capstone_report.md

Machine Learning Engineer Nanodegree

Capstone Project

Daniel Brandt July, 10 2019

Table of contents generated with markdown-toc

I. Definition

Project Overview

Background

Automation of grading multiple choice exams is trivial. For many reasons, however, short answer tests provide a better tool for helping students assess their knowledge. So far, this area remains a challenge and machine learning has not yet provided usable solutions. Several papers have discussed both algorithmic natural language solutions as well as deep learning approaches. None of the approaches so far have provided a general purpose solution to the many challenges faced. These challenges are in some ways very similar to some of the most important AI projects of our time: Intelligent Assistants like Apple Siri, Amazon Echo and the Google Assistant. Given a very broad set of possible subjects, any given question can be phrased in innumerable ways. How do you accurately interpret what the essential question is?

For short answer tests, a given question can be answered correctly in innumerable ways as well. Like with voice assistants, the quality of the response in terms of grammar, spelling, typos and word choice are extremely variable. What makes the short answer grading possibly more challenging than large scale intelligent assistants is that each individual test might have very limited responses and may not be in use for that long.

Because short answer tests are very tedious to grade, the success in automating related areas such as identifying plagiarism and essay grading has led to renewed efforts to attack this area. The problem is challenging because of the need to focus on identifying correctness in relatively short answers. Longer answers and essays require a broader set of criteria like grammar, ideas, structure and therefore provide more features for machine learning to work with.

If we had enough short answer test results graded by a human for a particular test, it would be relatively easy to model and grade a short answer test. Unfortunately, exhaustive student answer datasets rarely exist in the real world for a specific short answer test. The reality is that tests generally have to be refreshed regularly to reflect constantly changing content and, in some cases, to prevent unwanted distribution and cheating.

1. Some of the Challenges of grading short answer questions include: *2

- Teachers usually find the task of assessing respondents' answers very time-consuming.
- Students may have to wait for a long time to receive feedback on their responses
- When they finally get it, the grade can be different from another classmate's, who has given a very similar answer.

2. The challenges of grading short answer (SAS) compared to essays (AES) are: *3

- Response length. Responses in SAS tasks are typically shorter. For example, while the ASAP-AES data contains essays that average between about 100 and 600 tokens (Shermis, 2014), short answer scoring datasets may have average answer lengths of just several words (Basu et al., 2013) to almost 60 words (Shermis, 2015).
- Rubrics focus on content only in SAS vs. broader writing quality in AES.
- Purpose and genre. AES tasks cover persuasive, narrative, and source-dependent reading comprehension and English Language Arts (ELA), while SAS tasks tend to be from science, math, and ELA reading comprehension.

3. Defining correct: *1

- From my analysis, I believe one of the most difficult challenges is defining what correct actually means. There is no absolute measure of correctness for this kind of test, rather, we can only look at how human graders make the judgement. Human graders tend to look for key words and pattern matching rather than the specific order of words when grading large numbers of questions.
    - The number of correctly used words has more influence on marks than semantics or order of words.
    - If a large number of responses are being graded, it is not unreasonable that a human would move towards pattern recognition via key words rather than "reading for meaning".
    - Identifying words gives an idea about grades and students misunderstanding to teachers. Such an approach allows time saving for scoring, and to provide rapid feedback to students by checking the words used from model vocabulary.

The ideal case would be a solution that can use a subset of answers by the first batch of students that are manually graded to then automatically grade subsequent tests with the same questions. The goal of this project is to test that approach on some datasets with baseline results from academia to gain a better understanding of the problem and the build a baseline of code to iterate in future projects as our understanding grows and technology improves towards better solutions.

A complete solution is not the only valuable outcome. Machine learning can still add a lot of value today. While complete grading may not be possible, automation of some of the answers allowing the grader to focus on a smaller subset could also be a win. This exercise may also help design short answer tests to make them easer to automatically grade. For example, a recent paper recommended that it be used to supplement the quality, provide automation in some areas of grading and help target areas where more human involvement is needed. *1

This project originated from my interest in both memory and how memory impacts learning. We employ two types of memory while learning: recognition and recall. A multiple-choice test involves a larger percentage of recognition since the correct answer is usually provided and can be recognized. Short answer questions on the other hand require a more intense form of memory where, given no recognition prompt, an answer must be recalled. Another benefit of short answer over multiple choice questions is that they are often easer to write. They are easier to write because an educator must generate multiple answers for a multiple-choice exam some of which must be tricky to distinguish from the correct answer.

Overview

The purpose of this project is to validate approaches to grading short answer tests using machine learning models. The problem is to determine if a student answer is correct or incorrect based on a model that performs a binary classification or linear regresssion approach. In either case, the output of the model prediction is a probability that an answer is correct or incorrect on a scale of 0 to 1.

The first step in the project is to replicate results of a deep learning LSTM approach (Long Short Term Memory) with embedding from a recent state of the art research paper by Riordan referenced below (*3). Each student answer is represented as a vector of numbers, where an integer is uniquely mapped to a word based on a vocabulary of all words in the dataset of answers. Embedding referes to a step where the word vectors are used by the model to create a set of output vectors, one per word, where related words are represented by vectors closer together than unrelated words. This allows the model to handle simiilar words being used to represent the same correct answer. The model can do the embedding at the time its run or pretrained embeddings can be used which can be downloaded from other sources. Pretrained embeddings represent the relationship of words based on a large set of training data that might not be available with a particular problem. The output of the model is the probability between 0 and 1 that the answer is correct.

The results of state of the art machine learning approaches currently show moderate but limited success automating short answer grading. Most reaults are in the range of 70-75% accuracy. Something close to 95-100% accuracy is needed. To the extent it cannot be achieved, the prediction must error toward precision, which means avoiding false positive (grading incorrect answer as correct) or Type 1 errors in favor of Type 2 errors . Answers with a false negative, correct answers marked incorrect, are easier for a manual grader to review and then to credit the students test score. Answers graded correct that are incorrect will not likely be reported by students.

