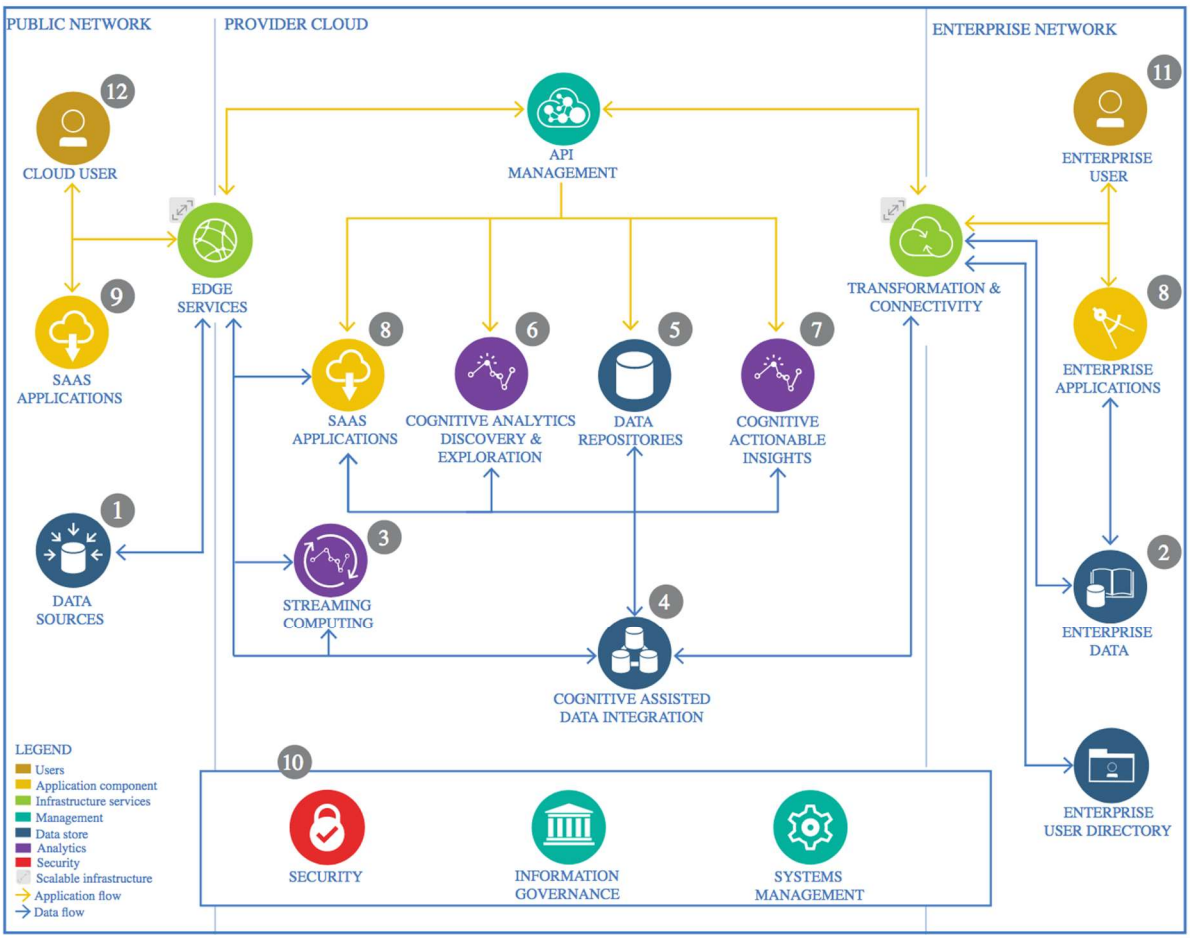
Identifying True vs. False News
Architectural Decisions Document

Objective: Identify news stories as true or false via text analysis using machine learning.

Repository: All files are found at our repository: https://github.com/adamx97/Data-Science-Advanced-Capstone

1 Architectural Components Overview



IBM Data and Analytics Reference Architecture. Source: IBM Corporation

1.1 Data Source

1.1.1 Technology Choice

Our data sources will be the following three data sources found in web repositories and provided for news analysis tasks like ours.

1) Buzzfeed dataset: this data contains 33 false news stories, with the articles covering a variety of topics. During our work, we discovered 3 Spanish language stories that we removed.
2) Rada Mihalcea dataset: (http://web.eecs.umich.edu/~mihalcea) has made a dataset available at http://web.eecs.umich.edu/~mihalcea/downloads.html#FakeNews This has two sets of files, one about news and another about celebrity gossip. News stories are found in six different topic areas: technology, education, business, sports, politics, and entertainment. They are sourced from mainstream news sites in the US such as ABCNews, CNN, USAToday, NewYorkTimes, FoxNews, Bloomberg, and CNET among others. Each item in the group of real stories has a matching fake news story created using the Mechanical Turk Amazon service. (This is a service that pays humans to do small piecework tasks, such as writing a fake news story that matches a true one.)
3) Snopes dataset: Snopes was one of the first online fact checking websites. It is rated high for Factual Reporting and to be in the center (Least Biased) of the political bias spectrum from http://mediabiasfactcheck.com This dataset consists of 312 news articles collected from Snopes classified as one of 5 values on a true/false spectrum (false, mostly false, mixed, mostly true, true) We will only use the 116 stories marked true or false. https://github.com/sfu-discourse-lab/Misinformation_detection/blob/master/snopes_checked_v02.csv.zip

1.1.2 Justification

These datasets provided clearly labelled true or false news stories and the full text of the stories was available. In arriving at these, we examined other datasets and considered using data from fact checking websites.

By combining the news stories from all three of the useable datasets, we obtain 1126 news stories, (555 true, 571 false) which should be enough to develop a model.

The other datasets were rejected due to a number of reasons: some were biased but not demonstrably false, some broadly categorized websites as unreliable, but didn't provide specific story labelling, and some of the other story datasets we found included links to the stories, not the full text. Due to age or other reasons, the full text of these stories was not reliably accessible.

We were not able to find a fact checking website that provided an API with links to the original stories. There is also an interesting ClaimReview website being created by several companies that provides an API, but the API concerns claims of fact, rather than the truthfulness of specific news stories.

A lot more detail on the rejected datasets and APIs investigated are found in the Data Exploration (https://github.com/adamx97/Data-Science-Advanced-

Capstone/blob/master/Data%20Exploration--Advanced%20Data%20Science%20Capstone.ipynb) and Data Cleansing notebook (https://github.com/adamx97/Data-Science-Advanced-Capstone/blob/master/Data%20Cleansing%20--%20Advanced%20Data%20Science%20Capstone.ipynb), as well as in the Prepare the Text section of https://github.com/adamx97/Data-Science-Advanced-Capstone/blob/master/NaiveBayesAndSVM.ipynb

1.2 Enterprise Data

1.2.1 Technology Choice

We are using widely available open source datasets.

1.2.2 Justification

The news stories we are analyzing are from internet sources and are not restricted to our enterprise.

1.3 Streaming analytics

1.3.1 Technology Choice

Not applicable.

1.3.2 Justification

We are performing a static text classification task and have no need for streaming analytics.

1.4 Data Integration

1.4.1 Technology Choice

Based on our dataset choices, we can use Panda Dataframes to do most of our data integration, cleansing and Feature Engineering tasks.

In some places we use the Natural Language Toolkit (http://www.nltk.org/) to do some of our language processing, tokenization, stemming, etc.

Using Python, pandas, and NLTK was the obvious easy choice for data cleansing. Manual inspection revealed 3 Spanish language stories, which we removed. If we had a greater volume of data, we could consider identifying based on detected language via an initial spoken language text classification task. If enough data was available, we could train separate models for each language.

Once we had collected our data into a single set, we were obliged measure a number of its attributes.

