Classifying Semantic Equivalency Across Patent Applications

Minsu Kim1 and Dave Friesen2

University of San Diego

Master of Science, Applied Data Science

ADS-502 Spring 2022

Section 02

April 18, 2022

Author Note

We have no conflicts of interest to disclose.

Correspondence concerning this paper should be addressed to Minsu Kim or Dave Friesen. Email: minsukim@sandiego.edu or dfriesen@sandiego.edu

Abstract

U.S. patent applications require significant manual effort to reconcile similarities across patents and patent domains. Modern data science and natural language processing techniques provide an opportunity to automate this review process. The objective of this study was to explore semantic similarity classification through several experimental approaches, from simple to near-state-of-the-art, toward a hypothesis that patent phrase pairs may be semantically classified. The study performed secondary data analysis of approximately 36,000 sampled U.S. patent phrase pairs, including preliminary data mining and pattern review. The study progressed through three experimental approaches, from simple word counts to context classification and, ultimately, semantic similarity using transformers (neural networks, or BERT). The study’s final BERT approach demonstrated that semantic similarity classification (automation) is possible for this use case, rejecting a null hypothesis. This conclusion supports promising additional refinement and research.

*Keywords:* Patent semantics, semantic similarity, USPTO patent phrase similarity

# Table of Contents

List of Figures 4

List of Tables 4

List of Equations 4

Classifying Semantic Equivalency Across Patent Applications 5

Objective 5

Method 5

Data Collection and Pre-Processing 6

Sample Characteristics 6

Multivariate Analysis 8

Results 8

Modeling for Context 8

Modeling for Word Similarity 10

Modeling for Semantic Similarity 11

Discussion 13

Hypothesis Review 13

Strengths, Weaknesses, and Opportunities 13

References 14

Appendix A – Project Code 15

Appendix B – Presentation 35

# List of Figures

**Figure 1** Labeled (Scored) Phrase Similarity by Context 7

**Figure 2** Phrase Similarity by Label (Score) 7

**Figure 3** Anchor à Context Confusion Matrix 9

**Figure 4** Duplicated Words in Training Data 11

**Figure 5** Semantic Similarity Score in Training Data 11

**Figure 6** Cosine Similarity vs. Labeled Score by Context 12

**Figure 7** Natural Distribution of Label vs. Model 13

**Figure 8** Re-scaled Distribution of Label vs. Model 13

# List of Tables

**Table 1** USPTO Patent Data Summary 6

**Table 2** Patent Dataset 7

**Table 3** Anchor à Context Classification Metrics 9

# List of Equations

**Equation 1** Logistic Regression for Context Classification 8

**Equation 2** Generalized Artificial Neural Network 12

# Classifying Semantic Equivalency Across Patent Applications

The Internet-enabled world is producing prodigious quantities of data, estimated at 79 *zettabytes* (7921 bytes) in 2021 and increasing to nearly 200 zettabytes annually by 2025 (Statista, 2022). Some estimate as much as 90% of this data to be *unstructured*, emphasizing a significant opportunity to “unlock” information through sophisticated Data Science techniques applied to the subfield of Natural Language Processing (NLP) (MIT Sloan School of Management, 2022).

This study is a *nascent step* into NLP, seeking to classify semantic relevancy (similarity) of phrase pairs across U.S. Patent and Trademark Office (USPTO) patent applications. Patent application reviews currently require significant manual reconciliation of synonymic phrases, and further automating this process promises to increase efficient approvals of distinct inventions. (Note that beyond the patent use case, it is easy to see how semantic similarity solution patterns apply to a spectrum of real-world challenges, from sentiment analysis to knowledge curation and artificial intelligence.)

## Objective

Framed by the above problem statement, this study uses data mining to explore classifying gradients of semantic similarity across patent application phrase pairs within patent application (subject) context and under a ‘reasonable’ margin of error. This objective translates to the [alternative] hypothesis () that patent phrase pairs may be classified as semantically similar within context, with a null hypothesis () of independence or no semantic classification possible (in this use case). Note that the “Results” section provides objective functions for these hypotheses.

# Method

The study followed a structured data research process, including exploratory data analysis, modeling, and evaluation, beginning with data acquisition, preparation, mining, and representation as described in this section.

## Data Collection and Pre-Processing

Secondary patent data was obtained from the USPTO indirectly via Kaggle (USPTO, 2022). These datasets were initially downloaded as “raw” comma-separated value (CSV) files, imported into a development Python Jupyter book, and preliminarily confirmed for essential data integrity. The following Table 1 summarizes the set of provided data assets:

### Table 1

USPTO Patent Data Summary

|  |  |  |  |
| --- | --- | --- | --- |
| **File Name** | **Dataset Description** | **Rows** | **Columns** |
| train.csv | Training set of phrases with pre-labeled scores | 36,473 | 5 |
| test.csv | Test set of phrases with pre-labeled scores | 36 | 4 |
| titles.csv | “Meanings” of USPTO context codes | 260,476 | 7 |
| sample\_submission.csv | Sample submission file (for context purposes) | 36 | 2 |

Based on a preliminary review, train.csv is most relevant to this exploratory study and was retained for further analysis and modeling.

## Sample Characteristics

As summarized in Table 1, the research dataset includes 36,473 unique observations[[1]](#footnote-2) or phrase pairs across eight Cooperative Patent Classifications (CPCs) or “context” areas (USPTO, 2022). The following Table 2 summarizes the patent dataset (train.csv) structure which was augmented with *context\_class* based on USPTO’s CPC definition:

### Table 2

Patent Dataset

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Description** | **Purpose** |
| id | Unique identifier for phrase pairs | index |
| anchor | First phrase | feature |
| target | Second phrase | feature |
| context | USPTO CPC | feature |
| context\_class | USPTO CPC definition (added) | feature |
| score | Labeled similarity score | target |

The USPTO defines *score* (similarity label) between sampled phrase pairs *anchor* and *target* as: 1.0 = Close Match; 0.75 = Close Synonym; 0.50 = Synonym; 0.25 = Somewhat Related; and 0 = Unrelated. Columns *anchor* and *target* collectively range in length from 1 to 15 words, with a mean *and* median of ~ two words. No missing or duplicative data were identified. Figures 1 and 2 show provided data coverage across contexts (CPCs) and labeled similarity scores, respectively:

|  |  |
| --- | --- |
| Figure 1 Labeled (Scored) Phrase Similarity by Context  Chart, bar chart  Description automatically generated | Figure 2 Phrase Similarity by Label (Score)  A picture containing text, weapon, knife  Description automatically generated |

Considering the study’s “nonlinear” semantic classification problem statement and hypothesis, the sample distributions illustrated in Figures 1 and 2 were only used to confirm sample representation (stratification) versus confirming overall data shape or outlier treatment for modeling.