To cover a variety of short answer data challenges, two different dataset were chosen from different research papers:

1. SciEntsBank (SEB) dataset. The dataset consists of science assessment questions with 2-way labels (correct/incorrect). *3 (Reordan)

2. Short Answer Grading (SAG): These assignments exams were assigned to an introductory computer science class at the University of North Texas and collected via an online learning environment. *1 (Suzen)

This project leverages both local computer testing as well as leveraging the AWS Sagemaker enviornment in order to uses Hypertuning. Hypertuning leverage the power of cloud computing to run a large number of concurrent models at once and find the optimal set of model configurations (tunning parameters) that result in the best predictions.

This project also compares the state-of-the-art deep learning model, LSTM, to a much simpler but powerful machine learning model, XGBoost.

The results will show that neither approach comes near to the desired accuracy. That is to say, neither apporach is currrently adequate to automate short answer grading. However, a deep learning model can be used to partially automate short answer grading and this addresses one major goal of this project which is to better understand the limitations and strenghts of the deep learning approach.

The results also show that future testing should focus on identifying the kinds of questions and answers that perform better by analyzing the results on a question basis.

Pretrained data was not of much use in this project due to user input errors. Future tests should be done with more predecitable short answer questions without typos and spelling issues. This could be accomplished by adding upfront algorithms to correct spelling and typos. Such preprocessing would make the pretrained embedding more useful.

Finally, it is possible that the machine learning could be used to identify questions that can be autograded while flaging questions or specific answers for human grading if they fall in a range of probabilities that are not as conclusive.

References

- **1** Suzen, Neslihan & Gorban, Alexander & Levesley, Jeremy & Mirkes, Evgeny. (2019). Automatic Short Answer Grading and Feedback Using Text Mining Methods. Page 1, 19.
- **2** Galhardi, Lucas & Brancher, Jacques. (2018). Machine Learning Approach for Automatic Short Answer Grading: A Systematic Review: 16th Ibero-American Conference on AI, Trujillo, Peru, November 13-16, 2018, Proceedings. 10.1007/978-3-030-03928-8_31. Page 380
- **3** Brian Riordan, Andrea Horbach, Aoife Cahill, Torsten Zesch, and Chong Min Lee. 2017. Investigating neural architectures for short answer scoring. In Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications. pages 159–168. Page 159

Problem Statement

Goal

Create a machine learning model and the supporting code to perform grading of an arbitrary short answer test, where each answer is graded as correct or incorrect, given a limited set of actual test results.

For this project I will focus on reproducing some results from the literature and implementing an approach with Sagemaker and a custom Sklearn deep learnign LSTM model. For comparison a basic XGBoost model will also be tested with the same data.

The problem will be broken into two pieces.

First, I will try to replicate the results from the Riordan (*3) paper referenced above using a baseline LSTM model and then apply hypertuning in SageMaker to optimize. The Riordan paper described various layers and techniques as part of the larger LSTM model. They identify improvements to a basic model such as pretrained turned embedding layer. The most promossing of these ideas suggested will be tested.

The desired outcome is to achive the most accurate prediction of the human grading of correct or incorrect for the student answers. The current state of the art from the Reordan paper achieves results with around 75% accurancy for the data sets used in the project. Ideally this bench mark can be matched or exceeded in this project.

Approach

The approach is to recreate the basic model variations from the Riordan paper which has a baseline for SEB data and repeat with a XGBoost model.
Once the two types of models are built, tuned and tested with the relatively small SEB data set of 4 questions, the same models and tuning will be applied to an entirely different and larger data set, SAG, which consists of 88 computer science questions and about 2500 answers.

Metrics

This is a basic binary classification problem, so the metrics are pretty straight forward. The baseline (Riordan *3) used F1 which is a measure of a test's accuracy. It considers both the precision P and the recall R of the test to compute the score. More specifically, they used a weighted calculatio for F1 that takes in to consideration any imbalance in the datasets. The datasets in these projects show some imbalance although nothing extreeme like a fraud detection problem. The results of F1 are not expected to differ drastacly from simple accuracy but are a more accurate measure.

F1 = 2 * (P * R) / (P + R)

When calculating weighted F1, the F1 Scores are calculated for each label and then their average is weighted by support - which is the number of true instances for each label.

For this project, We are not only interested in accuracy but also have a secondary goal of minimizing type 1 errors in our metric. False positives are less likely to be discovered or reported by students then false negatives which if reported can improve their grade.

Because of our bias toward Type II errors, precision should we weighted more than recal. This concept is supported by the F1 calcuation by adding a beta coeficient. or F1 the beta is 1. The more generalized version, where β van be any value > 0, is:

Fβ = (1 + β) * (P * R) / (β * P + R)

Selecting a value of β = 0.1 will bias our metric significantly in favor or precision and minimize false positives.

1. Baseline: For the purposes of comparing to the baseline we will use the F1 weighted value.
2. Best Model: For the purposes of selecting the best model variation we will also use F1 as a primary criteria.
    - For a set of results in a reasonable range of F1 (i.e > 75%) we will look at Fβ to choose between results thar have similar F1 values.

In addition to the F1 and Fβ, the key metrics we need to collect are:

- True Positives - np.logical_and(test_labels, test_preds).sum()
- False Positives - np.logical_and(1 - test_labels, test_preds).sum() - Type 1 errors
- True Negatives - np.logical_and(1 - test_labels, 1 - test_preds).sum()
- False Negatives - np.logical_and(test_labels, 1 - test_preds).sum() - Type 2 errors
- Recal - tp / (tp + fn)
- Precision - tp / (tp + fp)
- Accuracy - (tp + tn) / (tp + fp + tn + fn)

II. Analysis

Data Exploration

The datasets used for testing include two primary data sources. These are located in the /data/source_data directory.