Table of news story counts:

	Snopes	Buzzfeed Top	Rada Mihalcea News	Rada Mihalcea Celebrity gossip	Total
Mixed:	72				
Mostly False:	53				
False:	51	33	240	250	**574**
					574
Mostly True:	72				
True:	65		240	250	**555**
					555
		Remove 3 Spanish language Buzzfeed			-3
				Total:	**1126**

Using Python to analyze our stories, we find:

Vocabulary	
Total words in Corpus	370,038
Total unique words	26,595
Count of words that appear once	11,910
Words appearing more than once	14,685

Stories	
Average word length	328.6
Std Deviation	624.9
Shortest story word count	27
Longest story word count	14722

Of the 26,595 words in our articles, we have 11,910 words used once, leaving 14,685 words found in more than one article. Any analysis that depends on finding the same word multiple articles will not find any of these 11,910 words, so from that standpoint they are just noise. However, embeddings that rely on analysis broader than word or ngram repetition, such as sentence structure, or by inferring parts of speech, will likely find meaning in words that appear once, so we will keep or remove them based on the vectorization we are using.

We went further when tokenizing sentences for word2vec parsing:

Average sentence count per story	15.90
Average tokens per sentence	24.00

(Code is found in the Feature Engineering Notebook: https://github.com/adamx97/Data-Science-Advanced-Capstone/blob/master/Feature%20Engineering--Adv.%20Data%20Science%20Capstone.ipynb)

1.4.1.1 Data Acquisition

We used the internet to search for useful news datasets. When they looked promising, they were often offered in CSV files for download. Using Excel it was easy enough to open the .csv files and see whether they were suitable.

In the case of the FakeNewsNet dataset (https://github.com/KaiDMML/FakeNewsNet) we used Python scripts to download the stories from the URLs provided. Many were missing, but even the ones that were still available were not suitable. Examination indicated that in many cases the story we downloaded from the URL provided had no relation to the original story, which was since removed or replaced. In some cases the text is that of the home page of the website, due to the original link being missing, and the website returning its home page instead. Others may have been removed due to age or removed when they were called out as false on factchecking sites. Regardless, without the original text to use as training data, this dataset was not useful. (More details on these issues in he Data Exploration Jupyter notebook.)

Once we had our useable datasets, some further data cleansing was needed which was done using Python and pandas dataframes.

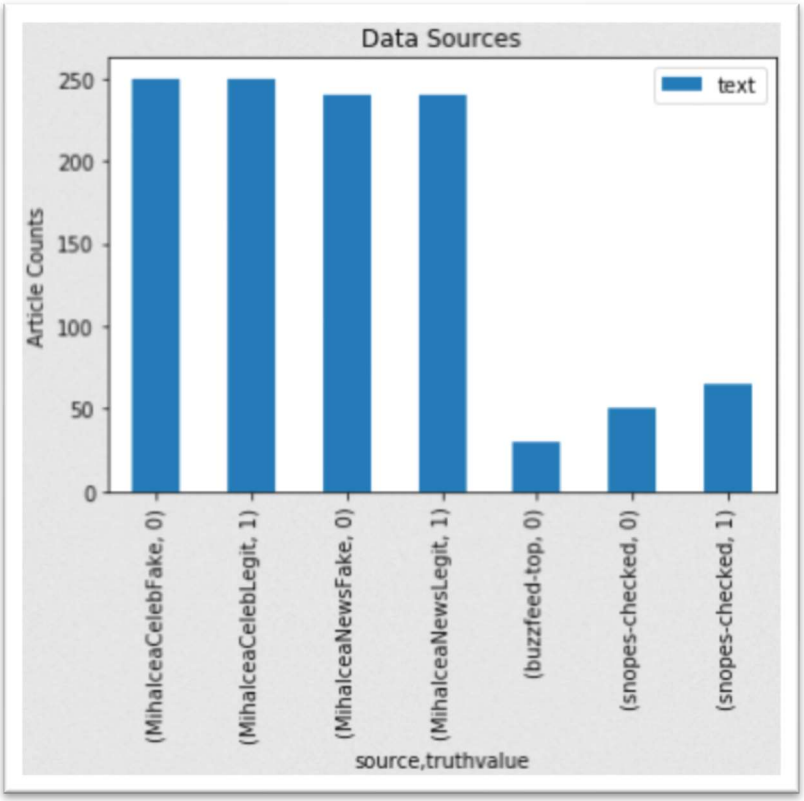
1.4.1.2 Data Cleansing

We needed to arrange our various data sources into a single DataFrame, ultimately settling on the following columns:

Column Name	**Description**
Index	A unique field identify the source and number of the original story.
Text	The original story text
Source	The source dataset
Truthvalue	Integer indicating that the story was True (1) or False (0)

Several steps were necessary to clean the data for best performance. Three story texts were in Spanish, these were removed. As the datasets were somewhat different to start with, each needed to be treated differently. These steps are shown in the Data Cleansing notebook: https://github.com/adamx97/Data-Science-Advanced-Capstone/blob/master/Data%20Cleansing%20--%20Advanced%20Data%20Science%20Capstone.ipynb

Ultimately this processing resulted in 1126 stories preserved in a Python dataframe pickle file that was reused throughout the project.



1.4.1.3 Feature Engineering

Feature engineering of the truthvalue was easily done by creating a suitable column and setting an integer value based on the directory/file name (Mihalcea data set) or based on other columns in the dataset (Snopes, Buzzfeed dataset).

(More information at the Data Cleansing notebook: https://github.com/adamx97/Data-Science-Advanced-Capstone/blob/master/Data%20Cleansing%20--%20Advanced%20Data%20Science%20Capstone.ipynb)

1.4.1.4 Encoding

Encoding types had a clear relationship to the accuracy of the model. There are a number of encoding and embedding types we tested, including ngrams, tf-idf vectorization and word2vec. There are others discussed in the literature, notably EXAM[1] which we didn't have time to test.

For the SVM and Naïve Bayes classification attempts, we used a bag of words style encoding, where training is based on vocabulary similarity between stories, with no regard to word ordering. For the bag of words encoding, we lowercased, tokenized into a set of words using NLTK's word_tokenize function, removed stop words, numeric values, and lemmatized them, and resulting in a vocabulary list simplified to a set of integers, one per vocabulary entry.

[1] https://arxiv.org/pdf/1811.09386.pdf

(Lemmatizing is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or *dictionary* form. This may result in resolving a word form to its most common form, often a singular noun, present form of a verb, for example: resolving the word “cacti” to “cactus,” “ran” to “run” and “better” to “good.”)

1.4.1.4.1 Text Vectorization with ngrams

In our initial convolutional 1d test, we tokenized using tensorflow’s Text Vectorization function. This function splits lowercases, strips punctuation, splits strings to words, mapped words to integer indexes, and transformed the sample into a vector of ints. Initially we used a maximum sequence length of 500, but after realizing our average story length was 328 words with a standard deviation of 624, we revisited this with a maximum sequence length of 960. Accuracy was 52.21%

Encoding with ngrams is the process of basing word mappings on 1 to N words. We used multiple ngram encodings but found no improvements with these. These are discussed in the results section.

In our second test with 2ngrams, we used the same TextVectorization layer, but this time the words were combined into pairs (2 ngrams) before being mapped to integers. This improved accuracy to 53.98%.

1.4.1.4.2 Text Vectorization with tf-idf

Our third test used the same TextVectorization processing, but output the vectors as floats using term frequency–inverse document frequency (tf-idf) is “a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.”[2]

> “The tf–idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. tf–idf is one of the most popular term-weighting schemes today. A survey conducted in 2015 showed that 83% of text-based recommender systems in digital libraries use tf–idf. Variations of the tf–idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query. tf–idf can be successfully used for stop-words filtering in various subject fields, including text summarization and classification.”[3]

Using tf-idf improved our accuracy to 57.96% after 3 epochs.

---

[2] https://en.wikipedia.org/wiki/Tf%E2%80%93idf

[3] Ibid.

