Document Classification Techniques for Aviation Letters of Agreement

Anna Batra*, Swetha Rajkumar†, David Nielsen‡, Stephen S. B. Clarke§, and Krishna M. Kalyanam¶
NASA Ames Research Center, Moffett Field, CA, 94035, USA

Kanvasi Tejasen‖, Melissa Ohsfeldt**, Mike Copp††
Federal Aviation Administration, 800 Independence Avenue, SW, Washington, DC 20591

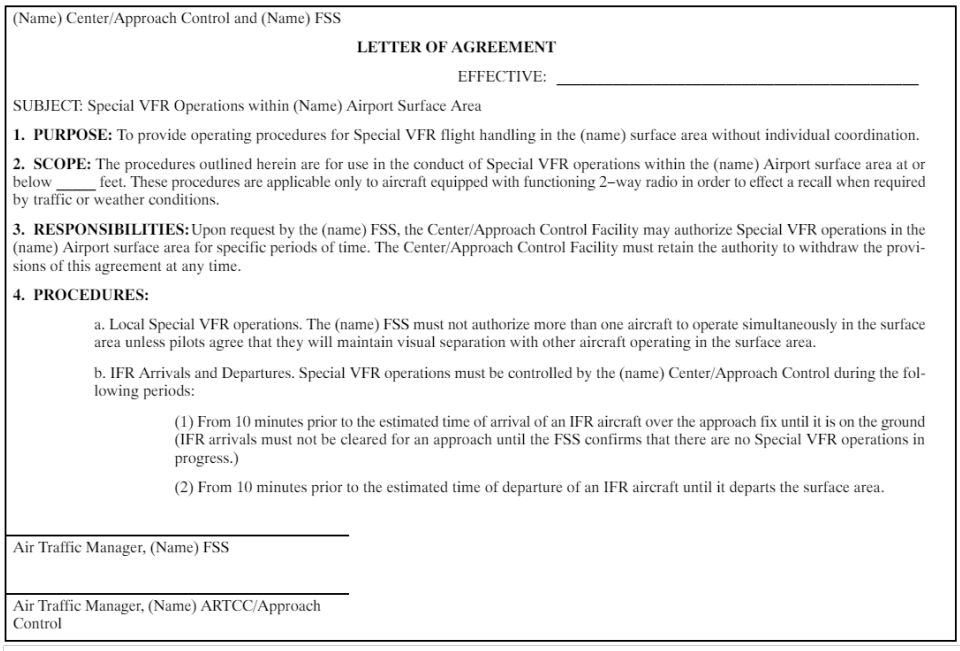
Often when working with historic air traffic management (ATM) documents, it is helpful to classify them into specific categories. In this paper, we conduct a thorough review of natural language processing techniques to perform this classification task on Letters of Agreement (LOAs), technical aviation documents outlining rules for utilizing US airspace. We evaluate multiple techniques for representing the text in the documents as embeddings: unigram and bigram Term Frequency Inverse Document Frequency (TFIDF), Word2Vec, Doc2Vec, GloVe and RoBERTa. We investigate a wide range of classification models: K-Nearest Neighbors, Random Forest, Support Vector Machines (SVM), Logistic Regression, Naive Bayes, Feed-Forward Neural Network, Convolutional Neural Networks (CNNs) and Long-Short Term Memory (LSTM). By comparing the different methods, we found the best overall approach for our task was to use unigram TFIDF representations with SVM while also gaining insight into how the other methodologies performed on a small technical datasets.

I. Introduction

The United States National Airspace System (NAS), managed by the Federal Aviation Administration (FAA), is subdivided into many different regions and areas of control which contribute to the safe management of air traffic. Users of the NAS range from commercial airlines, civilian operators, military airspace users, and independent companies such as sky-diving operators who all need to coordinate their movements with air traffic controllers. In order to formalize and standardize this coordination, the FAA established Letters of Agreement (LOA). These letters are made between two or more parties, and establish the default operations in or between adjacent airspaces. These rules or conditions that airspace users must follow are referred to as *constraints*.

Currently, LOAs are stored as portable document format (PDFs) and are made accessible to their responsible parties. In particular, controllers are trained on the LOAs that pertain to their airspace but other airspace users do not have direct access to the constraints contained within the LOAs; they may glean from experience what positions and altitudes they are likely to be assigned (i.e., based on routes they were allowed to fly in the past). To address this accessibility issue, we are exploring how best to digitize and share the LOA constraints with airspace users in a form that is compatible with existing aviation data models such as the Aeronautical Information Exchange Model (AIXM)*. These digitization efforts have been documented in a report to the International Civil Aviation Organization [1]. This report outlines the goal and steps taken towards the overall digitization effort.However, because LOAs can be made between the FAA and the military and/or private companies, not all documents should be made publicly available. We consider the LOAs that are only between the FAA and/or other public entities to be 'civil' LOAs and the constraints within can be digitized and shared. All LOAs that do not meet this criteria are 'not-civil'. This is due to at least one party to the LOA not being a public entity and therefore not being eligible for public release.

*Graduate Student, University of Washington, Computational Linguistics
†Student, University of California Berkeley, Electrical Engineering and Computer Science
‡Senior Aerospace Research Engineer, KBR, david.l.nielsen@nasa.gov
§Senior Aerospace Research Engineer, Flight Research Aerospace, stephen.s.clarke@nasa.gov
¶Senior Aerospace Research Engineer, NASA Ames Research Center, AIAA Associate Fellow
‖Project Manager, FAA, Info-Centric NAS (ICN) Technology & Advanced Concepts Division
**Managing Partner, Silver Maple LLC
††ATC Specialist, LS Technologies
*https://www.aixm.aero/



(Name) Center/Approach Control and (Name) FSS

LETTER OF AGREEMENT

EFFECTIVE: _____

SUBJECT: Special VFR Operations within (Name) Airport Surface Area

1. PURPOSE: To provide operating procedures for Special VFR flight handling in the (name) surface area without individual coordination.