1. **SciEntsBank** (SEB) dataset. This data was taken from Dzikovska et al., 2012. The SciEntsBank (SEB) dataset consists of science assessment questions and we will only use the set with 2-way labels (correct/incorrect). *3

- The data is stored in XML format, one file for each of four questions.
- Each question has approximately 30 to 40 answers.
- The ratio of correct to total is 37.5%. This indicates that the majority of the students did very poorly. Examining the data shows some very confused respones.
- Each file includes the questions text, correct answer text and a list of graded answers (correct/incorrect)

- To make processing easier when training, the data is converted to a simpler csv format:
    - **question data** (question_id (0..3), question text, reference answer) are combined into a single file, **questions.csv**
    - **answer data** (question_id, answer text, score - 0/1) are combined into a single file, **answers.csv**
- Example Question Data
    - **Question:** (id=0): Carrie wanted to find out which was harder, a penny or a nickel, so she did a scratch test. How would this tell her which is harder?
        - **Reference Response**: The harder coin will scratch the other.
        - **Correct**: The one that is harder will scratch the less harder one.
        - **incorrect**: She could tell which is harder by getting a rock and seeing if the penny or nickel would scratch it. Whichever one does is harder.
        - **incorrect**: Rub them against a crystal.
    - **Question:** (id=2) A solution is a type of mixture. What makes it different from other mixtures?
        - **Reference Response**: A solution is a mixture formed when a solid dissolves in a liquid.
        - **Correct**: It dissolves the solid into a liquid that is see through
        - **incorrect**: A solution is a different type of mixture. Then they are a solution is a mixtures that dissolves.
        - **incorrect**: When the mixture is mixed.

2. **Short Answer Grading**: University of North Texas short answer grading data set. These assignment exams were assigned to an introductory computer science class. The student answers were collected via an online learning environment. The answers were scored by two human judges using marks between 0 (completely incorrect) and 5 (perfect answer). Data set creators treated the average grade of the two evaluators as the gold standard to examine the automatic scoring task. *1 (Suzen)

- The data set as a whole contains 80 questions and 2242 student answers, about 30 per question.
- The ratio of correct to total answers is about 71%. This makes sense as you would expect an average passing grade.
- Answer lengths varied between 1 and 950 words with the bulk in the 25 to 100 word range.
- The data set is stored in a complex format of multiple text files in multiple sub-directories.
- Multiple versions of the same data is available in aggregated and file per question formats.
- Score are not aggregated into a single file, rather a file per questions.
- To produce binary results for scores in a range of 0 to 5, answers were considered correct when the score was greather than or equal to 4. Another option would have been to chose 3, the midpoint as the cuttoff or any other value greater than 0. The criteria for the decision is not availabe from the question creator so some rational is needed. Here the goal is to have a sufficient balance of correct and incorrect and try and filter out the more confusing answers which presumably have a lower score. For this reason, the value of 4 was chosen.
- The data has a number of text representations for punctuation that were removed for this exercise. ex. -LRB-, -RRB- and STOP
- The files used for this project are:
    - **Questions** - *ShortAnswerGrading_v2.0/data/sent/questions* which is in the format of questions_id and question text seperated by a space
    - **Reference Answers** - *ShortAnswerGrading_v2.0/data/sent/answers* which is in the format question_id and answer text, spaced
    - **Student Answers** - *ShortAnswerGrading_v2.0/data/sent/all* which is in the format question_id and answer text, spaced

    - **Files** - *ShortAnswerGrading_v2.0/data/docs/files*. A single list of question_ids used also for directory and filenames which identify the directories for scores by question_id.
    - **Scores** - Using the Files list, all scores were concatenated in order from the files located in *ShortAnswerGrading_v2.0/data/scores/<question_id>/ave*.

- To make subsequent processing easier the data was converted into simple questions and answer files similar to those for SEB.
    - **question data** (question_id, question, reference answer) were combined into a single csv file, **questions.csv**
    - **answer_data** (question_id,answer, score) were combined into a single csv file, **answers.csv**

- The simplified questions.csv and answers.csv are stored in */data/sag2*.

  Example Question Data

    - **Question:** (id=1.1) What is the role of a prototype program in problem solving?
        - **Reference Response**: To simulate the behaviour of portions of the desired software product.
        - **Correct**: you can break the whole program into prototype programs to simulate parts of the final program.
        - **incorrect**: To lay out the basics and give you a starting point in the actual problem solving.
    - **Question:** (id=5.1) In one sentence, what is the main idea implemented by insertion sort?
        - **Reference Response**: Taking one array element at a time, from left to right, it inserts it in the right position among the already sorted elements on its left.
        - **Correct**: insertion sort is were after k iterations the first k items in the array are sorted it take the k plus 1 item and inserts it into the correct position in the already sorted k elements.
        - **incorrect**: Take a number and choose a pivot point and insert the number in the correct position from the pivot point.
    - **Question:** (id=7.2) What is the main advantage of linked lists over arrays
        - **Reference Response**: The linked lists can be of variable length.
        - **Correct**: Array size is fixed, but Linked is not fixed.
        - **incorrect**: Linked lists have constant time insertion and deletion
        - **incorrect**: There is no limit as to how many you create where an array can only hold a given amount of information.
            - Note: This second incorrect answer is provided here because I would grade it as correct. This highlights the human error potential in grading which adds an additional random factor to the data.
    - **Question:** (id=11.1) What are the elements typically included in a class definition?
        - **Reference Response**: Function members and data members.
        - **Correct**: data members and function definitions.
        - **incorrect**: Class name,, semicoln at the end of the defination, private and bublic followed by :
    - **Question:** (id=12.10) How many steps does it take to search a node in a binary search tree?
        - **Reference Response**: The height of the tree.
        - **Correct**: to find a node in a binary search tree takes at most the same number of steps as there are levels of the tree.
        - **incorrect**: it depends on the install search tree then from there for whatever the case is the it repeats it back along the case of the primary node.

Exploratory Visualization

The data used in this project has been well vetted as a good use case for short answer grading in the papers cited at the top of this report.