These three tests are described in https://github.com/adamx97/Data-Science-Advanced-Capstone/blob/master/Model%20Definition2--Advanced%20Data%20Science%20Capstone.ipynb

1.4.1.4.3 Text Vectorization with gensim and word2vec

Finally, we used word2vec encoding, which encodes words on vectors based on word associations, attempting to derive meaning from those associations—word vectors with high cosine similarity are considered to be close in semantic similarity. Word associations are derived from the words nearby in the corpus text, learning context clues via unsupervised training. Vectors can be derived from any text--we experimented with training vectors based on our own relatively small corpus as well as pretrained vectors trained on much larger works: Stanford University's pre-trained word vectors based on 6 billion words (Wikipedia 2014 and Gigaword5--glove.6B.zip)[4]. It seems like this larger, more precise encoding should produce the best results, as the truth of a statement seems to rely as much or more on word meaning as word selection, which is the only criteria being used by the bag of words (ie 1-n ngrams) method.

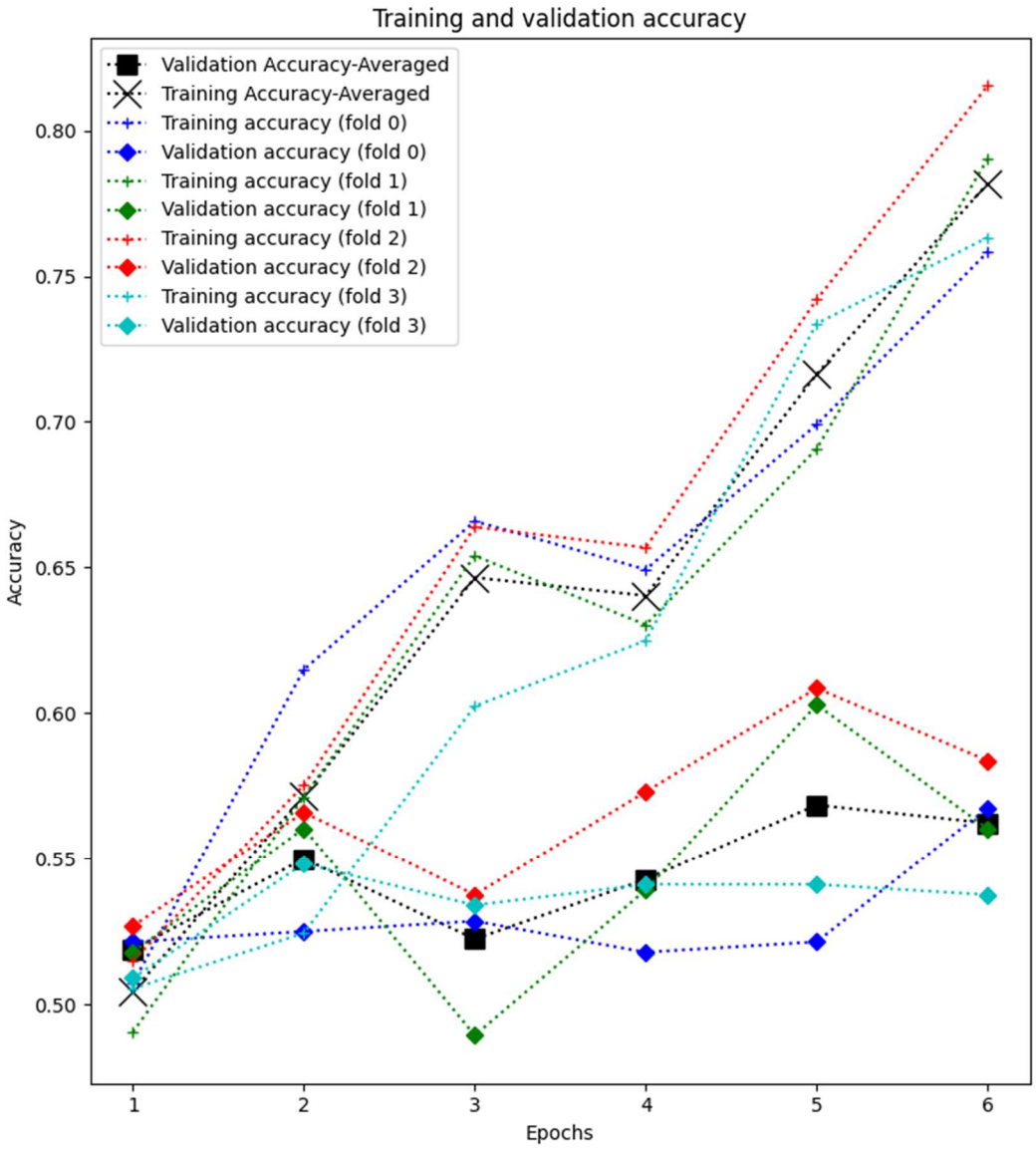
1.4.1.5 Methodology comparison: Folds vs Stratified Folds vs. a straight split

Using the Stanford University Glove pretrained Word2Vec embeddings with a deep learning model, we split out data with sklearn.Kfold and achieved as high as 75.86% accuracy on one fold. The KFold module split our data into folds, without randomization and not balanced between training classes.

[4] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

We revised our method with shuffling and used StratifiedKfold to create balanced training sets.

Figure 1 Results from deeplearningWithGlove300-20200923-1258.history



Using the Stratified Kfold method, our accuracy was more consistent between the folds, reached an average of 55% in Epoch 2, although the best fold achieved accuracy of 58.3%.

Upon further reflection, we realized that our test data of 1126 news stories was so minimal that we would be better off training on a greater amount of data, and the folds strategy was not called for in this case.

We settled on using sklearn.train_test_split to shuffle the data, again allowing it to stratify based on classes. This allowed us to train on 90% of the data and test on the remaining data. Here our accuracy peaked at Epoch 3 with 54.9%.