2. SCOPE: The procedures outlined herein are for use in the conduct of Special VFR operations within the (name) Airport surface area at or below _____ feet. These procedures are applicable only to aircraft equipped with functioning 2–way radio in order to effect a recall when required by traffic or weather conditions.

3. RESPONSIBILITIES: Upon request by the (name) FSS, the Center/Approach Control Facility may authorize Special VFR operations in the (name) Airport surface area for specific periods of time. The Center/Approach Control Facility must retain the authority to withdraw the provisions of this agreement at any time.

4. PROCEDURES:

a. Local Special VFR operations. The (name) FSS must not authorize more than one aircraft to operate simultaneously in the surface area unless pilots agree that they will maintain visual separation with other aircraft operating in the surface area.

b. IFR Arrivals and Departures. Special VFR operations must be controlled by the (name) Center/Approach Control during the following periods:

(1) From 10 minutes prior to the estimated time of arrival of an IFR aircraft over the approach fix until it is on the ground (IFR arrivals must not be cleared for an approach until the FSS confirms that there are no Special VFR operations in progress.)

(2) From 10 minutes prior to the estimated time of departure of an IFR aircraft until it departs the surface area.

Air Traffic Manager, (Name) FSS

Air Traffic Manager, (Name) ARTCC/Approach Control

Fig. 1 Truncated example LOA from FAA 7210.3

While it might seem easy to simply have a list of all official aviation entities entities and use a rule based filter to identify the documents, our previous work on named entity recognition (NER) showed that the variation in how agency names were written (fully spelled out, containing common acronyms, etc.) make this approach untenable. It is better to have a model-based document classification scheme due to its ability to generalize between similar entity names. The LOA documents contain metadata that gives us a straightforward way to create a labeled dataset. Subject matter experts (SMEs) use this metadata to label documents - see discussion in section II. This provides us with training data for a model-based approach. Labeled data is scarce in the aviation domain and these SME labels provide the framework to evaluate different NLP methods on the dataset and find techniques we can use in other low resource applications within LOA classification. As such, we cast this as a two-class classification problem with the labels *civil* and *non-civil*.

LOAs are written in a highly technical language that uses Aviation/Air Traffic Management (ATM) specific phraseology. There are many proper nouns, abbreviations, and technical aviation terminology that is not present in general English. Additionally, as a small dataset, there are few examples of each proper noun or technical term. This characteristic motivated us to evaluate methods built purely from the data using models trained on general English as well as *transfer learning* from general English to what we call 'Aviation English'. As such, we performed a survey of embedding methodologies as well as modeling techniques to establish the optimal approach for representing and classifying documents in a technical dataset of this size. This gives us the most insight into what representations to use for LOA data and what models preform well on the classification task. This paper documents the creation of the LOA dataset used for classification (Section II), the methodology used to pre-process the text data before creating our embeddings (Section III.A), creating embeddings (Section III.B), before training two-class classifiers to identify the civil documents (Section III.C), and finally evaluates the best performing techniques (Section IV).

II. Dataset

A. LOA documents

As noted, the LOAs are currently stored as PDFs which can be seen in the FAA Order JO 7210.3DD[†] and Figure 1. In order to create a test set of documents for our digitization effort, we focus on 493 documents from Dallas Fort Worth

[†]https://www.faa.gov/air_traffic/publications/atpubs/foa_html/chap4_section_3.html

Air Route Traffic Control Center (ZFW)[‡]. We chose ZFW because our SMEs had vast experience in that region and we could leverage their knowledge in document classification. The documents were mostly LOAs but a few other aviation procedure documents were included as well to allow our model to have a more real-world dataset on which to classify. After gathering these documents we used SME knowledge along with the document storage that provides metadata on whether a document is an LOA or a Standard Operating Procedure (SOP), civil or military, etc. Our SMEs were able to label each document in our dataset as either a 'civil LOA' or not. This resulted in a set of 222 civil LOA documents and 271 documents that are 'non-civil LOAs'. This is a relatively small dataset which will allow us to asses which techniques in Section III are best suited to its size. With our documents thus labeled, we could then begin to extract the text and pre-process them for two-class classification.

B. Document pre-processing

The text of the LOAs making up our dataset is considered an unstructured dataset as there is no agreed upon formal way to write a Letter of Agreement; there are guidelines to follow but these allow for significant variation. This means that the formatting of the documents tend to vary between facility to facility, and even letter to letter. In addition, we have non-LOA documents that made it into the dataset as well. There are recommended section headers (e.g. Purpose, Subject, Scope, Cancellation, Responsibilities, Procedures, and Appendix) that many of documents use, though there are still many that do not follow these conventions.

The LOA pre-processing methodology begins by converting the LOA dataset of PDFs to plain text using *Amazon Textract*[§]. This returns a JSON structure with all the document text captured either using PDF2Text or optical character recognition (OCR). This provided us with the raw text to begin pre-processing. We then proceeded to extract the most relevant document sections for civil classification from this raw text. We used rule based methods along with the LOA template as published in FAA Order JO 7210.3DD[¶] to identify section headers and group text with these headers when they are present. This creates a semi-structured dataset to begin our representation and model survey work.

III. Methodology

A. Text Pre-Processing

To convert our documents into a representation readable by our machine learning models we convert them to numerical embeddings, also known simply as *embeddings*. Embeddings that contain the specific information needed to discern whether a document is a civil LOA or not is pertinent to getting the best accuracy out of machine learning model for classification. Any extra or superfluous information may cause the embeddings to have less prominent features for the classification and will result in a poor model. For example, the beginning of civil LOAs tend to have more useful information pertaining to the document class within the Purpose, Subject, and Scope sections while the remainder of the document focuses on Procedures containing constraints that have less relevant information. As such, we will focus the pre-processing on collecting this relevant information over the full text.