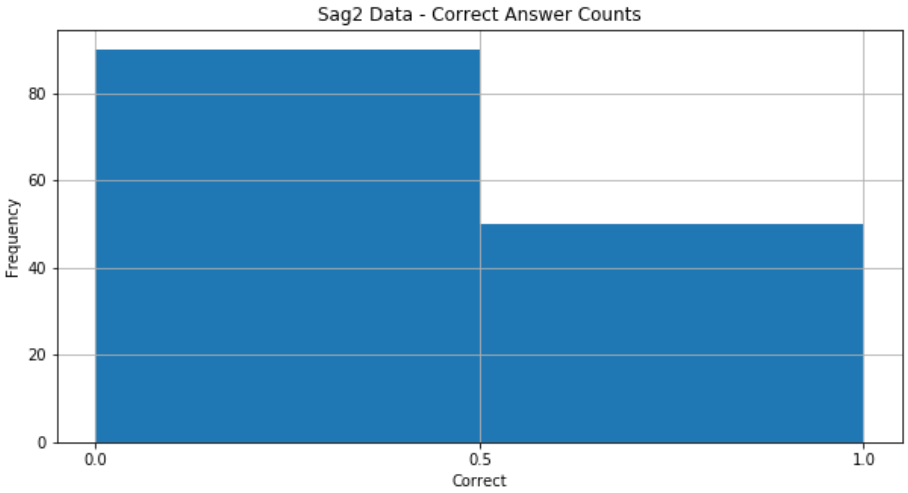
The feature or features used are essentiall the encoded representation of strings that were provided as answers. This enconding is achieved by creating a word dictionary and assigning the index from the dictionary to the word. The feature is then a list of integers representing the words. The dictionary size for the small SEB dataset is about 265 words. While the dictionary size for the large SAG dataset is about 2650 words.

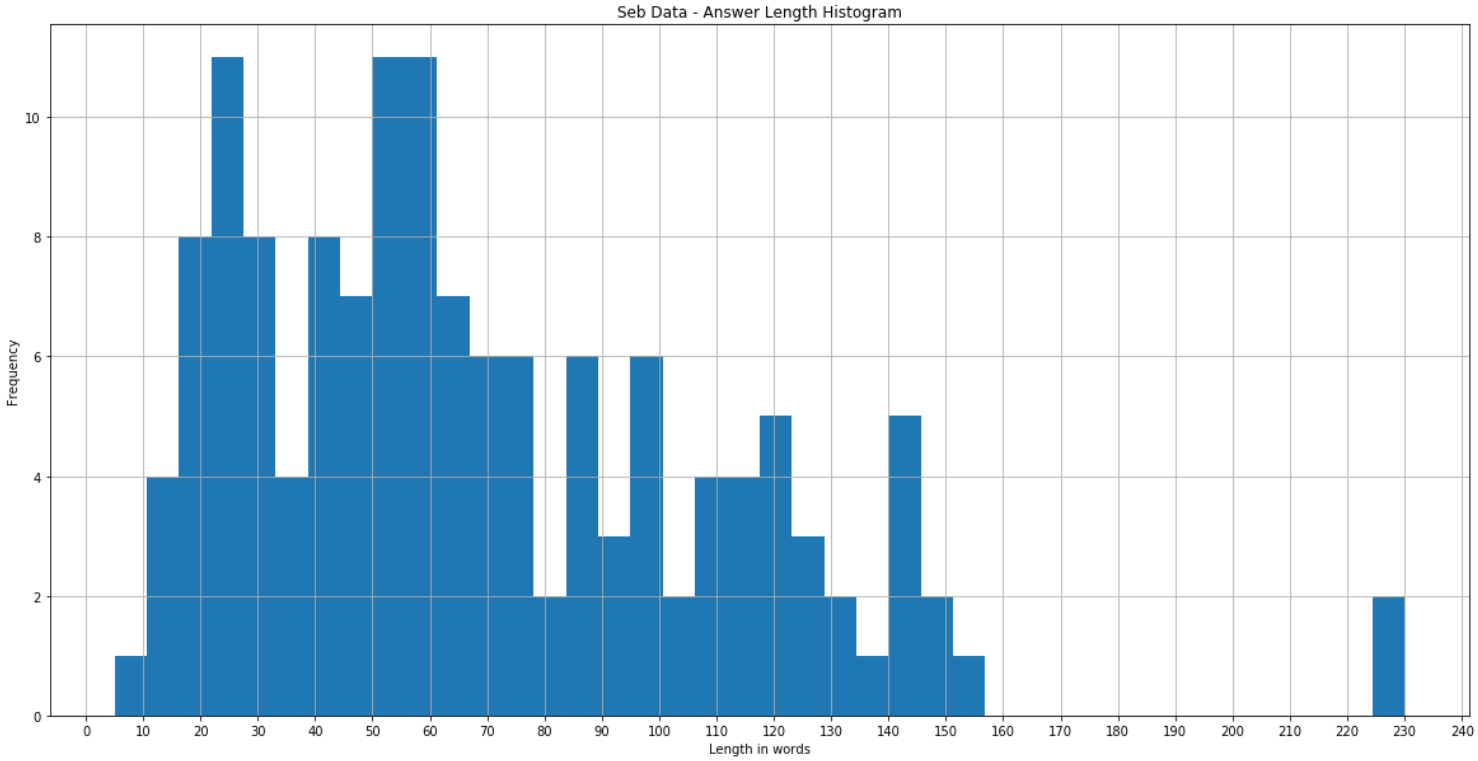
The most important data to visualize is the grads for the answers. If for example 95% of the answers were correct or incorrect, we whould need to consider this a unbalanced dataset and adjust for that. The visualization shows that the data, while not exactly balanced, has a signifiant result set for both correct and incorrect. In both cases it is a 2/3 to 1/3 ratio, For the SEB dataset, that ratio is in favor of incorrect while for SAG in favor of correct.

The visualization of answer lengths is also of some interest. This information is likely to play into the effectiveness of the model fitting. It has been documented in the research papers that shorter answers are more difficult to fit. From the historgrams we can see that a majority of the answers for both datasets fall in the range of 50 to 100 words. This confirms the relatively short nature of the answers as compared to essays or even short essay type response for which other modeling technical have proven successful (see *3 Riordan)

