# Machine-Learning Approach for Cross-Domain Acronym Definition Identification

Maya Varma and Rachel Gardner

December 2017

## 1 Introduction

The large quantity of data uploaded to the Internet daily presents numerous challenges in information retrieval and analysis. A major challenge of automated text analysis is the identification of acronyms. Acronyms, such as HTML and GAN, are abbreviations formed by combining the first letters of a series of words. Although commonly used, acronyms can often be ambiguous in meaning, with each acronym resulting in numerous possible definitions. According to Liu et al., almost 81% of acronyms used in MEDLINE abstracts are ambiguous, and we expect that this value is higher when considering all webpages on the Internet [4]. Thus, an automatic tool for identifying appropriate definitions for acronyms is essential.

To address this problem, we are designing a machine-learning based classifier to match ambiguous acronyms with accurate definitions based on context.

## 2 Related Work

In existing literature on acronym definition extraction, potential acronyms are known as "potential short forms," or PSFs. Potential definitions are known as "potential long forms," or PLFs. Because there is such a wide range of possible tasks to tackle in this space, it is important to enumerate the various goals often pursued.

1. identifying PSF and PLF pairs
   *(ex: identifying "HTML" as an acronym, and determining that the surrounding text likely contains its definition)*
2. determining which PLF is correct for a given short form
   *(ex: out of the possible long forms, determining one is the correct definition that matches the acronym, in this case "Hypertext Markup Language")*
3. determining the correct LF for an acronym that is undefined in the text
   *(ex: given just "HTML" and potentially the context it was used in, determining what the meaning of "HTML" is)*

Many of the existing commercial systems for identifying acronym definitions are simply a lookup table containing every possible definition for any given acronym, independent of context. The datasets used are created manually, whether by employees [1] or through crowd-sourcing [5]. Thus they are solving goal 3, but leave it to their users to choose between various definitions.

On the research side, most of the literature has been focused on goal 1 and 2. Because our research goal was primarily goal 3, the identification of acronym definitions based on context, we were looking to apply existing techniques to solve tasks 1 and 2. These techniques rely upon hand-coded programs to evaluate potential short forms and their potential long forms.

For example, in 2003, Schwartz and Hearst created a simple algorithm which starts at the end of a potential LF and at the end of a potential SF, then moves right to left, trying to find the shortest long form which matches the short form [7]. The algorithm matches any character in the LF with one in the SF, with the only constraint that they must appear in order. The approach performed well; they achieved 96% precision and 82% recall on the Medstract corpus. However, the Medstract corpus only contains SF-LF pairs in the following forms:

1. long form ( short form )
2. short form ( long form )

Thus this work ignored acronyms introduced in other ways, which proved unviable in the context of less structured data like the Wikipedia articles we were training on.

Because so much of the interest in identifying acronyms has come from the biomedical field, many authors rely upon the Medstract corpus and a related "gold standard" dataset published by the same group. Thus many of these approaches make the same initial assumptions as Schwartz and Hearst (though they perform less accurately, according to the literature review provided by Schwartz and Hearst) [3].

More recently, researchers have begun applying machine learning models to tasks 1 and 2 (extracting acronyms and their meanings). However, these approaches rely on an initial set of hand-labelled data to train the model to distinguish between correct definitions and random context [8] [2] [9].

The closest approach that we could find was that of Yeganova et. al [9], who automatically generated a dataset of naturally labelled data to teach a machine learning system how to differentiate between true LFs for a given SF, and random noise from the surrounding text. The approach did well on Medstract (93% recall, 95% accuracy). However, the model produced from this method (if it had been made public) would likely not have generalized to other domains (and still suffers from the initial assumptions about SFs and LFs in text).

Thus our work is the only paper to solve task (3) as far as we know, as well as the only cross-domain system to handle tasks (1) and (2).