Note that while all of the documents in our dataset were able to be processed by Amazon Textract, we were not always able to extract the expected section headers and associated section text. Those that were nicely extracted had the expected section headers and associated text. However, many times it failed to extract section information at all and we just had the unstructured Textract text to work with. This led us to use a rule-based schema to make sure we use the most informative text possible for our embeddings.

1. Rule-Based Schema Part I

The first part of our rule-based schema involved the documents that were able to be nicely extracted with section headers and section text using the following section headers: DOCUMENT, TITLE, FACILITY, EFFECTIVE, SUBJECT, PURPOSE, and SCOPE. The documents were picked under the following guidelines through manual exploration of what sections may be helpful towards identifying whether a document was a civil letter of agreement or not.

Only one of the following four criteria must apply to be applicable to Part I of the rule-based schema: (1) All of the above sections were present; (2) All of the above sections, except SCOPE, were present; (3) All of the above sections,

[‡]This dataset cannot be made public due to being an internal FAA dataset.

[§]`https://docs.aws.amazon.com/textract/latest/dg/what-is.html`

[¶]`https://www.faa.gov/air_traffic/publications/atpubs/foa_html/chap4_section_3.html`

except SCOPE and any ONE other section, were present; (4) All of the above sections, except TITLE and FACILITY, were present. It does not matter if SCOPE was present or not here as well.

The last guideline will make use of the section called RAW_HEADER as well, since a lot of the info missing from the sections was put into this unnamed header by *Amazon Textract* at the beginning of the document, but not parsed through accurately. For the embeddings we will use all of the available text from the sections gathered using the guidelines above, as well the RAW_HEADER for the last guideline.

2. Rule-Based Schema Part II

For the rest of the documents not matching the above criteria, text was extracted from only the first page of each document regardless of any sections. From manual exploration of what may help the classification, we found that the text here tended to match what informative text was used in *Rule-Based Schema Part I*. The length of one page also matched well with the same amount of text found in *Rule-Based Schema Part I*, if it was a letter of agreement. The first page also had valuable identifying information even if the document wasn't a letter of agreement.

3. Clean-Up and Tokenization

Clean-up is integral as this removes extra text structure that humans use purely for readability, and not needed for the model. This tends to include formatting and punctuation on documents. For our text, we combined paragraphs together. Regular expressions from the package *regex*[‖], were used to remove extra formatting bullet point numbers, bullet point letters, bullet point roman numerals, plain numbers on a line, page numbers, and all punctuation. The package NLTK *word_tokenize*[**], was then employed to tokenize all the text into words. As the last part of our clean-up, words in all uppercase or mixed case (past the first letter) were maintained to help preserve acronyms so that they are succinct from other words. The remaining were standardized to lowercase to help the computer recognize that they are the same words.

B. Embedding Approaches

In order to find the best hyperparameters for each embedding method, all embeddings were tuned on a downstream classification task. The details of the models used for the classification task can be found in the section III.C. Although there are intrinsic methods for embedding evaluation, such as analogy tasks [2], the majority of these focus on evaluating word-embeddings. Since we are creating document embeddings we decided to use the extrinsic evaluation task of classification to verify that our embeddings were learning the information needed. All embedding methods except the RoBERTA [3] transfer learning were tuned with a SVM, as defined in section III.C.3. Logistic Regression, defined in section III.C.4, was used to tune RoBERTa transfer learning embeddings.

In the classification task used for tuning, both SVM and Logistic Regression were trained with 10-Fold cross validation on F1-Macro score, with 90% of the full dataset being used as training data. The technique of cross validation is helpful for our problem, as it helps us make the most out of our small dataset by helping us re-sample our data for multiple runs. A Grid Search over the parameters mentioned in Section III.C was utilized for tuning with the downstream task. A random seed of 211 was used for the Python libraries *r*andom, numpy, pytorch, and *h*ugginface.

1. Frequency-Based Embeddings

The simplest embeddings we use for our survey of embeddings are Term Frequency Inverse - Document Frequency (TFIDF) embeddings. The TF-IDF weighting scheme is commonly used in many information retrieval systems. This should help the TF-IDF weights in the vector to have higher scores for words more rare in the document, and thus help indicate how that document's content may differ a bit from others. [4]

For this exploration, we used the *TFIDFVectorizer* implemented by the python library *scikitlearn*[††]. We used the n-gram ranges unigram (one word), bigram (two words), and unigram & bigram (both combinations of one and two words) and trained it over our own document corpus. N-grams are the number of words in sequence we used as the TF-IDF feature type. For simplicity, we will call these embeddings TFIDF Unigram, TFIDF Bigram, and TFIDF Unigram + Bigram respectively. For hyperparameters, we chose to include less frequent words from the documents by choosing to not eliminate any words from the TFIDF embeddings ($min\_df = 0$). We thought this may help the classification due to our small low-resource dataset within the airspace domain language. This is because a lot of civil

[‖]Package: `https://docs.python.org/3/library/re.html`

[**]Package: `https://www.nltk.org/api/nltk.tokenize.word_tokenize.html`

[††]Package: `https://scikit-learn.org/stable/`

N-gram	max_df	norm
Unigram	0.65	L2
Bigram	50	L2
Unigram + Bigram	70	L2

Table 1 TFIDF Tuned Embedding Parameters

Parameter	vector_size	window	workers	alpha	epochs	sample	negative
Value	350	20	6	0.01	1000	1e-4	15

Table 2 Word2Vec Tuned Embedding Parameters

documents could be classified by a variety of specific terms that may indicate a facility or vessel that conducts civil operations, but conversely each term may be rarely used due to the large variety of them. We also chose to remove stopwords with the given 'english' list provided by sklearn (stopwords='english') since these features may not be helpful toward classification as they are general terms. We tuned the other hyper-parameters, max_df and $norm$, using SVM to obtain meaningful embeddings that will work well end-to-end with classification, as shown in Table 1 below.

