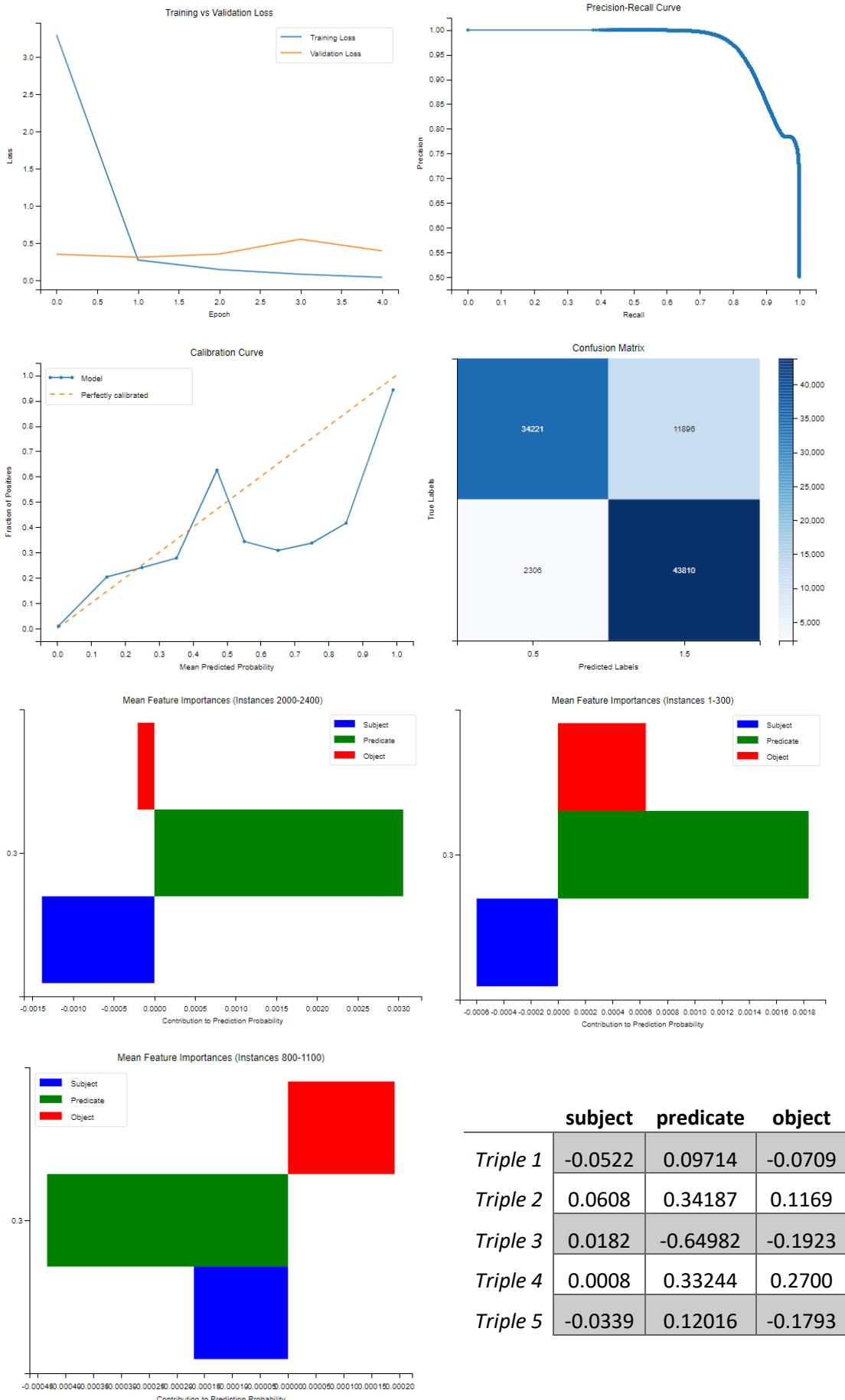
Relation with lowest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/dateUpdated',
 0.032192379236221313)
```

XII. Experiment 2

Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
0.7266	0.2940	2156.5329

C1	C2	C3	C4	C5	C6	C7	C8
25	510	331	21	0.5692	1.0000	0.0000	0.3720



Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/definition', 283),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 199),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 157),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/content', 72),
 ('http://www.w3.org/2000/01/rdf-schema#label', 64)]

Relations appearing only once:

['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/context',
 'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasTopic']

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCategoryOfGlossaryArticle',
 0.7590611591935158)

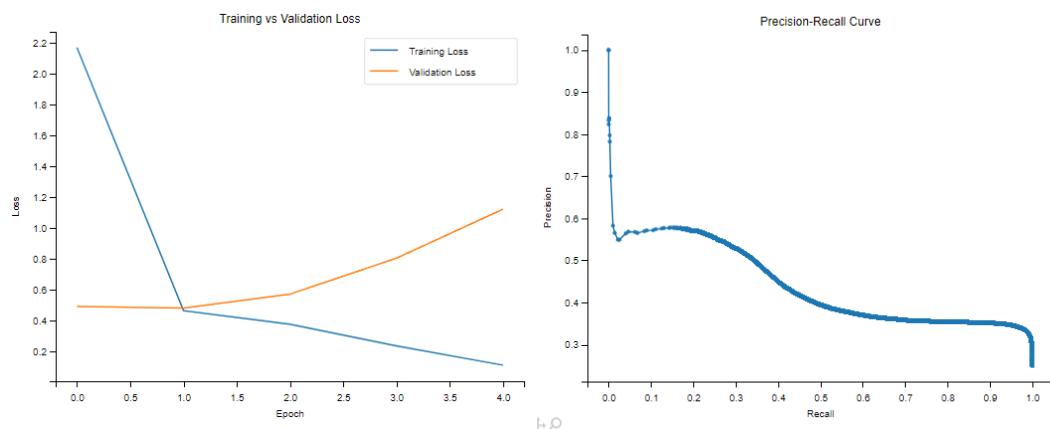
Relation with lowest average probability:

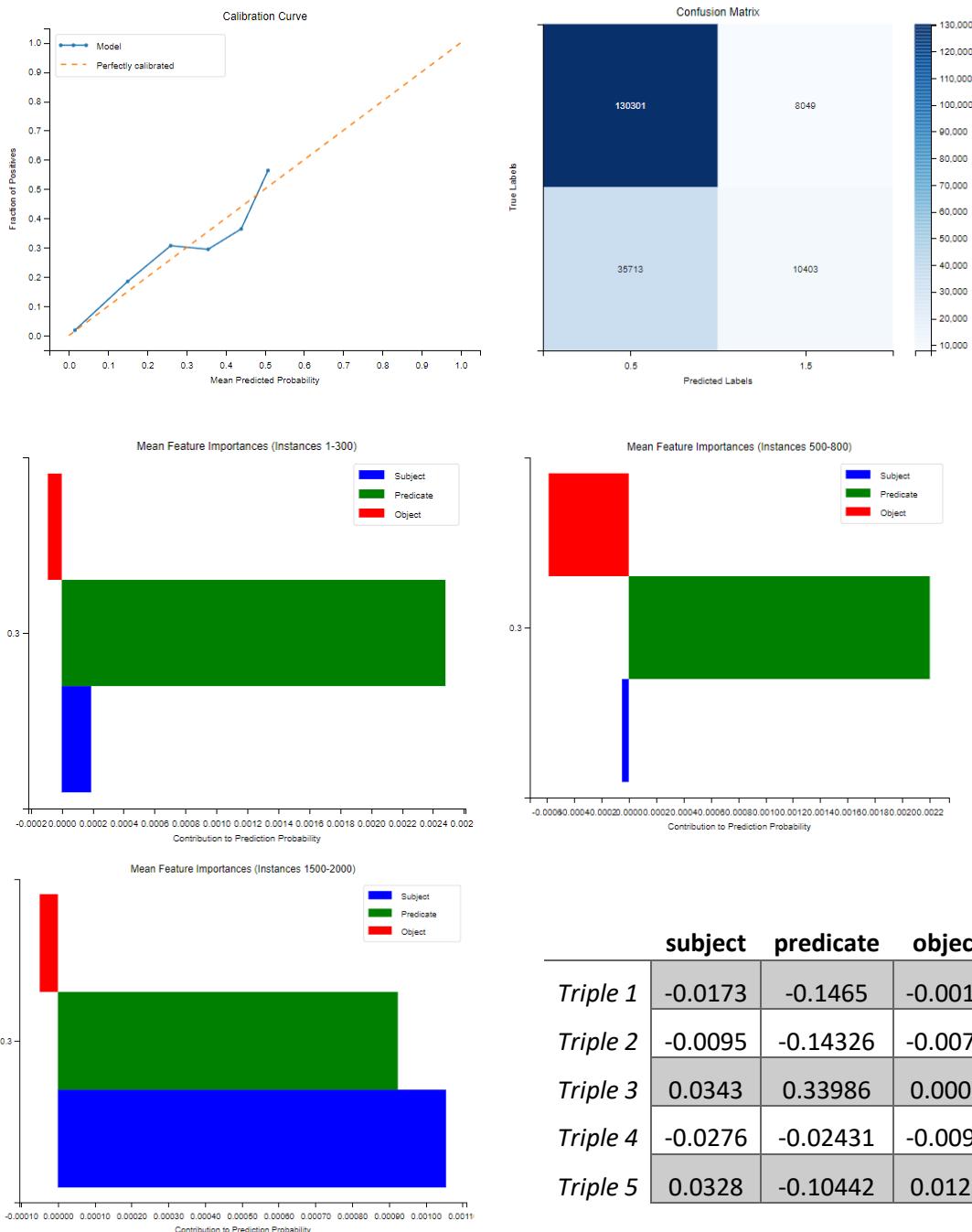
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 0.12462425937216419)

XIII. Experiment 3

Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
0.7541	0.2978	2558.1597

C1	C2	C3	C4	C5	C6	C7	C8
16	0	0	0	0.1592	0.4830	0.0001	0.1557





Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURL', 471),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 294),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 107),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 41),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/title', 38)]
```