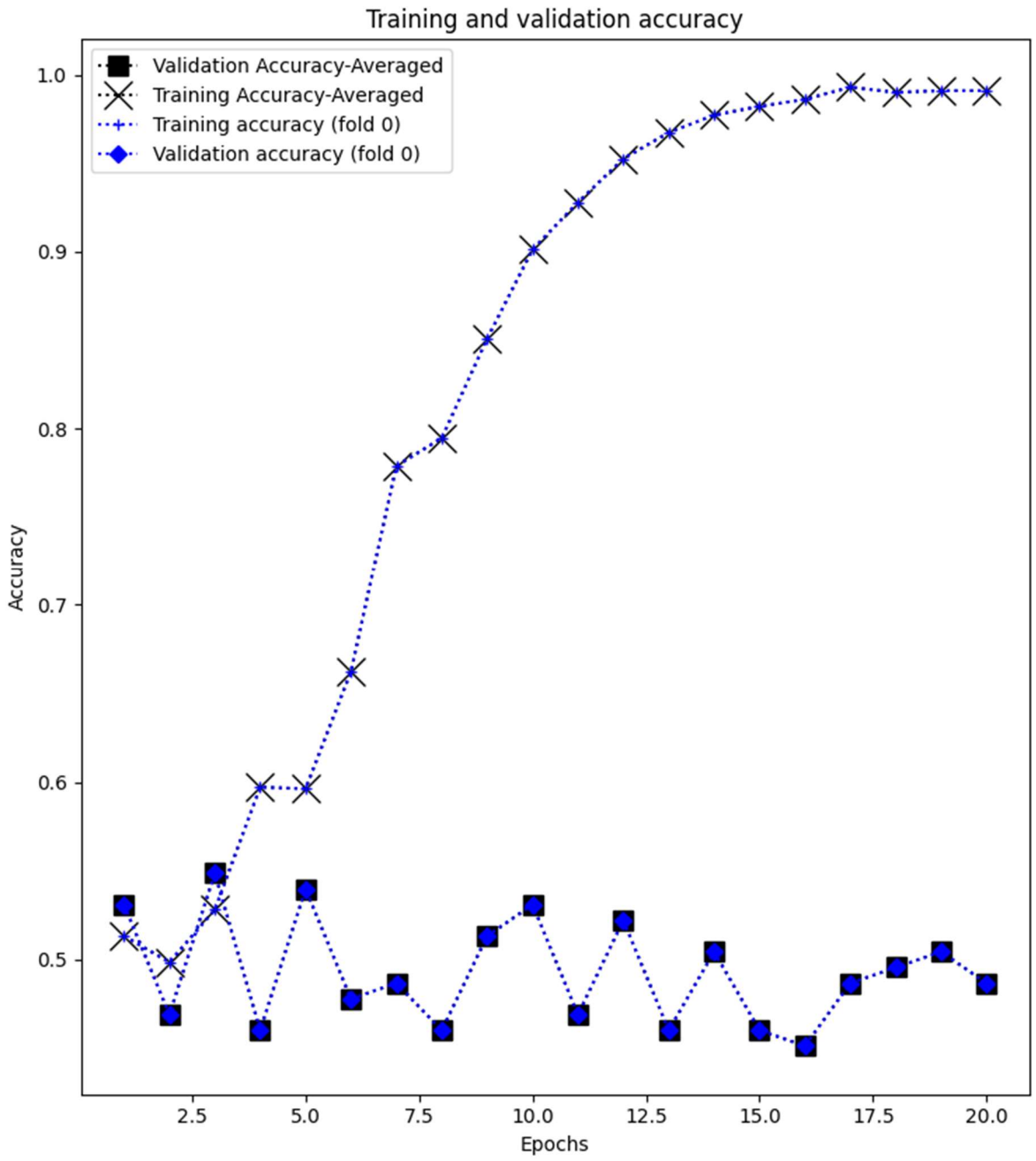


Figure 2 deeplearningWithGlove300_TrainTestSplit_-20200923-1409.history

1.4.2 Justification

We found that there are many options for TextVectorization: ngrams, tf-idf, word2vec, as well as others. Because we are essentially doing a sentiment analysis that is less dependent on word choice than possibly on word order, it seemed that word2vec could offer the best results. We tested with each of them.

We chose a test/train split method based on the one that resulted in us having the largest amount of training data with which to refine our model.

1.5 Data Repository

1.5.1 Technology Choice

Our training data is coming from static files that we have imported to object storage.

1.5.2 Justification

This provides an easy way to get to work. There is no need to store this data in any kind of database.

1.6 Discovery and Exploration

1.6.1 Technology Choice

We will use mostly IBM Watson Studio. When we use Tensorboard, we use Google's colab to create Jupyter notebooks and run our models because it doesn't easily run on Watson Studio. For longer running models, we are using our laptop to avoid running into cloud usage limits.

We use Tensorflow for our deep learning models, and scikit learn for the non-deep learning ones.

We tested with two non deep learning models and two deep learning models. Our Non-deep learning models were Naïve Bayes and Support Vector Machine. The deep learning models were densely connected networks and 1d convolutional models.

1.6.2 Models Used

1.6.2.1.1 Non Deep Learning: Naïve Bayes with 2ngrams

Leaving stop words in our encoding resulted in a 50% accuracy, equal to chance.

```
Accuracy: 0.5044
Precision: 0.5054
Recall: 0.5044
F1 Score: 0.5009

Model Classification report:
------------------------------
              precision    recall  f1-score   support

           1       0.50      0.59      0.54        56
           0       0.51      0.42      0.46        57

    accuracy                           0.50       113
   macro avg       0.51      0.51      0.50       113
weighted avg       0.51      0.50      0.50       113



Prediction Confusion Matrix:
------------------------------
          Predicted:
                   1   0
Actual: 1         33  23
        0         33  24
```

Removing stop words (not shown) reduced accuracy to 46%.

1.6.2.1.2 Non Deep Learning: Naïve Bayes with tf-idf

Taking a clue from Salvetti, Lowe, and Martin, we tried using Naïve Bayes.MultinomialNB() with the TfIdfVectorizer using our entire corpus vocabulary and allowing stop words produced an Accuracy of 46%.[5]

```
Accuracy: 0.4602
Precision: 0.4584
Recall: 0.4602
F1 Score: 0.4563

Model Classification report:
------------------------------
              precision    recall  f1-score   support

           1       0.45      0.38      0.41        56
           0       0.47      0.54      0.50        57

    accuracy                           0.46       113
   macro avg       0.46      0.46      0.46       113
weighted avg       0.46      0.46      0.46       113


Prediction Confusion Matrix:
------------------------------
          Predicted:
                   1   0
Actual: 1         21  35
        0         26  31
```

Removing common English words (stop words) prior to Tfidf vectorization did not improve training.[6]

1.6.2.1.3 Non-deep Learning: Support Vector Machine (SVM)

We created an SVM model and had our best results using no stop words and our entire vocabulary. SVM had an accuracy of 65.49%, amongst the best results we obtained.[7]

[5] https://github.com/adamx97/Data-Science-Advanced-Capstone/blob/master/NaiveBayesAndSVM.ipynb
[6] Ibid.
[7] https://github.com/adamx97/Data-Science-Advanced-Capstone/blob/master/NaiveBayesAndSVM.ipynb

Here, unlike Naïve Bayes, removing stop words had no improvements. Like Naïve Bayes, various ngram combinations lead to no improvements.

```
Accuracy: 0.5929
Precision: 0.6035
Recall: 0.5929
F1 Score: 0.5889

Model Classification report:
------------------------------
              precision    recall  f1-score   support

           1       0.64      0.49      0.56        59
           0       0.56      0.70      0.62        54

    accuracy                           0.59       113
   macro avg       0.60      0.60      0.59       113
weighted avg       0.60      0.59      0.59       113


Prediction Confusion Matrix:
------------------------------
          Predicted:
                   1   0
Actual: 1         29  30
        0         16  38
```

1.6.2.2 Deep Learning

We tested deep learning models with various embeddings. The challenge here is to determine the best combinations of layers, dropout, learning rates, bias, and normalization.

We also found a consistent tendency to overfit due to the small amount of our data. Most tests began to overfit within about 4 epochs.

1.6.2.2.1 Deep Learning Model: Convolutional 1d model with Basic Vectorization Bag of Words with 1 ngrams

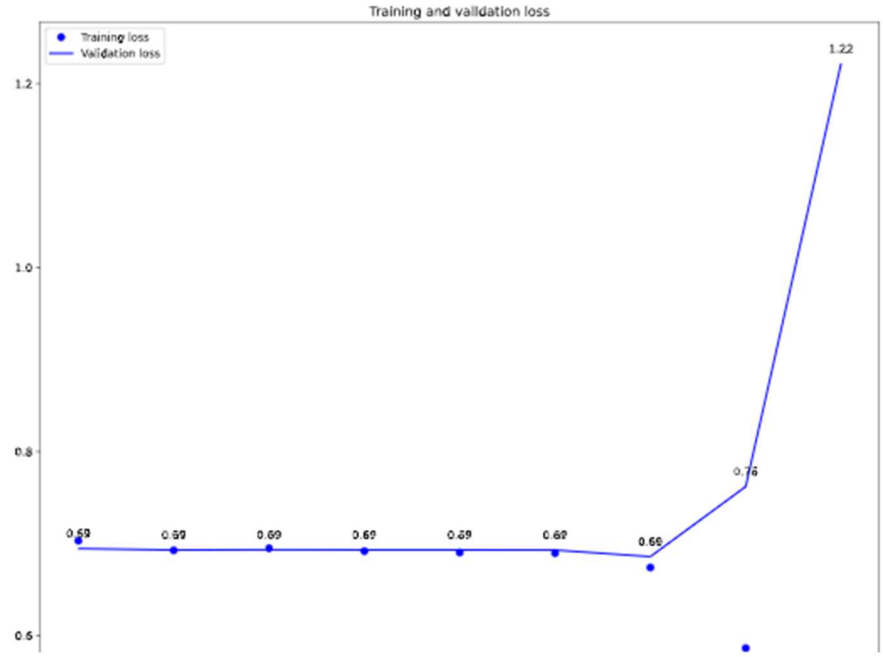
We started with a 1d convolution. Embedding was TextVectorization with the default 1 ngrams. Activation used was relu, which is commonly considered to be the recommended activation to start with. We used a stride of 1 to retain the maximum resolution in our convolution layer. Using a stride of 1 means that we pay maximal attention to each word and seems to improve accuracy, reaching 58% in epoch 7 before loss gets out of hand. Applying bias keeps the loss lower for more epochs. This is a binary classification task, so we use a sigmoid activation in the output layer.[8]