# 3 Task Definition

The purpose of our project is to design a machine-learning based classifier that can accurately identify definitions of acronyms based on surrounding context. To do so, we implemented a multiclass classification model, which allows us to generate accurate predictions when there are multiple possible definitions for an acronym; each possible definition represents a class. A classification model is appropriate for this task because the definitions are discrete instances.

We are utilizing a supervised learning approach to pair each acronym with a correct definition. The input for our classifier is a segment of text containing an acronym. When run on the training data, the model learns weights that correspond to context feature vectors (X) and the true definition (y). Then, when run on testing data, the model accurately predicts the appropriate definition for

each acronym based on surrounding context. The classifier outputs the appropriate definitions for each identified acronym in the input text.

We evaluated the accuracy of our model by utilizing F1 scores to measure the percentage of correctly classified acronym-definition pairs. F1 scores account for potential class imbalance and serve as a robust accuracy metric.

Figure 1 shows the processing pipeline for our project. Details on each stage of our project are explained below.

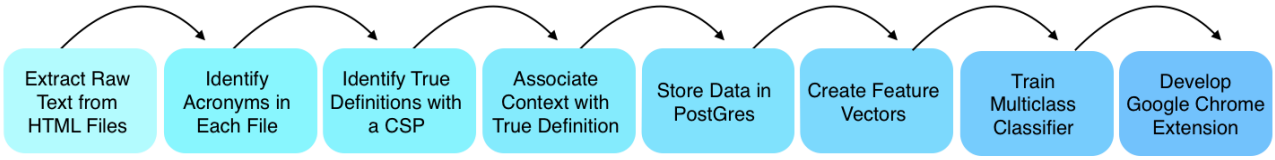


Figure 1: Processing Pipeline

# 4 Data Acquisition and Preprocessing

## 4.1 Wikipedia Text Mining

In order to generate a large corpus of text, we used Wikipedia HTML files as input for the classifier. To obtain the data, we created a web-crawler to query the Wikipedia API, recursively storing URLs for articles across eight STEM categories:

```
['Lists of medical abbreviations', 'Medicine',
'Health', 'Biology',
'List of computing and IT abbreviations', 'Computing',
'Electronics', 'Engineering']
```

The result was URLs for over 1000 Wikipedia articles. We then shuffled the URLs and divided them into training and test sets (with a 70/30 split). From there, we downloaded and cleaned the raw HTML, using a modified version of the code provided in the Python Natural Language Toolkit (NLTK) package. We then tokenized the string into words, keeping those that satisfied the following constraints:

1. all capital letters
2. length greater than 2
3. contains only alphabet characters
4. not in blacklist (e.g not "ABSTRACT", "RESULTS" etc.)
5. not part of a header (as determined by whether or not surrounding words are in all-caps)

Once we identified an acronym in this way, we stored both the acronym and its index in the array of words for future processing.

## 4.2 Automated Detection of Ground-Truth Definitions

Once we identified acronyms, the challenge came in determining what part of the surrounding text contained the proper definition (if any). Since this was the area focused on by the majority of other researchers, we hoped to be able to leverage their existing approaches. However, many of the rule-based models had neither sample code available nor public datasets. Since we were unwilling to manually label our own train and test data, we developed a simple rule-based approach, which involved determining every set of consecutive words surrounding an acronym with the same length as the acronym; then, a confidence score was assigned based on the number of words in each set with first letters matching a letter in the acronym. The set with the highest score was selected as the definition. However, this introduced a lot of noisy labels in our dataset. Then, in our attempt to solve this problem, we found open-source code for more elaborate rule-checking (from Schwartz and Hearst, as mentioned previously). Unfortunately, the code did not find nearly as many acronym-definition pairs, simply because it only worked for a very particular structure of acronym-definition pairs [7].