Relations appearing only once:

```
['http://www.w3.org/2000/01/rdf-schema#label',
'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink',
'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/dateUpdated']
```

Relation with highest average probability:

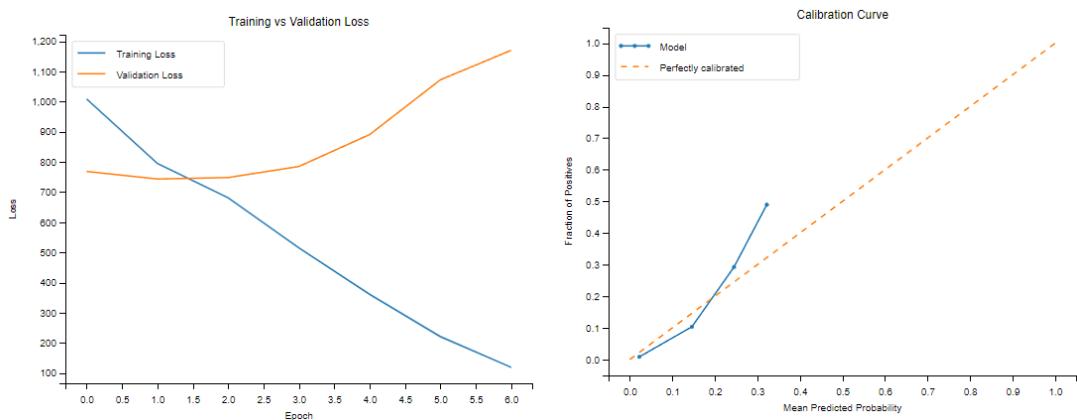
```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm',
0.25532177307953435)
```

Relation with lowest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink', 0.005369111895561218)
```

XIV. Experiment 4

	Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
C1	0.8118	0.2401	3526.3153
C2	2	0	0
C3	0	0.0961	0.2954
C4	0	0.0937	0.0001
C5			
C6			
C7			
C8			



Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 1417),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 25)]
```

Relation with highest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 0.09643133350456623)
```

Relation with lowest average probability:

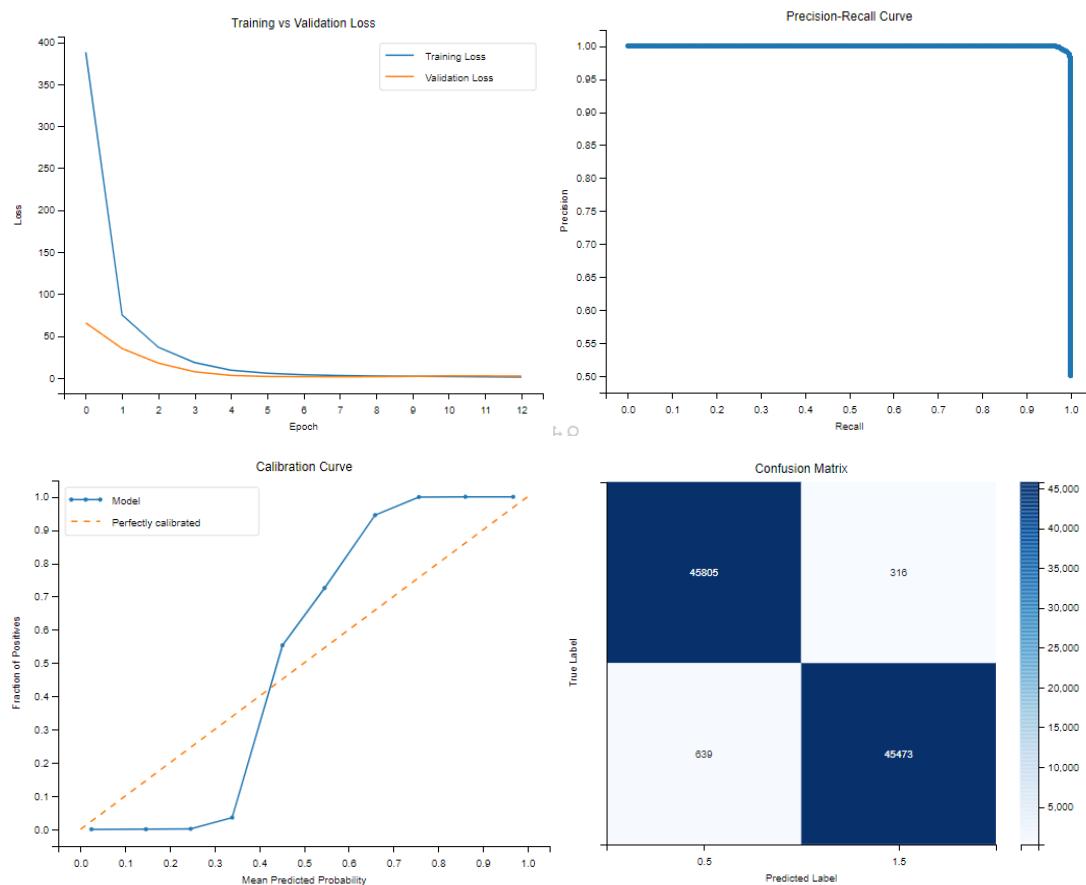
```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference',
0.07899965308519313)
```

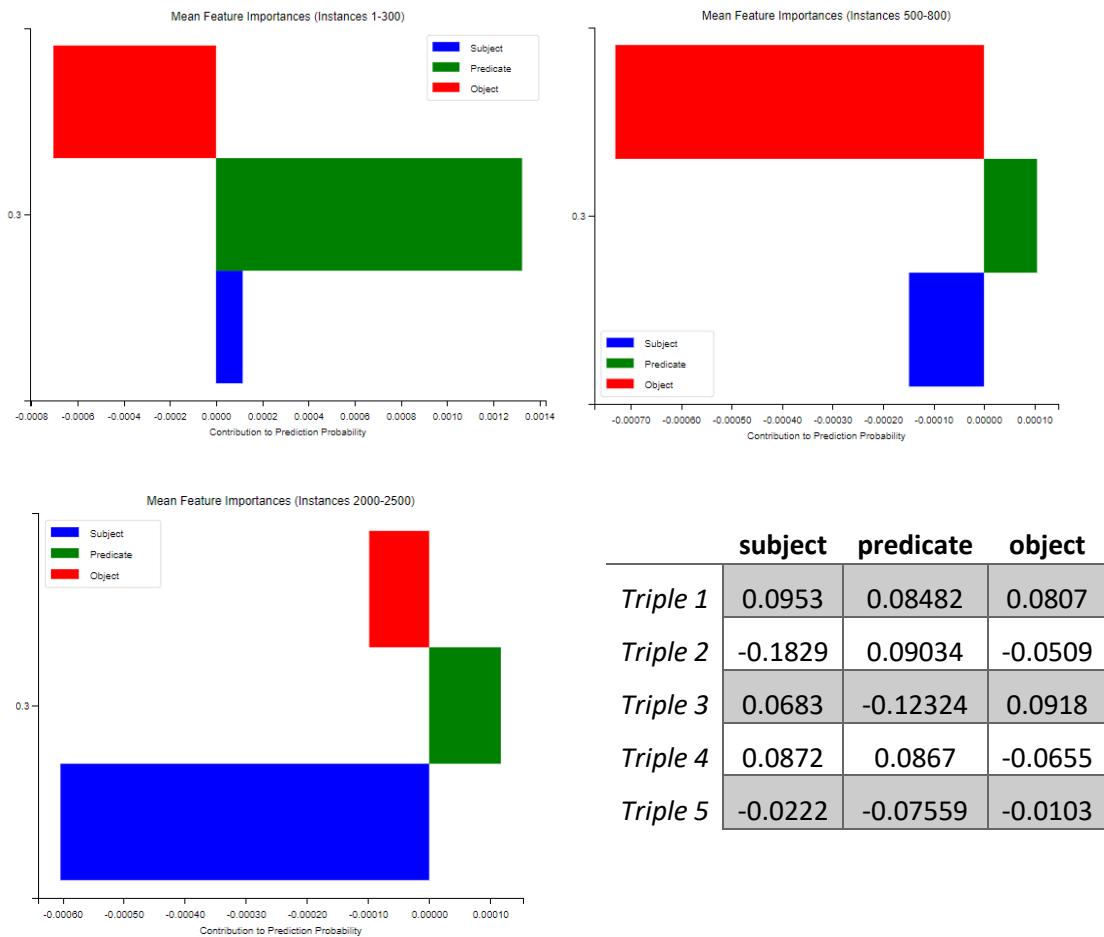
XV. Experiment 5

Mean Silhouette Mean Davies-Bouldin Mean Calinski-Harabasz

0.7967	0.2659	4258.8930
--------	--------	-----------

C1	C2	C3	C4	C5	C6	C7	C8
42	1409	436	40	0.8345	0.9965	0.1408	0.1238





Top 5 most frequent relations:

