A Sample of Extracted Triples

To illustrate the output of the triple extraction process used in this study, we include a representative sample of 20 triples extracted from different user reviews. These triples reflect a range of sentiment scores (positive, negative, and neutral) and capture various aspects of the reviews. Each triple is structured as $\langle$ *Subject, Relation, Object* $\rangle$, with an associated sentiment score. Inside our knowledge graph we remove all triples with relationships or nodes that have a text length of longer than 14 characters.

Table 11: Sample of Extracted Triples from Review Data

Review ID	Subject	Relation	Object	Sentiment
144	Great pool	is in	wonderful spot by beach	0.83
3798	bed	was comfortable with	excellent linen	0.79
992	positive	were many small issues with	room for improvement	0.77
6254	loft	best feature of was	bathroom	0.64
276	Breakfast	was outstanding for	British fryup	0.61
1299	we	were	most impressed	0.53
2185	Our 40th school reunion weekend	was	help	0.40
1513	hotel	is well located in	historical center	0.27
4683	Hotel	was accommodating to	our group	0.00
5108	We	brought along	our 8yearold	0.00
721	We	had	5night stay	0.00
6958	Ravenna	was	crowded	0.00
3744	same	can	can said of bathroom	0.00
8153	This	is	our 4th year staying here	0.00
5882	manager	moved with	only minor change fee	0.00
9326	it	is	too much trouble	-0.40
6644	Poor excuse	is in	need	-0.42
5672	water temperature	keeps	fluctuating dangerously	-0.46
8623	check	is	wrong	-0.48
4844	My complaint	was	very poor wireless internet service	-0.68

B Key Implementation Code

This section includes selected code snippets that are central to the implementation of our model. The complete source code can be found in our GitHub repository.

https://github.com/yourusername/your-repo-name

B.1 Triple Extraction Using OpenIE

```
def create_csv_triples_stanfordopenie(review_data):
    # initialize an empty DataFrame
    columns = ["Review_ID", "Subject", "Relation", "Object"]
    triple_df = pd.DataFrame(columns=columns)

    for index, row in review_data.iterrows():
        # assign review data
        review_id = index
        review = row[review_column_name]

        try:
            # translate the review to english (if applicable)
            review = translate_review(review)

            # expand contractions (can't to cannot)
            review = contractions.fix(review)
        except Exception as e:
            print(f"Skipping review due to error: {e}")

        # remove html newline symbols
        review = re.sub(r'<br\s*/?>', ' ', review)

        # remove hyphens and apostrophes
        review = review.replace('-', '')
        review = review.replace("'", "")
        review = review.replace("´", "")
        review = review.replace("'", "")

        # add spaces after periods, question marks, exclamation marks
        review = add_space_after_period(review)

        # extract triples
        triple_list = triple_extraction_stanfordopenie(review)

        # create a temporary DataFrame for extracted triples
        temp_df = pd.DataFrame(triple_list, columns=["Subject", "Relation", "Object"])
```

```
        temp_df["Review_ID"] = review_id  # Add the review ID

        # Append to main DataFrame
        triple_df = pd.concat([triple_df, temp_df], ignore_index=True)

    return triple_df, review_data
```

And the function to make use of the actual StanfordOpenIE library:

```
def triple_extraction_stanfordopenie(review):
# Connect to the CoreNLP server (ensure it is running)
nlp = StanfordCoreNLP('http://localhost', port=9000, timeout=30000)

# set up stanford openIE
output = nlp.annotate(review, properties=props)
review_output = json.loads(output)

triple_list = []
for sentence in review_output['sentences']:
    if 'openie' in sentence:
        for triple in sentence['openie']:
            subject = triple['subject']
            relation = triple['relation']
            object_ = triple['object']
            triple_list.append((subject, relation, object_))

return triple_list
```

B.2 Calculate the Sentiment Scores on Triples

```
def calculate_sentiments(triple_df):
analyzer = SentimentIntensityAnalyzer()
sentiments = []

for index, triple in triple_df.iterrows():
    triple = str(triple['Subject']) + " " + str(triple['Relation']) + " " + str(triple
    vs = analyzer.polarity_scores(triple)
    sentiments.append(vs['compound'])

triple_df['Sentiment'] = sentiments
return triple_df
```