2. Co-occurrence Embeddings

When using word sequence frequencies, as in the above section, contextual knowledge may be lost as words typically have different meanings in the context of other words. GloVe [5] makes use of co-occurrence frequencies to handle just that. But it is only able to handle words in the immediate vicinity of the feature/token. It is done by creating a co-occurrence matrix based on statistics of an existing corpus [5]. For this exploration we used the pre-trained implementation by Pennington et al.[‡‡]. We use the matrix that was pre-trained on Wikipedia 2014 + Gigaword 5 with 300 dimensions. Since these embeddings produced are actually word embeddings, we employed plain averaging and TF-IDF-weighted averaging as defined in Section III.B.6. For simplicity, we will call these embeddings Averaged Glove and TFIDF-Averaged Glove respectively.

3. Bigger Windows for Contextual Word Embeddings

Neural networks are able handle bigger windows while producing contextual word embeddings. Word2Vec is one the first examples of this seen in the field, as GloVe actually post dates it. Word2Vec makes use of a fake task to be able to obtain the weights for the layers, with a common fake task being the Continuous Bag of Words (CBOW). With CBOW we input several words (with no word-order hence it being a bag-of-words) and output a word that should be related as per the context. This fake task provides data to the network which gives the network the ability to fine-tune weights to predict based upon context words or produce the context words, and try to understand words at a contextual level [6]. For this exploration we used the *Gensim*[§§] Word2Vec implementation, and train it over our own document corpus using the CBOW approach. Since these embeddings produced are also word embeddings, we employed plain averaging and TF-IDF-weighted averaging as defined in Section III.B.6. For simplicity, we will call these embeddings Averaged Word2Vec and TFIDF-Averaged Word2Vec respectively. For hyper-parameters, we chose to stay with the CBOW model ($sg = 0$), and to not ignore any words with lower frequencies ($min\_count = 0$). We chose to keep all lower frequency words for the same reason as mentioned with the TFIDF vectors in section III.B.1. The number of workers was not tuned and kept at 6, as we are not using parallelism for faster training. The rest of the parameters, vector_size, window, alpha, epochs, sample, and negative, were tuned with and may be referenced in Table 2.

Doc2Vec is built over Word2Vec, and helps to create document or paragraph vectors. Similar to Word2Vec, Doc2Vec trains words with similar contexts to have closer vectors in space, but at the same time, also trains a document feature vector to try to capture what is missing from the context [7]. For this exploration, we used the Doc2Vec *Gensim* implementation as well. The Doc2Vec embeddings were actually tuned before the Word2Vec ones, and as you may notice, some of the values for hyper-parameters are similar since the models are trained similarly. We use the distributed memory version of the model, which is similar to CBOW in the Word2Vec model. This may contribute to why some of

[‡‡]Package: `https://nlp.stanford.edu/projects/glove/`

[§§]Package: `https://radimrehurek.com/gensim/index.html`

Parameter	vector_size	window	workers	alpha	epochs	sample	negative	dm_concat
Value	300	8	6	0.01	1000	1e-5	15	0

Table 3 Doc2Vec Tuned Embedding Parameters

Parameter	learning_rate	weight_decay	adam_beta1	adam_beta2
Value	0.0001	0	0	0.75

Table 4 TL RoBERTa Tuned Embedding Parameters

the best hyper-parameters are similar across embedding models. We choose $min\_count = 0$. The parameters we tuned matched those of Word2Vec, except for dm_concat which we have additionally tuned, as shown in Table 3.

4. Large Language Models

Large language models using transformer-based architectures not only handle bigger windows for more context, but also, as in the name, handle a large corpus as its training data. This can allow the model to have a much greater number examples to learn the context of language and may produce more accurate embeddings. However, it does come with a cost in that it will only learn from the type of corpus that it was trained on. Since our problem task is of a specific aviation domain, it might not work out to be the perfect embeddings. This leads us to the use of transfer learning with these types of pre-trained embeddings. For this exploration we used the *HuggingFace* implementation of the pre-trained RoBERTa[¶¶] model embeddings. The model was mainly based off of an improved BERT as the name itself stands for Robustly Optimized BERT pre-training Approach [3]. We also used transfer learning with this model, which is explained in the next section.

5. Transfer Learning

Transfer learning is a widely used technique in machine learning, often when we have a small or limited amount of training data to work with. Our dataset is small, so it is worth employing transfer learning to see if it can help us to improve the accuracy. For our purposes, we make use of transfer learning by using a pre-trained model on a large English corpora. The pre-trained corpora will have picked up on a good base for English patterns in language, which can help it to perform well on basic English tasks. This can be helpful to our problem as training from scratch with a small dataset may not be enough for the model to pick up on the patterns. To make use of the pre-trained model, we can use its existing weights (trained on the large corpora), and train over it more with our small training dataset to fine-tune it with our dataset's documents and have it learn more about our document's domain. Transfer learning can be helpful if the original corpora of the pre-trained model is similar in language to the dataset. [8]

For this task, the pre-trained Glove and RoBERTa models were pulled and fine-tuned on our own document corpus. The pre-trained GloVe vocab weights were input into an untrained Word2Vec model with it set to re-train over it using our own document corpus. This means our vocab from the train data will be incorporated while leveraging all the extra knowledge from the existing pre-trained GloVe. Since these embeddings with once again correspond with words we will use plain averaging and tfidf-weighted averaging defined in this section III.B.6. For simplicity, we will call these embeddings TL Glove, where TL stands for transfer learning. To train the hyper-parameters, we started by using the best parameters found by Word2Vec, while still accounting for the different pre-trained vector_size of 300. We found it to perform well with these beginning parameters and stuck with them. Similarly, RoBERTa was fine tuned over our own document corpus. We used the weights from the RoBERTa Base model and trained over it using the RoBERTaForMaskedLM model. We chose the $per\_device\_train\_batch\_size = 16$. We settled with $num\_train\_epochs = 10$. We then performed with a manual grid search to tune the the finalized hyper-parameters[***], as presented in Table 4. For simplicity, we will call these embeddings TL RoBERTa.