```
[('http://www.w3.org/2000/01/rdf-schema#comment', 134),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/sourcePublication', 108),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/remark', 81),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 79),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasFrequentTerm', 74)]
```

Relations appearing only once:

```
['http://www.w3.org/2002/07/owl#backwardCompatibleWith']
```

Relation with highest average probability:

```
('http://www.w3.org/2002/07/owl#equivalentClass', 0.9076344115393502)
```

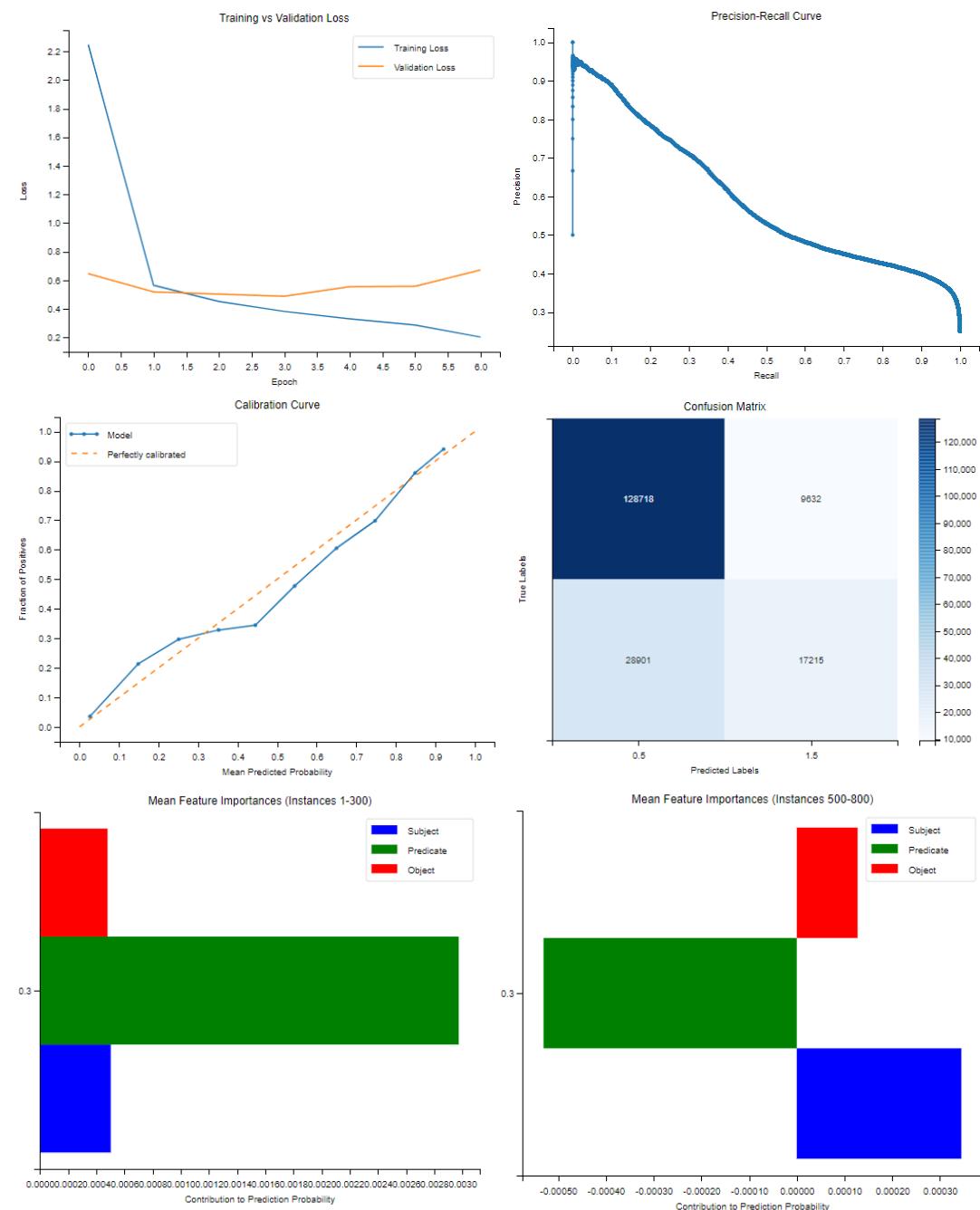
Relation with lowest average probability:

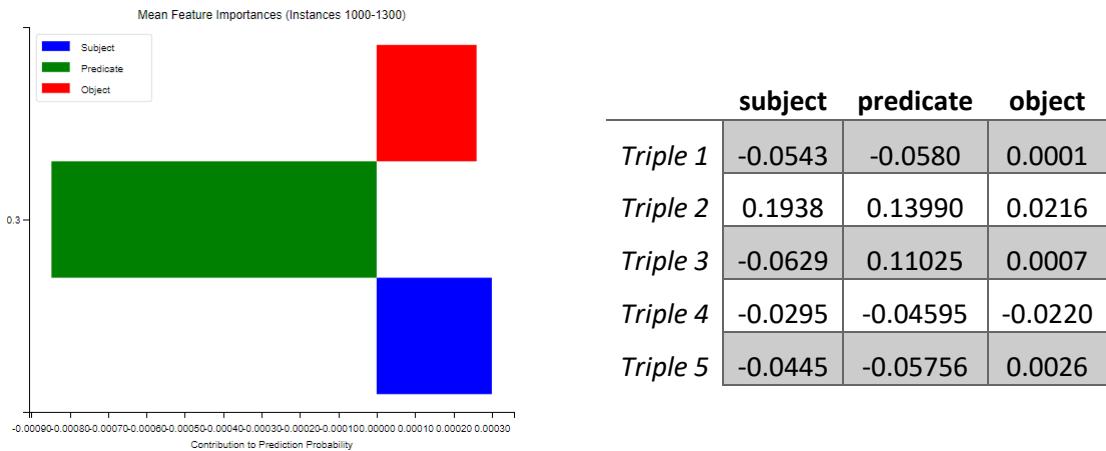
```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 0.7821886017918587)
```

XVI. Experiment 6

Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
0.8027	0.2564	3468.3677

C1	C2	C3	C4	C5	C6	C7	C8
22	2	0	0	0.0760	0.6515	0.0001	0.1211





Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/remark', 766),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/definition', 208),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 83),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/databasePath', 55),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 46)]

Relations appearing only once:

['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCategoryOfStatisticExplainedArticle', 'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasOECDTheme',
 'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/dataSource']

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCategoryOfStatisticExplainedArticle', 0.4231768250465393)

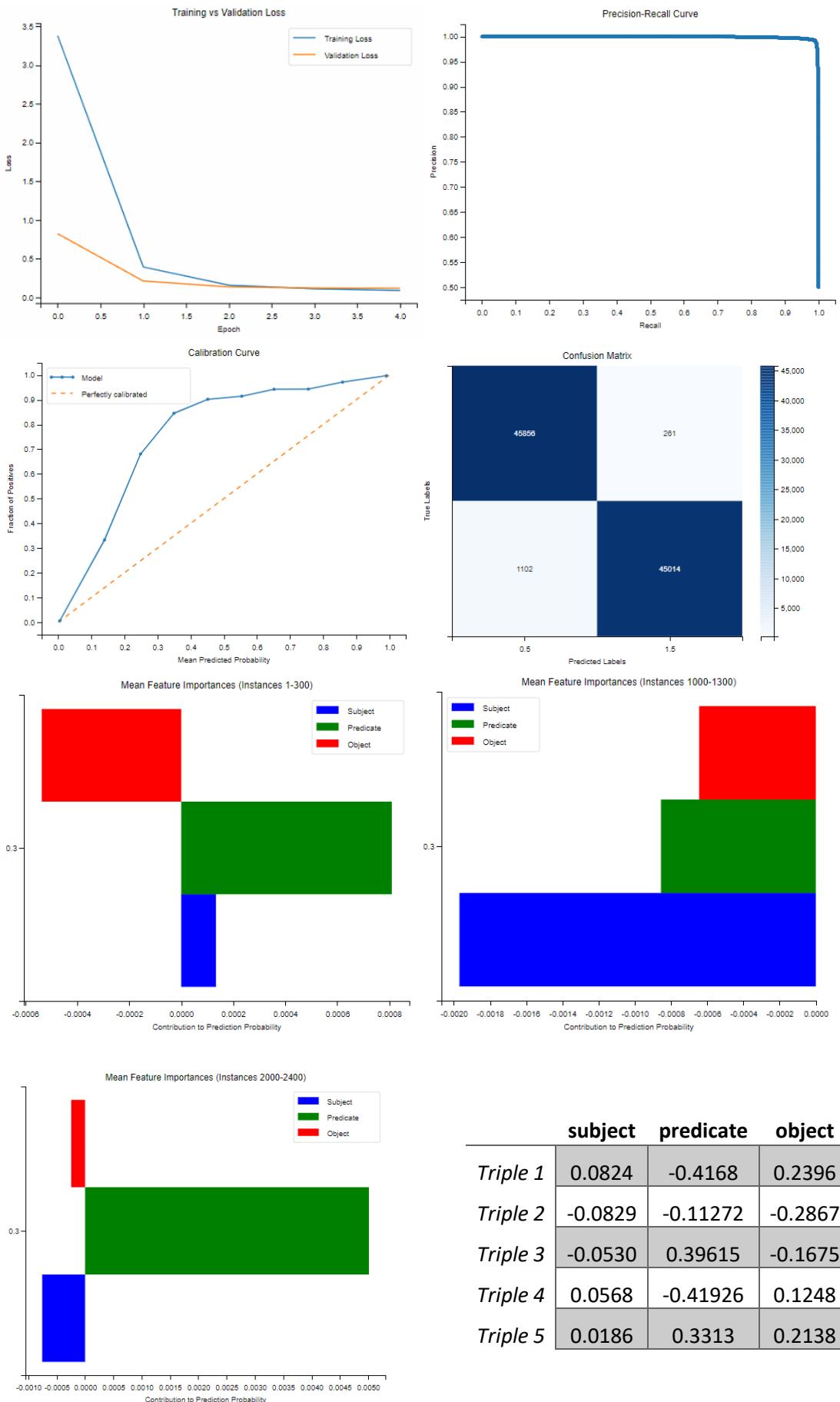
Relation with lowest average probability:

('http://www.w3.org/2000/01/rdf-schema#subClassOf', 0.007155439350754023)

XVII. Experiment 7

Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
0.8069	0.2479	3895.3840

C1	C2	C3	C4	C5	C6	C7	C8
11	937	659	11	0.6658	1.0000	0.0039	0.3460



Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 692),  
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/databasePath', 197),  
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 190),  
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 125),  
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/definition', 102)]
```