Finally, we designed a labeling algorithm based on a constraint satisfaction problem (CSP). Given a particular acronym, the code would run two CSPs: one on the words before the acronym, and one on the words after. Each variable in the CSP was one letter in the given acronym, and the domain of each variable was all possible characters in the surrounding sentence that could match that letter. We began by only implementing basic factors such as whether or not a character was uppercase, and whether or not the character was the first character in the word. We also enforced the constraint that the matching characters must be sequential. With only these simple factors, we were able to perform significantly better than the previous approaches. Future work could include adding constraints such as whether or not words were found in parentheses, or how far away from the acronym each individual word is.

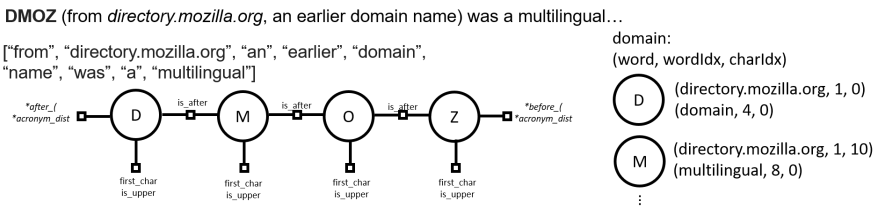


Figure 2: CSP for extracting acronym definition from surrounding words

After this process, we had a training dataset consisting of 3038 unique acronyms along with their respective definitions. In order to ensure that we had a broad range of acronyms for testing, we created two test sets. The "breadth" test set contains a variety of acronyms from different domains, each with an average of 1.084 definitions. Then, in order to validate performance of our system in cases of acronym definition collisions, we also constructed a targeted "depth" dataset with an average of 2.038 definitions per acronym.

| | Training | Test I - Breadth | Test II - Depth |
|---|---|---|---|
| **Unique Acronyms** | 3038 | 612 | 26 |
| **Number of Occurrences of Acronyms in Text** | 10946 | 2832 | 925 |
| **Average Number of Definitions Per Acronym** | 1.124 | 1.084 | 2.038 |

Figure 3: Description of Training and Testing Data

### 4.3 Data Storage

To handle the large volume of data, we created a PostgreSQL backend to store acronym-definition pairs, including their context. We created a specific `AcronymDatabase` class to wrap the necessary calls. At train time, the Python scripts call the database to get the ground truth labels. However, once a model has been trained, it can run inference directly from its trained weights. A visual representation of the database is shown in Figure 4.

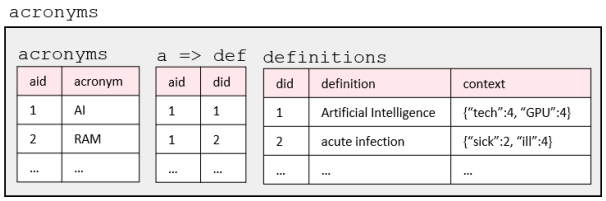


Figure 4: Database Layout

## 5 Methods

### 5.1 Feature Representations

When an acronym is encountered in text, a high-dimension feature vector is created to effectively characterize surrounding context. We utilized two approaches for representing features: frequency vectors and GloVe vectors.

#### 5.1.1 Frequency Feature Representations

Frequency feature vectors are sparse vectors that characterize context; for this feature representation, context is defined as a set of thirty words consisting of the fifteen words preceding and the fifteen words succeeding the acronym. The frequency of words that appear in the context is calculated and stored as a Counter object. Stop words, which include inconsequential words such as "the" and "an", are excluded from the feature vector. Since acronyms with multiple possible definitions often occur in different contexts, this method ensures that the classifier will be able to distinguish between multiple potential definitions.

