



SentNet

Next Generation Document Scoring
Using Advanced Graph Analytics

Executive Summary

The volatility of the world in the twenty-first century necessitates that our nation's leaders have access to detailed and timely intelligence products that allow them to make informed decisions in a rapid manner. However, the analysts that produce these products are increasingly being asked to collect, interpret and synthesize insights from ever growing volumes of data stretching their time and existing resources. Given this inherent tension between the IC customers' needs and analysts' limited time and resources, it is clear that ODNI and other IC agencies could benefit from new tools and methodologies to assist analysts in drafting accurate and timely intelligence products. While there are numerous existing Natural Language Processing (NLP) tools and methods, such as Automatic Essay Scoring (AES), none fully meet the requirements that ODNI outlined for a robust intelligence product scoring solution.

In response to this need, we have developed and implemented an analytical prototype called SentNet. In short, SentNet is an advanced document scoring tool designed for the Intelligence Community that allows agencies to define and develop highly accurate models to automatically score intelligence products providing analysts with instant feedback about the content and quality of their reports. Combining methods from academia, the commercial sector, and the open-source community, SentNet is able to identify more nuanced linguistic features that can be used to predict how an intelligence product will score against establish ODNI or agency specific criteria. In contrast to existing NLP methods SentNet offers analysts and agencies the following capabilities:

1. **Advanced Feature Extraction:** SentNet uses cutting edge machine learning and linguistic methods including WordNet, Doc2Vec, linguistic network analysis, Louvain clustering, and perceptual hashing to develop 12 classes of features for each document. When combined, these features allow SentNet to "see" more nuanced structures within documents compared to other NLP methods.
2. **Model Chaining:** Within SentNet document level predictions are dependent on not one, but multiple models, each tasked with a different function. SentNet then constructs a final model to ensemble (combine) outputs from these multiple input models and features resulting in more accurate and stable predictions.

In this paper, we discuss how we implemented a prototype of SentNet using standard open-source software libraries in conjunction with customized code. To test its performance, we developed 43 different models using SentNet and compared the predicted document scores from those models to scores assigned by professional human graders. These essays covered a

variety of topics and essays were scored according to a range of different criteria. Initial results from these models are promising. SentNet correctly classified 61% of essays, producing a score identical to that given by a human reviewer with an overall Root Mean Square Error (RMSE) of only 0.6. This means that, for most models, 98% of SentNet's document score predictions fall within one point of the professionally assigned score. Given the relatively limited amount of time allocated to model tuning/adjustment, this performance demonstrates that SentNet's methodology is fully capable of rapidly evaluating and numerically scoring brief analytic reports with no human intervention.

Additionally, given our team's decade of work in the intelligence industry and the data science domain, including the delivery of tools and solutions involving graph-based text analytics, we firmly believe that SentNet has the ability to transform the IC analytic toolset. Our unique solution, combined with industry-specific experience, offers a creative and robust solution to IC analysts and agencies.

Table of Contents

Executive Summary	0
Table of Contents	2
Overview	4
Existing Solutions	4
Proposed Solution	5
Features	6
Methodology	6
Scorecard Model Creation	6
Readability Features:	7
WordNet Feature Generation:	7
Graph Extraction:	8
Document Similarity Matching:	9
Perceptual Hashing:	9
Random Forest Modeling:	10
Scoring of New Documents	10
Solution Requirements	11
Implementation	12
Overview	12
Development Software	13
Data	13
Scorecard (Model) Development Process	14
Data Ingest	14
Readability Feature Calculation	15
Document Matching	15
Document Similarity	16
WordNet Generalization and Graph Extraction	17
Data Cleaning	18
Word-Level Feature Extraction	18
Word Graph Feature Generation	19
Synset-Level Feature Generation	20
Synset Graph Feature Generation	22
Perceptual Hashing Feature Extraction	24
Random Forest Modeling	24

Modeling Process	24
Modeling Results: Overview	25
Modeling Results: Performance	25
Modeling Results - Feature Importance	28
Modeling Results: Future Enhancements	30
Scoring of New Documents	30
Scorecard (Model) Export	31
New Scorecard Creation	31
Model Update Process	31
Risks & Mitigating Factors	32
Future Improvements	32
Scoring Visualization	32
Recommended Technology Stack	32
Front-End	32
Back-End	33
User Interface	33
RSEATS Analysis Landing Page	33
Visualize SentNet Output in User Interface	34
Parallelization	36
Summary	36
Appendix A. SentNet Technical Requirements	37
System Resources	37
Required Python Packages	37
Appendix B. SentNet Installation Instructions	39
Appendix C. Training and Validation Data Preparation	40
Data Description and Dictionary	40
Data Preparation	40
Data Ingestion and Conversion	41
Image Injection into Documents	42
Appendix D. Model Results	45
Modeling Metrics	45
Modeling Feature Importance	46

Overview

Natural Language Processing (NLP) made huge strides in the late twentieth and early twenty-first centuries and is now commonly utilized across a wide variety of industries including business, education and information technology. However, particular advances in the last few years have resulted in significant increases in the number and variety of NLP techniques that are available to analysts. In this document, we outline a new system called SentNet that utilizes many of these new innovations that will enable agencies to automatically and accurately revise and score intelligence products.

In the following sections, we first review existing solutions for reviewing documents. Next we outline SentNet's core features that improve upon these existing techniques as well as SentNet's underlying methods. Then we map SentNet's capabilities to the Challenge Requirements before moving on to a detailed description of our SentNet's implementation. The implementation section includes a thorough discussion of SentNet's full capabilities, including references to specific sections of our submitted code, enabling a complete understanding of the system. Finally, we will look toward the future and outline our vision of user workflows in SentNet as well as potential user-interface enhancements before offering some summarizing thoughts.

Existing Solutions

The ability to use computers to automate the task of reading, comprehending, and evaluating human-composed documentation has made significant strides in the last two decades. Advances in NLP, have enabled analytics professionals to derive insights from vast quantities of text-based data to solve a wide range of problems including translation, sentiment analysis, document summarization, topic extraction and document classification. NLP has applications across countless domains including marketing, information technology, healthcare, education, and defense.

Specific implementations of NLP for automatic document evaluation and scoring generally fall under the umbrella of two main methodologies. The first is a structure-based analysis found in Applicant Tracking Systems (ATS) commonly employed by human resource departments to automatically filter out electronic copies of resumes as part of the hiring process. Such systems focus their efforts on the identification, validation and/or extraction of predefined entities or structures within the document. The other approach, Automatic Essay Scoring (AES), is utilized in the educational field as a means of automatically grading computer-based essays such as the Graduate Management Admission Test (GMAT) and the Graduate Records Examination. Rather than concerning themselves with text structure, AES systems key in on semantics (i.e. topic relevance, flow, etc.) and text-production skills (i.e. grammar, vocabulary, spelling, etc.).

The primary benefits associated with both of these systems are speed and standardization. With the rapid increase in the amount of text-based data available to organizations, the ability to rapidly sift through data in a quick and efficient manner is vital. For example, ATS is a vital tool for modern human resource departments, enabling them to rapidly sift through the large amount of information generated by online job applications. Similarly, the use of automated

systems also has the added benefit of normalizing the process used to evaluate and/or score associated documents. In the case of AES systems, the last document graded will be treated as equally as the first utilizing the same methodology. It is much more difficult to ensure a consistent evaluation from a human evaluator for every document they inspect as they are subject to negative influences such as bias and fatigue.

Unfortunately, these solutions are not free of problems which have hindered their ability to fully replicate the performance of humans. The main weakness of structure-based/text-production solutions is their overreliance on objective, rule-based criteria for classifying and scoring. If the task at hand is subjective in nature, semantic-based analysis is required as systems constrained to predefined models will often struggle to recognize context and creativity. Furthermore, such systems will struggle to adapt to text and language it was not trained on such as changing style, structure, or language; events that would be expected to commonly occur in the IC.

Proposed Solution

While ATS and AES continue to be used across multiple industries, operational requirements at the Office of the Director of National Intelligence (ODNI) necessitate a next generation solution that meets the following requirements:

1. Rapidly evaluate and numerically score brief (1-2 page) analytic intelligence products against specified criteria with no human intervention.
2. Allow analysts and analytic managers to easily understand and validate these scores with minimal training.

To address this need, we are proposing a new system called SentNet. In short, SentNet is a next generation document analysis tool that utilizes advanced part-of-speech tagging and graph analysis techniques to evaluate and score documents using numerous contextual features. In doing so, SentNet fulfills the aforementioned ODNI requirements of rapidly evaluating, numerically scoring, and easily validating intelligence products with the potential to display results through an intuitive user interface.

Features

At its core, SentNet is an advanced network analysis tool. Using the power of graphs, SentNet is able to detect relationships between textual elements and identify analytical reasoning with respect to those elements. Representing a document as a graph allows SentNet to use individual and aggregate features from a document's derived word network to assign scores for each of the RSEATS (*Rating Scale of Evaluating Analytic Tradecraft Standards*) criteria. SentNet goes beyond existing document scoring methods to offer ODNI these additional capabilities:

- **Complex Document Scoring:** In comparison to ATS, AES, and other existing methods, SentNet allows for complex document scoring by revealing complex and latent textual and semantic features within a document. These features are extracted using established methods drawn from advanced natural language processing and social network analysis (described in more detail in the next section). Upon extracting these generalized features from an analytic product, SentNet's models can predict RSEATS

scores for that product based on generalized semantic features that have been found to represent arguments, judgments, assessments and assumptions.

- **Refinement of valuable, existing document analysis methods:** SentNet includes many valuable existing capabilities that are used for the scoring of documents including:
 - Ability to search for the presence or absence of specific n-grams (terms and/or a collection of terms)
 - Generation of readability scores to assess the understandability of text utilizing existing indices (i.e. Dale-Chall, Flesch, Flesch-Kincaid, Gunning Fog, Smog, etc.)
 - Comparing media (charts, graphs, and maps) in new documents to media used in previous reports using perceptual hashing
- **Intuitive Visual Representation:** SentNet allows analysts and analytic managers to view the relationships between a report's elements, as well as the RSEATS scores derived from those elements, using an intuitive user interface. While we have not provided a fully interactive version of this user-interface in the iteration of SentNet submitted with this proposal, we have provided the following in the Future Improvements section:
 - All the required data structures and summaries required to populate the UI
 - Prototype UI wireframes depicting the various recommended visualizations and features
 - Technology platform suggestions for use in the implementation of the UI

Methodology

Scorecard Model Creation

In order to generate RSEATS scores for new documents, SentNet must first generate a RSEATS Scorecard model. This requires that SentNet have access to a sample of previously scored analytic products (similar to the Sample Analytics Product provided for this challenge). Using sample scored documents, SentNet imports, transforms, analyzes, and summarizes these documents to generate complex textual features which are then used to develop a robust model that is capable of predicting RSEATS scores for a new document.

Developing such a model within SentNet involves the following steps:

1. Readability Features:

The five readability indexes utilized by SentNet are:

- a. Dale-Chall
- b. Flesch
- c. Flesch-Kincaid
- d. Gunning Fog
- e. Smog

The purpose of readability tests are to indicate how difficult a passage of text is to understand or comprehend. Different tests have different formulas but the general idea across all tests is the same: provide a score based on characteristics such as average word length or sentence length in order to assign a reading grade level or a measurement of linguistic difficulty. For example, the Flesch-Kincaid grade level index

presents a document score as a U.S. Grade level to imply the minimum level of education necessary to comprehend a document based on the total number of words, sentences and syllables. The Gunning Fog index attempts to determine the same score, the number of years of formal education necessary to understand a text on the first reading, but it does so by focusing on complex words (those containing more than three syllables) rather than just syllables.

2. WordNet Feature Generation:

SentNet will first take a sample of existing analytic products (defined by a user or agency) and “generalize” them using an advanced part-of-speech tagger based on the WordNet Lexical Database (an open source, license free, lexical database hosted by Princeton University)¹. The WordNet database groups nouns, verbs, adjectives and adverbs into sets of cognitive synonyms (synsets), each expressing a distinct concept. In this way WordNet resembles a thesaurus, given that it groups words together based on their meanings. However, there are some important differences between WordNet and a Thesaurus. First, WordNet interlinks not just word forms, but specific senses of words. As a result, words that are found in close proximity to one another in WordNet are semantically disambiguated. Second, WordNet labels the semantic relations among words, whereas the groupings of words in a thesaurus do not follow any explicit pattern other than meaning similarity.

SentNet makes use of these unique features by substituting all words in a document for their higher order (most generalizable) synset equivalents represented by a synset code. To assist with this process, it is recommended that future SentNet developers create a customized WordNet lexical database that has been tailored to include topics, terms, and acronyms that are commonly discussed in the US Intelligence Community. In contrast to other text generalization and ‘part-of-speech’ tagging solutions, WordNet enables SentNet to more easily identify and compare common semantic features and structures (including arguments, methods, assumptions and judgements) across analytic products. The WordNet generalization process is depicted in Figure 1, shown below.

Officials want to strengthen partnerships between the government and industry.