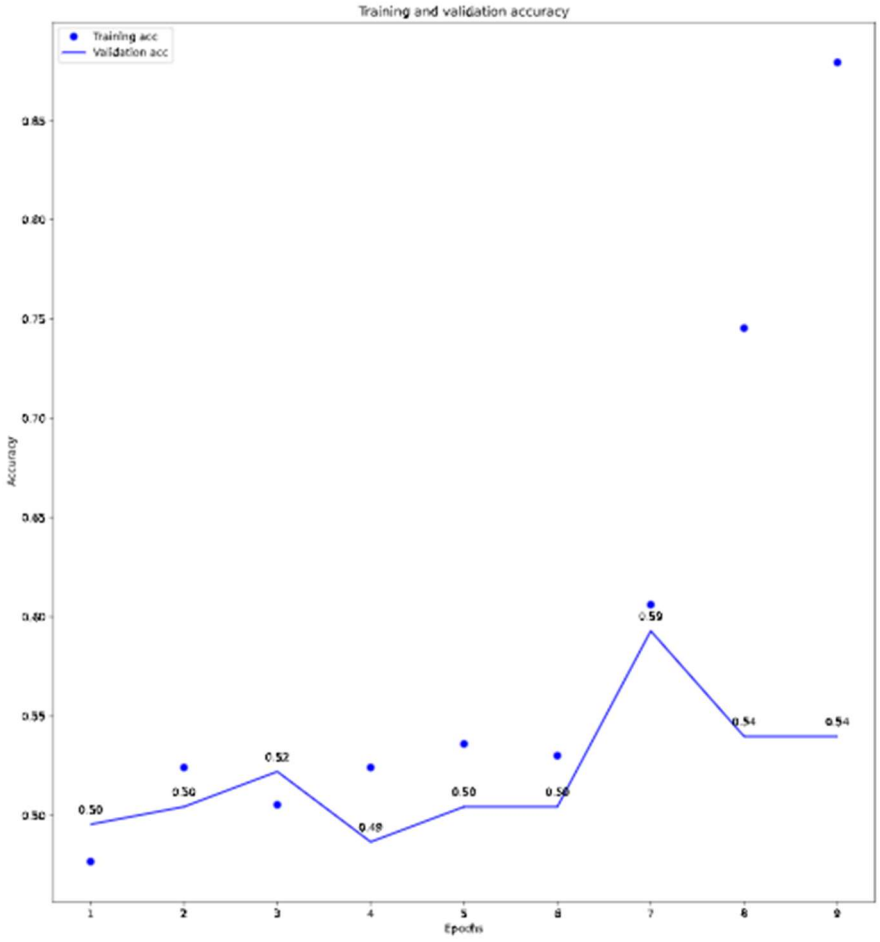
```
vectorize_layer1 = TextVectorization(max_tokens=max_features, output_mode='int',
    output_sequence_length=500)
```

[8] https://github.com/adamx97/Data-Science-Advanced-Capstone/blob/master/Model%20Definition2--Advanced%20Data%20Science%20Capstone.ipynb

```
vectorize_layer1.adapt(text_dataset.batch(64))
text_input = tf.keras.Input(shape=(1,), dtype=tf.string, name='text')
x = vectorize_layer1(text_input)
x = layers.Embedding(max_features + 1, embedding_dim)(x)
x = layers.Dropout(0.5)(x)

x = layers.Conv1D(128, 7, padding='valid', activation='relu', strides=1, use_bias=True,
    bias_initializer=tf.keras.initializers.GlorotNormal(seed=42))(x)

x = layers.Conv1D(128, 7, padding='valid', activation='relu', strides=1, use_bias=True,
    bias_initializer=tf.keras.initializers.GlorotNormal(seed=42))(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dense(128, activation='relu', kernel_initializer=tf.keras.initializers.GlorotNormal
(seed=42))(x)
x = layers.Dropout(0.5)(x)
predictions = layers.Dense(1, activation='sigmoid', name='predictions')(x)
model1 = tf.keras.Model(text_input, predictions)
```

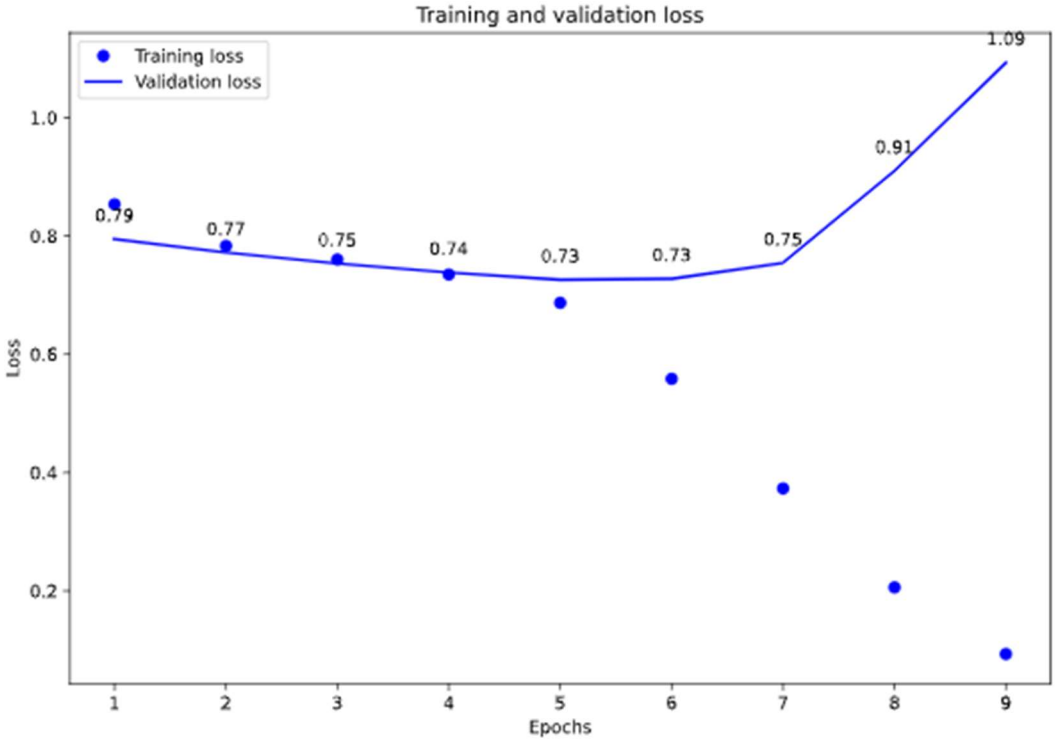
This model obtained a best accuracy of 59% in the seventh epoch, and then a decline due to overfitting.