For example, consider the case where our input data includes two Wikipedia articles titled "Cell adhesion molecule" and "Engineering". These two articles are included in our set of training data, and both contain the acronym CAM with two very distinct definitions:

```
CAM ['cell adhesion molecule', 'computer aided manufacturing']
```

Since the acronym CAM has two distinct definitions, we can see that the word is used in distinct contexts. A frequency feature vector is generated for each occurrence of CAM in text; the following are two of the frequency feature vectors generated for CAM.

```
{'computer aided manufacturing': Counter({'manufacturing': 2, 'engineering': 2,
'software': 2, 'tasks': 1, 'process': 1, 'tools': 1, 'management': 1, 'support': 1,
'production': 1, 'cnc': 1, 'generate': 1, 'instructions': 1, 'computer-aided': 1,
'machining': 1})}
{'cell adhesion molecule': Counter({'igsf': 3, 'binding': 1, 'heterophilic': 1,
'different': 1, 'fibrinogen': 1, 'platelet': 1, 'integrins': 1, 'increased': 1,
'aggregation': 1, 'superfamily': 1, 'bind': 1, 'immunoglobulin': 1, 'either': 1,
'homophilic': 1})}
```

Finally, since the acronym itself is the most important predictor of its definition, we included the corresponding acronym as a feature with a value of 10 in each feature vector.

### 5.1.2 GloVe Feature Representations

Global Vectors for Word Representation (GloVe) refers to an unsupervised machine learning algorithm utilized for obtaining vector representations for words [6]. Developed at Stanford in 2014, GloVe generates clusters of similar words and helps identify correlations. We downloaded a set of pre-trained GloVe word vectors from the Wikipedia text corpus, which consists of 400,000 unique words with corresponding 20-dimension vectors.

We define context to be a set of thirty words consisting of the fifteen words preceding and the fifteen words succeeding the acronym, similar to the frequency representation. Then, we identified the four words in the context that occurred most frequently; if multiple words occurred with the same frequency, ties were broken randomly. We then determined four corresponding GloVe vectors for each of the selected words, and concatenated them to form a single non-sparse 80-dimension feature vector for the acronym.

For example, consider the example from the previous section with the acronym CAM. A GloVe feature vector is generated for each occurrence of CAM in text; the following are two of the GloVe feature vectors generated for CAM. In the first case, where CAM is defined as "computer aided manufacturing", the algorithm concatenates four GloVe vectors for the context words "tasks", "support", "process", and "circuit". In the second case, where CAM is defined as "cell adhesion molecule", the algorithm concatenates four GloVe vectors for the context words "integrins", "superfamily", "increased", and "platelet". Vectors have been truncated in this report for clarity.

```
{'computer aided manufacturing': [-1.73, -1.366, -1.3504, -1.0219, -0.99729, …, 50],
 'cell adhesion molecule': [-1.991, -1.7809, -1.6889, -1.3528, -1.2691, …, 50]}
```

Finally, similar to the frequency representation, we included the corresponding acronym as a feature with a value of 50 in each feature vector. This introduces some sparsity into the feature representation.

## 5.2 Supervised Learning Approaches

We utilized a multiclass machine learning model, which allows us to generate accurate predictions when there are multiple possible definitions for an acronym.

We created two separate models for each of the four algorithms, where one utilized frequency feature representation and the second utilized GloVe feature representations. To train our models, we parsed HTML documents in our training set and identified 10,946 acronyms. The context for

each acronym (X) was recorded and used to generate feature vectors. Feature vectors were then paired with the corresponding true definition of the acronym (y) and then used to train the model.

For our baseline, we performed a naive look-up of each acronym in the STANDS4 acronym database without considering context. Our oracle involved manually identifying definitions by performing an Internet look-up of the acronym and any known identifying text. Performance results are presented in the next section. We then implemented four supervised learning algorithms: multinomial naive bayes, support vector classifier with hinge loss, decision tree, and random forest.

The multinomial naive bayes implements the naive bayes algorithm for multiple classes and estimates parameters based on maximum likelihood; this algorithm is typically used for text classification. However, this algorithm could not be used with GloVe feature representation since feature vectors include negative values. The support vector classifier minimizes hinge loss when performing multiclass classification; this algorithm is effective when feature representations are high-dimensional. Decision trees estimate decisions and possible consequences based on input feature vectors. The random forest algorithm constructs a variety of decision trees to determine the most frequently predicted class. We tuned hyperparameters for all four algorithms using 3-fold cross-validation.