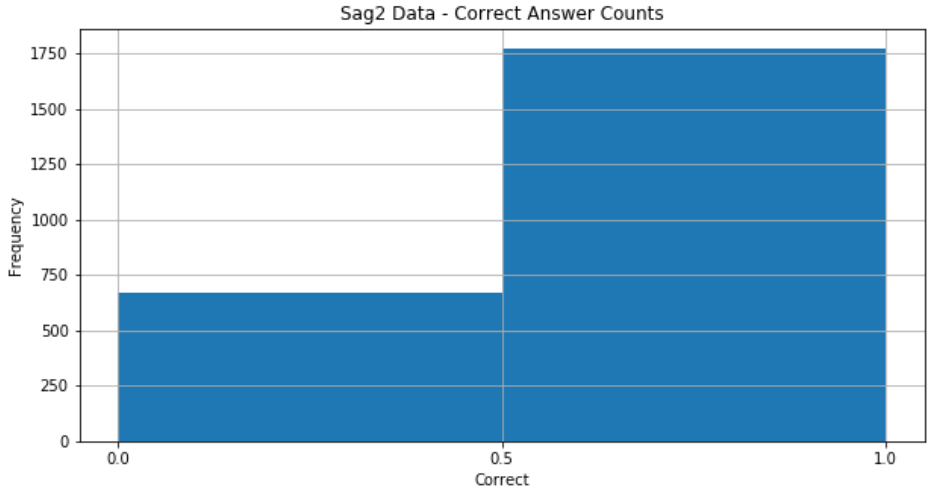
One final area that bears noting is the difficulty of the questions. This can be shown by a graph of the percentage correct answers by question. The chart for SAG below shows that while the average is around 80% it varies widely between 20% and 100%. This is likely to have an impact on the predictability of the data because the answer pool for each question has a different composition of scores.