## Multivariate Analysis

Considering the USPTO’s relatively simple, low-dimensional dataset and [primarily] textual nature, additional pre-model multivariate analysis – e.g., correlation or chi-square analysis – was not required.

# Results

As the study’s introduction highlights, NLP and this use case are a complex problem set, a perspective that helps to frame the *experimental* nature of the following model approaches.

## Modeling for Context

First, given a hypothesis that patent phrase pairs may be classified as similar *within context*, modeling began by analyzing how [provided] phrases relate to context – USPTO CPCs. Often, within NLP, this potential relationship is more *deterministic* than *probabilistic*, pre-established through a context-based corpus, for example. This relationship might also be explored through unsupervised methods like clustering. However, given a low-dimensional dataset and known problem, the study chose to provisionally evaluate the phrase-to-context relationship by using multinomial logistic regression to classify *context\_class* (CPCs) from *anchor* phrase, as illustrated in Equation 1. Recall from Table 1 that *anchor* and *target* are the provided phrase pair.

(1)

Note in Equation 1 that, while target *context\_class* is categorical, predictor *anchor* is textual. This required additional pre-processing to convert the *anchor’s* text to a numeric representation compatible with logistic regression. Multiple approaches are available for this text “vectorization” or embedding, including “bag of words” and others. The study chose TF-IDF, or *term frequency-inverse document frequency*, a method commonly used in search engines to find relevant documents. TF-IDF vectorization bases its approach on the product of a term’s frequency within a document and the term’s relative importance (Srinivasa-Desikan, 2018). In the study’s example, “document” is synonymous with “dataset” (set of phrase pairs).

Following text pre-processing, the provided training data was split 70/30 into training and test (validation) subsets. While USPTO technically provided a test dataset, this was unlabeled (unvalidatable) and very small, containing only one observation per context, or 36 unlabeled rows versus 10,942 labeled test observations resulting from the 70/30 split. Once training and test data subsets were created, k-fold cross-validation was performed against the [now subset] training data to confirm consistent mean accuracy across [subjectively] five folds. This cross-validation produced relatively stable values of 0.6904, 0.7043, 0.6992, 0.7004, and 0.7009. (Note that a *stratified* k-fold approach was used, which accommodates potential post-split class imbalance, even though it is clear from Figure 1 that full dataset target classes are balanced.)

Once split, the model was fit to [newly subset] training data and run against [newly subset] test data for evaluation. The following Figure 3 visualizes a multiclass confusion matrix for initial model results, supported by metrics across *context\_class* listed in Table 3:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Figure 3 Anchor à Context Confusion Matrix Application  Description automatically generated with low confidence | Table 3 Anchor à Context Classification Metrics   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Context** | **Precision** | **Recall** | **F1** | **Support** | | 0 | 0.78 | 0.83 | 0.81 | 1573 | | 1 | 0.73 | 0.77 | 0.75 | 1867 | | 2 | 0.58 | 0.47 | 0.52 | 433 | | 3 | 0.67 | 0.56 | 0.61 | 1252 | | 4 | 0.61 | 0.63 | 0.62 | 1176 | | 5 | 0.67 | 0.72 | 0.70 | 2458 | | 6 | 0.69 | 0.68 | 0.68 | 1821 | | 7 | 0.59 | 0.46 | 0.52 | 362 | | Accuracy |  |  | 0.69 | 10942 | |

According to these results (estimates), it appears possible to correctly classify context, or USPTO CPC, from provided phrasing nearly 70% of the time. Perhaps more useful, though, are the individual F1 scores or weighted averages of Precision and Recall, which vary widely across classes. Precision is the proportion of correct *context\_classk* predictions out of all *context\_classk* predictions, while Recall is the proportion of correct *context\_classk* predictions out of all actual *context\_classk* occurrences. For example, 78% of Chemistry and Metallurgy (0) classifications were this class, while 83% of actual Chemistry and Metallurgy occurrences were actually classified.

## Modeling for Word Similarity

Following phrase-to-context experimentation, steps were taken to better understand fundamental word similarity across *anchor* and *target* phrase pairs.

Our goal was to create a model that best predicts that semantic similarity between the anchor and the target by creating scores between 0 and 1, with 0 being not similar at all and 1 being identical. For all data analysis and modeling involved, we used Python, especially the Pandas software library.

Among the data sets obtained via Kaggle, “train.csv”, the training data set was explored in the attempts of creating the said model. The training data included the scores manually calculated by USPTO. The scores our models output in each experiment were compared to the scores in the training data to measure model performance.

As a preliminary approach, we directly compared and searched for overlapping words in anchor and target. Strings of words were broken into each word, and we identified if any same word appeared in both. For example, if the anchor is “abatement” and the target is “rent abatement”, we recognized the word “abatement” was repeated in both places. The scores were given based on the number of duplicated words divided by the number of maximum number of words possible. With this approach, we encountered issues of the model not being able to recognize synonyms. As an instance, while the anchor “abatement” and the target “act of abating” are highly synonymous, the model failed to recognize due to no overlapping exact words. Figure 4 shows the largest number of the cases do not share any same words, hence the cases had zero scores, as seen in Figure 5. As explained in Appendix A, this approach output scores were quite different from actual scores – hence we continued on exploring other modeling approaches such as logistic regression and BERT.

|  |  |
| --- | --- |
| **Figure 4**  *Duplicated Words in Training Data* | **Figure 5**  *Semantic Similarity Score in Training Data* |
|  |  |

## Modeling for Semantic Similarity

Recall from the “Modeling for Context” sub-section that TF-IDF was used to vectorize *anchor* phrases for logistic regression. TF-IDF works for this basic need with a limitation – it mechanically encodes words without considering *meaning* (semantics) (González-Carvajal & Garrido-Merchán, 2020). Enter *Bidirectional Encoder Representation from Transformers*, or BERT.