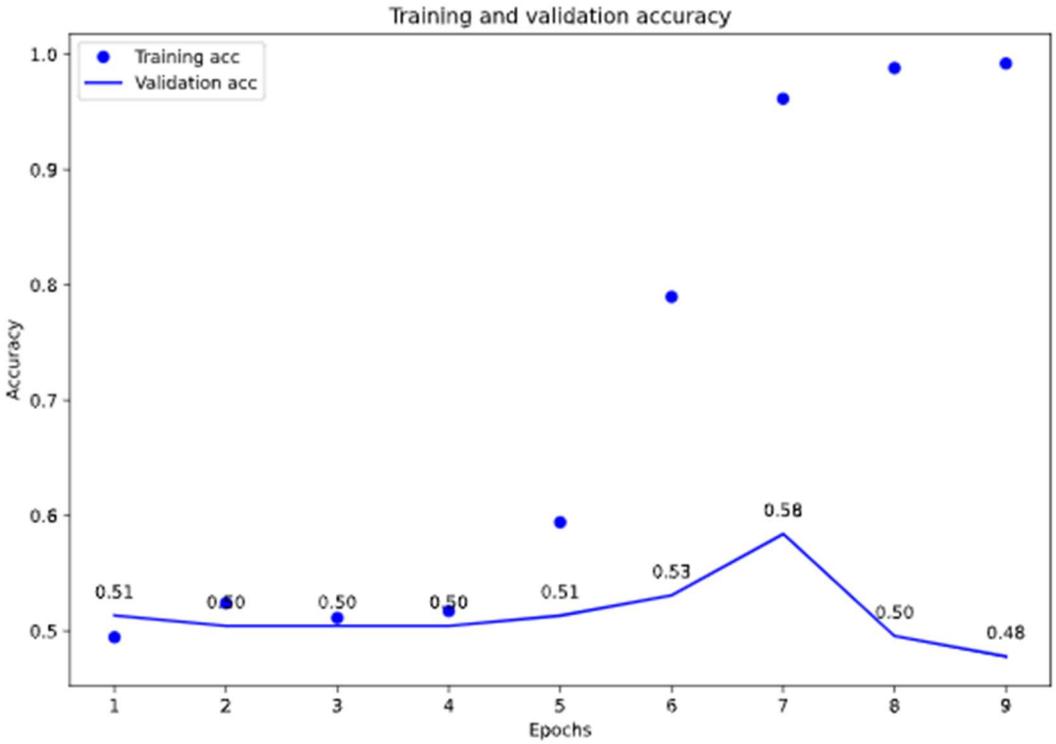




1.6.2.2.2 Deep Learning Model: Convolutional 1d model with, 2ngrams, with 1 stride and bias and regularization

We changed to 2ngrams. There was really no improvement between 1 and 2 ngrams.





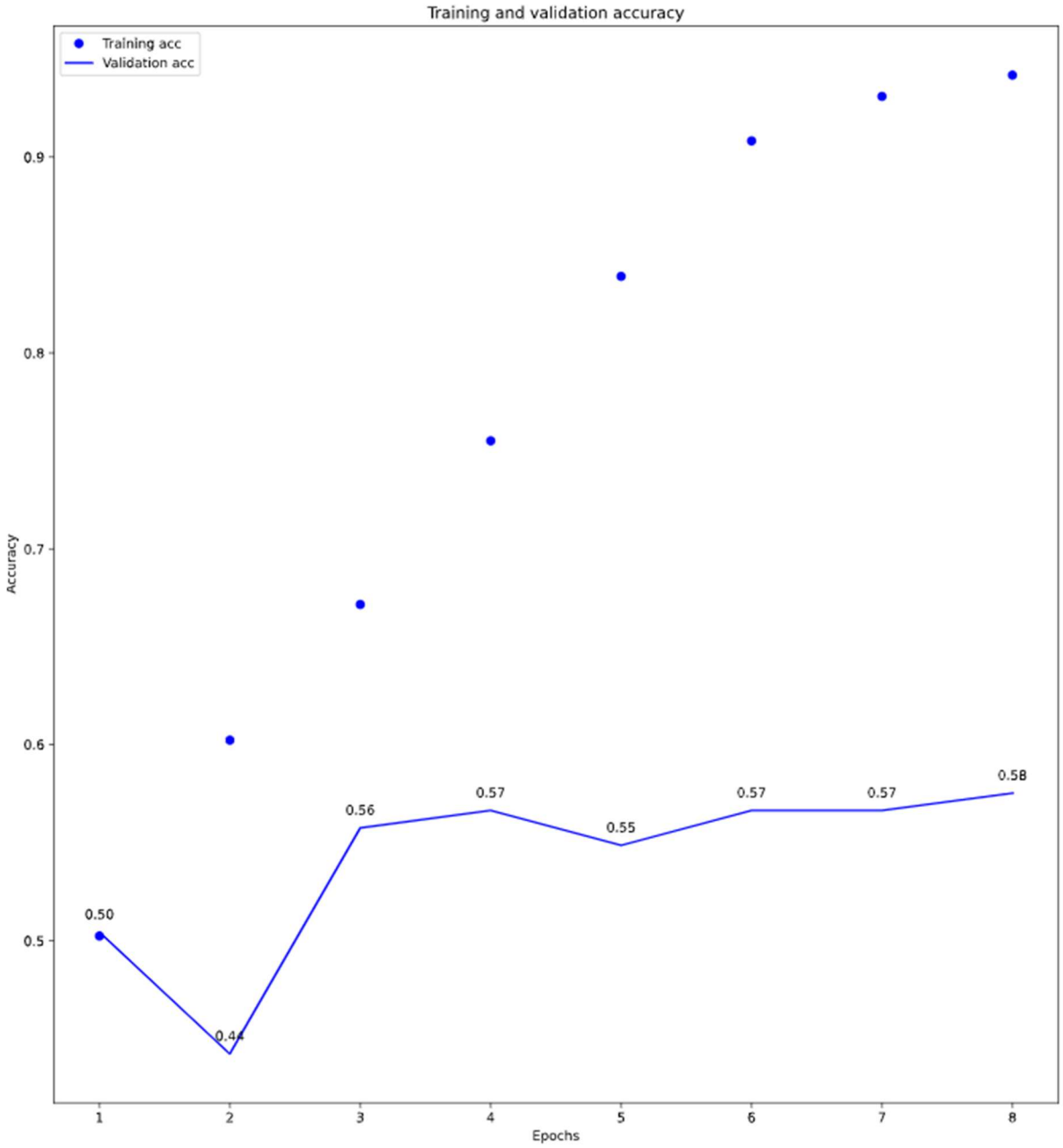
1.6.2.2.3 Deep Learning Model: Densely connected model with Tf-idf Vectorization with Bag of Words 2 ngrams [9]

As described above, tf-idf vectorization weights the word based on the frequency of their occurrences in the corpus. We tested with vocabulary size of 14745, the number of words found in more than one story. We continued to use bias to try to keep loss low.

```
vectorize_layer2 = TextVectorization(max_tokens=max_features, output_mode='tf-idf',
    ngrams=2   )
vectorize_layer2.adapt(text_dataset.batch(64))
text_input = tf.keras.Input(shape=(1,), dtype=tf.string, name='text')
x = vectorize_layer2(text_input)
x = layers.Dense(256, activation='relu', use_bias=True,
    bias_initializer=tf.keras.initializers.GlorotNormal(seed=42),
    kernel_initializer=tf.keras.initializers.GlorotNormal(seed=42))(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(256, activation='relu', use_bias=True,
    bias_initializer=tf.keras.initializers.GlorotNormal(seed=42),
    kernel_initializer=tf.keras.initializers.GlorotNormal(seed=42))(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(256, activation='relu', use_bias=True,
    bias_initializer=tf.keras.initializers.GlorotNormal(seed=42),
    kernel_initializer=tf.keras.initializers.GlorotNormal(seed=42))(x)
x = layers.Dropout(0.5)(x)
predictions = layers.Dense(1, activation='sigmoid', name='predictions')(x)
model2 = tf.keras.Model(text_input, predictions)
```

[9] https://github.com/adamx97/Data-Science-Advanced-Capstone/blob/4552518eb435928f9ded078b61e724856f93975b/Model%20Definition2--Advanced%20Data%20Science%20Capstone.ipynb