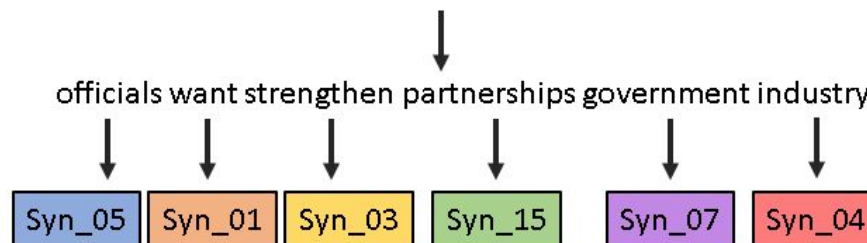


Figure 1. WordNet Generalization of Sentence

¹ Princeton University. (2018). *WordNet: A lexical database for English*. Retrieved from <https://wordnet.princeton.edu/>

The WordNet Feature Generation creates two categories of results: Word-based and Synset-based. More specifically, feature sets and graph networks (discussed in the next section) are produced for both categories, all of which serve as inputs to the SentNet classification model. The exact details of this feature generation process and the classification mode itself are covered in the Implementation section of this proposal.

3. Graph Extraction:

Once a document has been generalized into its synsets using the WordNet database, documents will be converted into textual graphs representing the relationships between all synsets in a document based on their co-location (i.e. synsets in the same sentence will be considered to be connected). This process is depicted in Figure 2, shown below.

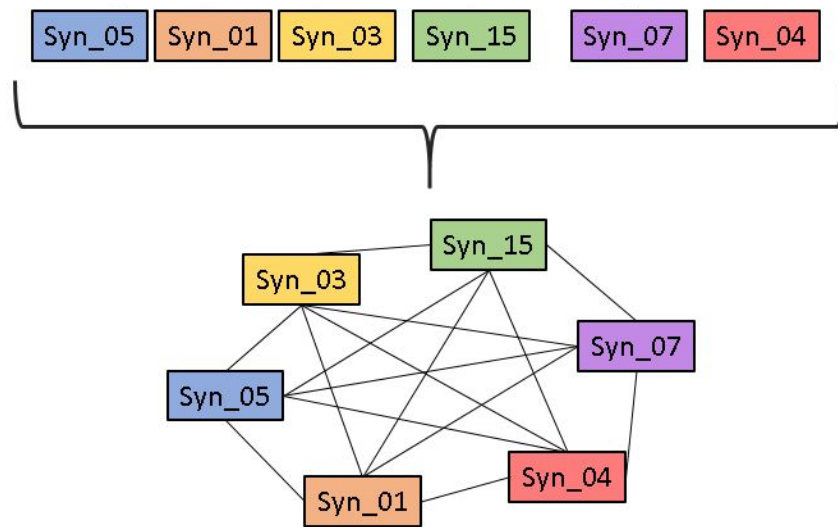


Figure 2. Graph Construction from Synsets

Once all analytic products in our sample are represented in this way all graphs are scanned to identify common synset features that pertain to key RSEATS criteria including arguments, methods, assumptions and judgements found within analytic products. Examples of such features could include:

- **Synset Presence/Absence:** SentNet will scan for the presence or absence of a single synset that relates to key RSEATS criteria such as assumptions, judgements, and logical argumentation (such as synsets that contain argumentation like ‘must’, ‘however’, ‘could’). SentNet could also be trained to scan for specific edges/connections between synsets in a graph (for example the synsets that include ‘must’ and ‘however’ occurring in the same sentence).
- **Synset Centrality:** Metrics about the relative location of synsets within a graph can also be useful. For example, SentNet could measure the betweenness centrality (a measure of importance/influence in a graph) for key terms in the graph. For example, the betweenness centrality of the ‘cyber’ synset would reflect the degree to which it is the central topic for a given document.
- **Synset Clusters & Communities:** Finally, SentNet can look for common “clusters” or “communities” of synsets within documents that represent some the most complex

and nuanced semantic features. Based on the presence/absence of these synset clusters, SentNet will be able to understand what types of arguments, methods, assumptions, judgements, and other logical features are used in an analytic product. The process of identifying common clusters and/or communities is accomplished using the Louvain method. The method simply attempts to optimize the “modularity” of a network by creating clusters of closely connected nodes. It does so by first partitioning every node into its own community. The method looks to create small communities by optimizing the modularity at a “local” level. Once the modularity has been fully optimized, it then combines nodes that belong to the same community, and repeats the optimization process treating these aggregated communities as individual nodes. This process is repeated until maximum modularity is achieved.²

4. Document Similarity Matching:

Once a document is scored, SentNet runs documents through a secondary document similarity matching algorithm. Documents that are highly similar (such as template reports that are updated weekly) will likely have similar RSEATS scores. Therefore, this similarity measure can be used to validate SentNet’s estimated scores for a given report if a similar/nearly identical report already exists in our sample.

5. Perceptual Hashing:

During the data ingest process, all images contained within every document are extracted and made available for perceptual hashing analysis. A perceptual hash is a method for fingerprinting images such that images that are similar to one another will generate comparable (but not identical) hash values. The degree of similarity between two hashes (determined by edit distance) represents the likeness of the two images that generated the hashes. It is important to note that similarity in this context refers to nearly duplicate images -- copyright violation type similarity rather than conceptual similarity.

6. Random Forest Modeling:

Once these features are derived, high and low performing analytic products in our sample will be compared. SentNet will use the 12 feature sets generated previously as inputs to multiple random forest models (one for each criterion) to estimate the RSEATS scores for each analytic product. These feature sets include:

- Readability Scores
- Predicted Class (based on matching documents)
- Predicted Class (based on similar documents)
- Word Counts (Word Features)
- Word Co-occurrence Counts (Word Edge Features)
- Word Graph Centrality Values
- Word Based Clusters
- Synset Counts (Synset Features)
- Synset Co-occurrence Counts (Synset Edge Features)

² <https://perso.uclouvain.be/vincent.blondel/research/louvain.html>

- Synset Graph Centrality Values
- Synset Based Clusters
- Perceptual (Image) Hashing Features

With all of these features as inputs, a Random Forest model allows SentNet to consider the importance of numerous features and find interactions between features while remaining highly generalizable (reduces the likelihood of overfitting). Once trained, this model can be used to predict the RSEATS scores for new reports.

Scoring of New Documents

Once an RSEATS scorecard model has been developed, new reports can be automatically scored by SentNet using the following process:

1. **Import:** A document(s) are uploaded to SentNet for analysis. This can be done by an analyst via the user interface or can be automatically flagged for scoring by copying documents to a predefined SentNet folder. Documents could be uploaded in a variety of formats including .txt, .doc, .docx. PDFs could be accommodated depending on ODNI's requirements.
2. **Select RSEATS Scorecard (Model):** SentNet is capable of storing multiple scorecards (given that different missions and agencies could have differing scoring criteria). Therefore, users must either develop a new scorecard or select the RSEATS scorecard model they would like to compare their report or reports against.
3. **Readability Features:** The first feature to be created in the scoring process is readability scores. These scores are generated using the entirety of each ingested document, in its raw form, and are created for each of the five aforementioned indexes.
4. **Document Cleaning:** Once imported, a document is cleaned and standardized for analysis. This involves:
 - a. **Extracting and Substituting Images** – Images (if any) are extracted and saved for analysis later on. A text identifier is substituted for the image in the document (i.e. Image #1)
 - b. **Clean Text** – The document's text is then cleaned by lower-casing all text, removing 'stop-words' from a provided list (i.e. 'a', 'the', 'at', etc). Common acronyms are substituted for their full equivalent to allow for easier term matching.
5. **WordNet Normalization:** Once a document has been cleaned it will be generalized by running it through the previously described WordNet Part-of-speech tagger. This serves to "generalize" the document by substituting terms for their synset equivalents.
6. **Construct Graph:** After a document is generalized into its component synsets the document is broken down by sentence. All synsets that are present within a sentence are assumed to be connected and will share a relationship in the synset graph. SentNet

will repeat this process of adding synsets and relationships between synsets until the entire document has been scanned.

7. **Extract Synset Graph Features:** Once a graph is built, SentNet will scan the graph to look for features as well as clusters present in the selected RSEATS scorecard model. Based on the presence/absence of these features SentNets' Random Forest Models will generate estimated scores for the document, one for each criteria.
8. **Document Similarity Scoring:** Finally, SentNet will compare the provided document to previously scored/validated documents in the provided corpus. This similarity scoring will take two forms:
 - a. **Document Similarity:** SentNet will calculate the Sequence Similarity between the synset graph for the submitted document and the synset graphs for all other documents in SentNet's sample. Scores from documents that are found to be highly similar are averaged and could be displayed in the user interface as a way of validating SentNet's model results.
 - b. **Perceptual Hashing for Images:** Any images that are extracted from the document in step three will be compared to images from our existing corpus using a perceptual hash. Using this method, SentNet will be able to estimate if similar charts, graphs, or maps found in other documents have been highly rated by reviewers.

Solution Requirements

Table 1, shown below, details how SentNet's major technical components will be utilized to identify logical, textual, and semantic features that correspond to all RSEATS criteria thereby meeting all Solution Requirements.

Requirement	SentNet Graph Model	Document Similarity	Perceptual Hash
1. Properly describes quality and credibility of underlying sources, data, and methodologies	X	X	
2. Properly expresses and explains uncertainties associated with major analytic judgments	X	X	
3. Properly distinguishes between underlying intelligence information and analysts' assumptions and judgments.	X	X	
4. Incorporates analysis of alternatives.	X	X	
5. Demonstrates relevance to customers and addresses implications and opportunities.	X	X	
6. Uses clear and logical argumentation.	X	X	

7. Explains change to or consistency of analytic judgments.	X	X	
8. Makes sound judgments and assessments.	X	X	
9. Incorporates effective visual information where appropriate.			X

Table 1. SentNet Solution Requirements

It is important to note that SentNet is capable of generating scores for all of these features assuming that it is given a training set of documents in which each analytic product has been scored for each of these features. Without this type of data set to learn from SentNet will not possess a way to properly determine what “clear and logical argumentation,” or any other standardized requirement, looks like or how it should be graded.

Implementation

Overview

As previously mentioned, rather than construct a pseudo-code implementation of SentNet, we opted to pursue a fully-functioning prototype capable of analyzing sets of scored documents, generating a document scorecard and subsequently predicting scores for new documents in a manner similar to RSEATS.

The following sections will walk-through the Python implementation of SentNet’s various components, methods and analytics as well and reference specific code implementations including links to relevant files and/or links in the project directory (where applicable).

For ease of reference, Figure 3 lists the full project directory for SentNet. When individual files and/or code scripts are referenced in the below sections, this diagram may serve as a useful reference.

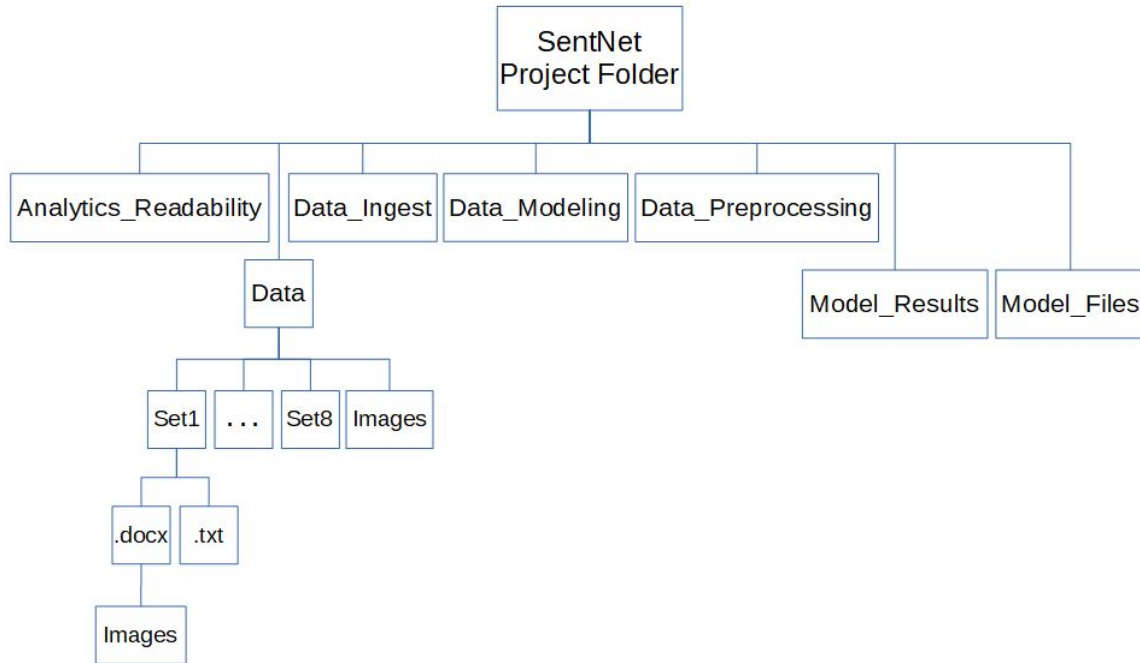


Figure 3. SentNet Project Directory

Development Software

SentNet was created entirely in Python3.6 utilizing open-source libraries/packages. While a detailed description of our methodology and its implementation is listed below, all code files are also heavily documented and contain links to outside references. These references were included where it was determined that a more significant level of detail about NLP-related topics or the libraries themselves would be beneficial to a future ODNI development team.

A full list of the Python packages necessary to run SentNet are listed in Appendix A along with links to package documentation. Recommend installation instructions for setting up SentNet can be found in Appendix B.

Data

In reality, ODNI analysts/developers will need to provide previously scored analytic products to SentNet before RSEATS specific models can be developed and tested. . For the purposes of developing and testing SentNet, we made the decision to seek out a fully realized data set that could serve as a substitute for these intelligence products.