BERT is a deep learning NLP technique created by Google and uses an approach similar to *recurrent neural networks* (RNNs), which are often used for language and speech (Tan et al., 2019). Of note, BERT has been shown to outperform traditional approaches like TF-IDF (González-Carvajal & Garrido-Merchán, 2020). As a neural network-based approach, BERT also likely comes with computational expense over relatively simple methods like TF-IDF. The study side-steps a portion of this expense by leveraging a BERT model[[2]](#footnote-3) pre-trained across a wide corpus (Reimers & Gurevych, 2018). A very generalized objective function for a neural-network-based approach like BERT *may* be considered as Equation 2:

(2)

Given the availability of a pre-trained BERT model, the study used the full provided training dataset to test (validate) the semantic similarity of phrase pairs *anchor* and *target*, including calculating cosine similarity between BERT-generated vectors to arrive at single scores. Cosine similarity is useful for sparse data and is often applied to word-term similarity, for example (Tan et al., 2019). Once computed, these similarity scores were compared to labeled *scores* from the training dataset, as illustrated in Figure 6:

## Figure 6

Cosine Similarity vs. Labeled Score by Context

A picture containing application

Description automatically generated

While the Figure 6 visualization highlights *directional* model performance versus labeled scores, there is room for improvement. As a final experimental step, the study reviewed the distribution of cosine similarity vs. labeled scores, both naturally and with cosine re-scaled by quartiles to match the labeled range of 0, 0.25, 0.50, 0.75, and 1. (Cosine similarity normally ranges from -1 to 1; however, it will not drop below 0 in this case given positive word counts.) Figures 7 and 8 visualize this distribution analysis:

|  |  |
| --- | --- |
| Figure 7 Natural Distribution of Label vs. Model  A picture containing company name  Description automatically generated | Figure 8 Re-scaled Distribution of Label vs. Model  A picture containing text, weapon, knife, scissors  Description automatically generated |

# Discussion

## Hypothesis Review

Section “Modeling for Context” *begins* to show that patent phrases are related to patent subject areas (contexts) in some cases. The “Modeling for Word Similarity” section follows by illustrating the benefits of common word analysis across patents. Lastly, the section “Modeling for Semantic Similarity” begins to illuminate the possibilities of using near-state-of-the-art deep learning to identify “meaning” similarities across patent phrases. Ultimately, this progression of experimental approaches qualitatively rejects the null hypothesis () of no semantic classification possible in this use case.

## Strengths, Weaknesses, and Opportunities

While the study leads to qualified results, it is only a first step into the NLP problem domain, as highlighted in the introductory section. That said, opportunities exist to improve results within the study’s model approaches, specifically BERT. Here, more robust consideration for patent domain-based corpora and overall context weighting may help achieve scores closer to true labels while retaining the neural network-based BERT approach’s generalization.

# References

González-Carvajal, S., Garrido-Merchán, E.C. (2020). Comparing BERT against traditional machine

learning text classification. https://doi.org/10.48550/arXiv.2005.13012

MIT Sloan School of Management (2022, April 10). *Tapping the power of unstructured data*. MIT Sloan School of Management. https://mitsloan.mit.edu/ideas-made-to-matter/tapping-power-unstructured-data

Reimers, N., Gurevych, I. (2018). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. https://doi.org/10.48550/arXiv.1908.10084

Srinivasa-Desikan, B. (2018). *Natural Lange Processing and Computational Linguistics.* Packt.

Statista (2022, April 10). *Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2025*. Statista. https://www.statista.com/statistics/871513/worldwide-data-created/

Tan, P. N., Steinbach, M., Karpatne, A., Kumar, V. (2019). *Introduction to Data Mining*. Pearson.

U.S. Patent and Trademark Office (USPTO) (2022). \*.csv [Data Set]. Kaggle Inc. https://www.kaggle.com/competitions/us-patent-phrase-to-phrase-matching/data; https://www.kaggle.com/datasets/xhlulu/cpc-codes

U.S. Patent and Trademark Office (USPTO) (2022, April 13). *USPTO Classification Resources*. USPTO. https://www.uspto.gov/web/patents/classification/cpc/html/cpc.html

# Appendix A

Project Code

**Minsu Kim and Dave Friesen - ADS-502-02-SP22**

\_\_author\_\_ = 'Minsu Kim, Dave Friesen'  
\_\_email\_\_ = 'misukim@sandiego.edu, dfriesen@sandiego.edu'  
\_\_version\_\_ = '1.0'  
\_\_date\_\_ = 'April 2022'  
\_\_license\_\_ = 'MIT'

# Setup

# Import Python libraries  
import pandas as pd  
import numpy as np  
  
from sentence\_transformers import SentenceTransformer  
from sklearn.metrics.pairwise import cosine\_similarity  
  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import cross\_val\_score, StratifiedKFold  
from sklearn.model\_selection import train\_test\_split  
from sklearn import metrics  
from sklearn.feature\_extraction.text import TfidfVectorizer  
  
from collections import Counter  
import statistics  
  
import matplotlib.pyplot as plt  
import seaborn as sns

# Set Pandas options for consistent output  
pd.set\_option('display.float\_format', lambda x: '%.3f' % x)  
  
# Set working directory (depending on coder)  
%cd '/Users/davidfriesen/Desktop/OneDrive/projects/patent-semantics/data'

/Users/davidfriesen/Library/CloudStorage/OneDrive-Personal/projects/patent-semantics/data