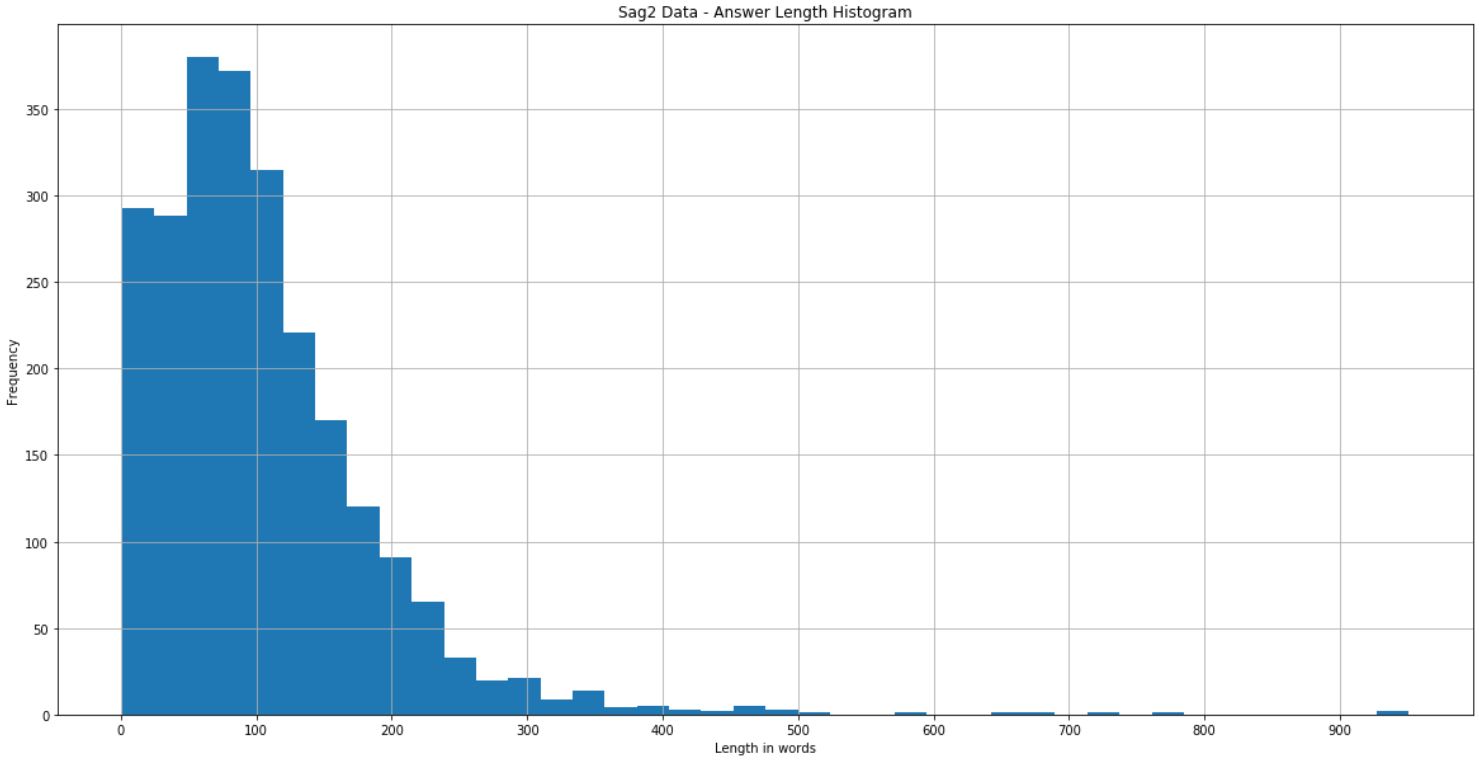
1. SciEntsBank

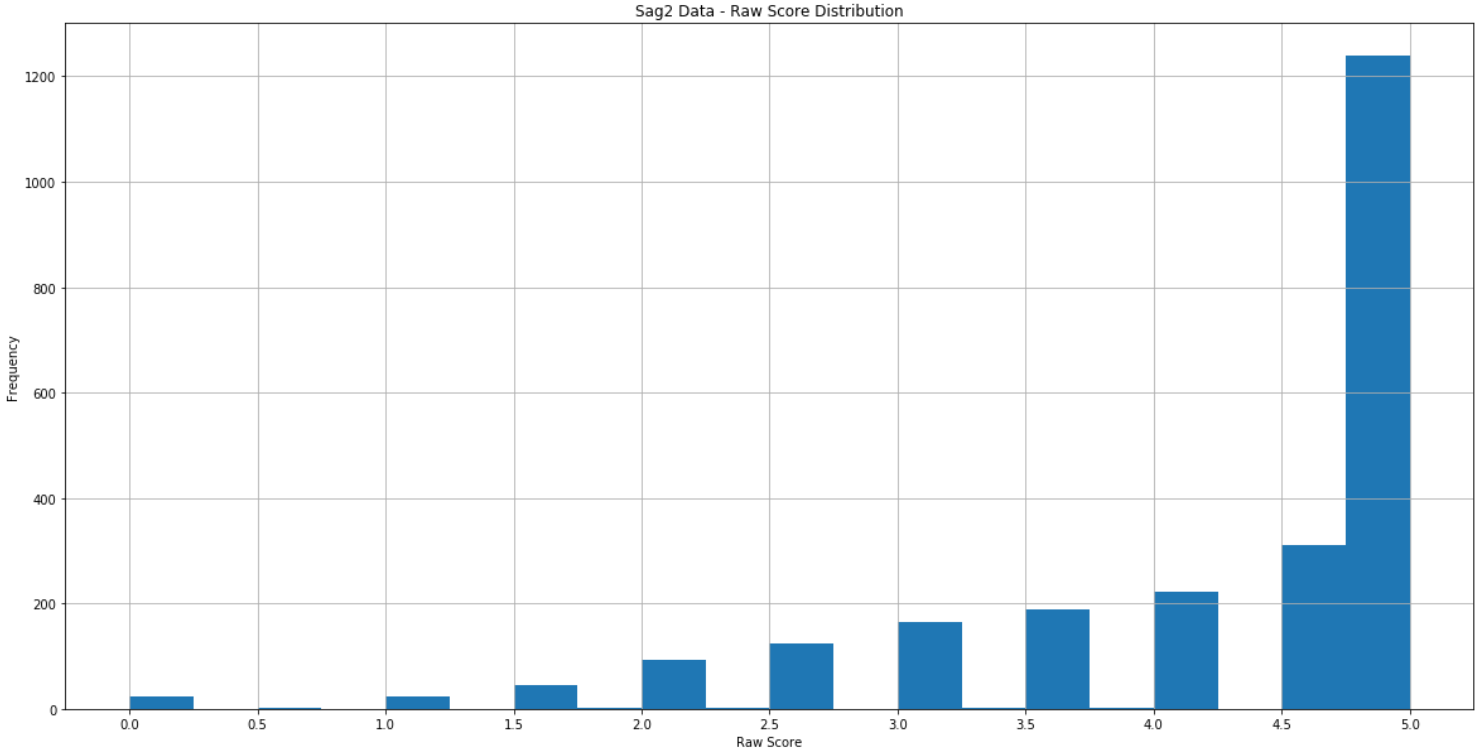




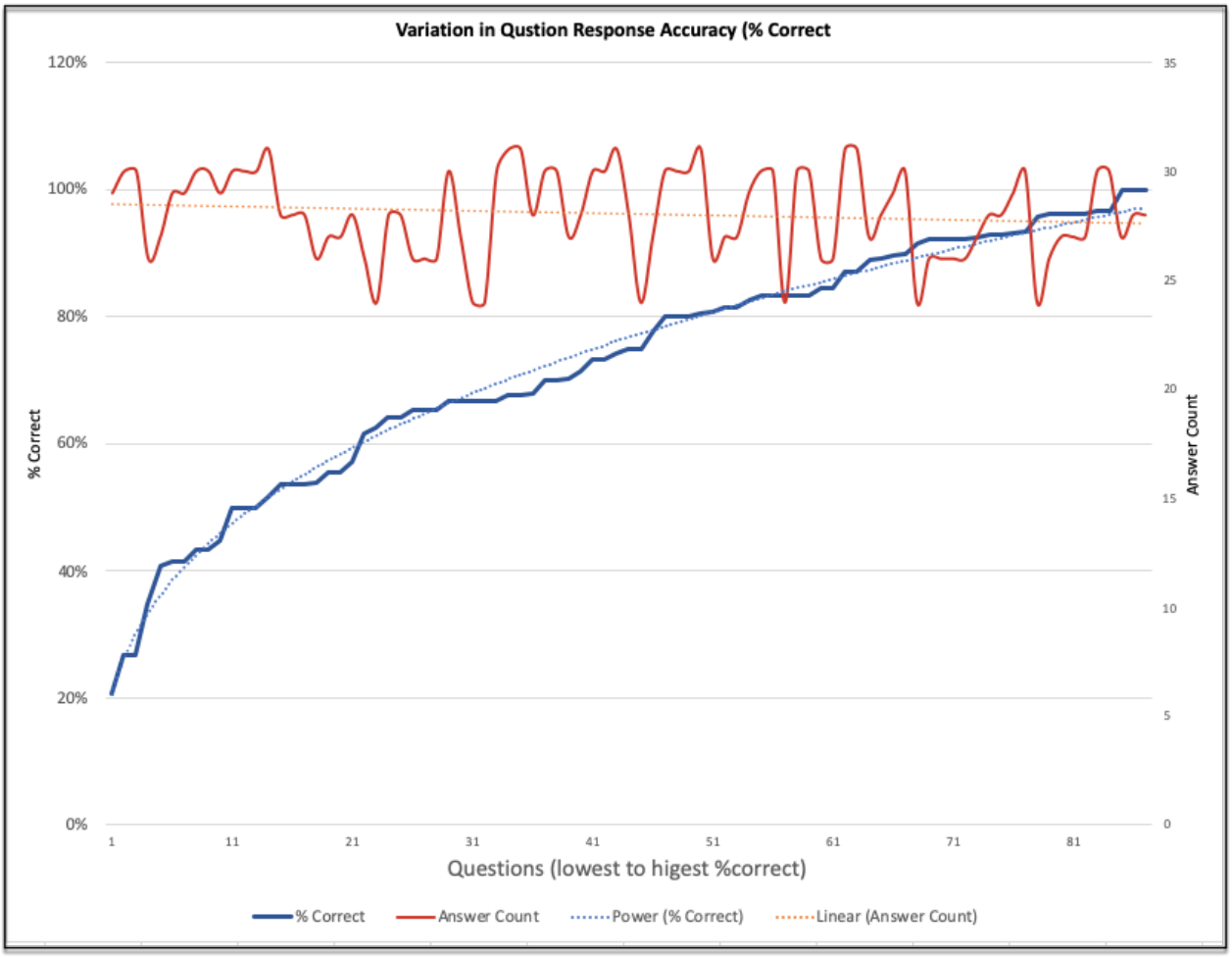
2. Short Answer Grading







Note: This disrtribution of answers uses a cutoff at 4.0 resulting in about 1770 correct and 670 incorrect.