[¶¶]Package: `https://huggingface.co/docs/transformers/model_doc/roberta`

[***]Note that the default optimizer is AdamW.

6. Word-Vector Averaging

Since both GloVe and Word2Vec are word embeddings, we must employ a word-vector averaging strategy to have one embedding to represent the full document. Two strategies, plain averaging and TF-IDF averaging, are discussed below. First as part of pre-processing, we first removed all the stopwords from the *NLTK* English stopwords list[†††] for both type of averaging. To calculate the plain average, we performed element-wise addition between all the produced word vectors and performed division of the resulting vector by the scalar n, where n is the number of words that are both present in the document (without stopwords) and in the pre-trained corpus if there was one.

To calculate the TFIDF average, we multiplied each vector by the scalar w, where w is the corresponding TF-IDF weight for the word take from the the unigram *TFIDFVectorizer* model. If there are unknown TFIDF word weights, $w = \ln(394)$. This was chosen as 394 is 80% of the number of documents in the entire data, and this number is around the number used for the training data in the "train/validation" split, where 399 training documents were used for the training when doing hyper-parameter tuning with the cross-validation datasets. The cross-validation datasets were made up of 0.1 of the original 493 documents in the training set. Finally we followed the same process as the plain average to produce an average of our vectors.

C. Classification Models

Using all the optimized representations we have just discussed in Section III.B we now want to use these embeddings to evaluate different modeling techniques. In this section we will survey the different types of models and our reasoning behind why we chose them. The models below each use all the possible embeddings in Section III.B. For all models which utilize randomness, a random seed or state of 211 was used for all applicable parameters or the seed that was set for the package.

All models make use of 10-Fold cross validation. The tuning technique used for sklearn models is Grid Search, which is also from sklearn[‡‡‡]. For Keras models[§§§] the technique used is Randomized Search, from sklearn, due to the much larger amount of parameters. For Randomized Search, lesser and lesser amounts of parameters were used each run to help try to find the optimum values for the parameters, while trying to center around the parameters that did well in the last round.

1. Classification Based on Known Labeling

We used K-Nearest Neighbors (KNN) as the algorithm is different from the others; it focuses heavily on the proximity of a known label. K-Nearest Neighbors looks at the K nearest already classified training document embeddings in space and what classes are being predicted by them, then it approximates the current document's class from a new dataset with that information. [9] Being built entirely on distance from known contrasts with the other approaches that attempt to build a discriminative or generative model for classification as described below.

For KNN we used the *sklearn* implementation and tuned all the hyper-parameters other than the type of *algorithm* to compute the nearest neighbors. Since some of our embeddings were in a sparse vector representation (TF-IDF), we used the algorithm *brute* as that was the only option for sparse vectors. Since some of our embeddings where sparse vectors, we decided to always use brute force for the algorithm. The rest of the parameters tuned are *n_neighbors, weights*, and *metric*.

2. Decision Tree

We also used a Random Forest (RF) model, which is an advanced Decision Tree model that makes use of sampling. A Random Forest model creates many decision tree models on subsets of the data and averages the predictions for the final prediction. We used the Random Forest implementation from *sklearn* and tuned all these parameters: *n_estimators, criterion, max_depth.* and *max_features*. [10]

3. Linear Model

We also used Support Vector Machines (SVM), which are a popular type of linear model. A SVM is a linear model that maximizes the distances between the classes in space (the decision boundary). It can also solve non-linear problems

[†††]Package: `https://www.nltk.org/`

[‡‡‡]Package: `https://scikit-learn.org/stable/`

[§§§]Package: `https://www.tensorflow.org/guide/keras`

using special constructed kernels. For SVM, we used the *sklearn* implementation and tuned all these parameters: *kernel, C, degree, gamma,* and *coef0*. [11]

4. Probabilistic

Logistic Regression (LR) is a discriminative classifier that makes use of an independence assumption and uses the logistic function to map instances to a probability it belongs in that class [11, 12]. Naïve Bayes (NB) is a generative classifier that uses an assumption of conditional independence (Bayes Rule) to map instances to a probability it belongs in the class [11, 12]. We used both these models to compare if a probabilistic classifier may do better than the others, and what what assumptions it might do better under. For both these models we used the *sklearn* implementation.

For Logistic Regression, in choosing the *solver*, *liblinear* was chosen due to it being usable with both the *l1* and *l2* penalties. *Saga* was not used as the solver as it's meant for much larger datasets. The parameter *dual* meant for an *l2 penalty* with *liblinear*, was always set to *False* since the number of samples tended to be greater than the number of features for most of the embeddings. This was true for all embeddings but the TFIDF and RoBERTa embeddings. *fit_intercept* was also set to *True* as it's useful with *liblinear*, and *intercept_scaling* set to *0.0001* after experimenting with it a bit and settling on this number that tended to do well for most of the embeddings. The *class_weight* was set to *balanced*, as our dataset is balanced for the most part, but we let the model figure out the proportions through cross validation. Finally, we chosen the max number of iterations, $max\_iter = 1000$, as the solver tended to work best around those number of iterations, even if was not able to converge to the solution. The parameters that we tuned using grid search were *penalty* and *C*.

For Naive Bayes, the *sklearn* implementation of *MultinomialNB* was used. The parameter *fit_prior* was chosen to be *True* so probabilities will be learned from the given data. Otherwise, we focused on tuning the *alpha* parameter with grid search.

5. Neural Networks

Neural networks are a subset of machine learning where the model does not rely on human patterns and probability theory and instead tries to approximate a function, usually using more data than a non-deep learning algorithms to do so [13].