# Set Matplotlib options for consistent visualization look 'n' feel  
plt.style.use('default')  
  
plt.rcParams['lines.linewidth'] = '1.5'  
  
plt.rcParams['axes.edgecolor'] = '#999999'  
plt.rcParams['axes.titlelocation'] = 'left'  
plt.rcParams['axes.titlesize'] = '12'  
plt.rcParams['axes.labelsize'] = '9'  
plt.rcParams['axes.labelcolor'] = '#999999'  
plt.rcParams['axes.spines.top'] = 'False' # Seaborn automatically despines for relplot()  
plt.rcParams['axes.spines.right'] = 'False' # Seaborn automatically despines for relplot()  
plt.rcParams['axes.prop\_cycle'] = plt.cycler(color = ['#0070c0'])   
PLT\_ALTERNATE\_COLOR = 'orange'  
  
plt.rcParams['xtick.color'] = '#cccccc'  
plt.rcParams['xtick.labelcolor'] = '#777777'  
plt.rcParams['xtick.labelsize'] = '9'  
plt.rcParams['ytick.color'] = '#cccccc'  
plt.rcParams['ytick.labelcolor'] = '#777777'  
plt.rcParams['ytick.labelsize'] = '9'  
  
plt.rcParams['figure.figsize'] = [7, 7 / (13 / 9)]  
plt.rcParams['figure.subplot.left'] = 0.17  
plt.rcParams['figure.subplot.right'] = 1  
plt.rcParams['figure.subplot.bottom'] = 0.19  
plt.rcParams['figure.subplot.top'] = 0.82  
plt.rcParams['figure.subplot.wspace'] = 0.05  
plt.rcParams['figure.subplot.hspace'] = 0.05

# Data Acquisition and Validation

# Instantiate base training and test dataframes  
train\_df = pd.read\_csv('train.csv')  
titles\_df = pd.read\_csv('titles.csv')  
test\_df = pd.read\_csv('test.csv')  
sample\_submission\_df = pd.read\_csv('sample\_submission.csv')

# Perform basic data validation steps by reviewing dataframe subset and metrics  
print('\ntrain\_df:\n', train\_df.head(5), '\n', train\_df.shape, sep = '')  
print('\ntitles\_df:\n', titles\_df.head(5), '\n', titles\_df.shape, sep = '')  
print('\ntest\_df:\n', test\_df.head(5), '\n', test\_df.shape, sep = '')  
print('\nsample\_submission\_df:\n', sample\_submission\_df.head(5), '\n', sample\_submission\_df.shape, sep = '')

train\_df:  
 id anchor target context score  
0 37d61fd2272659b1 abatement abatement of pollution A47 0.500  
1 7b9652b17b68b7a4 abatement act of abating A47 0.750  
2 36d72442aefd8232 abatement active catalyst A47 0.250  
3 5296b0c19e1ce60e abatement eliminating process A47 0.500  
4 54c1e3b9184cb5b6 abatement forest region A47 0.000  
(36473, 5)  
  
titles\_df:  
 code title section class \  
0 A HUMAN NECESSITIES A NaN   
1 A01 AGRICULTURE; FORESTRY; ANIMAL HUSBANDRY; HUNTI... A 1.000   
2 A01B SOIL WORKING IN AGRICULTURE OR FORESTRY; PARTS... A 1.000   
3 A01B1/00 Hand tools (edge trimmers for lawns A01G3/06 ... A 1.000   
4 A01B1/02 Spades; Shovels {(hand-operated dredgers E02F3... A 1.000   
  
 subclass group main\_group   
0 NaN NaN NaN   
1 NaN NaN NaN   
2 B NaN NaN   
3 B 1.000 0.000   
4 B 1.000 2.000   
(260476, 7)  
  
test\_df:  
 id anchor target context  
0 4112d61851461f60 opc drum inorganic photoconductor drum G02  
1 09e418c93a776564 adjust gas flow altering gas flow F23  
2 36baf228038e314b lower trunnion lower locating B60  
3 1f37ead645e7f0c8 cap component upper portion D06  
4 71a5b6ad068d531f neural stimulation artificial neural network H04  
(36, 4)  
  
sample\_submission\_df:  
 id score  
0 4112d61851461f60 0  
1 09e418c93a776564 0  
2 36baf228038e314b 0  
3 1f37ead645e7f0c8 0  
4 71a5b6ad068d531f 0  
(36, 2)

**These summary dataframe views show relatively simple structures as identifed by the data provider, including:**

id - unique identifier for phrase set  
anchor - first phrase  
target - second phrase  
context - CPC classification, which indicates subject within which the similarity is to be scored  
score - given similarity, sourced from a combination of one or more manual expert ratings

**Interesting to note that small size of the *provided* test dataset, which later analysis confirms to be simply a [single] representation of all contexts.**

# Univariate Analysis and Data Preparation

## Data Characteristic Review

# Understand sample characteristics (prospective model features and outcome(s)), including data types  
# (numerical - discrete, continuous; categorical - ordinal, nominal)  
print('\ntrain\_df data types:\n', train\_df.dtypes, sep = '')  
print('\ntest\_df data types:\n', test\_df.dtypes, sep = '')  
print('\nsample\_submission\_df data types:\n', sample\_submission\_df.dtypes, sep = '')

train\_df data types:  
id object  
anchor object  
target object  
context object  
score float64  
dtype: object  
  
test\_df data types:  
id object  
anchor object  
target object  
context object  
dtype: object  
  
sample\_submission\_df data types:  
id object  
score int64  
dtype: object

## Variable Type Transformations

**Given data definitions (see Data Acquisition and Validation above) and types, we determined that variable type transformations are not required for this simple dataset and problem statement.**

## Aggregate, Descriptive Statistic, and Distribution Review

# Categorize word pairs by 'Cooperative Patent Classification' (CPC), per provided context code  
context\_class = {'A': 'Human Necessities',  
 'B': 'Operations and Transport',  
 'C': 'Chemistry and Metallurgy',  
 'D': 'Textiles',  
 'E': 'Fixed Constructions',  
 'F': 'Mechanical Engineering',  
 'G': 'Physics',  
 'H': 'Electricity',  
 'Y': 'Emerging Cross-Sectional Technologies'}  
train\_df['context\_class'] = train\_df['context'].apply(lambda x: context\_class.get(x[:1]))