We tested our algorithm on both our breadth and depth datasets, using F1 scores to measure performance.

### 5.3 Models

We created a `serve.py` program based on the frequency feature representation with a Naive Bayes algorithm. This program is called by both a Moxel model and our Chrome extension. The Chrome extension allows users to highlight text in a webpage, then right click to "look up" the acronym. This process then calls `serve.py` (via a web server) with the selected text. The returned definition for the acronym(s) identified in the text are then displayed as a small pop-up underneath the selected text.

## 6 Results and Analysis

### 6.1 Frequency Feature Representation

| | Training | | | | Test I - Breadth | | | | Test II - Depth | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | Recall | F1 | Accuracy | Precision | Recall | F1 | Accuracy | Precision | Recall | F1 | Accuracy |
| Naive Bayes | 0.997 | 0.997 | 0.997 | 0.997 | 0.946 | 0.941 | 0.933 | 0.941 | 0.931 | 0.928 | 0.9155 | 0.928 |
| SVC | 0.999 | 0.999 | 0.999 | 0.999 | 0.944 | 0.933 | 0.927 | 0.933 | 0.937 | 0.919 | 0.9092 | 0.919 |
| Decision Tree | 0.999 | 0.999 | 0.999 | 0.999 | 0.899 | 0.848 | 0.853 | 0.848 | 0.935 | 0.873 | 0.8748 | 0.873 |
| Random Forest | 0.992 | 0.994 | 0.993 | 0.994 | 0.813 | 0.805 | 0.786 | 0.805 | 0.867 | 0.844 | 0.8222 | 0.844 |

Figure 5: F1 Scores for Frequency Feature Representation

F1 scores and raw accuracy measures for the models trained on frequency feature representations are shown in Figure 5. The Multinomial Naive Bayes model showed the greatest performance,

with an F1 score of 0.99 on the training set, 0.93 on the breadth testing set, and 0.912 on the depth testing set.