After researching publicly available repositories, we uncovered a large and diverse scored essay set provided by The Hewlett Foundation as part of their Automated Essay Scoring contest. Specifically, this essay set contains long-form constructed essays written by 7th-10th grade students and scored by humans utilizing a variety of metrics. In total, eight independent sets (each covering a different topic) contained more than 21,000 essays that were scored by a minimum of two (and sometimes three) human raters against up to 8 different metrics. A more thorough description of the data along with the methodology used to prepared the essays to serve as simulated analytic intelligence documents can be found in Appendix C.

Scorecard (Model) Development Process

As previously discussed in the Methodology section, RSEATS scorecard model creation is the first of SentNet’s three major capabilities. In practice, the process for creating new scorecard models is as follows:

1. Data Ingest
2. Readability Feature Calculation
3. Document Matching Score Generation
4. Document Similarity Score Generation
5. WordNet Generalization and Graph Extraction
 - a. Data Cleaning
 - b. Word-Level Feature Extraction
 - c. Synset-Level Feature Generation
 - d. Word Graph Feature Generation
 - e. Synset Graph Feature Generation
6. Perceptual Hashing Feature Extraction
7. Random Forest Modeling

In the proposed implementation of SentNet the Python script “SentNet_Training_Master.py,” located in the Data_Modeling folder, controls this development process. A detailed discussion of each step in the process is contained in the sections listed below.

Data Ingest

In theory, SentNet would have the capability to ingest a wide variety of document file types. However, given the data we had available, the ingest processes of SentNet have only been developed and tested against Microsoft Word (.docx) and Microsoft Excel (.csv) file types. For the purposes of this documentation, we will restrict our discussion to the .docx file type. Future work on SentNet could easily expand this capability, given the vast amount of document-related libraries in the Python ecosystem.

In the current prototype of SentNet, the data ingest process for the creation of a new scorecard (model) begins in the “Data_Ingest” section of the “SentNet_Training_Master.py” Python script located in the *Data_Modeling* folder. The script takes as input the desired dataset specified by the user/analyst (in the current prototype, the dataset is ‘hard-coded’ and must be manually changed inside the script) and calls the function “Ingest_Training_Data” from the “SentNet_Data_Prep_Functions.py” Python script located in the *Data_Ingest* folder. This function relies on eight other sub-functions in the same script to extract text and images from every .docx file contained within a specific folder.

Note: All eight of these functions contain detailed comments that explicitly walk through the process of data ingest. This section serves as a high-level summary of the this process. For a more thorough description of the actual analytic implementation, please refer to the aforementioned “SentNet_Data_Prep_Functions.py” file.

To generate image features (discussed in more detail in the Perceptual Hashing Feature Extraction section), the process makes use of another script: “Goldberg_Perceptual_Hashing.py” located in the *Data_Preprocessing* folder. This script and its associated functionality, in combination with all the aforementioned functions, results in a dataframe that contains the title of every document, the full text of every document, and the hashes of every image within each document.

In order to assess the quality of SentNet’s predictive model during the development process, the newly created dataframe is partitioned into training and test sets. Approximately 80% of the data is utilized for training with the remaining 20% utilized for testing. This takes place in the “Train and Test Split” section which is positioned directly after the “Data Ingest” section of the “SentNet_Training_Master.py” Python script.

Readability Feature Calculation

SentNet utilizes two open source Python packages, Textatistic and TextStat, to generate “readability” metrics for every document, as previously discussed.

The functions responsible for these calculations are located in the “readability_score_generation_packages.py” Python script located in the *Analytics_Readability* folder. The first function, “textatistic_scores,” utilizes the Textatistic Python package to create a dictionary that contains the aforementioned metrics for a specific document. The function “textstat_scores” performs the same task but utilizes the “Textstat” Python packages instead. A third function, “textacy_scores,” was constructed for the “Textacy” Python package but difficulties were encountered with installing dependencies across a variety of development platforms and its inclusion in the modeling process was ultimately scrapped.

These functions are called by the function “Readability_Features” in the “SenNet_Data_Feature Extraction.py” Python script. “Readability_Features” is located in the “Document Features & Readability Tests” section of the Python file. The features derived from this process are stored in the *readability_features* set and serve as the first predictive input for SentNet’s Random Forest classification model.

Document Matching

As outlined in the proposed solution section of this proposal, the structure of text-based documents (or naturally sequenced objects) makes it a prime candidate for graph-level comparison. Documents with graphs that share many of the same sequences will have a high-score and can then be classified as potential “matches.” In the case of SentNet, this is incredibly useful as it will allow the program to validate estimated RSEATS scores for a given report if a similar/nearly identical report already exists in our sample.

The implementation for the document matching process and all associated functions are located in the “SentNet_Document_Matching.py” Python script located in the *Data_Modeling* folder. Five functions reside in the script: “read_corpus,” “Doc2Vec_Training_Model,” “Doc2Vec_Estimates_Training,” “Document_Matching_Training,” and “Document_Matching_Testing.”

Note: All five of these functions contain detailed comments that explicitly walk through the process of document matching. The sections written below present a high-level summary of the this process. For a more thorough description of the actual analytic implementation, please refer to the aforementioned “SentNet_Document_Matching.py” file.

The matching process begins when the function “Document_Matching_Training” is called in the “SenNet_Training_Master.py” Python script (located in the “Feature Extraction” section). This function combines all the required document matching functions to produce target estimates of matches for every document in a provided training set. It accomplishes this through the following steps/functions:

1. Develops a training corpus for the Doc2Vec model using the “read_corpus” function
2. Trains a Doc2Vec model using the “Doc2Vec_Training_Model” function
3. Produces and returns target estimates for every document in the training set using the “Doc2Vec_Estimates_Training” function
 - a. This function produces an estimated score for every document by attempting to identify the closest related/matching document to a specified target document (Note: SentNet will attempt to find the “second” most similar/closest-matching document; selecting the first would result in a document matching itself).

These functions make heavy use of the gensim Python library, a library used for topic modelling and similarity retrieval and adopted by companies around the world from Cisco and Capital One to Amazon and NIH. The general concept behind Doc2Vec modeling is to create a numeric representation of a document (i.e. a vector). As previously mentioned, these vectors can then be compared to produce similarity scores which enables the matching process.

The scores derived from this process are stored in the *matching_docs* dataframe and serve as the second predictive input for SentNet’s Random Forest classification model.

Document Similarity

In comparison to the document matching functionality within SentNet, this set of functions attempts to estimate a score for a given document based on generalized features about that document.

This differs from document matching in that this estimate uses inputs from the entire corpus of training documents as opposed to simply finding the most “similar” document in the corpus (and assuming that “similar” provides us the correct score).

The implementation for the document matching process and all associated functions are located in the “SentNet_Document_Similarity.py” Python script located in the *Data_Modeling* folder. Five functions reside in the script: “read_corpus_sim,” “Doc2Vec_Training_Model_Sim,” “Doc2Vec_Sim_Estimates,” “Document_Similarity_Training,” and “Document_Similarity_Testing.”

Note: All five of these functions contain detailed comments that explicitly walk through the process of document matching. The sections written below present a high-level summary of the this process. For a more thorough description of the actual analytic implementation, please refer to the aforementioned “SentNet_Document_Matching.py.”

The similarity scoring process begins when the function “Document_Similarity_Training” is called in the “SenNet_Training_Master.py” Python script (located in the “Feature Extraction” section). This function combines all the required document similarity functions to produce target estimates for every document in a provided training set. It accomplishes this through the following steps/functions:

1. Develops a training corpus for the Doc2Vec model using the “read_corpus_sim” function
2. Trains a Doc2Vec model using the “Doc2Vec_Training_Model_Sim” function
3. Produces and returns target estimates for every document in the training set using the “Doc2Vec_Estimates_Training” function
 - a. These estimates are derived for every document by attempting to identify which group/cluster of documents it is most similar to (based on the score of the documents within the group/cluster).

The scores derived from this process are stored in the *similar_docs* dataframe and serve as the third predictive input for SentNet’s Random Forest classification model.

WordNet Generalization and Graph Extraction

SentNet relies heavily on the concepts of word-level features, synsets (sets of cognitive synonyms), and textual graph features that are all described in detail below. The processes/functions which derive all of these features from ingested documents are located in the “SentNet_Data_Feature_Extraction.py” Python script in the *Data_Preprocessing* folder. These functions are all utilized by the “SentNet_Training_Master.py” Python script (located in the *Data_Modeling* folder) during the “Feature Extraction” section of the modelling process. Using the functions contained within the “SentNet_Data_Feature_Extraction.py” file, a corpus of documents can be transformed into the following feature sets:

- Word Matrix: Word counts for the most common/important words within each document
- Word Edge Matrix: Counts of the most common/important co-occurring terms within each document
- Word Centrality Matrix: The centrality/importance of key terms within each document (based on a graphical translation of that document)
- Word Clusters: The common topic/argument clusters found across all essays and their concentration within each document.
- Synset Matrix: The prevalence of the most common/important synsets (a WordNet lexicographic representation of words) contained within each document
- Synset Edge Matrix: Counts of the most common/important co-occurring synsets within each document

- Synset Centrality Matrix: The centrality/importance of key synsets within each document (based on a graphical translation of that document)
- Synset Clusters: The common topic/argument clusters found across all essays and their concentration within each document.

To derive these features from every document, SentNet proceeds as follows:

1. Data Cleaning

Every document is cleaned/transformed so that it can be properly ingested by follow-on feature generation functions. Data cleaning involves three functions, located in the “Data Cleaning” section of “SentNet_Data_Feature_Extraction.py” that are executed in combination with one another.

The functions “clean_sentence,” “clean_words,” and “remove_punc” sequentially iterate through every sentence in a document, removing punctuation and stop words. Punctuation removed from the document are defined at the top of the script and are listed below in Figure 4, for reference:

! () - [] { } ; : ' " \ , < > / ? @ # \$ % ^ & * _ ~

Figure 4. Punctuation Removed from Documents

Stop words, or commonly used words such as “the”, “a”, “an”, or “this” are words that do not meaningfully contribute to the text and can be removed from text-based analysis. SentNet sources its stop-word dictionary from the Natural Language Toolkit (NLTK) Python package which contains stopwords in 16 different languages.

2. Word-Level Feature Extraction

After having cleaned/transformed the contents of each document, the next step is to identify and extract all word/token based features.

The function, “Word_Features,” located in the “Word Feature Extraction” section of “SentNet_Data_Feature_Extraction.py” is responsible for this extraction process. After calling the aforementioned cleaning functions, “Word_Features” then extracts the unique words from all documents. It then creates a feature vector for each term that holds the number of times the term was observed in each document. This is done utilizing the Sklearn Matrix builder from the sklearn Python package (i.e. scikit-learn machine learning library).

SentNet then selects the subset of terms that occur more than a user defined threshold (currently $0.01 \times$ the length of the training set). This is done to exclude terms that are sufficiently rare enough to not offer any predictive value. In this case, a limit parameter of 0.01 means that a words must appear in at least 1% of all documents to be included in the term matrix. This also has the added effect of screening for misspellings or other possible grammar related abnormalities.

Terms that meet the defined threshold remain in the count matrix and are then added to the *word_features* dataset. This set represents the fourth predictive input into SentNet's classification model.

3. Word Graph Feature Generation

With word-level features now identified, SentNet can now begin the process of creating a word network for every document. This work occurs in the "Word Graph Feature Generation" section of "SentNet_Data_Feature_Extraction.py" and contains five functions: "pd_edge_extractor," "tuple_edge_extractor," "edge_translation," "Word_Edge_Features," and "Word_Centrality_Features."

Note: All five of these functions contain detailed comments that explicitly walk through the process of generating word graph features. The sections written below present a high-level summary of the word network creation process. For a more thorough description of the actual analytic implementation, please refer to the aforementioned "Word Graph Feature Generation" section of "SentNet_Data_Feature_Extraction.py."

All terms that occur within the same sentence are considered to be "connected" and are represented as an edge/connection within the given document's network. Translating a document into a network based form allows us to extract two feature sets: Edge counts and Term Betweenness Centrality.

- A. Edge Counts:** First we calculate the occurrence of every pair of co-occurring words, within the same sentence, across all documents. Next we exclude those co-occurring words that are rare and not common enough to be a reliable feature to model on (this limiting parameter can be tuned by the user). Finally, we look across all document to calculate the presence/absence of selected co-occurring terms. The entirety of this edge count process is completed by the master function "Word_Edge_Features" utilizing two helper functions: "pd_edge_extractor" and "edge_translation." The final representation of these counts are stored in the *word_edges_features* dataset.
- B. Term Betweenness Centrality:** For every document we will be able to extract the "centrality" or importance of every term within that document. This is done by first identifying the shortest path between all terms in the network. Next we calculate the proportion of shortest paths that traverse each term (node). This proportion (bounded between 0 and 1) is referred to as the "betweenness centrality" of that synset. This calculation is performed for every word in every document to ascertain that term's importance within that document. The entirety of the term betweenness centrality process is completed by the master function "Word_Centrality_Features" utilizing a single helper function: "topule_edge_extractor." The final representation of the betweenness centrality for all selected terms within all documents is stored in the *word_centrality_features* dataset.