B.3 Create a Graph From the Triple Data

```
def create_graph(triple_df, review_data):
# Returns graph G
```

```
# Create a directed graph
G = nx.DiGraph()  # Directed graph since relationships are directional

for index, triple in triple_df.iterrows():
    review = triple['Review_ID']
    review_row = review_data[review_data['Review_ID'] == triple['Review_ID']]

    subject = normalize_text(triple['Subject'])
    relation = normalize_text(triple['Relation'])
    object = normalize_text(triple['Object'])
    sentiment = triple['Sentiment']

    if not isinstance(subject, str) or not isinstance(object, str) or not isinstance(re
        continue
    if len(subject) > 14 or len(relation) > 14 or len(object) > 14:
        continue

    # Split the string by 'Reviews-' and take the first part of the split
    hotel_name = review_row['hotel_url'].iloc[0].split('Reviews-')[1].split('.html')[0]

    # author = review_row['author'].iloc[0]

    # Add review_id node to the graph
    if not G.has_node(review):
        G.add_node(review, type='review', rating=review_row['rating'].iloc[0], date=rev

    # Add hotel node to the graph
    if not G.has_node(hotel_name):
        G.add_node(hotel_name, type='hotel')

    # Add nodes for subject and object
    for node in [subject, object]:
        node_type = 'amenity' if node in amenities_list else 'word'

        if not G.has_node(node) or not G.has_edge(review, node):
            G.add_node(node, type=node_type)

            # Add edges from review_id to subject and object
            G.add_edge(review, node, relation='contains')

    if not G.has_edge(review, hotel_name):
        # Add edge from review to hotel
        G.add_edge(review, hotel_name, relation='reviewed')
```

```
        # Add edge between subject and object
        G.add_edge(subject, object, relation=relation, sentiment=sentiment)

    # Save the graph to a file
    with open('graph.pkl', 'wb') as f:
        pickle.dump(G, f)

    return G
```

C Complete Tables of Results

C.1 Train Test Split Results

Classifier Model	Type	Dim	Sampling	Accuracy	MAE	RMSE	Cohen's Kappa
Random Forest	Node2Vec	100	Oversampling	0.4709	0.8135	1.6409	0.0352
Logistic Regression	Node2Vec	100	Oversampling	0.1968	1.7331	4.6121	0.0180
Neural Network (MLP)	Node2Vec	100	Oversampling	0.1757	1.3282	2.4874	-0.0025
Random Forest	Node2Vec	5	Oversampling	0.4060	0.8722	1.6388	0.0574
Logistic Regression	Node2Vec	5	Oversampling	0.2334	1.6249	4.3699	0.0307
Neural Network (MLP)	Node2Vec	5	Oversampling	0.1664	1.8537	5.0644	0.0043
Random Forest	Node2Vec	5	No Sampling	0.4673	0.8315	1.7105	0.0188
Logistic Regression	Node2Vec	5	No Sampling	0.3060	1.5281	4.2998	0.0562
Neural Network (MLP)	Node2Vec	5	No Sampling	0.2257	1.3735	3.2643	-0.0070
Dummy (Most Frequent)	Node2Vec	5	No Sampling	0.5059	0.8465	1.9201	0.0000
Random Forest	Node2Vec	100	No Sampling	0.4951	0.8542	1.9134	-0.0076
Logistic Regression	Node2Vec	100	No Sampling	0.3344	1.4776	4.2751	0.0751
Neural Network (MLP)	Node2Vec	100	No Sampling	0.2700	1.2081	2.6321	0.0049
Dummy (Most Frequent)	Node2Vec	100	No Sampling	0.5059	0.8465	1.9201	0.0000
Random Forest	N2V+Sentiment(min/max)	5	No Sampling	0.5590	0.5992	1.0185	0.2607
Logistic Regression	N2V+Sentiment(min/max)	5	No Sampling	0.3632	1.3060	3.5018	0.1688
Neural Network (MLP)	N2V+Sentiment(min/max)	5	No Sampling	0.4626	1.0752	2.8300	0.0937
Dummy (Most Frequent)	N2V+Sentiment(min/max)	5	No Sampling	0.5059	0.8465	1.9201	0.0000
Random Forest	N2V+Sentiment(min/max)	5	Oversampling	0.5404	0.6301	1.0896	0.2829
Logistic Regression	N2V+Sentiment(min/max)	5	Oversampling	0.4250	1.1345	3.0057	0.2107
Neural Network (MLP)	N2V+Sentiment(min/max)	5	Oversampling	0.5039	0.8238	1.7965	0.0862
Dummy (Most Frequent)	N2V+Sentiment(min/max)	5	Oversampling	0.0433	3.1535	11.1484	0.0000
Random Forest	Word2Vec	–	–	0.5705	0.5775	0.9805	0.2988
Logistic Regression	Word2Vec	–	–	**0.6035**	**0.5010**	**0.7920**	**0.3793**
Neural Network (MLP)	Word2Vec	–	–	0.5275	0.6325	1.0555	0.2828
Dummy (Most Frequent)	Word2Vec	–	–	0.4765	0.9025	2.0405	0.0000
Random Forest	Bag of Words	–	–	0.5350	0.7335	1.5405	0.1704
Logistic Regression	Bag of Words	–	–	0.5960	0.4855	0.6875	0.3802
Neural Network (MLP)	Bag of Words	–	–	0.5920	0.5010	0.7360	0.3698
Dummy (Most Frequent)	Bag of Words	–	–	0.4765	0.9025	2.0405	0.0000
Random Forest	TF-IDF	–	–	0.5265	0.7550	1.6010	0.1500
Logistic Regression	TF-IDF	–	–	0.5955	0.5120	0.7990	0.3474
Neural Network (MLP)	TF-IDF	–	–	0.5880	0.5015	0.7165	0.3607
Dummy (Most Frequent)	TF-IDF	–	–	0.4765	0.9025	2.0405	0.0000
LLM Model	LLM (n=200)	–	–	0.5165	0.8050	1.7748	0.0427
LLM Model	LLM (n=2000)	–	–	0.5867	0.5795	1.0622	0.2839