Feed-forward networks (FFN) are the first type of neural network that we make use of. We used both the Multi-Layer-Perceptron (MLP) implementation from *sklearn* and a FNN created by *Keras*. A MLP is a subset of FNNs in that it is a densely connected feed-forward network. It differs from our Keras version that make use of a dropout of 0.2. Additionally, we allow the MLP to tune the number of layers it uses, while with the FNN we restrict it to one layer. For the MLP an $alpha = 0.1$ was chosen. The *learning_rate* was chosen to be constant, with a $learning\_rate\_init = 0.08$ and $max\_iter = 300$. *Shuffle* was set to *False* with the *solver* adam. The *activation*, *beta_1*. *beta_2*, and *hidden_layer_sizes* were hyper-parameters that we tuned. We also chose to tune the *batch_size* to be between 25 and 30, along with other hyper-parameters. Our FNN architecture in *Keras* is built out of the Sequential model and starts with the giving embedding passed into a Dense Layer with relu activation, which is then passed to a Dropout Layer of 0.2, and then finally at a final Dense layer of size 1 with sigmoid activation for the classification result.

Dropout is one way a neural network can try to combat over-fitting. When using dropout over an input layer, it is often better to keep between 0.5 to 1 of the nodes, and better if it's closer to 1 [14]. Additionally, dropout can become more and more computationally expensive as the number nodes drop since that will make the number of iterations to converge bigger [15]. We chose $dropout = 0.2$ for now, but we could tune it more for the future. We chose a $learning\ rate = 0.001$ with $epochs = 10$ and early stopping with a $patience = 3$ monitoring the max validation accuracy. $Batch\ size = 20$ was also chosen. Due to the large compute power of neural networks, size of the singular dense layer was chosen to be about half of the size of the embedding input shape. For the TFIDF Unigram + Bigram and TFIDF Bigram embeddings, a $dense\ layer\ size = 1024$ was chosen due to the extremely large input embeddings sizes, causing the model to take a prohibitive amount of time to train. *beta_1*, *beta_2*, *weight_decay*, and *ema_momentum* were hyper-parameters.

Convolutional Neural Networks (CNN) have a different type of neural architecture using filters that is mainly used to extract different types of features in images. In an NLP context, we must reshape the embeddings to model an image matrix, to see if we can hopefully get the model to also recognize low-level and high-level patterns in text to help with our document classification. Our CNN architecture in *Keras* is also built out of the Sequential model and starts with the giving embedding passed into a 1D convolutional Layer with relu activation, a Flatten layer, a $dropout = 0.2$, and finally a $dense\ layer\ size = 1$ with sigmoid activation for the classification result. Early stopping was also utilized.

The CNN parameters that align with FNN, were chosen and tuned in the same manner. However, we also focus on tuning the kernel for the Convolutional 1D Layer, and the maxpooling size that follows after it. The filter size for the CNN matched that of the FNN's dense layer sizes respectively. [16]

Long Short-Term Memory (LSTM) neural networks are a type of Recurrent Neural Network made to be used with sequential data such a text. This makes this model better at handling dependencies between different words in text. Our LSTM architecture in *Keras* is built out of the Sequential model and starts with the giving embedding passed into an LSTM layer with activation tanh. It is then passed in a $dropoutlayer = 0.2$, and finally a $dense\ layer\ size = 1$ with sigmoid activation for the classification result. Early stopping was not utilized here. The LSTM parameters that align with both FNN and CNN, were chosen and tuned in the same manner. However, the size of the LSTM layer was chosen to match the size of the FNN's dense layer and CNN's filter sizes. [16]

IV. Results

For the results tables, F1-Macro scores in plain bold are the top score for the model. Scores that are also italicized were top F1-Macro scores overall, but were not the first-place for the model.

A. Cross Validation Results

Cross Validation Mean F1 Macro results for all embedding-model combinations on are shown in Table 5. These were computed using 90% of the LOA dataset using the cross validation methods discussed in Section III.C. The top F1-Macro scores were found to be between 0.841 to 0.854. The TFIDF Unigram is the embedding that most consistently has the top cross validation F1-Macro score over all the models. Averaged Word2Vec is the next most consistent embeddings for top scores.

B. Test Results: Best Embeddings

The Final Test F1 Macro Results, shown in Table 6, were generated on the held-out test set which amounts to 10% of our total data. We used the hyper-parameters found by the best cross validation score. The top F1-Macro scores were found to be between 0.877 to 0.919.

The top embeddings are both TFIDF Unigram and TFIDF Unigram + Bigram consistently across the different models. For our task, we will choose TFIDF Unigram as the best performing embedding due to it being consistent with our top cross validation results. It is also the more efficient embedding during training time, as it has a length of 4447 compared to 23,482, the length of TFIDF Unigram + Bigram. The other embeddings–TFIDF Bigram, Averaged Word2Vec, and TFIDF-Average Word2Vec perform well on some models too, though not as consistently. All of our TFIDF and Word2Vec embeddings work well for our task. This may be due to these embeddings being fully trained on our dataset from scratch. Remember that our civil document classification will most probably rely on similar entity names, as explained in Section I. This makes sense why TFIDF works well as the top embeddings, since it's providing weighted frequencies on the terms in our documents to aid in the classification. Doc2Vec did not perform well despite its genesis from Word2Vec. It is possible that this is due to Doc2Vec's focus to general document topic. The other embeddings are either pre-trained or use transfer learning from the pre-trained embeddings, and don't perform well for our task. This may be due to the language of these documents being different from the original corpus used to train the pre-trained embeddings.

C. Test Results: Best Models

Based on the results in Table 6, we have observed that the models that perform the best and most consistently are SVM followed by Logistic Regression. KNN, Random Forest, and Naive Bayes also perform well with some embeddings, but not as consistently as SVM. So, we choose SVM as our top performing model.