The final step of the Word Graph feature generation process involves the creation of topic clusters. This work occurs in the “Word Based Clustering” section of “Term_Clustering.py” located inside the *Data_Modeling* folder and contains three functions: “Clustering_Features,” “cluster_concentration,” and “Cluster_Concentrations.” To execute this process, the “SentNet_Training_Master.py” script calls the function “Clustering_Features” and uses the returned information to then call the function “Cluster_Concentrations.”

Note: All three of these functions contain detailed comments that explicitly walk through the process of generating clustering features. The text written below presents a high-level summary of the cluster creation process. For a more thorough description of the actual analytic implementation, please refer to the aforementioned “Word Based Clustering” section of “Term_Clustering.py.”

Utilizing the existing edge table generated by the processes detailed above, SentNet can identify topic clusters via the process described below.

- A. Document Clusters:** After translating all documents into their network representation, we aggregate all networks from all documents into a single network that represents the entire corpus. From this network, we run the Louvain clustering algorithm to identify “clusters” of words within all documents. The clusters that emerge represent common topics/arguments found across all essays. After identifying these clusters we calculate the concentration/presence of each cluster within each document. The final representation of these cluster features is stored in the *word_cluster_features* dataset.

These three sets of features (*word_edge_features*, *word_centrality_features*, and *word_cluster_features*) serve as the fifth, sixth, and seventh predictive inputs into SentNet’s classification model.

4. Synset-Level Feature Generation

Having identified and extracted all word-based features within each document, SentNet then moves on to extracting all synset based features. This process is contained in the “Synset Feature Generation” section of “SentNet_Data_Feature_Extraction.py” and contains four functions that work in combination to translate terms into their synset equivalent and then to ultimately extract the synset counts for each document.

The process begins with the function “Synset_Features” which takes as input the entire collection of cleaned documents utilized for extracting word features.

“Synset_Features” iterates over every document and, for every sentence, calls the function “synset_translated_string.” This function processes each sentence and returns a synset translated sentence in which all terms have been replaced with all the synset codes along their hypernym path.

To accomplish this process, SentNet makes use of a system called WordNet. WordNet is a lexical database which attempts to model the relationships between words including

word/topic hierarchies. Hypernyms of a term in WordNet represent a generalization of the word itself. In other words, hypernyms are high level representations of a term or object. For example, the hypernym path for the word “official” can be seen below in Figure 5.

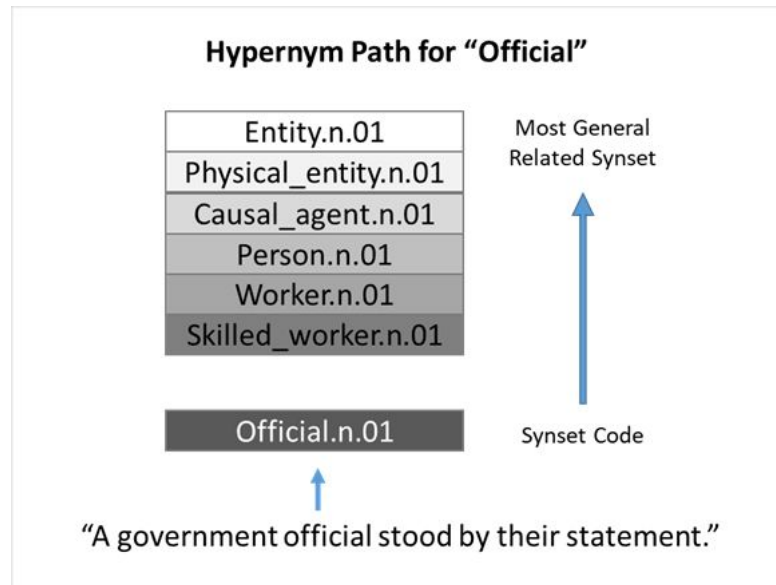


Figure 5. Hypernym Path for the Word “Official”

To construct the hypernym path, “synset_translated_string” utilizes the “WordNet_Translator” function which takes as input a single word and returns a list of synset codes that are contained within the provided term’s hypernym path. More specifically, this function extracts the “hypernym path” for the provided term and returns a list of synset codes that are contained within the hypernym’s path (removing any period delineators for ease of analysis via the function “return_hyponym_codes.” For reference, the previous example of the hypernym_path for official would be returned from the function “WordNet_Translator” as shown below in Figure 6.

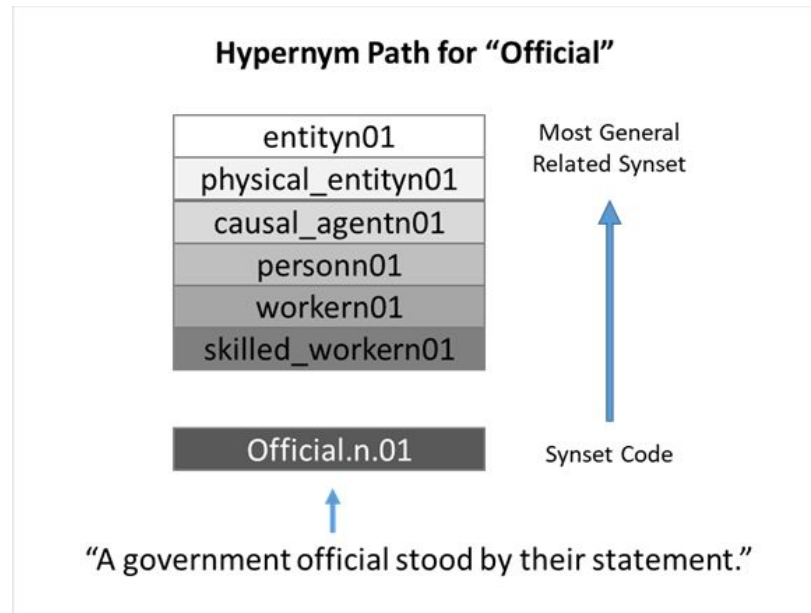


Figure 6. Hypernym Path for the Word “Official” after “WordNet_Translator” Processing

With that process in place, the function “Synset_Features” operates in a manner similar to that detailed above for Word Feature extraction. “Synset_Features” builds a synset translation for each document and then creates a matrix which contains the counts of all synsets found in every document. It then employs a minimum threshold to filter out only those synsets that are common in nature (i.e. are found in at least 1% of all documents). As previously stated, this process helps to remove extremely rare features that will provide little value during the modeling process.

These “common synsets” are then added to the *synset_features* dataset; the eighth input into SentNet’s classification model.

5. Synset Graph Feature Generation

The final step of the WordNet Generalization and Graph Extraction process involves the creation of a synset network for every document. This work occurs in the “Synset Graph Feature Generation” section of “SentNet_Data_Feature_Extraction.py” and contains six functions:

- synset_translated_sentence
- pd_edge_extractor_synset
- tuple_edge_extractor_synset
- edge_translation_synset
- Synset_Edge_Features
- Synset_Centrality_Features

Note: All six of these functions contain detailed comments that explicitly walk through the process of generating synset graph features. The sections written

below present a high-level summary of this network creation process. For a more thorough description of the actual analytic implementation, please refer to the aforementioned “Synset Graph Feature Generation” section of “SentNet_Data_Feature_Extraction.py.”

Synsets that occur within the same sentence share an edge within a network. Translating documents into a network based form allows us to extract two additional feature sets:

- A. Synset Edge Counts:** First we calculate the occurrence of every pair of co-occurring synsets (within the same sentence) across all documents. Next we exclude those co-occurring synsets that are rare and not common enough to be a reliable feature to model on (this limiting parameter can be tuned by the user). Finally, we look across all document to calculate the presence/absence of selected co-occurring synsets. The entirety of the synset edge counts process is completed by the master function “Synset_Edge_Features” utilizing three helper functions: “pd_edge_extractor_synset,” “edge_translation_synset,” and “synset_translated_sentence.” The final synset edge counts are stored in the *synset_edge_features* dataset.
- B. Synset Betweenness Centrality:** The next step is to extract the “centrality” or importance of every synset within a document, for every document in the sample. This is done by first identifying the shortest path between all synsets in the network. Next we calculate the proportion of shortest paths that traverse each synset. This proportion (bounded between 0 and 1) is referred to as the “betweenness centrality” of that synset. This calculation is performed for every synset in every document to ascertain that synset's importance within that document. The entirety of the synset betweenness centrality process is completed by the master function “Synset_Centrality_Features” utilizing a single helper function: “topule_edge_extractor.” The final representation of these centrality/importance values are stored in the *synset_centrality_features* dataset.

Having finalized the creation of a synset network, SentNet follows the same steps as outlined in the creation of Word Cluster Features in order to create topic clusters using synsets. This work occurs in the “Synset Based Clustering” section of “Term_Clustering.py” located inside the *Data_Modeling* folder and contains three functions: “Synset_Clustering_Features,” “synset_cluster_concentration,” and “Synset_Concentrations.” To execute this process, the “SentNet_Training_Master.py” script calls the function “Synset_Clustering_Features” and uses the returned information to then call the function “Synset_Concentrations.”

Note: All three of these functions contain detailed comments that explicitly walk through the process of generating clustering features. The text written below presents a high-level summary of the cluster creation process. For a more thorough description of the actual analytic implementation, please refer to the aforementioned “Synset Based Clustering” section of “Term_Clustering.py.”

Utilizing the existing edge table generated by the synset graph creation process detailed above, SentNet can identify topic clusters via the following process:

- A. **Synset Document Clusters:** After translating all documents into their synset network representation, we aggregate all networks from all documents into a single network that represents the entire corpus. From this network, we run the Louvain clustering algorithm to identify "clusters" of synsets within all documents. The clusters that emerge represent common topics/arguments found across all essays. After identifying these clusters, we calculate the concentration/presence of each cluster within each document. The final representation of these cluster features is stored in the *synset_cluster_features* dataset.

These three sets of features (*synset_edge_features*, *synset_centrality_features* and *synset_cluster_features*) serve as ninth, tenth, and eleventh predictive inputs into SentNet's classification model.

Perceptual Hashing Feature Extraction

Perceptual Hashing functionality enables SentNet to evaluate visual information (e.g., charts, graphs, maps, etc.) embedded within ingested documents for the ultimate purposes of image comparison/matching. The generation of features based on image hashes occurs in the "Image Hashing" section of "SentNet_Data_Feature_Extraction.py" and contains two functions: "Unpack_Image_Hashes" and "Return_Image_Score."

The first function, "Unpack_Image_Hashes," unpacks the Goldberg Perceptual Hashing Array's that were originally extracted from document images during the ingest process. Given that more than one image can be present in a document, this function flattens this list of all hash arrays from all documents into a single pandas DataFrame and maps that hash to its source document's target value. This new pandas DataFrame is used as an input to the "Return_Image_Score" function.

The "Return_Image_Score" function attempts to calculate a target score for every document in the aforementioned dataframe, identifying documents that have highly similar/matching images. As previously discussed in the methodology section, this similarity value will be the most accurate when the target feature being modeled relates to the images contained within a document. This function calculates this score using the following methodology:

1. For every image hash extracted from a specific document we calculate the similarity between that image hash and all other image hashes contained within the flattened image hash table and append the "match probability" to a temporary dataframe.
2. We then sort that dataframe by the match probability and remove any matches that are less than .75 (not likely to be a matching/similar image).
3. Finally, we take the mean of the target scores for all of the remaining "high likelihood" matches/similarities to generate the final value.

These average target scores for every document with highly similar/matching images are then returned as a new column ('Image_Avg_Score') in the features dataframe that will serve as the final input to SentNet's classification model.

Random Forest Modeling

1. Modeling Process

Having ingested the data, split it between training and test sets, and finalized the extraction of all relevant features, SentNet is now ready to begin modeling the data.

SentNet utilizes a Random Forest Classification Model to consider the importance of all twelve predictive features outlined above and to find interactions between these features while remaining highly generalizable (i.e. it reduces the likelihood of overfitting its training data). The modeling process is contained within the "modeling" section of the "SentNet_Training_Master.py" Python script located in the *Data_Modeling* folder.

To begin, all twelve feature sets are collated into one large Pandas DataFrame (called `train`). Any missing values or values marked as 'N/A' are replaced with 0's to facilitate the modeling process. An initial random forest classifier is run to determine the importance of every possible variable/feature. This process is run approximately 10,000 times and features that exceed a predetermined level of importance are extracted from the list of all available variables.

These variables serve as the predictors for the final scoring model which is used to predict the relevant scores for each document (in the case of ODNI analytic products, this would be the RSEATS scores).

2. Modeling Results: Overview

After having training a Random Forest model using the provided training data, new documents can be scored by specifying a testing partition within SentNet_Training_Master.py or by feeding documents through the provided SentNet_Scoring_Only.py script. In the section below, we discuss the initial performance of SentNet's models using the features described previously. To establish a performance baseline for SentNet we developed and evaluated 43 different models within SentNet, one for each human generated score for each essay contained within our example data. For each test we randomly partitioned our data with 80% of observations going into our training set while the remaining 20% went into our testing/evaluation set. After developing each model we evaluated its performance using the following metrics:

- A. **Square Root Mean Squared Error (RMSE):** For this measure, we look at the absolute "distance" between the estimated target score and the true target score for every observation within our test set (sometime referred to as the error or residual). We then calculate this distance across our entire test set, and take the average, to judge our model's performance. Given this, lower RMSE values are more desirable.