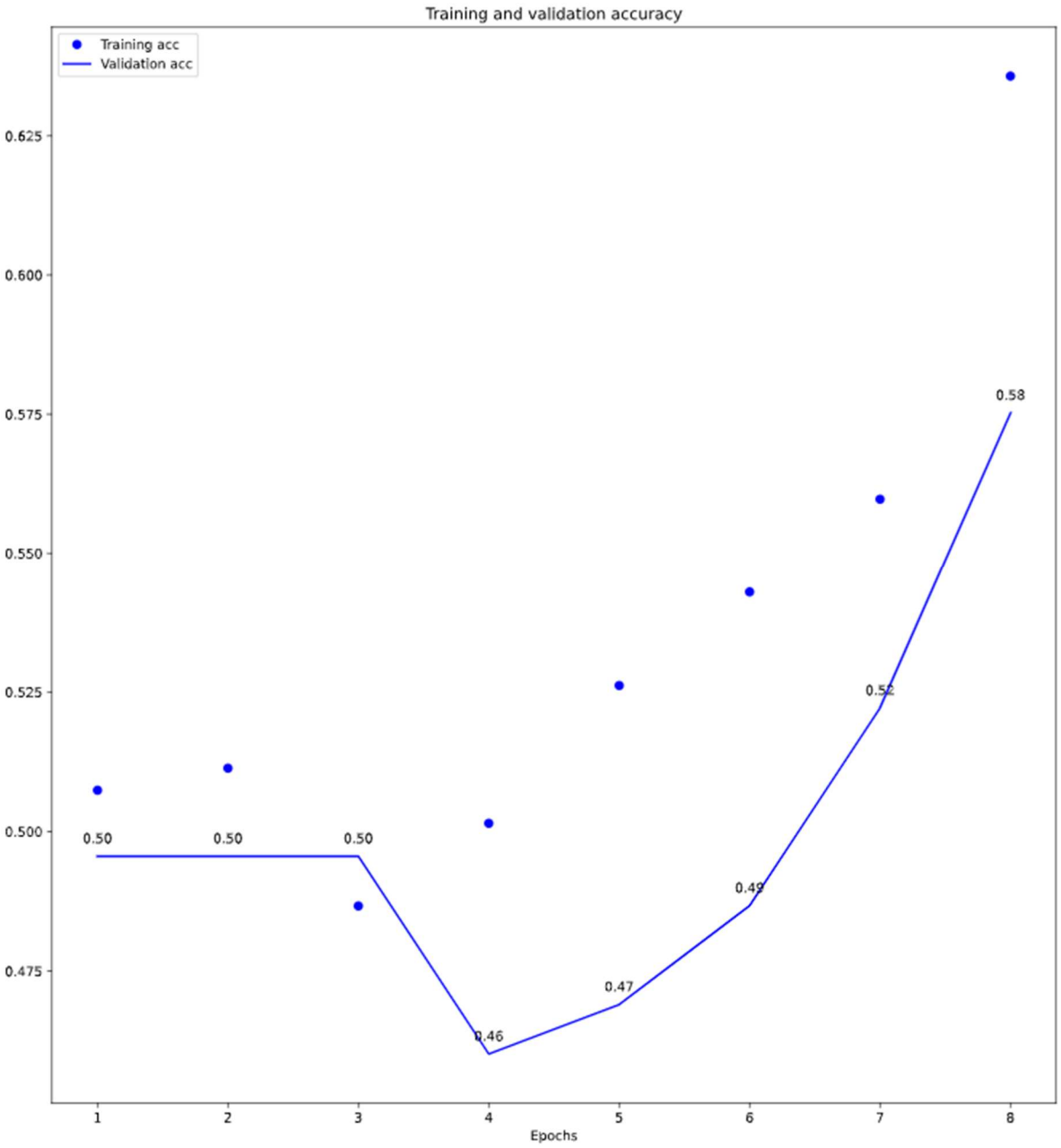
As described above, tf-idf vectorization weights the word based on the frequency of their occurrences in the corpus. We tested with vocabulary size of 26595, the total number of words in all stories. This produces accuracy of 56% by the third epoch. [10]



[10] https://github.com/adamx97/Data-Science-Advanced-Capstone/blob/4552518eb435928f9ded078b61e724856f93975b/Model%20Definition2--Advanced%20Data%20Science%20Capstone.ipynb

1.6.2.2.4 Deep Learning Model 6: Convolutional 1d model with Tf-idf Vectorization with Bag of Words 2 ngrams [11]

We combined deep in learning model used model in Models 1, 2, and 3, with the Tf-idf vectorization used in Models 4 and 5 and obtained good results.



[11] https://github.com/adamx97/Data-Science-Advanced-Capstone/blob/4552518eb435928f9ded078b61e724856f93975b/Model%20Definition2--Advanced%20Data%20Science%20Capstone.ipynb

1.6.2.2.5 Justification

IBM Watson Studio is free for our use and allows us to scale up our compute power if needed. It allows us to use compute and storage infrastructure at scale without having to spend too much time on logistics, thus allowing us to focus on the problem at hand.

Tensorflow is an optimized implementation of Keras will allow us to build a neural network quickly, is free and open source, and is one of the standard for tasks like ours. Colab offers optimized hardware options.

1.7 Actionable Insights

1.7.1 Evaluation metric used

We used accuracy as the evaluation metric, since accuracy is the most intuitive metric for binary classification tasks.

We used binary crossentropy as our loss function.

1.7.2 Discussion

Detecting the veracity of a text is a notoriously difficult problem.

As Salvetti and Lowe state:

> It is important to point out that unlike other Artificial Intelligence tasks, this is a task for which it is known … that humans—even trained law enforcement agents—perform at chance or even below chance."[12]

This is a difficult task, especially with the limited number of news stories. On some of our models we approached 60% accuracy, which is only 10% better than the 50% accuracy we would have achieved by chance.

1.7.2.1 Dataset size observation

Our dataset is far too small to obtain good results. The authors of number of the datasets we used commented on the difficulty of obtaining clear true and false news stories to use. As Asr and Taboada state, "The most important challenge in automatic misinformation detection using modern NLP techniques, especially at the level of full news articles, is [obtaining] data."[13]

It is a laborious task to create datasets with news stories that have been fully checked fact checked. Checking news for fact vs. fiction is not a case of mass production. There are a number of fact checking websites (Snopes, Politifact, ClaimReview) that have been established

---

[12] Salvetti, Franco, et al. "A Tangled Web: The Faint Signals of Deception in Text - Boulder Lies and Truth Corpus (BLT-C)." *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, European Language Resources Association (ELRA), May 2016, www.aclweb.org/anthology/L16-1558/.

[13] Torabi Asr, M. (2018). The Data Challenge in Misinformation Detection: Source Reputation vs. Content Veracity. In *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)* (pp. 10–15). Association for Computational Linguistics.

to do this, and the time and effort expended in each story's analysis is clear. Similarly, collecting stories that have been checked is not an easy matter. None of the websites mentioned provide an API that would allow for easy data analysis.

1.7.2.2 Difficulty of the task

Detecting truth vs. false information is a very difficult task, as is made clear by the Salvetti and Lowe quote above. Some of our models obtained rea

Comparison with similar published results:

In the published paper "A Tangled Web: The Faint Signals of Deception in Text Boulder Lies and Truth Corpus (BLT-C)" [14], Salvetti, Lowe, and Martin describe the creation of a corpora for use in detecting deception in text, and describe the differences between true, false, and deceptive text. The paper contains a good description of the challenges met by both machines and people in identifying truth vs. fiction vs. deception. To validate their corpora, they use a bag of words feature space, and use Naïve Bayes, Naïve Bayes Multinomial, and an open source version of the C4.5 Decision Tree models. Their paper contains results of their validation, parts of which are in included in the table below.

In "FakeNewsTracker: A Tool for Fake News Collection, Detection, and Visualization," Kai Shu, Deepak Mahudeswaran, and Huan Liu describe a similar task, attempting to identify news from Politifact and Buzzfeed with intentionally false information, using both the published text of the articles and social media mentions of the article (Tweets, retweets, and replies. They use a LSTM deep learning model and encode their textual data using word2vec. They merge the results of the text analysis with their Tweet analysis to come up with a final score of truth vs. fiction, finding the best results when using both the text and the Tweets, but they present the results both separately and in the aggregate.[15]

[14] https://www.aclweb.org/anthology/L16-1558.pdf

[15] Shu, K., Mahudeswaran, D. & Liu, H. FakeNewsTracker: a tool for fake news collection, detection, and visualization. *Comput Math Organ Theory* **25,** 60–71 (2019). https://doi.org/10.1007/s10588-018-09280-3

1.7.3 Conclusions

Our overall results are summarized in the table below, along with published results on similar tasks.