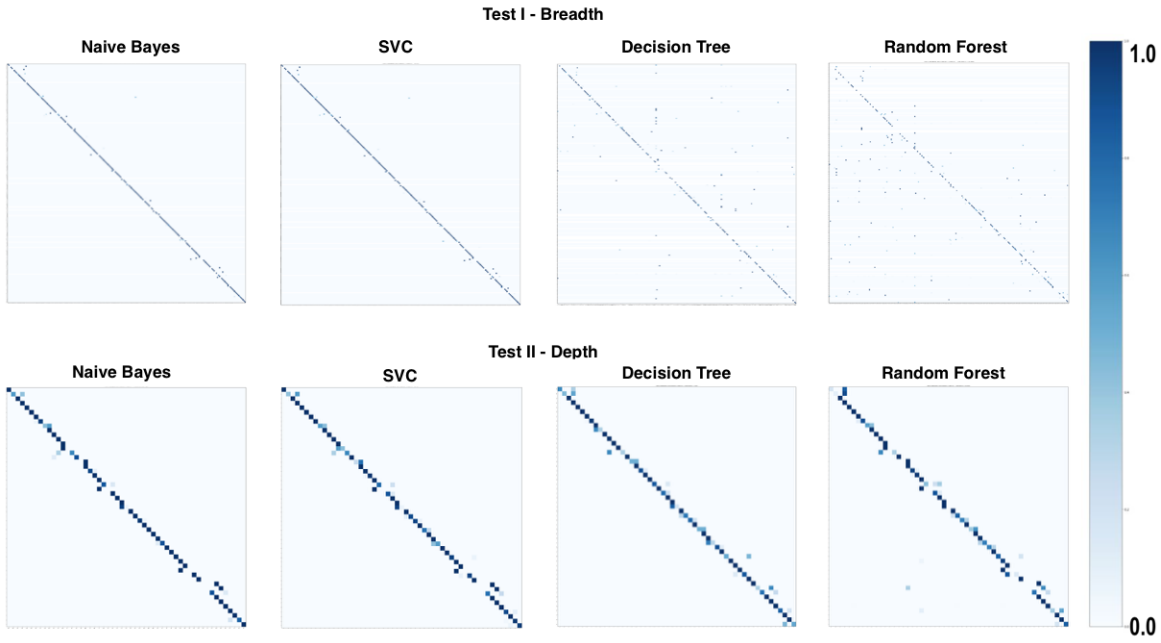


Figure 6: Confusion Matrices for Frequency Feature Representation

We created confusion matrices for each model to visually represent error on each test set. A perfect classifier with an error of 0% will have a confusion matrix with a perfect diagonal; deviations from the diagonal indicate error. Figure 6 shows confusion matrices for each machine learning classifier trained with frequency feature representations.

As mentioned above, our results show that the Multinomial Naive Bayes algorithm had the greatest performance. Typically, models trained with the multinomial naive bayes algorithm are utilized for machine-learning problems involving text classification; for example, most email spam clasisfiers are based on Naive Bayes. Since identification of definitions for acronyms is inherently a text classification problem, it makes sense that the Naive Bayes algorithm delivered the best results.

In addition, our results are consistent between the breadth and depth testing sets, indicating that the model can perform accurately across a wide range of distinct acronyms that may have multiple definitions.

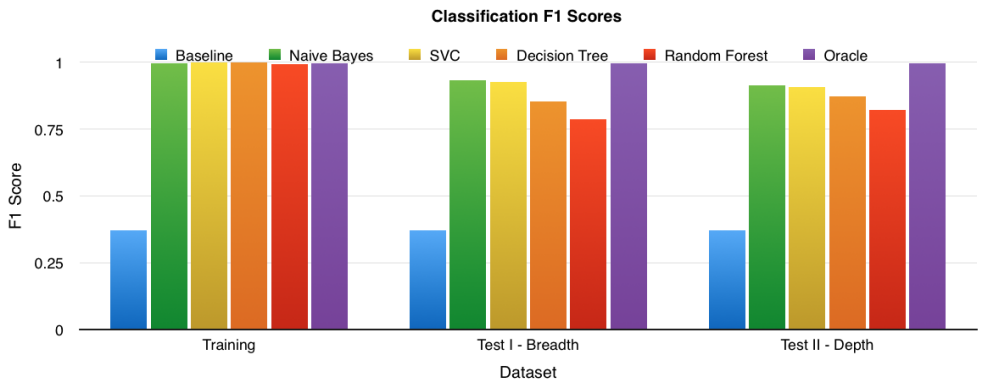

Figure 7: F1 Scores by Classifier (Frequency Feature Representation)

A visual representation of F1 scores across all classifiers is shown in Figure 7. Our baseline accuracy was 0.37 and our oracle accuracy was 1; since the baseline and oracle could not be measured with f-1 scores (since they were not classifiers), the raw accuracy values are reported instead. We see that all four classifiers show a significant improvement from the baseline algorithm.

## 6.2 GloVe Feature Representations

| | Training | | | | Test I - Breadth | | | | Test II - Depth | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | Recall | F1 | Accuracy | Precision | Recall | F1 | Accuracy | Precision | Recall | F1 | Accuracy |
| SVC | 0.999 | 0.999 | 0.999 | 0.999 | 0.907 | 0.902 | 0.896 | 0.902 | 0.874 | 0.860 | 0.853 | 0.860 |
| Decision Tree | 0.999 | 0.999 | 0.999 | 0.999 | 0.867 | 0.810 | 0.821 | 0.810 | 0.805 | 0.739 | 0.749 | 0.739 |
| Random Forest | 0.993 | 0.994 | 0.993 | 0.994 | 0.658 | 0.604 | 0.578 | 0.604 | 0.764 | 0.664 | 0.665 | 0.664 |

Figure 8: F1 Scores for GloVe Feature Representation

F1 scores and raw accuracy measures for the models trained on GloVe feature representations are shown in Figure 8. The support vector classifier with hinge loss showed the greatest performance, with an F1 score of 0.999 on the training set, 0.896 on the breadth testing set, and 0.853 on the depth testing set.