# Generate target word counts for analysis  
def word\_count(s):  
 l = len(str(s).split())  
 return l  
  
train\_df['target\_wc'] = train\_df[['target']].apply(lambda s: word\_count(\*s), axis = 1)

df = train\_df  
df[df['anchor'].duplicated() == False] # returns each individual row that "is not" a duplicate of other rows  
df[df['anchor'].duplicated() == False].count() # returns a total count of rows that are not duplicates

id 733  
anchor 733  
target 733  
context 733  
score 733  
context\_class 733  
target\_wc 733  
dtype: int64

# Categorize given scores (for analysis)  
score\_levels = [-1, 0, 0.25, 0.5, .75, 1]  
score\_labels = ['Unrelated', 'Somewhat Related', 'Synonym', 'Close Synonym', 'Close Match']  
train\_df['score\_match'] = pd.cut(  
 train\_df['score'],  
 bins = score\_levels,  
 labels = score\_labels  
)

# One other way to look at this is a boxplot view across contexts  
fig, ax = plt.subplots()  
sns.boxplot(  
 x = 'score', y = 'context\_class',  
 data = train\_df,  
 orient = 'h'  
).set(  
 title = 'Distribution of Training Scores across Contexts',  
 xlabel = 'Training Score',  
 ylabel = 'Context Classification (CPC)'  
)  
plt.tight\_layout()  
plt.show()  
fig.savefig('context-distribution.png')  
fig.savefig('context-distribution.svg')

Chart, bar chart

Description automatically generated

# Visualize/review distribution of provided lables - FOR UNDERSTANDING OF SAMPLE REPRESENTATION,  
# VS. EXPECTED REQUIREMENT FOR MODEL APPROACH  
fig, ax = plt.subplots()  
sns.kdeplot(  
 data = train\_df['score'],  
 fill = True  
).set(  
 title = 'Distribution of Training Scores (Target Sample Representation)',  
 xlabel = 'Training Score',  
)  
plt.xticks(score\_levels[1:], score\_labels)  
plt.tight\_layout()  
plt.show()  
fig.savefig('training-distribution.png')  
fig.savefig('training-distribution.svg')

A screenshot of a computer

Description automatically generated with low confidence

**Note that we don't review summary statistics in further detail here given the nature of our semantic classification problem statement and hypothesis. i.e., as noted in code comments, we're more interested in sample representation (stratification) than data distribution per model algorithm requirements.**

## Missing Value Review and Treatment

# Analyze null values - show, along with beginning row count of training data set  
rows = train\_df.shape[0]  
   
print('\nTotal Rows == ', rows, sep = '')  
print('\ntrain\_df nulls:\n', train\_df.isna().sum(), '\n', sep = '')  
print('\netest\_df nulls:\n', test\_df.isna().sum(), sep = '')

Total Rows == 36473  
  
train\_df nulls:  
id 0  
anchor 0  
target 0  
context 0  
score 0  
context\_class 0  
target\_wc 0  
score\_match 0  
dtype: int64  
  
  
etest\_df nulls:  
id 0  
anchor 0  
target 0  
context 0  
dtype: int64

# Analyze missing values - show, along with beginning row count  
print('\ntrain\_df missing\n\n\n', train\_df.info(show\_counts = True), sep = '')  
print('\ntest\_df missing\n\n\n', test\_df.info(show\_counts = True), sep = '')

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 36473 entries, 0 to 36472  
Data columns (total 8 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 id 36473 non-null object   
 1 anchor 36473 non-null object   
 2 target 36473 non-null object   
 3 context 36473 non-null object   
 4 score 36473 non-null float64   
 5 context\_class 36473 non-null object   
 6 target\_wc 36473 non-null int64   
 7 score\_match 36473 non-null category  
dtypes: category(1), float64(1), int64(1), object(5)  
memory usage: 2.0+ MB  
  
train\_df missing  
  
  
None  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 36 entries, 0 to 35  
Data columns (total 4 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 id 36 non-null object  
 1 anchor 36 non-null object  
 2 target 36 non-null object  
 3 context 36 non-null object  
dtypes: object(4)  
memory usage: 1.2+ KB  
  
test\_df missing  
  
  
None

## Outlier Review and Treatment

**With the key assumption that our training dataset is valid per the data provider, outlier treatment is n/a in the context of our problem statement, hypothesis, and model approach.**

## Duplicate Value Review and Treatment

# Do we have any duplicates in our core training and test datasets?  
print('\ntrain\_df duplicates\n\n', train\_df[train\_df.duplicated()], sep = '')  
print('\ntest\_df duplicates\n\n', test\_df[test\_df.duplicated()], sep = '')

train\_df duplicates  
  
Empty DataFrame  
Columns: [id, anchor, target, context, score, context\_class, target\_wc, score\_match]  
Index: []  
  
test\_df duplicates  
  
Empty DataFrame  
Columns: [id, anchor, target, context]  
Index: []

## Categorical Feature Standardization

**We do not have categorical features to factor/"standardize"**

## Data Normalization/Standardization/Rescaling

**We do not have have variables (prospective features) to normalize/standardize.**

## Data Augmentation (Other)

# Join extended training data with context detail, for further analysis  
train\_df = pd.merge(  
 train\_df,  
 titles\_df[['code', 'title']].set\_index('code'), how = 'left', left\_on = 'context', right\_index = True  
)  
train\_df.rename(columns = {'title': 'context\_title'}, inplace = True)

# Multivariate Analysis, Model and Feature Selection/Refinement

**NOTE: We are retaining this section for consideration and coverage; however, "page is intentionally blank" given our problem statememt, hypothesis, and model approach.**

### Key Q n

### Category Variable "Dummy" Encoding

### Model Feature Selection

### Model Feature Preparation

### Model Collinearity Check

# Model Train, Test, and Evaluation

## Modeling for Context - Logistic Regression

**Key Question: Can we classify context based on "anchor" phrases?**

# Create independent predictors and dependent response (target, label)  
X = train\_df['anchor']  
y = train\_df['context\_class']  
  