Relations appearing only once:

```
['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasFrequentTerm']
```

Relation with highest average probability:

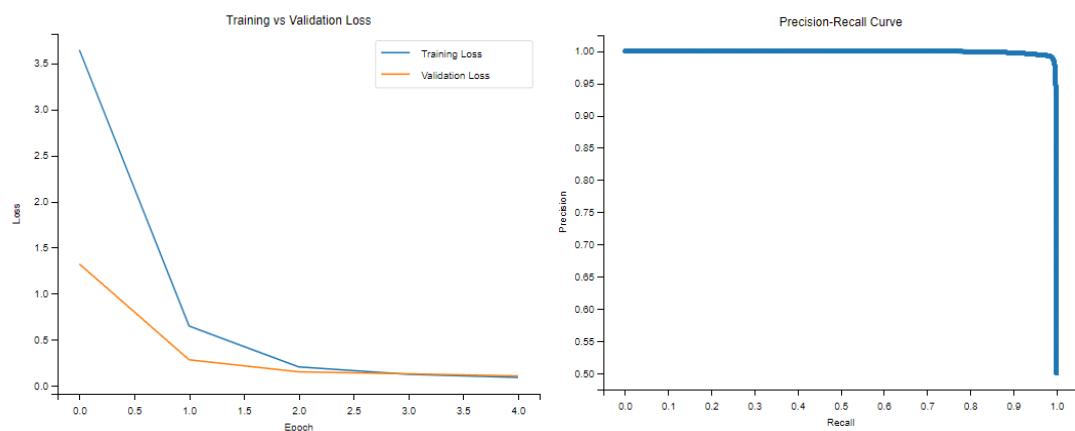
```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasFrequentTerm',  
 0.947577178478241)
```

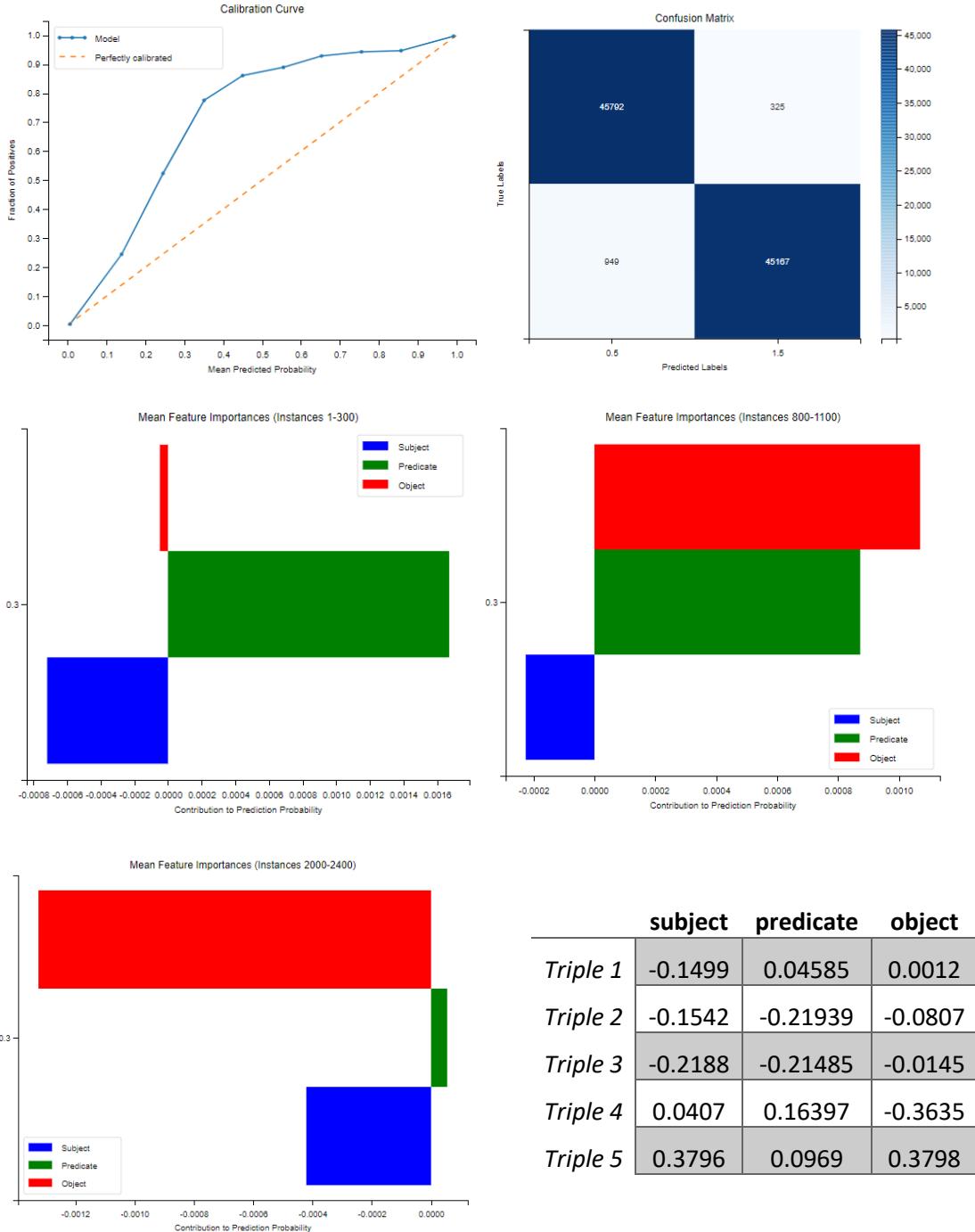
Relation with lowest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/definition', 0.6081479691498566)
```

XVIII. Experiment 8

Mean Silhouette		Mean Davies-Bouldin		Mean Calinski-Harabasz			
C1	C2	C3	C4	C5	C6	C7	C8
5	1104	842	5	0.7571	1.0000	0.0017	0.3068





Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 1369),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 122),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink', 3),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm', 3),
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/content', 3)]
```