- B. **Accuracy Rate:** Next we looked our model's accuracy rate. In short, this is the number of times our model correctly predicted a document's human generated score, divided by the total number of observations within our test set.
 - C. **Generalized Accuracy Rate:** Given that our documents were scored by humans along an ordinal scale we expect there to be some natural variation in scores, particularly for documents that are judged to be at a breakpoint between two scores (such as a document that could score either as a 4 or 5). To allow for this variation without overly penalizing our models we created a more generalized accuracy rate that assumes a prediction is correct if it falls within 1 point/level of the true score. We feel this could also be helpful framing for end users so they could see the range of scores a document would likely receive.
- 3. Modeling Results: Performance**
- Across all 43 iterations of our models, SentNet had an average accuracy rate of 61%, with a generalized accuracy rate of 92% and on overall RMSE of only 0.595. However, model performance varied significantly depending on the complexity of its target. As seen in Table 2, shown below, 35 of our models that focused on human generated ratings with 6 or fewer levels/scores had an accuracy rate of 64%, a generalized accuracy rate of 98.7% and a RMSE of only .353. In contrast, models with targets that had 10 or more levels/scores had an accuracy rate of 43.3%, a generalized accuracy rate of 63.8% and a RMSE of 1.65.

Performance Measure	Overall	<=6 Levels	>= 10 Levels
<i>Number of Models</i>	43	35	8
<i>RMSE</i>	0.595	0.353	1.65
<i>Accuracy Rate</i>	60.60%	64.50%	43.30%
<i>Accuracy Rate +-1</i>	92.20%	98.70%	63.80%

Table 2. Prototype Model Results

This variance in performance in comparison to “target complexity” (target variables with many potential values) is even more clearly seen in Figure 7, shown below. In this graph we have plotted the Accuracy and Generalized Accuracy for all 43 models we built using SentNet. From this plot it is easy to see that SentNet does significantly better with less complex target features.

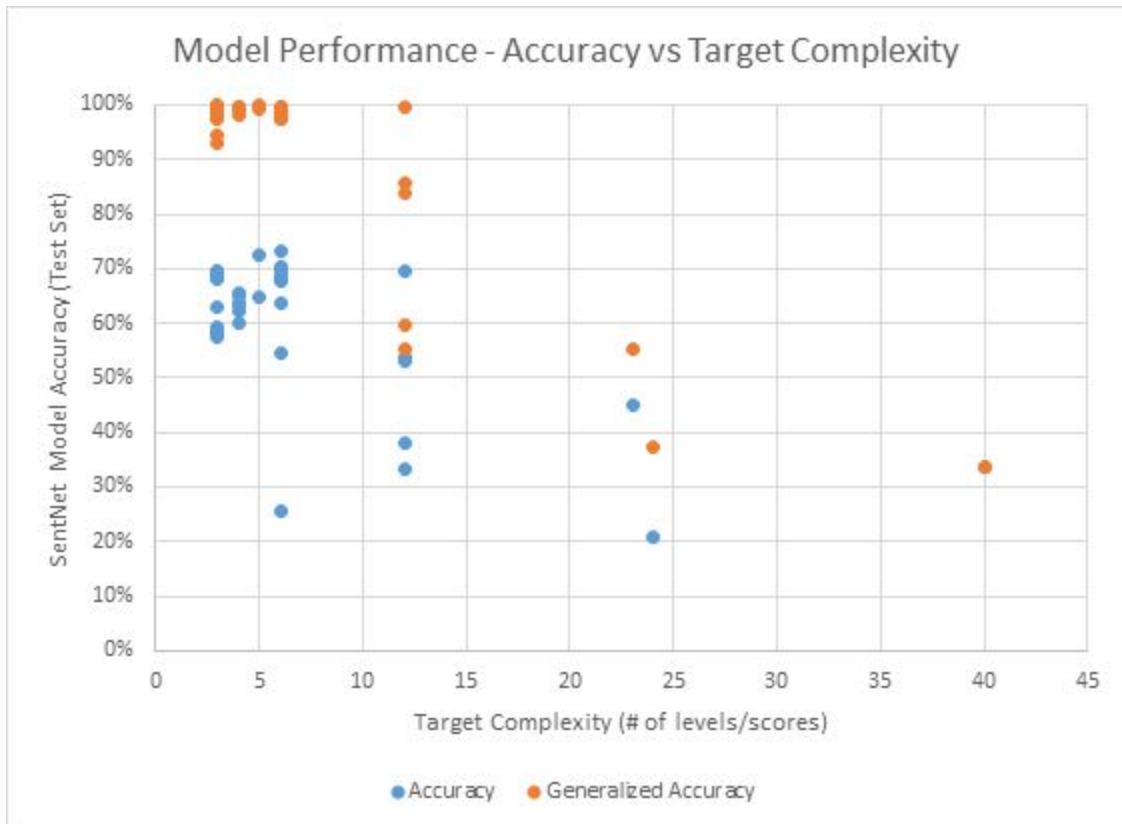


Figure 7. Prototype Model Accuracy as a Function of Data Complexity

This same trend is also seen in this next plot, Figure 8, where we compare the square root mean square error (RMSE) of our models in contrast to the complexity of our target variable. From this we can see that as the number of potential target values increases the RMSE of our models increases as well.

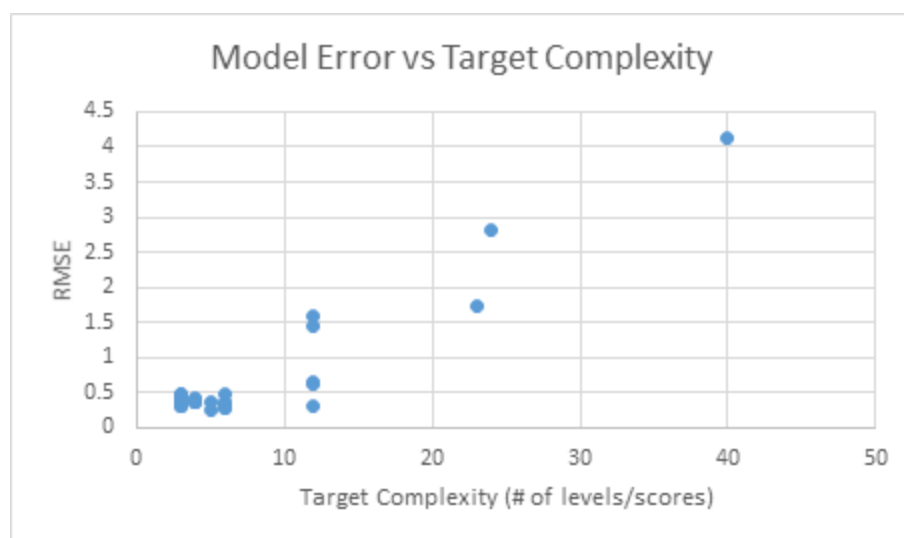


Figure 8. Prototype Model Error as a Function of Data Complexity

However, it is important to note that this increase in the RMSE of our models appears to be (nearly) linear when compared to the complexity of our target variable. In other words, more complex targets do result in an exponential decrease in performance. We believe that this variance in the performance of SentNet's models stems from two factors:

- A. Source Variability:** It is important to recognize that we are attempting to model scores that were originally graded by human reviewers. While these reviewers likely had a rubric or scorecard to help with the assignment of scores, there is still some natural variation we expect the reviewers would introduce. While this variation is likely minimal when reviewers have less than 6 categories it could be much higher when they have 20 or more categories/scores they need to choose from. This natural variability is difficult, if not impossible to accurately model, contributing to a decrease in our model's performance as the number of target factors increases.
- B. Classification vs Regression Models:** Currently, all of SentNet's models are built using Random Forests which generally perform better at classification tasks with fewer potential categories to choose from. For targets that are more complex (those with 10+ levels/scores) we believe a regression based model would likely be a good alternative to a random forest model (discussed more in the future improvements section).

4. Modeling Results - Feature Importance

Finally we wanted to determine the importance (or predictive power) of the features that went into our models. This is done for two reasons:

- A. Run time:** If it seems that a specific feature set is not a valuable input to our models, we can remove it from the feature generation process to decrease SentNet's run time and model complexity.
- B. Model Accuracy:** Finding a subset of features that are more predictive of our target can help to improve the accuracy of our models. The number of possible features will vary greatly depending on the size and length of the training corpus given to SentNet. On average, for of our training sets (~1,700 standardized test student essays) SentNet extracted around 80,000 features. This is simply too many features to feed into a Random Forest directly because it could result in a phenomenon called model "overfitting" which decreases the predictive power of our models.

To address this risk, we first feed all the features generated during our feature development process into a Feature Selection Random Forest. This random forest contains 10,000 "trees" which all vote on the importance of our features. Any feature that is chosen by a sufficient number of trees is put into our "selected feature set" which typically contains around 2,000 features. Using only this "selected feature set" we develop a second model that predicts the target value we are interested in.

While there is not enough space to explore all the features developed by SentNet here, we wanted to determine which features were consistently ranked in the top 15 features across all 43 of our test models. In Figure 9, shown below, we show the top 20 features from this list and the number of models that they appear in.

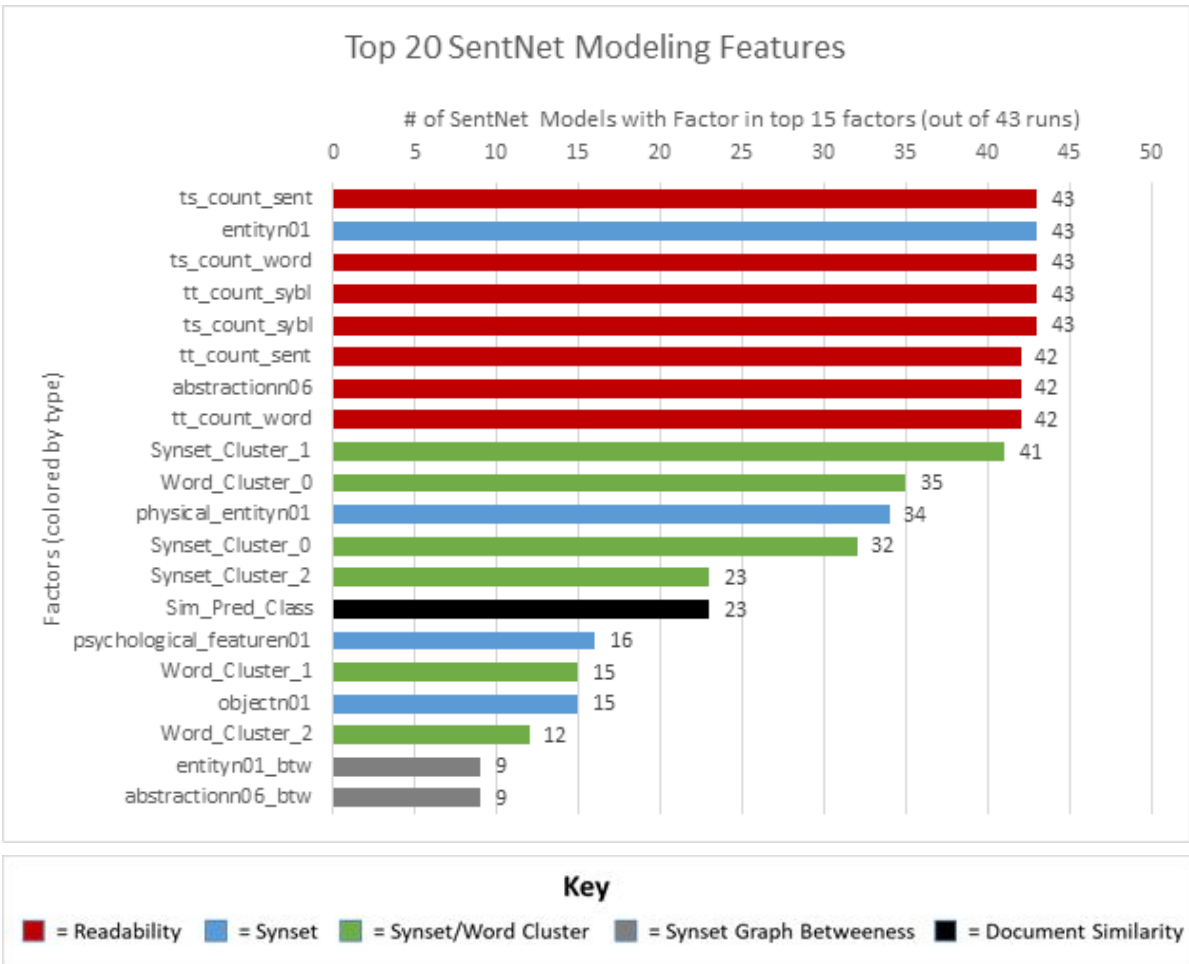


Figure 9. Prototype Model Features

Overall our “Readability” features were the top scoring, presumably because well written documents tend to score better overall. However, we can see that all of our “core” feature sets (readability, sysnets, word graph clusters, synset graph clusters, graph betweenness centralities, and document similarities) all made it into the top 20 features across our 43 test models. Therefore, from this we can see that all of SentNet’s features have some predictive value that help to increase model performance.

Within SentNet, for each model, these results are exported to a comma-separated value (CSV) file available for the analyst/developer to interact with outside of an active Python session. All results files are saved to the

Model_Results folder in the format
“SentNet_Scoring_Predictions_*SetFolderName*_YYYY_MM_DD_HH:MM:SS.csv.”

More detailed results for each of our 43 test models, and the associated feature importance metrics, are included in Appendix D.