Algorithms and Techniques

Data Pre-Processing: The first and non-trivial task was to preprocess the raw datasets, some of which are XML, and generate encoded train and test datasets. One of the goals of the encodining of answers is to build in the ability to decode the results. After data is randomly broken in to train and test and prediction are made, we can reverse the encoding to see if errors in answers predicted show any patterns. This decoding helps to better understand and visualize the challenges with some of the answers as compared to others. To support that, the question_id is added to the answer embedding vector.

To simplify working in the AWS Sagemaker enviroment, the process also generates a test.csv and train.csv suitable for use in Sagemaker. That is, the label for correct answer (0 or 1) is prepended to the features (word vectors).

While not done for this project, it would be simple to add calls to the methods that process in data set to the SageMaker Jupyter notebooks. This was not done at the time of this writing. Instead, the files are stored in GitHub and retrieved in the AWS environment as needed.

LSTM - Deep Learning

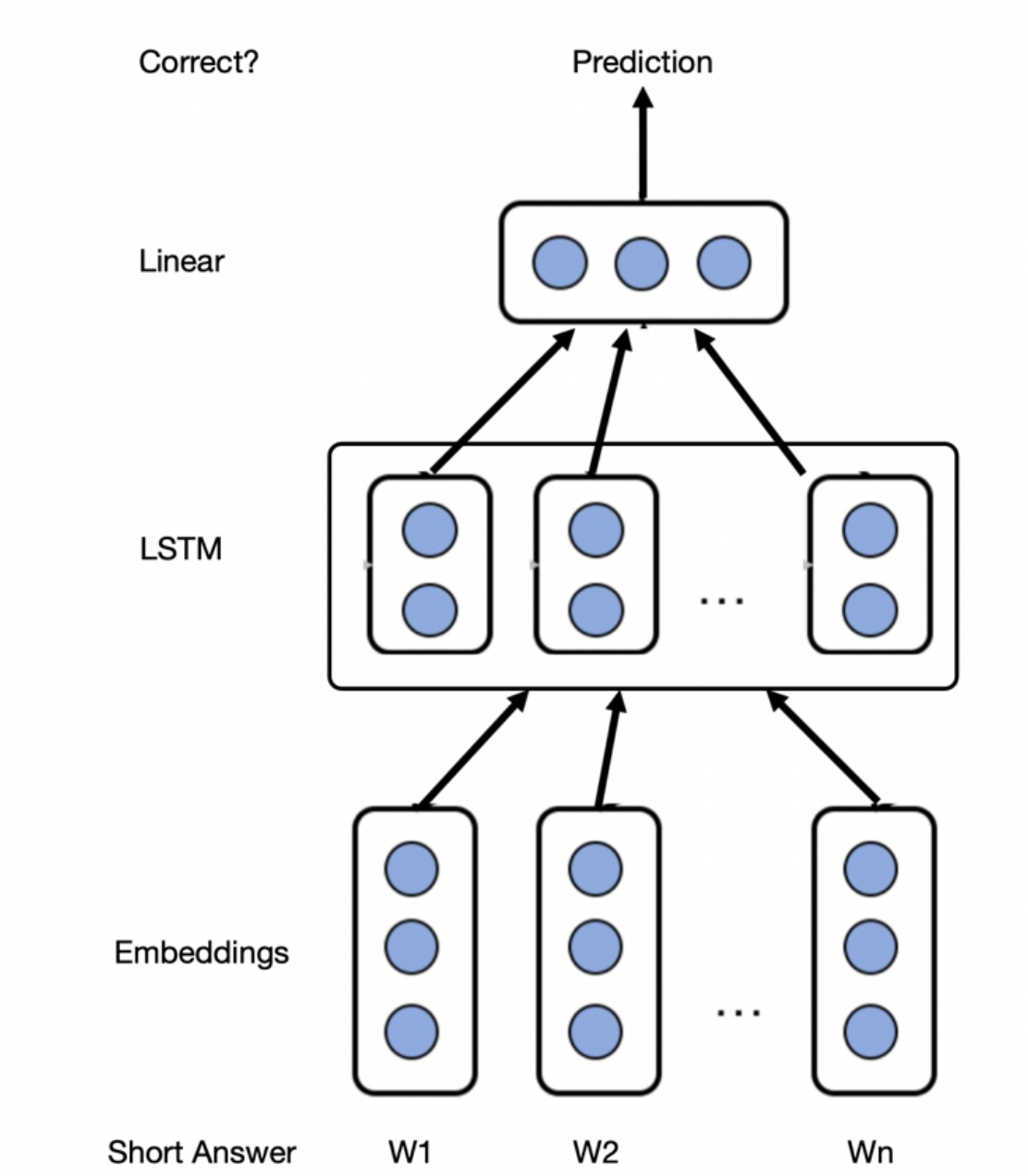
The basic framework for the LSTM deep learning model comes from the Reordan (*3) paper.

The basic model used here consists of:

- An embedding layer that process the encoded sentences and produces word vectors for each word in the vocabulary.
  - After testing the generated embeddings some effort was also put into testing pretrained embeddings taken from the Glove data file with 50 dimensions (data/glove/glove.6B.50d.txt).
    - This required deriving a custom dictionary from the Glove file by selecting the words from the locally generated answer dictorionary and using the glove id's to lookup the embedding_matrix for each word and creating a custom_embedding_matrix for our dictionary.

    - The pretrained custom embedding matrix was used to preload the embedding layer. The code for this can be found in */source/glove.py*.
    - One challenge to note is that mispelling, concatinations, garbage input and jargon were not found in the glove database and 0's were used insetead. This represented about 18% of the words.
    - It was out of scope for this project to correct misspelled or concatenated words to reduce the glove databases misses.
- An LSTM layer with one or more sub-layers to model more complexity.
- Dropout layers to help tuning and beter managing overfitting.
    - This proved quite valuable when dealing with problems of models converging at all.
    - This permits overfitting and then the use of dropout increases to make the test predictions more accurate.
- Activation layer and Objective. Linear activation rather than sofmax/binary_crossentropy is used based on the Riordan finding. This was validated with some initial tests. It also has the benefit of giving more visability into the probabilities.

The figure below is visualization of the model approach.



Tuning Parameters and Variations:

In addition to the basic model above, some variations and adjustments to the model are tested. These variations are documented in the Riordan paper as having a positive impact on accuracy.