# Create a TF-IDF (term frequency-inverse document frequency) transformer, with word/phrase relevance  
# based on sklearn's "standard" corpus  
transformer = TfidfVectorizer(  
 stop\_words = 'english',  
 lowercase = True,  
 max\_features = 150000  
)  
  
# Create training and test datasets, all from training dataset since it's essentially our full  
# working dataset  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.30, random\_state = 0)  
  
# Apply TF-IDF transformation to [string-concatenated] phrase combination (predictor)  
X\_train = transformer.fit\_transform(X\_train)  
X\_test = transformer.transform(X\_test)

# Create model  
model = LogisticRegression(  
 solver = 'lbfgs',  
 max\_iter = 1000,  
 multi\_class = 'multinomial',  
 random\_state = 17  
)  
  
#  
skf = StratifiedKFold(n\_splits = 5, shuffle = True, random\_state = 17)  
cv = cross\_val\_score(model, X\_train, y\_train, cv = skf, scoring = 'f1\_micro')  
print(cv, cv.mean())  
  
# Fit model to training data  
model = model.fit(X\_train, y\_train)  
  
# Run model predictions (validation)  
y\_pred = model.predict(X\_test)

[0.69042491 0.70426949 0.69917744 0.70035253 0.70094007] 0.6990328858242464

# Generate classification report, confusion matrix, and multilabel confusion matrix (effectively  
# 2x2 for all classes) and summarize model results  
cr = metrics.classification\_report(y\_test, y\_pred)  
cm = metrics.confusion\_matrix(y\_test, y\_pred)  
mcm = metrics.multilabel\_confusion\_matrix(y\_test, y\_pred)  
score = model.score(X\_test, y\_test)  
  
# Output each  
print(cr)  
print(cm)  
print(mcm)  
  
# Visualize confusion matrix  
fig, ax = plt.subplots()  
sns.heatmap(  
 data = cm,  
 cmap = 'Blues\_r',  
 annot = True,  
 fmt = ".0f", linewidths = .5,  
 annot\_kws = {"fontsize":8}  
).set(  
 title = 'Confusion Matrix; Accuracy Score: {0:.4f}'.format(score),  
 xlabel = 'Predicted Label',  
 ylabel = 'Actual Label'  
)  
labels = pd.DataFrame(pd.DataFrame(y\_train).groupby('context\_class'))  
labels = labels[labels.columns[0]].values.tolist()  
#ax.set\_xticklabels(labels, rotation = 90)  
ax.set\_yticklabels(labels, rotation = 0)  
plt.tight\_layout()  
plt.show()  
fig.savefig('confusion-matrix.png')  
fig.savefig('confusion-matrix.svg')  
  
# Do standard calcs (for practice)  
FP = cm.sum(axis = 0) - np.diag(cm)   
FN = cm.sum(axis = 1) - np.diag(cm)  
TP = np.diag(cm)  
TN = cm.sum() - (FP + FN + TP)  
  
# Sensitivity, hit rate, recall, or true positive rate  
TPR = TP / (TP + FN)  
# Specificity or true negative rate  
TNR = TN / (TN + FP)   
# Precision or positive predictive value  
PPV = TP / (TP + FP)  
# Negative predictive value  
NPV = TN / (TN + FN)  
# Fall out or false positive rate  
FPR = FP / (FP + TN)  
# False negative rate  
FNR = FN / (TP + FN)  
# False discovery rate  
FDR = FP / (TP + FP)  
  
# Overall accuracy  
ACC = (TP + TN) / (TP + FP + FN + TN)

precision recall f1-score support  
  
Chemistry and Metallurgy 0.78 0.83 0.81 1573  
 Electricity 0.73 0.77 0.75 1867  
 Fixed Constructions 0.58 0.47 0.52 433  
 Human Necessities 0.67 0.56 0.61 1252  
 Mechanical Engineering 0.61 0.63 0.62 1176  
Operations and Transport 0.67 0.72 0.70 2458  
 Physics 0.69 0.68 0.68 1821  
 Textiles 0.59 0.46 0.52 362  
  
 accuracy 0.69 10942  
 macro avg 0.67 0.64 0.65 10942  
 weighted avg 0.69 0.69 0.69 10942  
  
[[1305 51 0 41 25 105 33 13]  
 [ 14 1432 15 21 91 95 184 15]  
 [ 20 7 202 38 38 77 41 10]  
 [ 105 82 41 706 56 177 65 20]  
 [ 40 55 41 54 740 188 41 17]  
 [ 93 101 38 80 169 1782 167 28]  
 [ 64 208 8 97 59 143 1230 12]  
 [ 28 18 4 15 26 83 20 168]]  
[[[ 9005 364]  
 [ 268 1305]]  
  
 [[ 8553 522]  
 [ 435 1432]]  
  
 [[10362 147]  
 [ 231 202]]  
  
 [[ 9344 346]  
 [ 546 706]]  
  
 [[ 9302 464]  
 [ 436 740]]  
  
 [[ 7616 868]  
 [ 676 1782]]  
  
 [[ 8570 551]  
 [ 591 1230]]  
  
 [[10465 115]  
 [ 194 168]]]

Application

Description automatically generated with medium confidence

## Modeling for Word Similarity

# counting words in anchor column  
wordcount\_anchor = train\_df['anchor'].str.split().str.len()  
wordcount\_anchor

0 1  
1 1  
2 1  
3 1  
4 1  
 ..  
36468 2  
36469 2  
36470 2  
36471 2  
36472 2  
Name: anchor, Length: 36473, dtype: int64

wordcount\_anchor.max()

5

# using the maximum number of words in 'anchor' plus one for the number of columns so each word in a row gets its own column  
split\_anchor = train\_df['anchor'].str.rsplit(' ', 6, expand=True)  
split\_anchor