5. Modeling Results: Future Enhancements

While we believe SentNet’s current performance demonstrates that it has the capability to effectively score IC analytic products, we recognize there is always the potential for additional improvements. Potential future enhancements to the SentNet’s models include:

- A. Additional Features:** Models are completely dependent on the features that you provide them. While we have done a satisfactory job developing generalized features within the current implementation of SentNet, more robust and IC specific features could improve SentNet’s performance even further if it were to be applied to IC related products. Possible features include developing IC related readability tests or an IC tailored implementation of WordNet to generate more accurate synsets.
- B. New Modeling Methods:** As discussed previously, we see a decrease in performance in SentNet’s current models when attempting to predict target variables with many potential levels/scores. To address this, future implementations of SentNet could include new modeling methods for these more “complex” targets. These new methods would likely be regression based, and therefore could only be applied with larger document corpuses or with a more aggressively pruned “selected feature set.”

Scoring of New Documents

Once a scorecard model has been created, new reports can automatically be scored by SentNet. The process outlined in the proposed methodology section described the following scoring process:

1. Document(s) Import
2. RSEATS Scorecard (Model) Selection
3. Readability Feature Creation
4. Document Cleaning
5. WordNet Normalization
6. Graph Construction
7. Synset Graph Feature Extraction
8. Document Similarity Scoring

With the exception of RSEATS Scorecard (Model) Selection process, which is not implemented in the current prototype of SentNet as it is a UI-based feature, all of these steps have been previously discussed in detail including references to functional code.

While models cannot be selected as part of an existing user interface in this prototype, a fully-functional implementation of this code does exist in the Python script “Document_Scoring_Only.py” located in the *Data_Modeling* folder. It follows a very similar process to the “SentNet_Training_Master.py” script that was detailed above in that it ingests a directory of documents, extracts features, and runs those features through a model. The only difference is that the model has already been generated and doesn’t need to be created. In the current implementation, the model is loaded via a Python pickle file (discussed below) and the new documents are scored utilizing the specified parameters. This level of functionality proves that SentNet has the capability to score any new document utilizing an existing scorecard.

Scorecard (Model) Export

In the current prototype of SentNet, we have implemented a model export function in the form of a Python Pickle. A pickle is a way to “serialize” a Python object such that it can be written to a file and then imported/reconstructed at a later time. This implementation can be found in the “Saving Models & Modeling Features” portion of the Python script “SentNet_Training_Master.py” in the *Data_Modeling* folder. Pickled models are stored in the *Model_Files* sub-directory of the main project folder using the format: “SentNet_Model_SetFolderName_YYYY_MM_DD_HH:MM:SS.pkl.” Future developers of SentNet could easily adapt the functionality of SentNet to simply score new models based on the model of a pickled file rather than having to create a model from scratch every time the program is run as the prototype currently stands.

In addition, the features of each model are also saved to the *Model_Files* sub-directory using the format: “SentNet_Modeling_Feature_Set_SetFolderName_YYYY_MM_DD_HH:MM:SS.csv.” These files provide analysts/developers with a quick and easy reference to the variables that serve as the predictors for each exported model.

New Scorecard Creation

Should the need for new scorecards (i.e. models) be necessary to accommodate different types of analytic products and/or agencies who utilize different scoring standards, future analysts/developers would simply need to repeat the training process detailed above. SentNet would only require a new set of training data that contains data evaluated by the new scoring standard and the resulting scores of those documents. This data could be fed into SentNet following the processes outlined above which would result in the generation of a new scorecard model. This model, along with other models generated by other data sets, could then be exported and stored in a directory (as detailed above) which SentNet could then access whenever documents utilizing that scoring standard are ready to be graded.

Model Update Process

The ability for SentNet to learn from each new report that is evaluated in order to improve the scoring platform requires a developer to regularly monitor the results of SentNet’s scoring process. As new reports are continuously fed into the system, and the associated results are verified by humans to be consistently accurate, it is recommended that these newly scored documents be combined with the previous documents used to create the initial model. A new, updated, model can then be created utilizing this larger sample of data resulting in richer

features for evaluating/scoring new documents. This model creation process would be identical to the one detailed in the above sections. A developer/analyst would simply need to direct SentNet to the updated data folder and the model will be created in the same exact manner. The result should be a more capable model that can be exported to the *Model_Files* sub-directory.

Risks & Mitigating Factors

Given that the proposed methods have not been tested in conjunction with one another on an IC related data set, it is important that we give credence to potential issues we foresee should these techniques be implemented with IC data. The first major problem could arise from a simple lack of available training and testing data. As previously outlined in the methodology section, SentNet will rely heavily on a collection of previously human-generated analytic product evaluations in order to establish a scoring baseline. This initial sample is vital in setting the framework for the system and must be properly curated to ensure a sufficient sample size for each possible scoring outcome across all criteria. External “proxy” data could be used to train the models if no IC related data is available, but this would likely reduce the accuracy of the models.

Additionally, small or reduced sample sizes could have the follow-on effect of creating overfit models. In other words, our methods could generate techniques which are too customized to the particulars of our sample data and fail to respond properly to new intelligence products. While the risk of this issue diminishes as the sample size increases, it is important to note given the number of RSEATS criteria that must be modeled.

Finally, as demonstrated above, SentNet is capable of accurately estimating essay scores within our example set. While we are optimistic we could achieve similar or better performance using IC generated documents, it is possible that additional model tuning and feature sets would be required to accurately model all of the RSEATS criteria.

Future Improvements

Scoring Visualization

The final major proposed feature of SentNet is the ability to display the completed results from the previously described scoring process. It is our belief that a qualified team of UI/UX developers would be able to easily create a simple, effective user interface capable of integrating with SentNet due to its functionalized nature. Listed below are our recommendations for UI features that would enable potential users to seamlessly specify documents for analysis and then visualize the associated results.

Recommended Technology Stack

Front-End

- [React](#)
 - React is a JavaScript library that facilitates the creation of interactive User Interfaces that will efficiently update components as data changes

- The use of components allows for maximum reusability of various UI elements
- A one-way data flow ensures that the only changes to the UI are a result of changes to the underlying data (e.g., ingested documents, model results, etc.)
- [D3](#)
 - D3 is a JavaScript library that generates powerful, interactive data visualizations that bring to life various elements of the web: HTML, SVG, CSS, etc.
 - D3 would be incredibly useful in developing an interactive network visualization for use in the “Document Visualizer” section detailed below
 - An example network graph produced using D3 can be found here: <https://bl.ocks.org/mbostock/4062045>

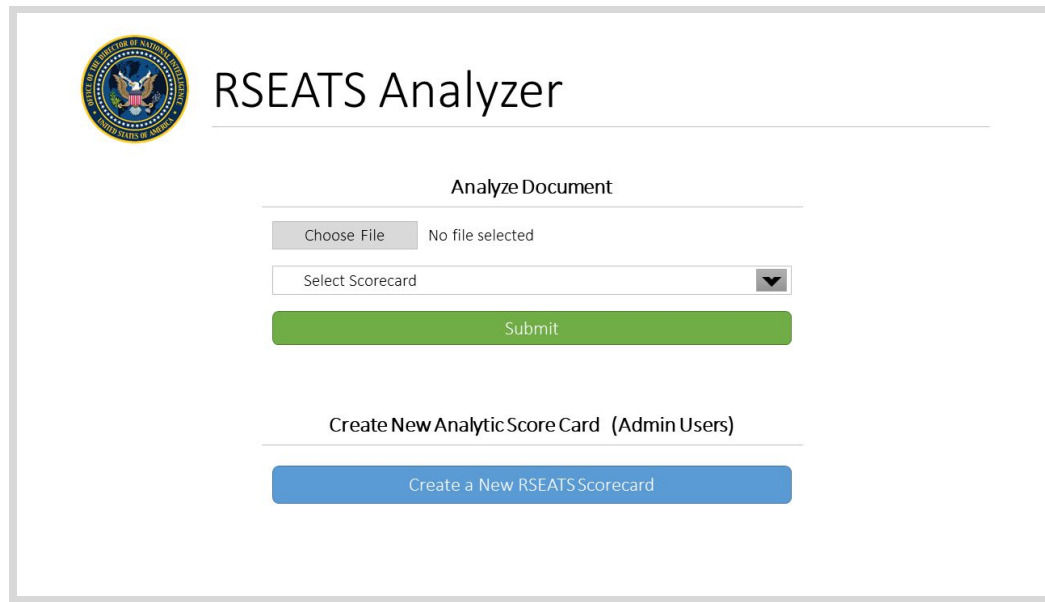
Back-End

- [MongoDB](#):
 - Models Table
 - Contains model based information such as features and parameters
 - Removes the need for the *Model_Files* folder and the “pickle” process
 - Documents Table
 - Contains original documents and their associated scores
 - Scored documents will also have relevant-extracted features and recommended scores based on the scorecards that were used to evaluate them
 - Recommend variables for each document that indicates whether a document has been scored, which model scored it, and whether or not it was used in the creation of a scorecard (and if so, what scorecard)
- [Docker](#)
 - Allows for easy deployment and package maintenance across both Windows and Linux
 - Docker containers are stand-alone, executable packages that include everything needed to run a piece of software (in this case SentNet): code, system tools, Python packages, etc.
 - this would remove the need to manage the package installations listed in Appendix A

User Interface

1. RSEATS Analysis Landing Page

Rather than requiring users to edit code to specify which datasets should be scored or which scorecard models should be utilized to grade the aforementioned documents, a basic UI homepage would facilitate both tasks via a simple system file-browser extension in combination with a drop-down box. Additional ‘admin’ functionality would enable developers or power users to create new scorecards anytime new scoring requirements are generated or new datasets with new scoring standards are received. A mock-up of this landing page can be found below in Figure 10.



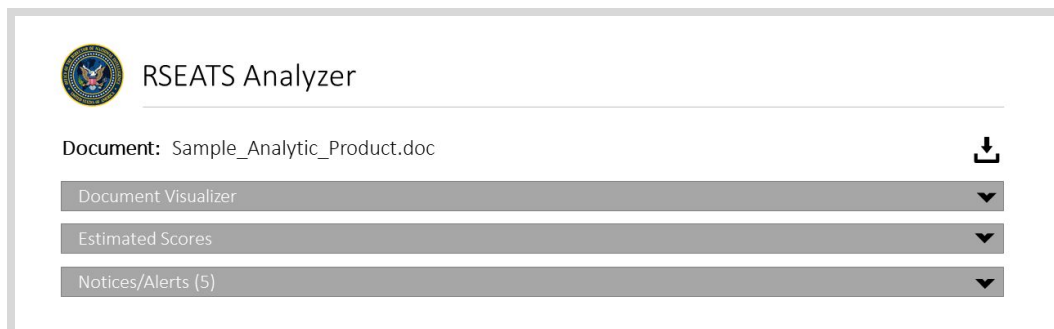
The RSEATS Analyzer homepage features the Department of Homeland Security seal in the top left corner. The main heading is "RSEATS Analyzer". Below this, there is a section titled "Analyze Document". This section contains a "Choose File" button, a status indicator "No file selected", a "Select Scorecard" dropdown menu, and a green "Submit" button. Below the "Analyze Document" section is a section titled "Create New Analytic Score Card (Admin Users)" which contains a blue button labeled "Create a New RSEATS Scorecard".

Figure 10. RSEATS Analyzer Homepage

2. Visualize SentNet Output in User Interface

Once SentNet's analysis is complete results can be pushed to a user interface such as the RSEATS analyzer depicted below. An analysts workflow through this tool would be as follows:

- A. **Load RSEATS Analyzer Page** - Once a document analysis is complete users will be directed to the RSEATS analyzer page. On this page SentNet's findings are broken into three sections: Document Visualizer, Estimated Scores, and Notices/Alerts as seen in Figure 11:



The RSEATS Analyzer Review Page displays the Department of Homeland Security seal and the title "RSEATS Analyzer". Below the title, it shows the document name "Document: Sample_Analytic_Product.doc" with a download icon to its right. There are three expandable sections, each with a downward arrow icon: "Document Visualizer", "Estimated Scores", and "Notices/Alerts (5)".

Figure 11. RSEATS Analyzer Review Page

- B. **Document Visualizer** - Using the document visualizer, shown below in Figure 12, analysts can explore an interactive network visualization of their document to learn how synset features are related, how they are used in context within their document, and how these features contribute to their overall RSEATS scores.