Table 12: Performance comparison across all classifier models, feature types, sampling methods, and dimensions. Bold values indicate the best overall scores across all configurations.

C.2 10-Fold Cross Validation Results

Classifier Model	Type	Dim	Sampling	Accuracy	MAE	RMSE	Cohen's Kappa
Random Forest	Node2Vec	5	Oversampling	0.4094	0.8766	1.6609	0.0558
Logistic Regression	Node2Vec	5	Oversampling	0.3575	1.4730	4.3033	0.0807
Neural Network (MLP)	Node2Vec	5	Oversampling	0.2765	1.3625	3.2776	0.0601
Dummy (Most Frequent)	Node2Vec	5	Oversampling	0.0406	3.1612	11.1686	0.0000
Random Forest	Node2Vec	5	No Sampling	0.4771	**0.8121**	**1.6404**	0.0400
Logistic Regression	Node2Vec	5	No Sampling	0.5085	0.8355	1.8657	0.0073
Neural Network (MLP)	Node2Vec	5	No Sampling	0.4917	0.8190	1.7327	0.0254
Dummy (Most Frequent)	Node2Vec	5	No Sampling	0.5080	0.8388	1.8789	0.0000
Random Forest	N2V+Sentiment(min/max)	5	Oversampling	0.5243	0.6379	1.0605	0.2615
Logistic Regression	N2V+Sentiment(min/max)	5	Oversampling	0.4913	0.7569	1.4742	0.2649
Neural Network (MLP)	N2V+Sentiment(min/max)	5	Oversampling	0.4543	0.8389	1.6516	0.2249
Dummy (Most Frequent)	N2V+Sentiment(min/max)	5	Oversampling	0.0406	3.1612	11.1686	0.0000
Random Forest	Word2Vec	–	–	0.5942	0.5365	0.8971	0.3213
Logistic Regression	Word2Vec	–	–	0.6286	0.4605	0.6981	**0.4055**
MLP Classifier	Word2Vec	–	–	0.5461	0.5874	0.9336	0.2957
Dummy (Most Frequent)	Word2Vec	–	–	0.5060	0.8380	1.8686	0.0000
Random Forest	Bag of Words	–	–	0.5612	0.6751	1.3739	0.1888
Logistic Regression	Bag of Words	–	–	0.6113	0.4608	**0.6334**	0.3855
MLP Classifier	Bag of Words	–	–	0.6056	0.4746	0.6694	0.3751
Dummy (Most Frequent)	Bag of Words	–	–	0.5060	0.8380	1.8686	0.0000
Random Forest	TF-IDF	–	–	0.5619	0.6839	1.4245	0.1848
Logistic Regression	TF-IDF	–	–	**0.6367**	**0.4536**	0.6942	0.4007
MLP Classifier	TF-IDF	–	–	0.5966	0.4817	0.6695	0.3591
Dummy (Most Frequent)	TF-IDF	–	–	0.5060	0.8380	1.8686	0.0000

Table 13: 10-Fold Cross-Validation Performance comparison across all classifier models, feature types, sampling methods, and dimensions. Bold values indicate the best overall scores across all configurations.