Relation with highest average probability:

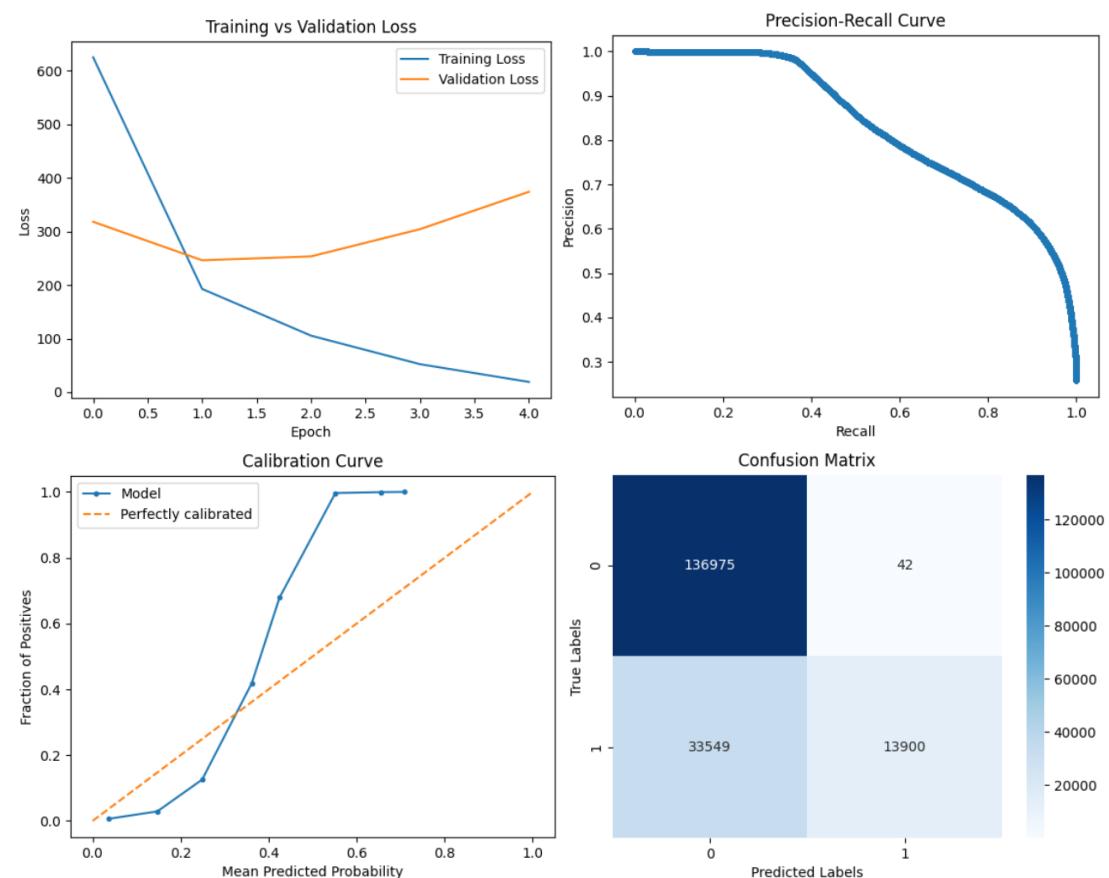
```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm', 0.9976281921068827)
```

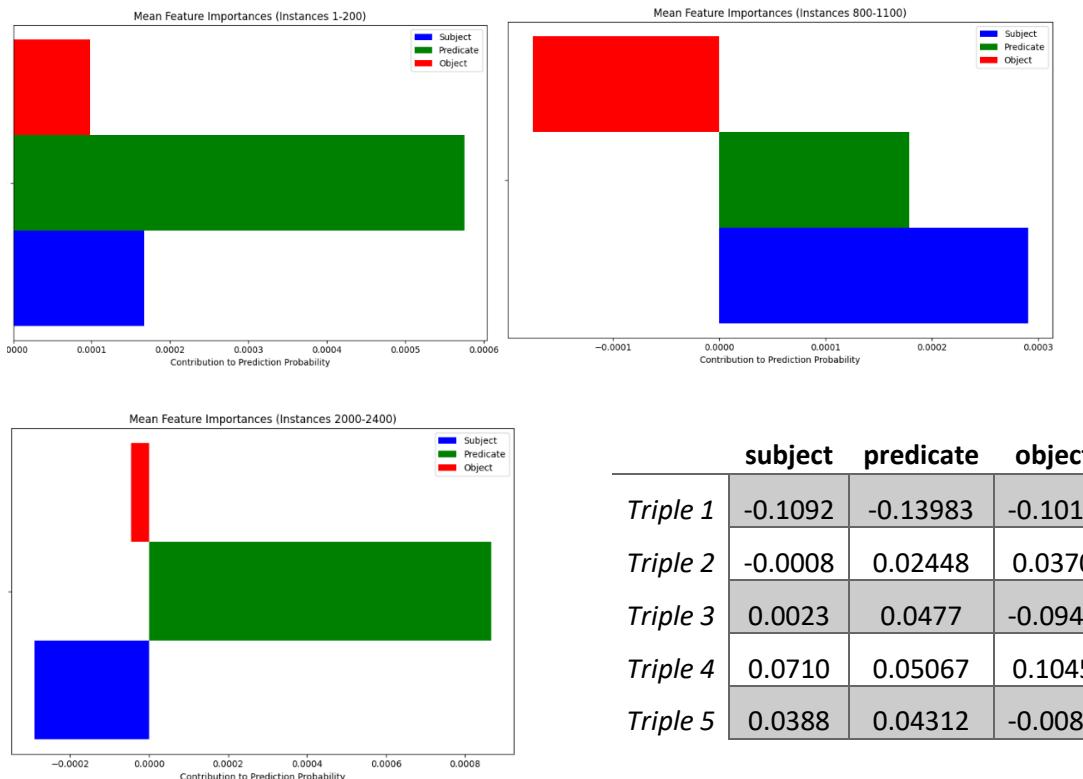
Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/content', 0.5336959660053253)

XIX. Experiment 9

	Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
C1	0.8068	0.2487	3582.4176
C2	4		
C3	0		
C4	0		
C5	0.2678		
C6	0.4549		
C7	0.0050		
C8	0.1117		





	subject	predicate	object
Triple 1	-0.1092	-0.13983	-0.1011
Triple 2	-0.0008	0.02448	0.0370
Triple 3	0.0023	0.0477	-0.0946
Triple 4	0.0710	0.05067	0.1045
Triple 5	0.0388	0.04312	-0.0085

Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 709),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/term', 468),
 ('http://www.w3.org/1999/02/22-rdf-syntax-ns#type', 275),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 48)]

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/level', 0.29218049774256843)