0 1 2 3 4  
0 abatement None None None None  
1 abatement None None None None  
2 abatement None None None None  
3 abatement None None None None  
4 abatement None None None None  
... ... ... ... ... ...  
36468 wood article None None None  
36469 wood article None None None  
36470 wood article None None None  
36471 wood article None None None  
36472 wood article None None None  
  
[36473 rows x 5 columns]

# counting words in target column  
wordcount\_target = train\_df['target'].str.split().str.len()  
wordcount\_target

0 3  
1 3  
2 2  
3 2  
4 2  
 ..  
36468 2  
36469 2  
36470 2  
36471 2  
36472 2  
Name: target, Length: 36473, dtype: int64

wordcount\_target.max()

15

# doing the same as we did for anchor  
split\_target = train\_df['target'].str.rsplit(' ', 16, expand=True)  
split\_target

0 1 2 3 4 5 6 7 8 \  
0 abatement of pollution None None None None None None   
1 act of abating None None None None None None   
2 active catalyst None None None None None None None   
3 eliminating process None None None None None None None   
4 forest region None None None None None None None   
... ... ... ... ... ... ... ... ... ...   
36468 wooden article None None None None None None None   
36469 wooden box None None None None None None None   
36470 wooden handle None None None None None None None   
36471 wooden material None None None None None None None   
36472 wooden substrate None None None None None None None   
  
 9 10 11 12 13 14   
0 None None None None None None   
1 None None None None None None   
2 None None None None None None   
3 None None None None None None   
4 None None None None None None   
... ... ... ... ... ... ...   
36468 None None None None None None   
36469 None None None None None None   
36470 None None None None None None   
36471 None None None None None None   
36472 None None None None None None   
  
[36473 rows x 15 columns]

def dupe\_wc(x):  
 x = str(x).split()  
 d = 0  
 for key, val in Counter(x).items():  
 d = d + (val > 1)  
 return d  
  
train\_df['anchor\_target'] = train\_df['anchor'] + ' ' + train\_df['target']

#Each row would represent each patent case, showing all words in anchor and target.  
train\_df['anchor\_target']

0 abatement abatement of pollution  
1 abatement act of abating  
2 abatement active catalyst  
3 abatement eliminating process  
4 abatement forest region  
 ...   
36468 wood article wooden article  
36469 wood article wooden box  
36470 wood article wooden handle  
36471 wood article wooden material  
36472 wood article wooden substrate  
Name: anchor\_target, Length: 36473, dtype: object

train\_df['dupecount'] = train\_df['anchor\_target'].apply(dupe\_wc)

#This shows the number of words that appear in both anchor and target  
train\_df['dupecount']

0 1  
1 0  
2 0  
3 0  
4 0  
 ..  
36468 1  
36469 0  
36470 0  
36471 0  
36472 0  
Name: dupecount, Length: 36473, dtype: int64

df1=train\_df['dupecount']

#Most cases (18,246) have no exact words that appear both in anchor and target.   
#However 13,701 cases have 1 word duplicated in both anchor and target  
# in 4 cases, 5 words are duplicated. Considering the max word count in anchor is 5, the sets of words would be   
# identical between anchor and target.  
df1.value\_counts()

0 18246  
1 13701  
2 4036  
3 440  
4 46  
5 4  
Name: dupecount, dtype: int64

df1.value\_counts().plot(kind= 'barh')  
plt.xlabel("Number of cases")  
plt.ylabel("Number of words duplicated in both anchor and target")

Text(0, 0.5, 'Number of words duplicated in both anchor and target')

Chart, bar chart

Description automatically generated

#Now we divide the number of duplicated words by 5, which is the maximum number of words duplicated,  
# to achieve the scores for each cases between 0 and 1.   
#i.e. for the 4 cases where all 5 words are duplicated, the score would be 1.  
train\_df['dupecount']/5

0 0.200  
1 0.000  
2 0.000  
3 0.000  
4 0.000  
 ...   
36468 0.200  
36469 0.000  
36470 0.000  
36471 0.000  
36472 0.000  
Name: dupecount, Length: 36473, dtype: float64

df2=train\_df['dupecount']/5  
df2.value\_counts().plot(kind= 'barh')  
plt.xlabel("Number of cases")  
plt.ylabel("Semantic Similiary Scores")

Text(0, 0.5, 'Semantic Similiary Scores')

Chart, bar chart

Description automatically generated

avg = sum(df2) / len(df2)  
round(avg, 4)

0.1277

train\_avg = sum(train\_df['score']) / len(train\_df['score'])  
round(train\_avg, 4)

0.3621

statistics.median(df2)

0.0

statistics.median(train\_df['score'])

0.25

**Based on the simple comparison of average and medians, we can conclude that calculating the semantic scores using the number of exact words duplicated in anchor and target is not a good approach, since such method yields significantly different results from actual scores given in the training data.**

## Modeling for Semantic Similarity - BERT

**(Bidirectional Encoder Representation from Transformers)**

**Key Question: To what extent can we leverage open source Artificial Neural Networks (and supporting corpora) to classify phrase pairs in terms of semantic similarity?**

# Generate text embeddings and resulting similarity scores for full training set  
# Note we are using SBERT (SentenceBERT), a Python neural-network based. We are starting with a  
# model pre-trained across a wide corpus, leveraging this vs. narrowly building and training here.  
model\_name = 'bert-base-nli-mean-tokens'  
model = SentenceTransformer(model\_name)  
  
# Function for determining/applying phrase pair similarity  
def sim\_score(a, b):  
 sentence\_vecs = model.encode([a, b])  
 sim = cosine\_similarity([sentence\_vecs[0]], sentence\_vecs[1:])[0, 0]  
 print(a, b, sim)  
 return sim  
  
# The following was completed (long execution time) and written (below) to an extended training data set  
#train\_df['sim'] = train\_df[['anchor', 'target']].apply(lambda t: sim\_score(\*t), axis = 1)  
#train\_df.to\_csv('train\_ext.csv', index = False)  
  