None of our neural models perform as well as the non-neural models. This may be due to having many more parameters that need to be tuned. This is much harder for the models to tune well, since we have a small dataset. Interestingly, both the large language model embeddings–RoBERTa and TL RoBERTa–performed extremely poorly compared with the Neural Network models. While we were able to tune it well to the cross validation, it is still over-fitting on the cross validation. This can also be seen with the low F1-Macro scores we are getting on the test set. The F1-Macro scores for the train set are also high compared to test results, due to over-fitting, and can be seen in the Appendix.

	KNN	RF	SVM	LR	NB	MLP	FNN	CNN	LSTM
TFIDF Unigram	0.818	**0.824**	**0.846**	***0.841***	**0.837**	**0.854**	***0.843***	**0.835**	**0.847**
TFIDF Unigram + Bigram	0.831	0.813	***0.844***	0.829	0.823	0.828	0.837	0.837	0.838
TFIDF Bigram	0.809	0.782	0.813	0.822	0.812	0.807	0.814	0.824	0.804
Averaged GloVe	0.808	0.812	0.807	0.816	0.734	0.814	0.787	0.792	0.793
TFIDF-Averaged GloVe	0.799	0.801	0.793	0.800	0.703	0.801	0.800	0.799	0.806
Averaged Word2Vec	0.806	0.821	***0.843***	0.832	0.745	***0.845***	**0.847**	0.835	0.836
TFIDF-Averaged Word2Vec	0.813	0.823	0.836	0.825	0.724	0.836	0.833	0.835	0.840
Doc2Vec	0.813	0.788	0.814	0.799	0.741	0.792	0.808	0.807	0.810
RoBERTa	**0.853**	0.812	0.828	0.808	0.711	0.768	0.802	0.776	0.801
TL Averaged GloVe	0.808	0.806	0.826	0.815	0.697	0.812	0.833	0.828	0.829
TL TFIDF-Averaged GloVe	0.806	0.806	0.824	0.806	0.676	0.803	0.835	0.819	0.830
TL RoBERTa	0.804	0.824	0.826	**0.845**	0.699	0.761	0.792	0.810	0.806

Table 5 10-Fold Mean Cross Validation F1-Macro Scores

	KNN	RF	SVM	LR	NB	MLP	FNN	CNN	LSTM
TFIDF Unigram	**0.919**	0.75	***0.877***	**0.880**	0.860	**0.854**	0.714	0.840	0.788
TFIDF Unigram + Bigram	0.859	***0.878***	0.859	***0.879***	**0.879**	0.828	**0.853**	**0.853**	**0.833**
TFIDF Bigram	0.797	0.853	**0.878**	***0.879***	0.859	0.807	0.830	0.819	0.807
Averaged GloVe	0.818	0.814	0.816	0.778	0.740	0.814	0.407	0.359	0.569
TFIDF-Averaged GloVe	0.819	0.855	0.816	0.739	0.714	0.801	0.760	0.643	0.661
Averaged Word2Vec	0.838	0.816	***0.877***	0.800	0.797	0.845	0.699	0.754	0.756
TFIDF-Averaged Word2Vec	0.840	**0.898**	***0.877***	0.799	0.731	0.836	0.760	0.758	0.814
Doc2Vec	0.853	0.758	0.820	0.836	0.407	0.792	0.644	0.407	0.706
RoBERTa	0.756	0.740	0.838	0.820	0.720	0.359	0.359	0.359	0.359
TL Averaged GloVe	0.816	0.773	0.731	0.754	0.660	0.812	0.714	0.714	0.696
TL TFIDF-Averaged GloVe	0.699	0.773	0.811	0.731	0.700	0.803	0.773	0.660	0.760
TL RoBERTa	0.836	0.811	0.795	0.838	0.760	0.685	0.359	0.359	0.359

Table 6 Test F1-Macro Scores

D. Conclusion

Overall, choosing the TFIDF Unigram embedding with the SVM model works most consistently for this task. Bias from fine-tuning embeddings with SVM classification as the downstream task is a possible reason why SVM is performing the best. This is with the exception of TL RoBERTa, which utilized cross validation on Logistic Regression for the downstream task. More experimental work could be done on this by training the RoBERTa embedding-model combinations end-to-end.

V. Future work

This thorough survey of embedding methods has provided a solid foundation for future work that falls into two categories: 1) increasing the scope of the methods evaluated on this dataset and 2) using the results from this dataset on similar, or larger, aviation NLP tasks and datasets.

The first category includes additional methods such as RoBERTa with a text classification head, BerTopic, FastText, and low resource classification methodologies [17–19]. This will allow us to compare additional approaches to our existing methodologies.

Additionally, we are interested in trying the best performing methods found on this dataset for other aviation text classification examples. Within the LOA documents, in addition to classifying documents, we also have research goals to classify sentences or lines within the documents. This is a similar two-class problem with one relevant class and a non-relevant class to capture the remainder. From the work performed here, we would expect to be able to use TFIDF combined with SVM or Logistic Regression and obtain good results. This would be working with smaller text segments but the representations would be similar and seeing techniques leading to 0.87 F1-macro scores on this problem gives us high confidence of good performance on the sentence classification.

Similarly, we have been researching several speech-based (audio) datasets. After doing speech to text (transcription) on the audio data, we often would like to perform intent classification and slot filling. Intent classification is a multi-class classifier problem but in the same domain and with similar dataset sizes and limitations as our LOA work. The work reported in this paper once again suggests evaluating a document-based embedding approach with similar classification models but set up as a multi-class problem instead.

Acknowledgments

We are grateful for the data collection and model validation support provided by subject matter experts and other stakeholders affiliated with the FAA Office of NextGen.

Appendix

The F1 Macro scores over the full training data for the model, using the hyper-parameter results from the cross validation, are presented in Table 8.