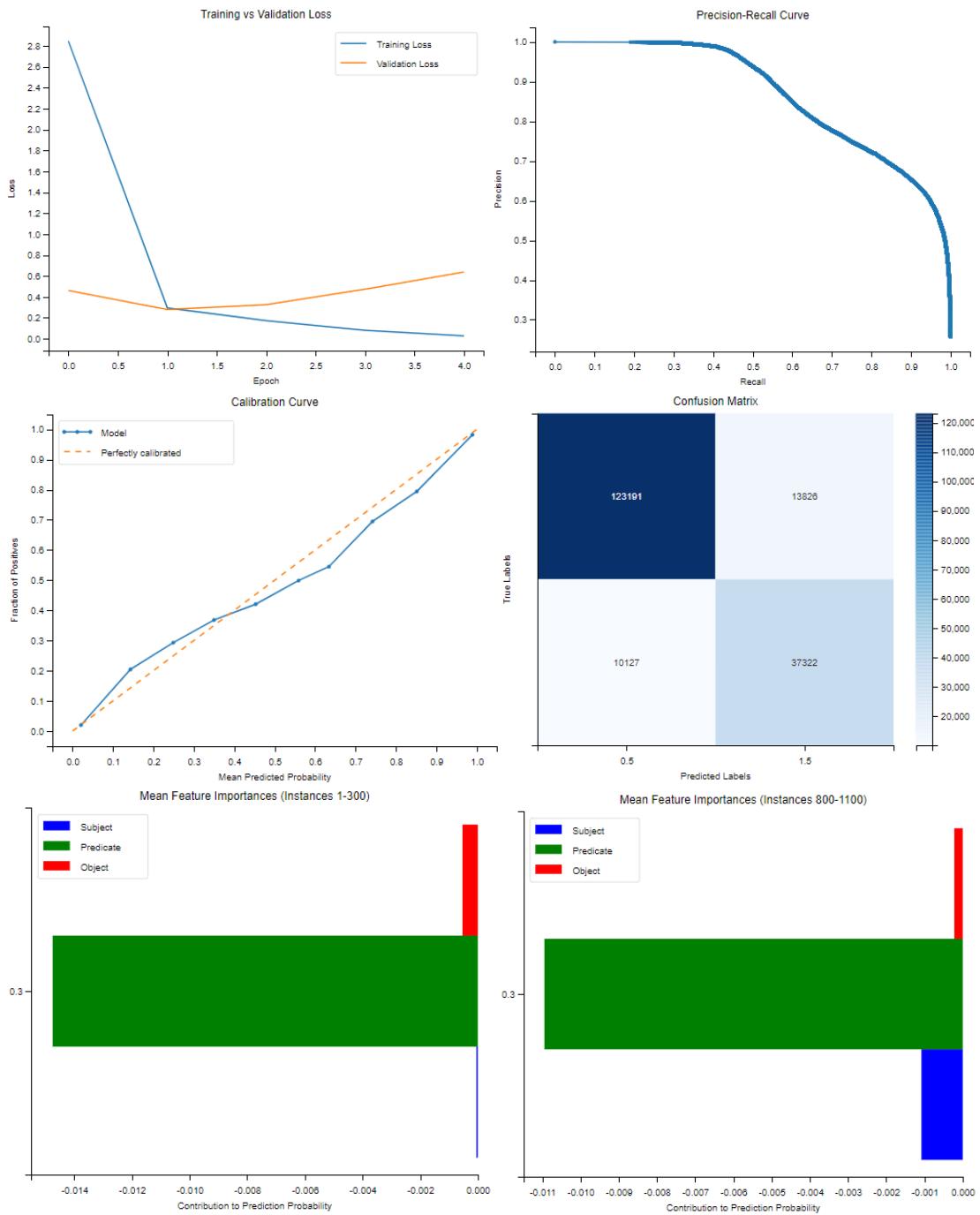
Relation with lowest average probability:

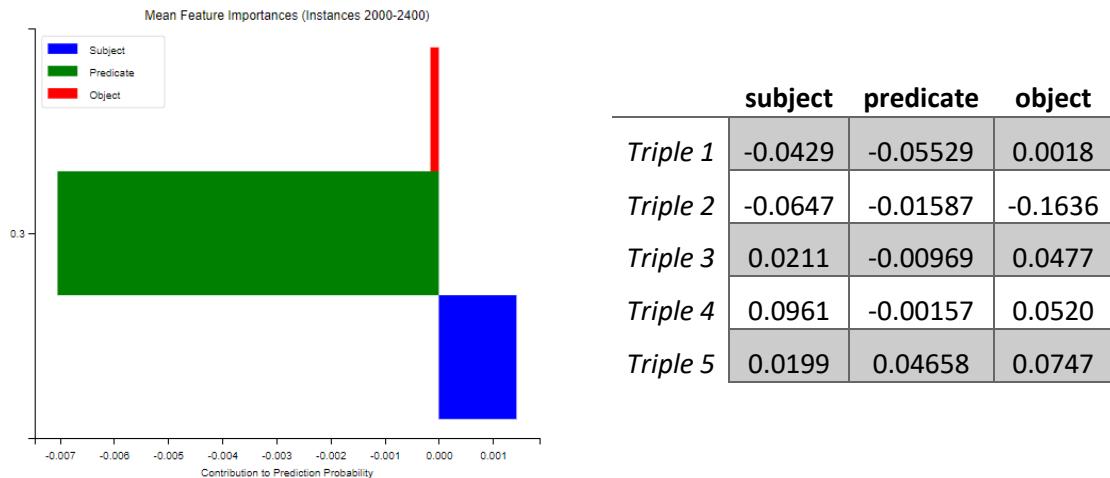
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasReference', 0.2641741103671323)

XX. Experiment 10

Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
0.8086	0.2453	4373.2786

C1	C2	C3	C4	C5	C6	C7	C8
30	76	0	0	0.2180	0.8965	0.0004	0.2173





Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasFrequentTerm', 256),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/keyword', 207),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/id', 204),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 137),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasSubTheme', 116)]
```

Relations appearing only once:

```
['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCategoryOfGlossaryArticle',
 'http://www.w3.org/2002/07/owl#unionOf',
 'http://www.w3.org/2002/07/owl#versionInfo',
 'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm',
 'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasTopic',
 'http://www.w3.org/2002/07/owl#backwardCompatibleWith']
```

Relation with highest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasTopic', 0.5259029865264893)
```

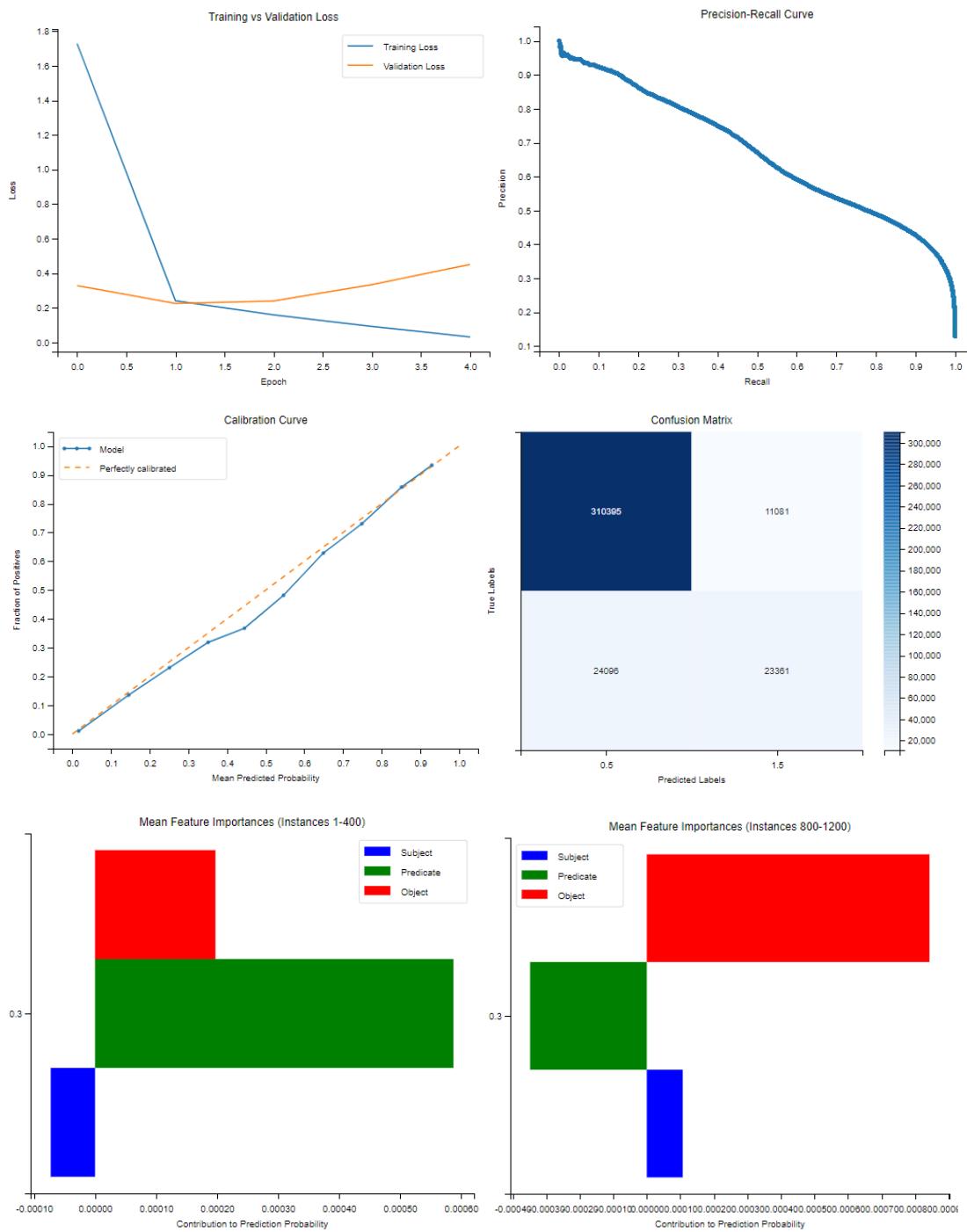
Relation with lowest average probability:

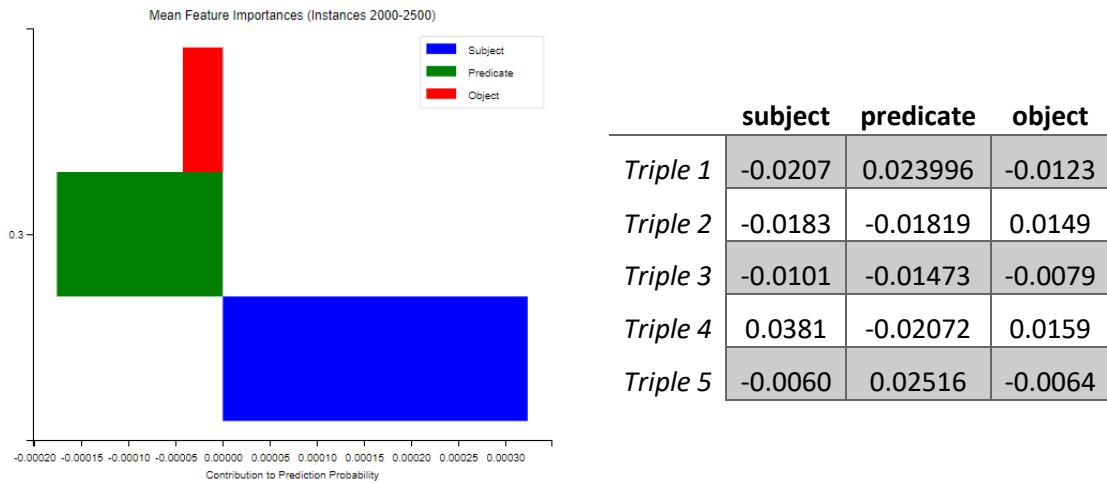
```
('http://www.w3.org/2002/07/owl#unionOf', 0.015426167286932468)
```

XXI. Experiment 11

Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
0.8097	0.2443	4150.0788

C1	C2	C3	C4	C5	C6	C7	C8
21	3	0	0	0.0425	0.7365	0.0003	0.0872





Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/remark', 612),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 170),
 ('http://www.w3.org/2000/01/rdf-schema#subClassOf', 129),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 125),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/title', 119)]
```

Relations appearing only once:

```
['http://www.w3.org/2000/01/rdf-schema#label',
 'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink',
 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
 'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTheme']
```

Relation with highest average probability:

```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph',
 0.1194011508487165)
```

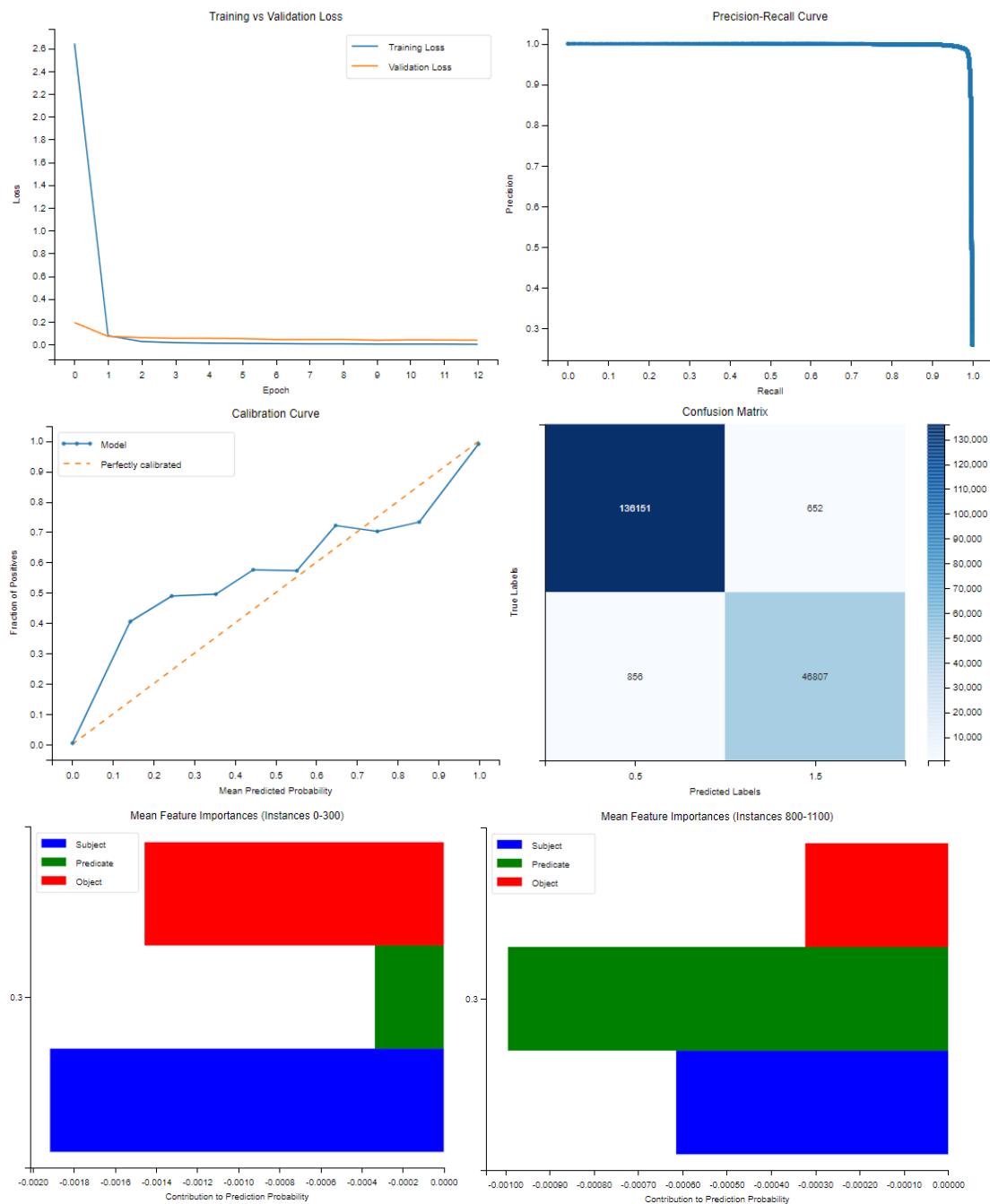
Relation with lowest average probability:

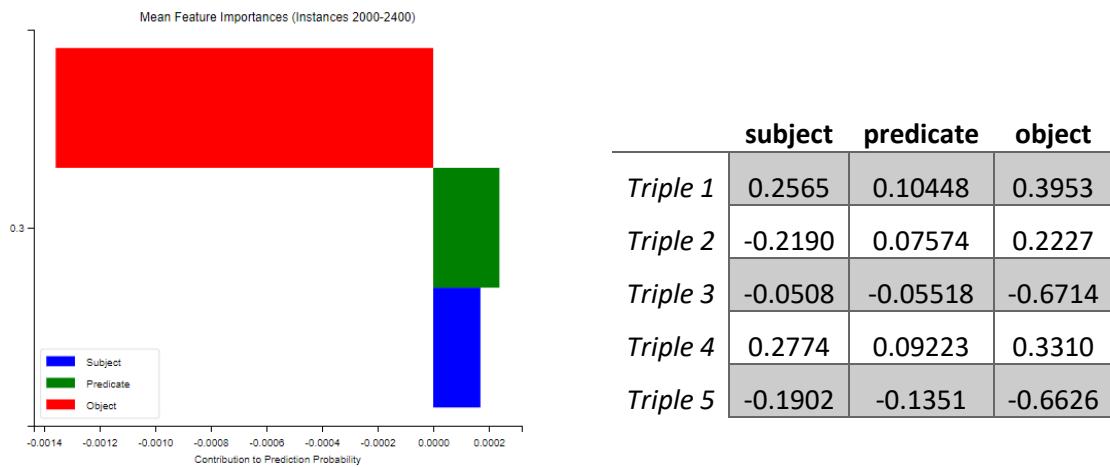
```
('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTheme',
 0.002175799338147044)
```

XXII. Experiment 12

Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
0.8049	0.2518	4022.4616

C1	C2	C3	C4	C5	C6	C7	C8
20	413	356	17	0.3006	1.0000	0.0001	0.4236





Top 5 most frequent relations:

[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasURI', 319),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/definition', 301),
 ('http://www.w3.org/2000/01/rdf-schema#subClassOf', 189),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 167),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/relatedTerm', 143)]

Relations appearing only once:

['https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasOECDTheme',
 'https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink']

Relation with highest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph',
 0.5013474763836712)

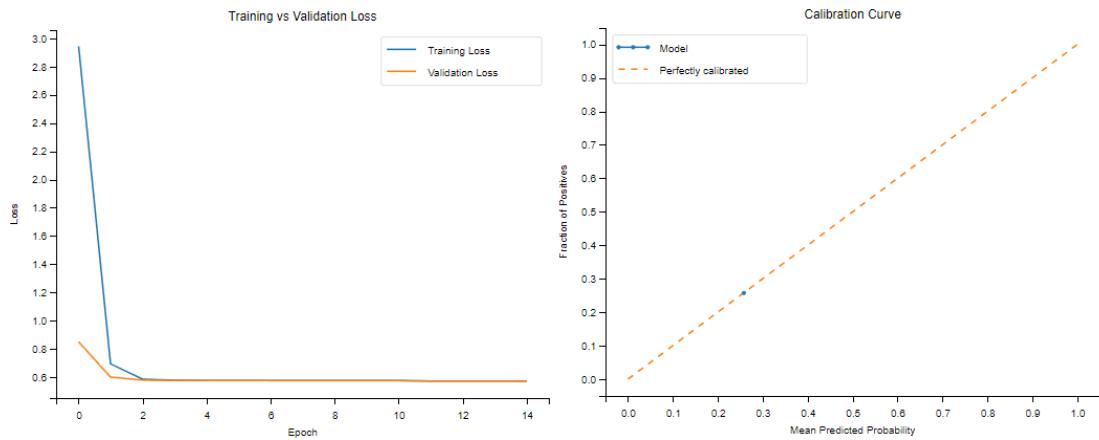
Relation with lowest average probability:

('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/fileLink', 0.0003957330482080579)

XXIII. Experiment 13

	Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
	0.7835	0.2856	3054.1802

C1	C2	C3	C4	C5	C6	C7	C8
1	0	0	0	0.2580	0.2580	0.2580	0.0000



Top 5 most frequent relations:

[('http://www.w3.org/2000/01/rdf-schema#subPropertyOf', 1500)]

Relation with highest average probability:

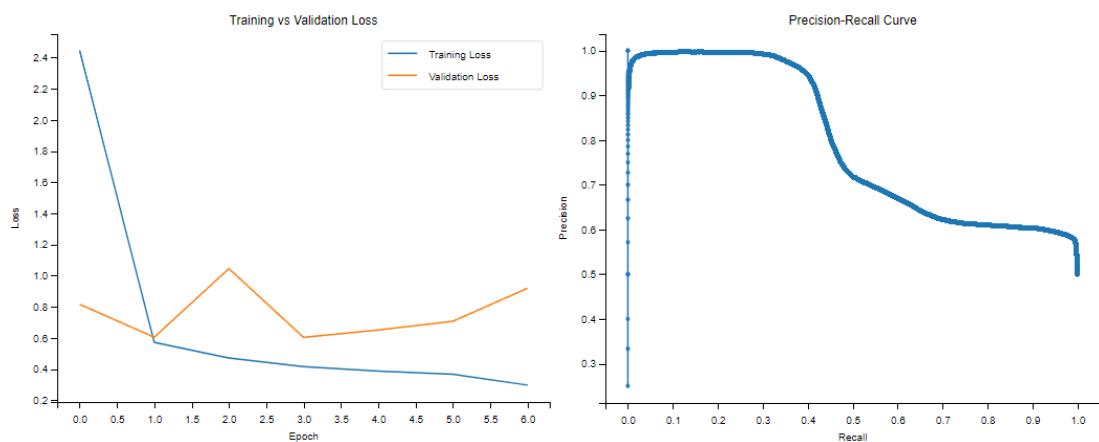
('http://www.w3.org/2000/01/rdf-schema#subPropertyOf', 0.2579798400402069)

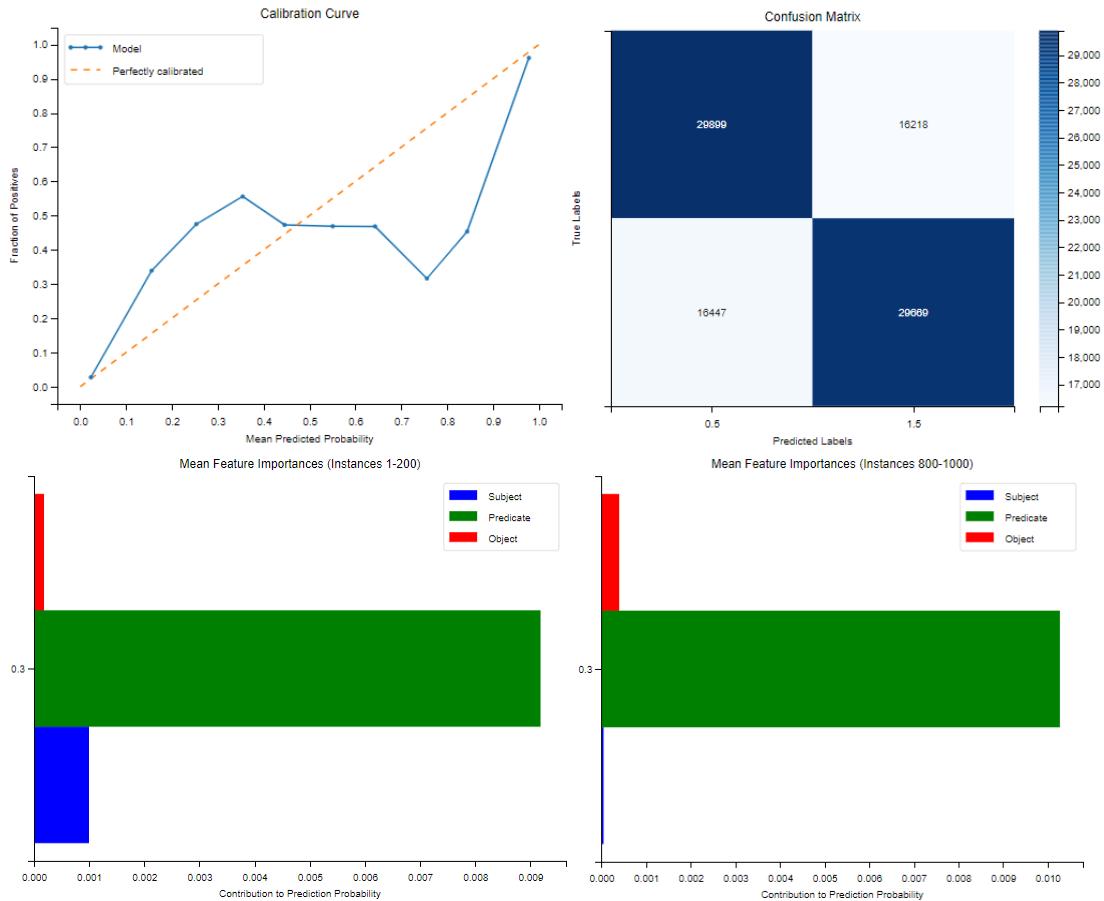
Relation with lowest average probability:

('http://www.w3.org/2000/01/rdf-schema#subPropertyOf', 0.2579798400402069)

XXIV. Experiment 14

	Mean Silhouette	Mean Davies-Bouldin	Mean Calinski-Harabasz
C1	0.8125	0.2416	3715.9306
C2	8	386	0
C3	0	0	0.4228
C4	0	0.8762	0.0014
C5	0.2588	0.5	0.2588
C6	0.4228	0.7	0.4228
C7	0.8762	0.6	0.0014
C8	0.0014	0.5	0.2588





	subject	predicate	object
<i>Triple 1</i>	0.01115	0.0160	-0.24187
<i>Triple 2</i>	-0.01493	0.2559	-0.07520
<i>Triple 3</i>	-0.00351	-0.41166	-0.07640
<i>Triple 4</i>	0.01936	0.25983	-0.03920
<i>Triple 5</i>	-0.01073	0.03038	0.42009

Top 5 most frequent relations:

```
[('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/databasePath', 1124),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasFrequentTerm', 350),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasCode', 14),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/content', 4),
 ('https://ec.europa.eu/eurostat/NLP4StatRef/ontology/hasParagraph', 3)]
```

Relations appearing only once:

```
['http://www.w3.org/2000/01/rdf-schema#label']
```

Relation with highest average probability:

```
('http://www.w3.org/2002/07/owl#equivalentClass', 0.6497162580490112)
```

Relation with lowest average probability:

('http://www.w3.org/2000/01/rdf-schema#label', 0.009083980694413185)