# Subsequently, just read in this extended training data with similarty score and merge join with core set  
train\_ext\_df = pd.read\_csv('train\_ext.csv')  
train\_df = pd.merge(  
 train\_df,  
 train\_ext\_df[['id', 'sim']].set\_index('id'), how = 'left', left\_on = 'id', right\_index = True  
)

# Visualize/review distribution of our SentenceTransform/cosine similarity scores vs. given training scores  
fig, ax = plt.subplots()  
sns.boxplot(  
 x = 'sim', y = 'score\_match',  
 data = train\_df,  
 order = ['Close Match', 'Close Synonym', 'Synonym', 'Somewhat Related', 'Unrelated']  
).set(  
 title = 'Similarity Score Distribution relative to Training Scores (Target)',  
 xlabel = 'Similarity Score',  
 ylabel = 'Training Score\n(labeled confidence level)'  
)  
plt.tight\_layout()  
plt.show()  
fig.savefig('similarity-v-training.png')  
fig.savefig('similarity-v-training.svg')

Chart, box and whisker chart

Description automatically generated

# Subset and pivot table for heat map visualization  
hm = train\_df.pivot\_table(  
 values = 'sim',  
 index = 'score\_match',  
 columns = 'context\_class',  
 aggfunc = np.mean,  
 sort = False)  
  
# Visualize similarity score vs. given score by context  
fig, ax = plt.subplots()  
sns.heatmap(  
 data = hm, annot = True  
).set(  
 title = 'Similarity Score Performance across Contexts',  
 xlabel = 'Patent Context Classification (CPC)',  
 ylabel = 'Training Score\n(labeled confidence level)'  
)  
ax.invert\_yaxis()  
plt.tight\_layout()  
plt.show()  
fig.savefig('similarity-v-training-by-context.png')  
fig.savefig('similarity-v-training-by-context.svg')

A picture containing table

Description automatically generated

# 'Normalize' similarity scores in quartiles for ~distribution comparison to given scores  
def q(val, q25, q50, q75, q100):  
 return (((q25 < val <= q50) \* 0.25) +   
 ((q50 < val <= q75) \* 0.5) +   
 ((q75 < val <= q100) \* 0.75) +  
 ((q100 < val) \* 1.0))  
  
a = np.quantile(train\_df['sim'], 0.25)  
b = np.quantile(train\_df['sim'], 0.50)  
c = np.quantile(train\_df['sim'], 0.75)  
d = np.quantile(train\_df['sim'], 1.0)  
train\_df['sim\_norm'] = train\_df['sim'].apply((lambda x: q(x, a, b, c, d)))  
  
# Visualize/review distribution of provided lables  
fig, ax = plt.subplots()  
sns.kdeplot(  
 data = train\_df['score'],  
 fill = True  
).set(  
 title = 'Distribution of Training vs. Similarity (Modeled) Scores',  
 xlabel = 'Training Score',  
)  
  
sns.kdeplot(  
 data = train\_df['sim'],  
 fill = True,  
 color = PLT\_ALTERNATE\_COLOR  
)  
  
plt.xticks(score\_levels[1:], score\_labels)  
plt.tight\_layout()  
plt.show()  
fig.savefig('training-vs-similarity-distribution.png')  
fig.savefig('training-vs-similarity-distribution.svg')

A picture containing chart

Description automatically generated

# There is some statistical analysis to do here, including understanding the scale of cosine similarity vs.  
# the given scale, which is more probability-based.  
# We may have a case to make about lack of context within semantic similarity, which either needs to  
# be overcome in the model itself or in the context within which the model is used?  
# An alternative is we could train based on a contextual corpus. . .  
# etc.

# Re-scale similarity scores in quartiles for ~distribution comparison to given scores  
def q(val, q25, q50, q75, q100):  
 return (((q25 < val <= q50) \* 0.25) +   
 ((q50 < val <= q75) \* 0.5) +   
 ((q75 < val <= q100) \* 0.75) +  
 ((q100 < val) \* 1.0))  
  
a = np.quantile(train\_df['sim'], 0.25)  
b = np.quantile(train\_df['sim'], 0.50)  
c = np.quantile(train\_df['sim'], 0.75)  
d = np.quantile(train\_df['sim'], 1.0)  
train\_df['sim\_norm'] = train\_df['sim'].apply((lambda x: q(x, a, b, c, d)))  
  
# Visualize/review distribution of provided lables  
fig, ax = plt.subplots()  
sns.kdeplot(  
 data = train\_df['score'],  
 fill = True  
).set(  
 title = 'Distribution of Training vs. Re-scaled Similarity (Modeled) Scores',  
 xlabel = 'Training Score',  
)  
  
sns.kdeplot(  
 data = train\_df['sim\_norm'],  
 fill = True,  
 color = PLT\_ALTERNATE\_COLOR  
)  
  
plt.xticks(score\_levels[1:], score\_labels)  
plt.tight\_layout()  
plt.show()  
fig.savefig('training-vs-norm-similarity-distribution.png')  
fig.savefig('training-vs-norm-similarity-distribution.svg')

A picture containing text, knife, weapon, scissors

Description automatically generated

# Appendix B

Presentation

A picture containing arrow

Description automatically generatedGraphical user interface, application, Word

Description automatically generatedGraphical user interface, application, Word

Description automatically generatedGraphical user interface, application

Description automatically generatedGraphical user interface, application, Word

Description automatically generatedA picture containing chart

Description automatically generatedGraphical user interface, application

Description automatically generatedTable

Description automatically generatedGraphical user interface, application, Word

Description automatically generatedGraphical user interface, text, application

Description automatically generatedGraphical user interface

Description automatically generated with low confidenceGraphical user interface, application, Word

Description automatically generatedGraphical user interface

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

1. Note that, while the source refers to this dataset as a population (), the study views the dataset as a sample () – or subpopulation – of the overall population, to allow for inference beyond the dataset to the larger domain (“corpus”) of patent applications. [↑](#footnote-ref-2)
2. For experimental purposes, the study used a pre-trained model called “bert-base-nli-mean-tokens”, which is now deprecated in favor of higher-quality embeddings in this rapidly evolving area. [↑](#footnote-ref-3)