Embeddings	size
TFIDF Unigram	4447
TFIDF Unigram + Bigram	23482
TFIDF Bigram	19011
Averaged GloVe	300
TFIDF-Averaged GloVe	300
Averaged Word2Vec	350
TFIDF-Averaged Word2Vec	350
Doc2Vec	300
RoBERTa	768
TL Averaged GloVe	300
TL TFIDF-Averaged GloVe	300
TL RoBERTa	768

Table 7 Embedding sizes

	KNN	**RF**	**SVM**	**LR**	**NB**	**MLP**	**FNN**	**CNN**	**LSTM**
TFIDF Unigram	0.993	0.989	0.989	0.932	0.941	0.941	0.896	0.905	0.867
TFIDF Unigram + Bigram	0.995	0.995	0.929	0.993	0.934	0.927	0.938	0.901	0.938
TFIDF Bigram	0.799	0.995	0.943	0.989	0.952	0.807	0.947	0.907	0.936
Averaged GloVe	0.995	0.995	0.882	0.878	0.748	0.814	0.462	0.354	0.522
TFIDF-Averaged GloVe	0.998	0.995	0.890	0.870	0.709	0.801	0.729	0.655	0.660
Averaged Word2Vec	0.995	0.995	0.927	0.882	0.767	0.845	0.812	0.803	0.787
TFIDF-Averaged Word2Vec	0.864	0.995	0.911	0.903	0.754	0.836	0.760	0.766	0.855
Doc2Vec	1.0	1.0	0.927	0.923	0.783	0.792	0.600	0.427	0.739
RoBERTa	0.829	0.993	0.872	0.912	0.742	0.354	0.354	0.354	0.354
TL Averaged GloVe	0.995	0.995	0.898	0.866	0.704	0.812	0.775	0.730	0.724
TL TFIDF-Averaged GloVe	0.995	0.995	0.934	0.875	0.694	0.804	0.786	0.725	0.810
TL RoBERTa	0.834	0.993	0.895	0.941	0.726	0.697	0.354	0.354	0.354

Table 8 Train F1-Macro Scores

References

[1] "Innovative Technology use in the Extraction of Flight Constraints recorded in Letters of Agreement (LOA)," *International Civil Aviation Organization Assembly — 41st Session*, August 2022. URL `https://www.icao.int/Meetings/a41/Documents/WP/wp_496_en.pdf`.

[2] Levy, O., and Goldberg, Y., "Linguistic Regularities in Sparse and Explicit Word Representations," *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, edited by R. Morante and S. W.-t. Yih, Association for Computational Linguistics, Ann Arbor, Michigan, 2014, pp. 171–180. https://doi.org/10.3115/v1/W14-1618.

[3] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V., "RoBERTa: A Robustly Optimized BERT Pretraining Approach," , 2019. https://doi.org/10.48550/arXiv.1907.11692.

[4] Sammut, C., and Webb, G. I. (eds.), *TF–IDF*, Springer US, Boston, MA, 2010, pp. 986–987. https://doi.org/10.1007/978-0-387-30164-8_832.

[5] Pennington, J., Socher, R., and Manning, C., "GloVe: Global Vectors for Word Representation," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Association for Computational Linguistics, Doha, Qatar, 2014, pp. 1532–1543. https://doi.org/10.3115/v1/D14-1162.

[6] Mikolov, T., Chen, K., Corrado, G., and Dean, J., "Efficient Estimation of Word Representations in Vector Space," , 2013.

[7] Le, Q. V., and Mikolov, T., "Distributed Representations of Sentences and Documents," , 2014. https://doi.org/10.48550/arXiv.1405.4053.

[8] Hosna, A., Merry, E., Gyalmo, J., Alom, Z., Aung, Z., and Azim, M., "Transfer learning: a friendly introduction," *Journal of Big Data*, Vol. 9, 2022. https://doi.org/10.1186/s40537-022-00652-w.

[9] Cover, T., and Hart, P., "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, Vol. 13, No. 1, 1967, pp. 21–27. https://doi.org/10.1109/TIT.1967.1053964.

[10] Breiman, L., "Random Forests," *Machine Learning*, Vol. 45, 2001, pp. 5–32. https://doi.org/10.1023/A:1010950718922.

[11] Manning, C. D., Raghavan, P., and Schütze, H., *Introduction to Information Retrieval*, Cambridge University Press, Cambridge, UK, 2008.

[12] Ng, A., and Jordan, M., "On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes," *Advances in Neural Information Processing Systems*, Vol. 14, MIT Press, 2001. URL https://proceedings.neurips.cc/paper_files/paper/2001/file/7b7a53e239400a13bd6be6c91c4f6c4e-Paper.pdf.

[13] Aggarwal, C., *An Introduction to Neural Networks*, Springer International Publishing, Cham, 2023, pp. 1–27. https://doi.org/10.1007/978-3-031-29642-0_1.

[14] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, Vol. 15, No. 56, 2014, pp. 1929–1958. URL http://jmlr.org/papers/v15/srivastava14a.html.

[15] Pauls, A. R., and Yoder, J. A., "Determining Optimum Drop-out Rate for Neural Networks," 2018. URL https://api.semanticscholar.org/CorpusID:247595430.

[16] Vajjala, S., Majumder, B., Gupta, A., and Surana, H., *Practical Natural Language Processing: A Comprehensive Guide to Building Real-world NLP Systems*, O'Reilly Media, 2020.

[17] Grootendorst, M., "BERTopic: Neural topic modeling with a class-based TF-IDF procedure," *arXiv preprint arXiv:2203.05794*, 2022. https://doi.org/10.48550/arXiv.2203.05794.

[18] Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A., "Advances in Pre-Training Distributed Word Representations," *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018. https://doi.org/10.48550/arXiv.1712.09405.

[19] Jiang, Z., Yang, M., Tsirlin, M., Tang, R., Dai, Y., and Lin, J., ""Low-Resource" Text Classification: A Parameter-Free Classification Method with Compressors," *Findings of the Association for Computational Linguistics: ACL 2023*, Association for Computational Linguistics, Toronto, Canada, 2023, pp. 6810–6828. https://doi.org/10.18653/v1/2023.findings-acl.426.