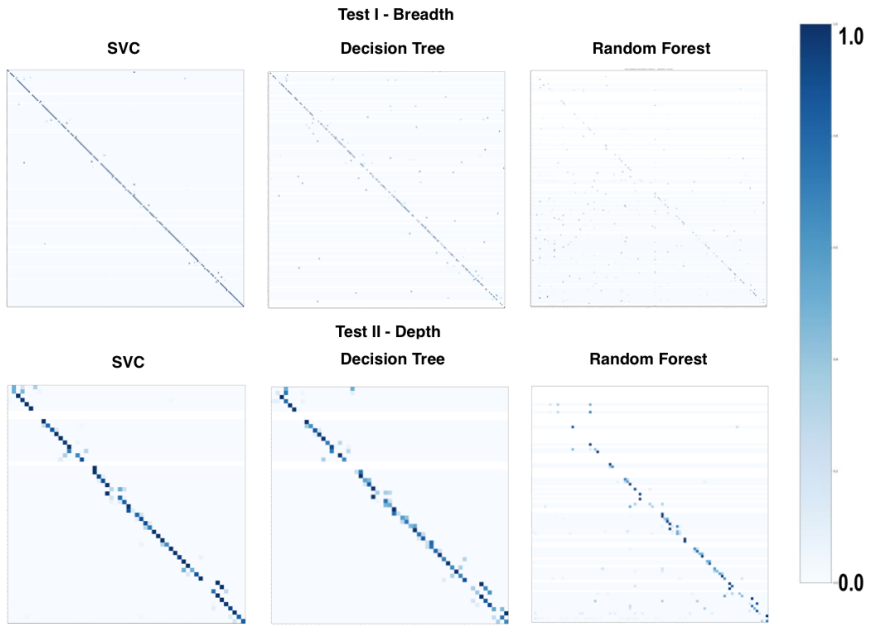


Figure 9: Confusion Matrices for GloVe Feature Representation

We again created confusion matrices for each model to visually represent error for each test set. Figure 9 shows confusion matrices for each machine learning classifier trained with GloVe feature representations.

As mentioned above, our results show that the Support Vector Classifier with hinge loss algorithm had the greatest performance. SVCs are typically used when there is a large number of features and training instances; this explains the high performance on this dataset. However, we see that the GloVe feature representation generally performed worse than the frequency feature representation. This could be because the GloVe vectors do not always characterize context in an effective manner; for example, if we examine the example from Section 5.1.2 with the acronym 'CAM', we see that the algorithm identified the generic context words "support" and "process", which were later used to generate GloVe vectors. It is possible that a better subset of words could have been selected to more effectively characterize the acronym, and we hope to address this problem in the future.