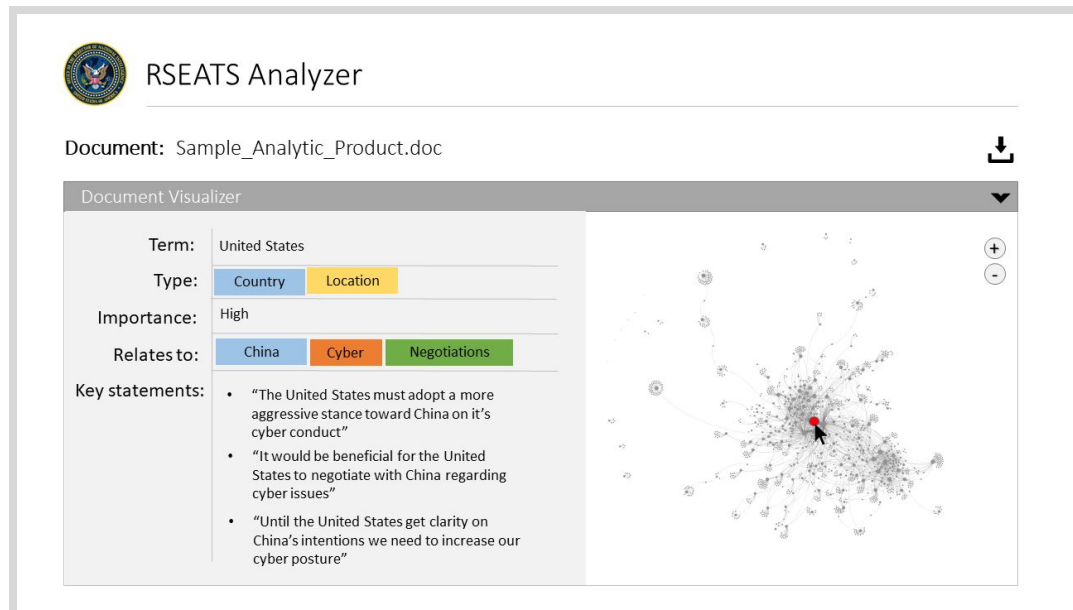


Figure 12. RSEATS Analyzer Document Viewer

- C. Estimated Scores** - Using the estimated scores tab, analysts can review the estimated scores for each criteria generated by SentNet's two scoring models (random forest and similarity matching) as seen in Figure 13. Analysts can also review other interpretability and readability metrics for a specified document.

RSEATS Analyzer

Document: Sample_Analytic_Product.doc

Document Visualizer

Estimated Scores

RSEATS Criterion	SentNet	Similarity
Literal Response Criterion 1: Properly describes quality and credibility of underlying sources, data, and methodologies.	2	1
Literal Response Criterion 2: Demonstrates customer relevance and addresses implications.	3	4
Inferential Response Criterion 1: Properly distinguishes between factual reporting and assumptions and judgments.	1	1
Inferential Response Criterion 2: Properly expresses and explains uncertainties associated with major analytic judgments.	1	2
Evaluative Response Criterion 1: Uses clear and logical argumentation.	2	2
Evaluative Response Criterion 2: Incorporates analysis of alternatives.	2	2

Readability Statistics

Flesch-Kincaid —Readability test designed to indicate how difficult a passage in English is to understand (higher better).	.97
Gunning Fog Index —Estimates the years of formal education a person needs to understand the text on the first reading.	12
Cloeman-Liau Index —Index that gauges the understandability of a text (lower better)	10

Figure 13. RSEATS Analyzer Estimated Scores

- D. Notices/Alerts** - Any notices or alerts that are generated by SentNet during its review of the document can be viewed in the final Notices/Alerts tab, as shown below in Figure 14. These alerts are based on SentNet's interpretation of a

document's synset graph and if any significant elements (or groups of elements) are missing in that graph.

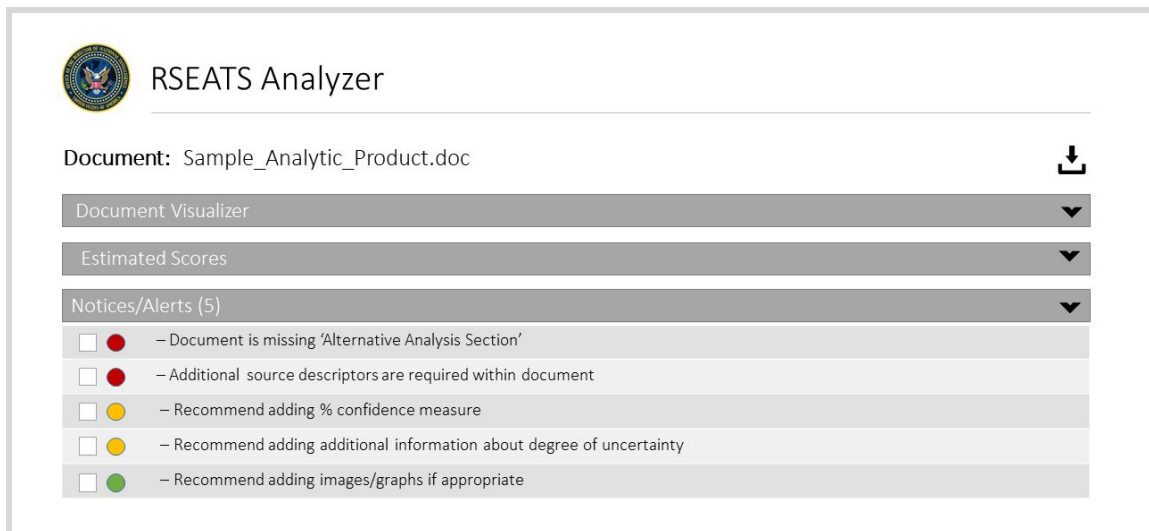


Figure 14. RSEATS Analyzer Notices & Alerts

Parallelization

The current SentNet modeling process can take upwards of five hours to process a single dataset containing roughly 1000 documents. Given this lengthy time requirement, it is susceptible to faults which can significantly lengthen the overall processing time. As such, our team recommends that future developers look into options for parallelizing the data processing pipeline. Our initial research has been limited but recommended packages within the Python ecosystem include Python's multiprocessing³ and/or threading⁴ module.

Summary

In conclusion, recent advances in Natural Language Processing have the ability to revolutionize the way IC products are currently evaluated and scored. Building off existing analysis methods, SentNet creates textual features in order to create a robust RSEATS scoring model. Utilizing a dataset that was hand-selected to mimic the characteristics of potential IC analytic products, our classification model produced an accuracy rate of 92% percent when determining what an essay's score should be within +/- 1 point of the essay's actual score.

Scorecard models, like the one we developed, can be utilized to automatically grade new reports with no human intervention. Future developments should bring together SentNet's system of scoring with a user interface ensures that enables analysts and analytic managers to quickly and easily evaluate the trustworthiness and accuracy of every grade with minimum additional training.

³ <https://docs.python.org/3/library/multiprocessing.html>

⁴ <https://docs.python.org/3/library/threading.html>

Appendix A. SentNet Technical Requirements

System Resources

SentNet was developed on a system with the following resources. When attempting to run SentNet it is recommended that you have a system with similar specifications:

- Operating System: Windows 10 or Ubuntu 16.04+
- Processor: Intel Core i7-7700 - 3.6 Ghz
- RAM: 16 GB
- Hard Drive: 10 GB of Hard Drive Space Available

Required Python Packages

The Python packages required to run SentNet are listed below. Links to associated documentation have been included for every package in the respective package name.

- [cairosvg](#)
 - `svg2png`
- [datetime](#)
- [docx](#)
 - `pip install python-docx`
- [docx2txt](#)
- [gensim](#)
- [io](#)
 - `BytesIO`
- [itertools](#)
- [networkx](#)
- [nltk](#)
 - `nltk - PunktSentenceTokenizer` (requires a separate download using the nltk downloader)
 - `nltk - stopwords` (requires a separate download using the nltk downloader)
 - `nltk - wordnet` (requires a separate download using the nltk downloader)
 - `nltk - word_tokenize` (requires a separate download using the nltk downloader)
 - these 4 corpus-addons can be downloaded via the following commands:
 - `import nltk`
 - `nltk.download('stopwords')`
 - `nltk.download('wordnet')`
 - `nltk.download('wordnet_ic')`
 - `nltk.download('punkt')`
 - [Download Instructions](#)
- [numpy](#)
- [os](#)
- [pandas](#)
- [pathlib](#)
- [pickle](#)
- [PIL \(Python Imaging Library\)](#)

- [Image](#)
- MpImagePlugin.MpImageFile
- Conflicts may exist on Windows machines when attempting to use Python 3.6+ with this library; recommend installing Pillow version 4.2.1
 - `pip install pillow==4.2.1`
- [python-louvain](#)
- [random](#)
- [re](#)
- [skimage](#)
 - [color.rgb2gray](#)
 - [io.imread](#)
 - `pip install scikit-image`
- [sklearn](#)
 - [ensemble.ExtraTreesClassifier](#)
 - [ensemble.GradientBoostingClassifier](#)
 - [ensemble.RandomForestClassifier](#)
 - [model_selection.train_test_split](#)
 - `pip install scikit-learn`
- [textstat](#)
- [textatistic](#)
- [xml.etree.ElementTree](#)
- [zipfile](#)

Appendix B. SentNet Installation Instructions

1. Unzip the contents of SentNet.zip into its own directory (e.g., SentNet)
2. Via the command line, navigate to the SentNet project directory then execute the following commands to create evaluation data identical to that which we utilized in the creation and testing of SentNet:
 - a. `python SentNet.py`
3. To create a new model, navigate to the SentNet project directory then execute the following command to create a training model for Essay Set 1:
 - a. `python SentNet_Training_Master.py`
4. If you wish to train a model for a different data set, simply open “SentNet_Training_Master.py” in a code/text editor of your choice and edit the variable ``data_set`` located in the Data Ingest section.
 - a. Models may be built for any of the other sets in the *Data* folder
 - i. Set2
 - ii. Set3
 - iii. Set4
 - iv. Set5
 - v. Set6
 - vi. Set7
 - vii. Set8

Notes:

- A. Depending on your installation of Python and your operating system, edits to the code may be required to resolve various path issues. The `pathlib` package was used extensively throughout our code to mitigate the risk of any such issues but the potential does exist for undiscovered conflicts.
- B. Python 3.6+ and the Pillow library have known issues that may result in image processing issues during the ingest process. These issues do not appear to exist on Linux. If such errors are encountered, switching to a Linux OS (if available) may be the preferred course of action.

Appendix C. Training and Validation Data Preparation

Data Description and Dictionary

The data upon which SentNet was developed, trained and tested was discovered in an expired Kaggle competition originally run in 2012.⁵ The data, “training_set_rel3,” came in three formats: a tab-separated value (TSV) file, a Microsoft Excel 2010 spreadsheet and a Microsoft Excel 2003 spreadsheet. For ease of reference, the files are all named “training_set_rel3.” We converted the TSV file to a comma-separated value (CSV) file for the purposes of function compatibility in Python.

All versions of the file contained the same information: 21,633 rows of essay information spanning 28 columns⁶:

- **essay_id**: A unique identifier for each individual student essay
- **essay_set**: 1-8, an id for each set of essays
- **essay**: The ascii text of a student's response
- **rater1_domain1**: Rater 1's domain 1 score; all essays have this
- **rater2_domain1**: Rater 2's domain 1 score; all essays have this
- **rater3_domain1**: Rater 3's domain 1 score; only some essays in set 8 have this.
- **domain1_score**: Resolved score between the raters; all essays have this
- **rater1_domain2**: Rater 1's domain 2 score; only essays in set 2 have this
- **rater2_domain2**: Rater 2's domain 2 score; only essays in set 2 have this
- **domain2_score**: Resolved score between the raters; only essays in set 2 have this
- **rater1_trait1 score - rater3_trait6 score**: trait scores for sets 7-8

Data Preparation

To more accurately simulate finalized analytic intelligence products, python scripts were written to ingest the CSV version of the file and subsequently create a Microsoft Office Word 2007-2013 document (.docx) and a plain text file (.txt) for every essay in the file. These files, located in the *Data_Ingest* folder, are as follows:

- SentNet_Convert_File_Types.py
 - Reads in the original .csv file and converts each essay into .docx and .txt files
- SentNet_Doc_Txt_Ingest.py
 - Reads a folder of .docx and .txt files and converts those files into a .csv file
- SentNet_Insert_Images_Into_Data.py
 - Injects a random number of images into all .docx files to simulate intelligence product visual information

A detailed description of how these files are utilized and in what order are listed in the sections below.

⁵ The Hewlett Foundation: Automated Essay Scoring Challenge Overview: <https://www.kaggle.com/c/asap-aes>

⁶ The Hewlett Foundation: Automated Essay Scoring Competition Data: <https://www.kaggle.com/c/asap-aes/data>

Data Ingestion and Conversion

Newly created .docx and .txt files are separated into different folders based on their respective “essay_set” by the Python script: “SentNet_Convert_File_Types.py” which is located in the *Data_Ingest* folder. Within this script, two functions handle this process: “convert2docx” and “convert2txt.” As their names imply, “convert2docx” will convert a row from the .csv file into its own Microsoft Word document while “convter2txt” will convert a row from the .csv file into its own individual plain text file.

The essay file, “training_set_rel3.csv” and all other file type variants of the same file, are located in the *Data* folder. Within the *Data* folder, subfolders are created for each essay set (e.g., “Set1”, “Set2”, “Set3”, etc.) during the ingest process. When individual essay files are created and placed in their respective essay set folder, they are further sorted into either a “.docx” folder or a “.txt” folder based on their respective file type.

Note: While .txt files were created for every essay in the training set as a proof of concept, only .docx files are utilized by SentNet in its current incarnation.

The finalized file structure for the project directory that pertains to training data creation can be seen, below, in Figure 15. This figure is a subset of that originally shown in Figure 3.