	Embedding	Accuracy
Published Results		
Salvetti, Lowe, Martin		
True vs False (Naïve Bayes)	Bag of Words	51.14%
True vs False (Naïve Bayes Multinomial)	Bag of Words	42.71%
Shu, Mahudeswaran, Liu		
Politifact	Word2vec	63.3%
BuzzFeed	Word2vec	62.3%
Our results:		
Model: Naïve Bayes	Bag of Words 2 ngram	50%
Model: Naïve Bayes – 2 ngrams	Bag of Words Tf-idf	46%
Support Vector Machine	Bag of Words Tf-idf	59%
Deep Learning:		
Convolutional 1d	Bag of Words 1 ngram	59%
Convolutional 1d	Bag of Words 2 ngram	58%
Densely connected	Bag of Words tf-idf 2ngrams	54%
Convolutional 1d	Bag of Words tf-idf 2ngrams	58%

The above shows an accuracy range of 42.71% - 63.3%, similar to our 60% number.

Our best results were obtained using a Support Vector Machine with rates of 65% accuracy, and secondly our convolutional model.

1.8 Applications / Data Products

1.8.1 Technology Choice

This project leads to possible applications in identifying false vs true news. I think it is not ready for production use until it has been trained and tested on more data.

1.8.2 Justification

Credibiility is everything when it comes to news. Any tool that claims to identify true vs. false news could easily lose credibility in the public's eye if its accuracy is questioned. Accuracy rates of 2/3 at best are not sufficient to put forward in a public forum.

1.9 Security, Information Governance and Systems Management

1.9.1 Technology Choice

Not needed

1.9.2 Justification

As this is just an academic exercise, deployed on a small scale in a laboratory environment, we are not concerned with security issues.

1.10 Appendix: Notebooks and Code

1.10.1 All relevant code is checked into Github at https://github.com/adamx97/Data-Science-Advanced-Capstone Some of the key files are listed below relative to this route.

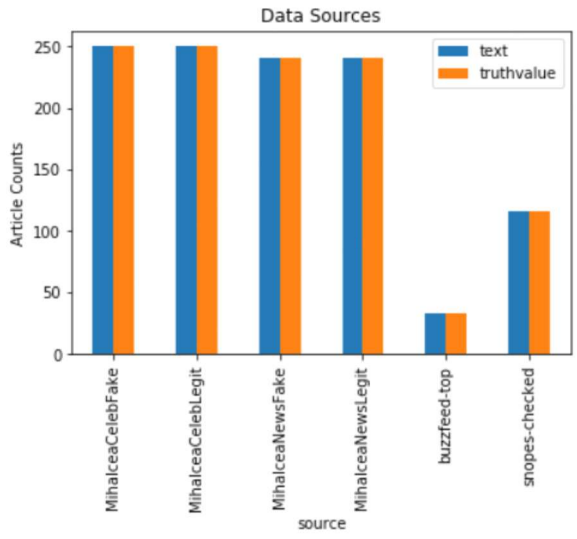
1.10.1.1 README.md

A general page of notes and pointers to relevant articles.

1.10.1.2 Data Exploration--Advanced Data Science Capstone.ipynb

Contains discussion of various datasets and pros and cons, and initial analysis

```
In [9]: download_file_cos(credentials,'dfTrueFalseNews.pkl','dfTrueFalseNews.pkl')

        File Downloaded

In [15]: dfTrueFalseNews = pd.read_pickle('dfTrueFalseNews.pkl')
         print (dfTrueFalseNews.columns, dfTrueFalseNews.shape)

         Index(['text', 'source', 'truthvalue'], dtype='object') (1129, 3)

In [17]: dfTrueFalseNews.groupby('source').count().plot(kind='bar')
         plt.title("Data Sources")
         plt.ylabel("Article Counts")

Out[17]: Text(0, 0.5, 'Article Counts')
```



1.10.1.3 Data Cleansing -- Advanced Data Science Capstone.ipynb

Contains data cleansing work needed to impute truthvalue from datasets. Code combines data from 3 sources into a single dataframe, and removes three Spanish language stories.

1.10.1.3.1 NewsDataUnifier.py

Python script related to above, brings three datasets into a single dataframe.

1.10.1.4 Feature Engineering--Adv. Data Science Capstone.ipynb

Determine optimum vocabulary size based on word occurrences. Identify words that only appear once in the corpus.

Vocabulary:

Vocabulary size: 26,595

Words that only appear once: 11,910

Identify story lengths:

Average story length: 328.6305506216696

Minimum story length: 27

Maximum story length: 14722

Standard deviation: 624.980898772334

Total words in corpus: 370038

Vocab size: 26595

Longest story: celeb060legit

Shortest story: celeb131fake

1.10.1.5 NaiveBayesAndSVM.ipynb

Contains two non-deep learning models: Naïve Bayes and Support Vector Machine for binary text classification.

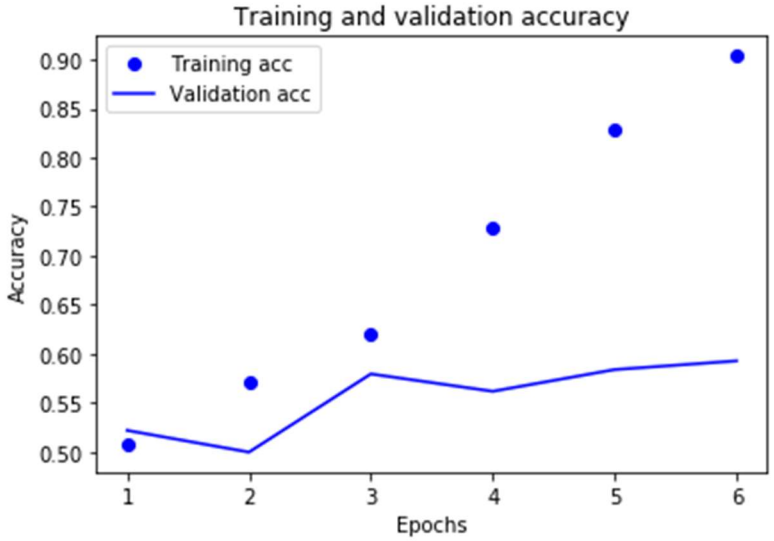
1.10.1.6 Model Definition2--Advanced Data Science Capstone.ipynb

Contains first deep learning models with both ngram and tf-idf TextVectorization

Model 1: Convolutional 1 D model

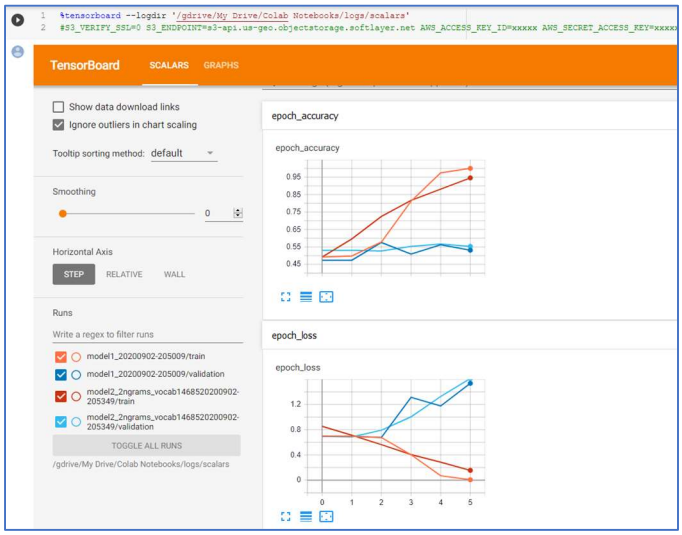
Model 2: Densely connected model.

Has a number of tests of other models with various layers.



1.10.1.7 FW_1_Add_Tensorboard_Colab_Version.ipynb

As Tensorboard didn't work on IBM Watson studio, we moved to Google's Colab notebooks for this effort.



1.10.1.8 FW_2_HyperparamTuning_Colab_Version.ipynb

We wanted to use Tensorboard to tune hyperparameters, but found it didn't work in IBM Watson Studio. We used Google's Colab facility instead.