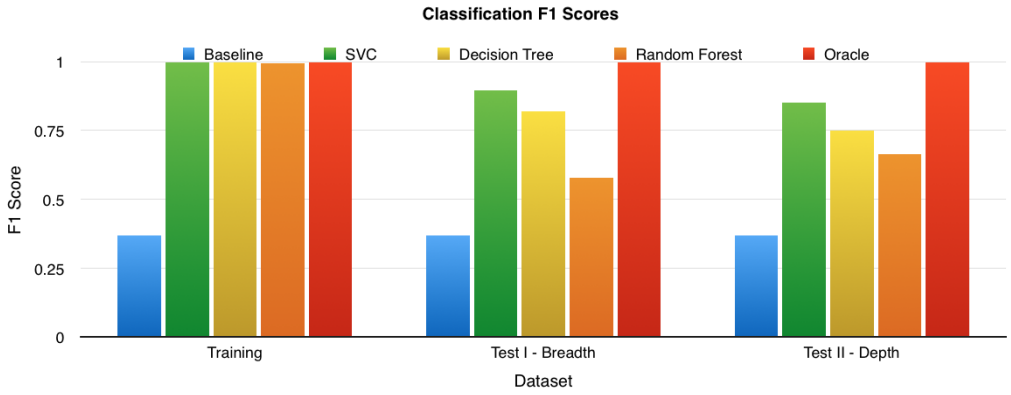


Figure 10: F1 Scores by Classifier (GloVe Feature Representation)

Again, our results are consistent between the breadth and depth testing sets. A visual representation of F1 scores across all classifiers is shown in Figure 10. We see that all four classifiers show a significant improvement from the baseline algorithm accuracy of 0.37.

# 7 Discussion and Conclusion

The purpose of this project was to design a machine learning-based classifier to identify definitions for ambiguous acronyms in text. To do so, we created datasets by mining text from Wikipedia, labeled ground truth by developing a CSP-based algorithm, created a PostgreSQL backend for storing data, and trained four machine learning classifiers based on two different feature representations. The Naive Bayes algorithm with frequency feature representations obtained the highest performance on testing sets, with an F1 score greater than 0.9 on both testing sets. Frequency feature representations generally performed better than GloVe feature representations.

Overall, our results show that we were able to create a highly-accurate machine-learning classifier that can identify definitions for acronyms based on surrounding text. We have deployed our model as a Google Chrome extension called Acronym Lookup, and we hope that in the future, we will be able to host the Chrome extension on the web and deploy the system to the public.

Please see the following links for additional details about this project.
- Moxel Model: http://beta.moxel.ai/models/maya/acronyms/latest
- Video Demonstration of Google Chrome Extension: https://youtu.be/LvdDEqyZvpQ
- Source Code (GitHub Repository): https://github.com/maya124/AcronymLookup

# References


[1] *Acronym Finder*. URL: https://www.acronymfinder.com/.

[2] Cheng-Ju Kuo et al. "BIOADI: a machine learning approach to identifying abbreviations and definitions in biological literature". In: *BMC Bioinformatics* 10.15 (Dec. 2009), S7. ISSN: 1471-2105. DOI: 10.1186/1471-2105-10-S15-S7. URL: https://doi.org/10.1186/1471-2105-10-S15-S7.

[3] Leah Larkey et al. "Acrophile: An Automated Acronym Extractor and Server". In: Proceedings of the Fifth ACM Conference on Digital Libraries. (July 2–7, 2000). San Antonio, Texas, USA: ACM, pp. 205–214.

[4] H. Liu, A. R. Aronson, and C. Friedman. "A study of abbreviations in MEDLINE abstracts". In: *Proc AMIA Symp* (2002), pp. 464–468.

[5] The STANDS4 Network. *Abbreviations.com*. URL: http://www.abbreviations.com/.

[6] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543.

[7] A.S. Schwartz and M.A. Hearst. "A Simple Algorithm for Identifying Abbreviation Definitions in Biomedical Text". In: Pacific Symposium on Biocomputing 8. 2003, pp. 451–462.

[8] Jun Xu and Ya-Lou Huang. "A machine learning approach to recognizing acronyms and their expansion". In: *2005 International Conference on Machine Learning and Cybernetics*. Vol. 4. Aug. 2005, 2313–2319 Vol. 4. DOI: 10.1109/ICMLC.2005.1527330.

[9] L. Yeganova, D. C. Comeau, and W. J. Wilbur. "Machine learning with naturally labeled data for identifying abbreviation definitions". In: *BMC Bioinformatics* 12 Suppl 3 (June 2011), S6.