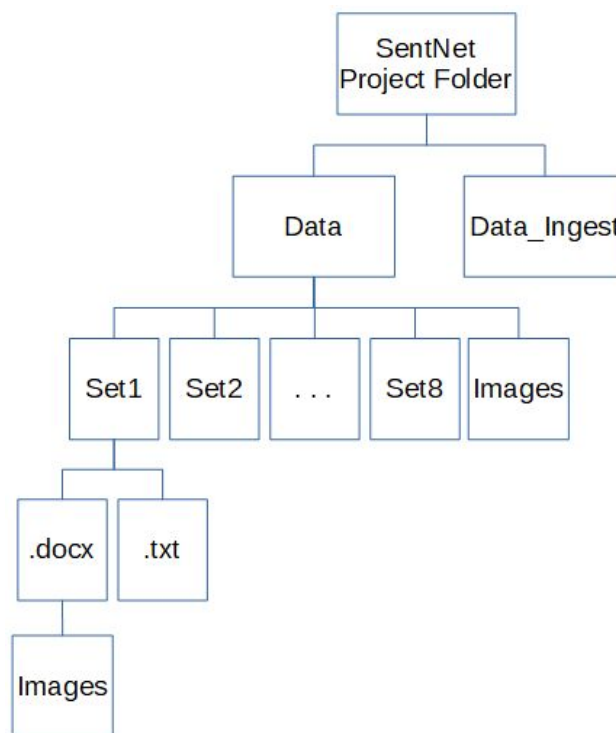


Figure 15. Folder Structure for SentNet Data

The aforementioned Python script, “SentNet_Convert_File_Types.py,” will check for the existence of all folders outlined in Figure A before creating any individual essay files. If the files do not exist, the folders will be created to ensure the proper organizational structure.

The Python script, “SentNet_Docx_Txt_ingest.py”, also located in the *Data_Ingest* folder reverses the previously described process. It ingests individual .docx files and .txt files and stores them in a dataframe that matches the original format of the training data (i.e. “training_set_rel3.csv” in the *Data* folder). This script utilizes only one function, “ingest_files” which detects the file type (e.g., .docx or .txt) and follows a specific process for each type to read in the file and ultimately place it in a Pandas DataFrame within Python.

To read-in Word Document files (i.e. .docx) the script utilizes the docx2txt Python library which breaks down every element of the document into separate pieces which can be easily parsed through to identify and extract the desired components for future analysis.

Image Injection into Documents

In order to accomplish our previously stated goal of accurately simulating finalized analytic intelligence products and to simultaneously address solution requirement number nine which stated the need for the solution to evaluate visual information in analytics products, we decided to identify publicly available images that could serve as stand-ins in our essay documents. We collected 30 images in total comprising 10 different images across three different chart types: bar, line, and pie charts. All images are located in the *Data/Images* folder in the project directory.

Sample images of each type are shown below in Figures 16, 17, and 18.

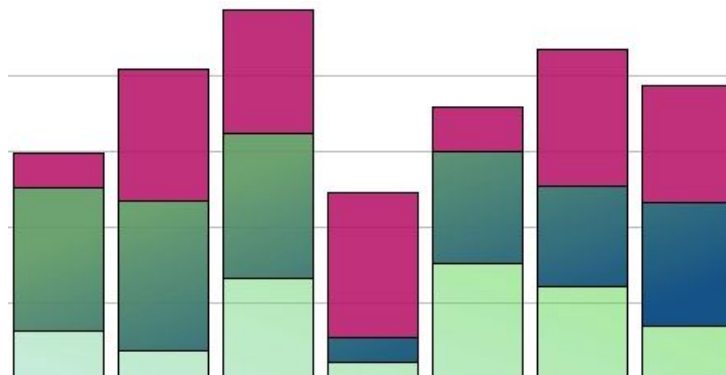


Figure 16: Example Bar Chart (BarChart1.png)

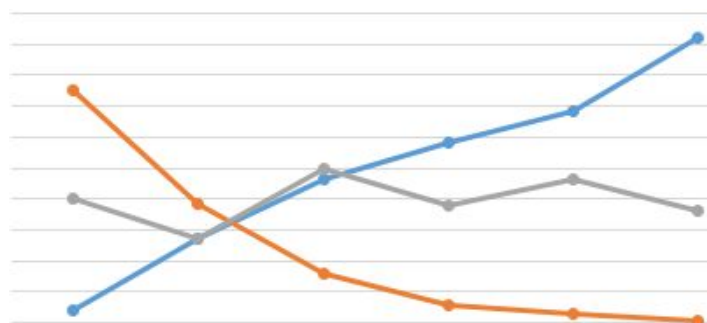


Figure 17: Example Line Chart (LineChart1.png)

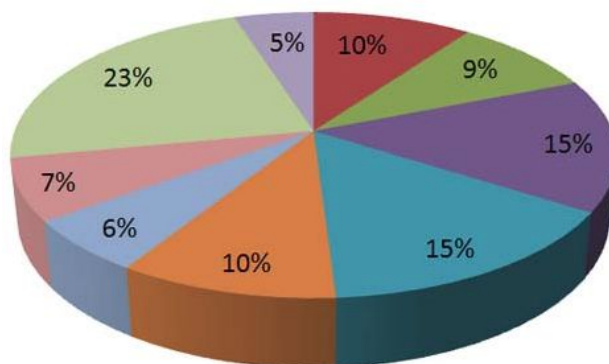


Figure 18: Example Pie Chart (PieChart1.png)

The process of inserting these images into Microsoft Word documents (i.e., .docx) is executed within the “SentNet_Insert_Images_Into_Data.py” Python script contained in the *Data_Ingest* folder. This script relies on two non-standard Python libraries (“Python-Docx” and “docx2txt”) and contains only one function: “insert_image_into_docx.”

For each directory set (e.g., *Data/Set1/docx*), the function calls the “ingest_files” function from the aforementioned “SentNet_Docx_Txt_ingest.py” and reads in the contents of each .docx document into a dataframe. The function then makes a random determination of how many files should be put into a file (0, 1, 2 or 3 images -- no more than one of each type: line, bar, pie). Images are then randomly selected from the *Data/Images* folder, based on the aforementioned file types that were chosen, and inserted into the document utilizing the “Python-Docx” library.

During this process, we made the assumption that visual information contained in above-average analytic products would often exhibit the same visual characteristics of those found in other above-average analytic products. Given that assumption, the same could then be said for average and below-average analytic products.

In an attempt to make our stock images reflect these characteristics, we added a decision process to our random image insertion pipeline. After randomly choosing how many images and what type of images to insert into a document, we then assess the overall “quality” of the document based on its “domain1_score.” Documents with a “domain1_score” in the bottom 30% of all scores from that set would receive an image with postfix value of 1, 2, 3 or 4. A score in the middle 40% will result in an image with a postfix value of 3, 4, 5, 6, 7 or 8. Finally, a score in the top 30% will pull from images with a postfix value of 7, 8, 9 or 10.

Implementing overlapping scores across percentile ranges was an intentional decision to reflect the assumption that overall document quality may not always be associated with the quality of visual information in the document. Additionally, this decision was made to prevent our images from being perfectly correlated with document scores and negatively impacting our modeling efforts. Tables 3, 4, and 5, shown below, lists the file names for each image in the *Data/Images* folder and their respective scoring distributions.

BarChart1.png	LineChart1.png	PieChart1.png
BarChart2.png	LineChart2.png	PieChart2.png
BarChart3.png	LineChart3.png	PieChart3.png
BarChart4.png	LineChart4.png	PieChart4.png

Table 3: Possible Images for Documents with a `domain1_score in their Set's Bottom 30%

BarChart3.png	LineChart3.png	PieChart3.png
BarChart4.png	LineChart4.png	PieChart4.png
BarChart5.png	LineChart5.png	PieChart5.png
BarChart6.png	LineChart6.png	PieChart6.png
BarChart7.png	LineChart7.png	PieChart7.png
BarChart8.png	LineChart8.png	PieChart8.png

Table 4: Possible Images for Documents with a `domain1_score in their Set's Middle 40%

BarChart7.png	LineChart7.png	PieChart7.png
BarChart8.png	LineChart8.png	PieChart8.png
BarChart9.png	LineChart9.png	PieChart9.png
BarChart10.png	LineChart10.png	PieChart10.png

Table 5: Possible Images for Documents with a `domain1_score in their Set's Top 30%

All images inserted into the documents are placed at the end of the document under the header(s) "Image1", "Image2" and/or "Image3." In the event the script randomly decides to not insert an image into a document, this entire process is skipped and the script simply moves onto the next file. All processed documents are stored in a subfolder called *Images* within each set's *docx* folder. Thus, the full path for .docx files in Set 1 injected with images (to include those not randomly chosen to have images) would be *Data/Set1/docx/Images*.

Appendix D. Model Results

Modeling Metrics

Run	Essay	Target	Levels	Correct	Total	True Accuracy	Adjusted Accuracy (+/- 1)	Model Error
1	1	domain1_score	12	192	357	53.8%	85.7%	0.6134
2	1	domain1_score	12	189	357	52.9%	84.0%	0.6359
3	1	rater1_domain1	6	228	357	63.9%	98.6%	0.3754
4	1	rater1_domain1	6	245	357	68.6%	99.7%	0.3165
5	1	rater2_domain1	6	251	357	70.3%	99.2%	0.3053
6	1	rater1_domain1	6	242	357	67.8%	99.2%	0.3305
7	1	rater2_domain1	6	250	357	70.0%	99.7%	0.3025
8	2	domain1_score	12	251	360	69.7%	99.7%	0.3056
9	2	rater1_domain1	6	250	360	69.4%	99.7%	0.3083
10	2	rater2_domain1	6	247	360	68.6%	99.7%	0.3167
11	2	rater1_domain2	4	235	360	65.3%	99.4%	0.3528
12	3	rater1_domain1	3	241	346	69.7%	97.4%	0.3295
13	3	rater2_domain1	3	238	346	68.8%	98.8%	0.3237
14	3	domain1_score	3	237	346	68.5%	97.7%	0.3382
15	4	rater1_domain1	3	242	354	68.4%	99.2%	0.3249
16	4	rater2_domain1	3	246	354	69.5%	98.6%	0.3192
17	4	domain1_score	3	242	354	68.4%	98.9%	0.3277
18	5	rater1_domain1	4	217	361	60.1%	99.2%	0.4072
19	5	rater2_domain1	4	224	361	62.0%	99.4%	0.3850
20	5	domain1_score	4	236	361	65.4%	99.7%	0.3490
21	6	rater1_domain1	4	228	360	63.3%	99.7%	0.3694
22	6	rater2_domain1	4	229	360	63.6%	98.1%	0.3833
23	6	domain1_score	4	234	360	65.0%	99.2%	0.3583
24	7	domain1_score	24	65	314	20.7%	37.3%	2.8121
25	7	rater1_domain1	12	104	314	33.1%	55.4%	1.5860
26	7	rater2_domain1	12	119	314	37.9%	59.6%	1.4586
27	7	rater1_trait1	3	184	314	58.6%	94.6%	0.4682
28	7	rater1_trait2	3	184	314	58.6%	98.7%	0.4268
29	7	rater1_trait3	3	214	314	68.2%	100.0%	0.3185
30	7	rater1_trait4	3	183	314	58.3%	99.4%	0.4236
31	7	rater2_trait1	3	186	314	59.2%	93.0%	0.4809
32	7	rater2_trait2	3	198	314	63.1%	99.0%	0.3790
33	7	rater2_trait3	3	217	314	69.1%	99.7%	0.3121
34	7	rater2_trait4	3	181	314	57.6%	100.0%	0.4236
35	8	domain1_score	40	49	145	33.8%	33.8%	4.1172
36	8	rater1_domain1	23	65	145	44.8%	55.2%	1.7172

37	8	rater1_trait1	6	37	145	25.5%	99.3%	0.2828
38	8	rater1_trait3	6	102	145	70.3%	97.9%	0.3172
39	8	rater1_trait4	6	106	145	73.1%	99.3%	0.2759
40	8	rater1_trait5	6	99	145	68.3%	97.2%	0.3448
41	8	rater2_trait1	6	79	145	54.5%	97.2%	0.4828
42	8	rater1_trait2	5	105	145	72.4%	99.3%	0.2621
43	8	rater1_trait6	5	94	145	64.8%	100.0%	0.3517

Table 6: Testing Results of all 43 Models Produced by SentNet v1.0 Prototype

Modeling Feature Importance

As mentioned in the modeling results section, 43 different models were developed and evaluated. Table 7, shown below, details the number of times each synset feature was used in a model and its average importance as a feature across all 43 tested models.

Synset	Count	Average Importance
ts_count_sybl	43	61.07%
tt_count_sybl	43	60.75%
tt_count_word	43	58.90%
ts_count_word	43	57.80%
entityn01	43	51.19%
Sim_Pred_Class	23	50.06%
Word_Cluster_2	12	49.78%
ts_score_smog	4	48.55%
Synset_Cluster_0	32	46.59%
personn01	2	46.52%
Synset_Cluster_1	41	46.35%
abstractionn06	43	45.14%
physical_entityn01	34	44.86%
objectn01	15	44.07%
Synset_Cluster_2	23	43.96%
attributen02	4	43.32%
causal_agentn01	1	42.61%
staten02	4	42.45%
ts_count_sent	43	42.35%
objectn01_btw	5	41.84%
Word_Cluster_3	4	41.74%
psychological_featuren01	16	40.95%
tt_count_sent	43	40.64%
Word_Cluster_0	35	40.07%
ts_score_fleschkincaid	1	39.19%

Word_Cluster_4	4	39.03%
Word_Cluster_1	15	38.36%
wholen02	2	33.97%
physical_entityn01_btw	6	32.96%
artifactn01	2	32.45%
phenomenonn01	1	31.96%
entityn01_btw	9	30.63%
craftn02	1	29.76%
instrumentalityn03	1	29.50%
abstractionn06_btw	9	28.66%
psychological_featuren01_btw	5	17.92%

Table 7: Average Variable Importance Across All 43 Tested Models