The key variations tested in this project are:

- Pretrained v.s. Generated embeddings.
- Flatening and Shaping (Yes or No).
  - These layers can reduce overfitting or improve fitting in general.
  - The embedding layer which is 2D for each entry is flattened.
- LSTM layers and layer sizes.
  - One or two layers of varying sizes.
- Dropout layers: Two drop out layers after embedding and LSTM with variable dropout percentages.

Sagemaker and Keras LSTM

The LSTM models required custom coding to work in SageMaker. For ease of local testing I chose to use as Keras Sequential model which uses Tensorflow as the engine. It was quite straight forward to do setup locally, but took a lot of research, trail and error to get the Keras/Tensorflow custom model working in Sagemaker. Additional work was also needed to capture the necessary metrics from the training jobs for use by the Hypertuning process. The resuls provide a useful framework for future testing and an easy path to quickly move Keras models from local to Sagemaker implementations.

XGBoost

As a comparison and sanity check, XGBoost was run on the same train.csv and test.csv files used with LSTM. XGBoost specific tunning parementers are used to optimize the models.

Benchmark

The benchmark for this project is well documented in the Riordan paper for the SEB dataset. The table provided below includes two datasets they tested and both 2- and 5-way grading. They also have their own benchmark from anther research paper. In summary, their results all fell in the range of 70-75% acurancy which is the benchmark accuracy for this project.

The highlighed section in yellow is the first of two datasets tested in this project which is referred to here as SEB through the report.

								SRA Beetle 2-way	5-way	SRA SEB 2-way	5-way
Experiment	**Condition**	**Emb**	**CNN**	**Dim**	**Dir**	**MoT**	**Att**	**Mean wF1**	**Mean wF1**	**Mean wF1**	**Mean wF1**
Benchmark	Baseline							0.7438	0.5815	0.7011	0.5415
	T&N best	tun	no	300	uni	yes	no	0.7805	0.6184	0.7386	0.6175
MoT	no MoT	tun	no	300	uni	no	no	0.7803	0.6163	0.7384	0.6159
Embeddings	fixed	fix	no	300	uni	yes	no	0.7803	0.6119	0.7112	0.5730
	learned	lea	no	300	uni	yes	no	0.7396	0.5929	0.7285	0.5855
CNN	win len 3	tun	no	300	uni	yes	no	0.7786	0.6048	**0.7431**	0.5874
Directionality	bi	tun	no	300	bi	yes	no	0.7699	**0.6461**	**0.7511**	0.6171
LSTM dims	50	tun	no	50	uni	yes	no	0.7603	0.5954	**0.7395**	0.5992
	100	tun	no	100	uni	yes	no	0.7679	**0.6192**	0.7341	0.5925
	150	tun	no	150	uni	yes	no	**0.7816**	0.6168	**0.7389**	0.6039
	200	tun	no	200	uni	yes	no	0.7768	**0.6186**	0.7336	0.6080
	250	tun	no	250	uni	yes	no	0.7663	0.6106	0.7334	0.6160
Attention	T&N-sum	tun	no	300	uni	no	yes	**0.7915**	**0.6469**	**0.7454**	0.5941
Combination		tun				no	yes	0.7691	**0.6246**	0.7308	0.6109

Table 3: Parameter experiment results on SRA datasets on the *development* set. "wF1" is the weighted F1 score. "Baseline" is the baseline non-neural system. "T&N best" is the best-performing parameter set in Taghipour & Ng (2016): tuned embeddings (here, GLOVE 100 dimensions), 300-dimensional LSTM, unidirectional, mean-over-time layer. Scores are bolded if they outperform the score for the "T&N best" parameter setting.

From page 166, (Riordan *3)

III. Methodology

Data Preprocessing

The preprocessing of data was done so that both data source (SEB and SAG) can share the modeling and testing code. The goal of the preprocessing was to generate the following file formats which represent the minimum fields required. Additional fields can be put in the files for analysis purposes but will not be used to generate the train and test files.

- Questions (questions.csv)
    - question_id - unique indentifier for the question
    - question - text of the question
    - answer - text of the reference answer
- Answers (answers.csv)
    - question_id - unique identifier for the question
    - answer - text of the user response
    - correct - 0 or 1

  Notes:
    - For SEB which was used during the developoment, question text and reference answer were also store to help with debugging.
    - For SAG the orignal score was also stored (0-5) for analysis purposes

SEB Preprocessing

The preprocessing can be found in one method in */source/utils* called *load_seb_data().

File Processing

This method reads the original raw source XML and writes the standard question.csv and answer.csv for use in model training.

As described in the **Data Exploration** section, SEB data for the four questions is stored in four XML files which each contain all the data for the question. This includes question text, reference answer, student answers and scores. Because XML is fairly easy to process, the code was written to load the data direcly from the XML files each time the model was tested. The files were also quite small, so this provided little overhead.

Helper funtions that leverage *xml.dom.minidom* were written to extract the desired fields from the SEB question file. These functions leverage DOM parsing to extract the single and multiply occuring data (student answers) by iterating through the XML DOM.

Answer Encoding

The next step is to generate a vocabulary and map the text words to a list of numbers. This step also determinew the longest sentence used later for padding all answer vectors to the same length.

The enconding is done by first building the vocuabulary. The vocabulary is generated by concatenating all the answers into a single text string, cleansing the string and creating a word dictionary with unique integer ids.The answers are then maped the words basd on unique integer index to the vocabulary.

The encoded answer vectors are padded based on the longest answer found in the test data using a function provided by the Keras library (keras.preprocessing.sequence.pad_sequences).

The data is then randomized by adding the predictor column (correct answer) to the answer vectors and using the DataFrame reindexing feature with np.random.permutations. After randomizing the order, the data is then split back into answsers and labels read for dividing into train and test datasets.

Train and Test Data The final step is to use sklearn.model_selection.train_test_split using a 30% split to generate the X_train, y_train, X_test and y_test data. These along with the vocabulary and the length of the answer vector (needed for the Embedding layer) are returned for use in the modeling phase. This data is also stored in train.csv and test.csv with the label prepended in the first column for use in Sagemaker models.

SAG Preprocessing

Because of the complexity of the SAG raw data as documented in the **Data Exploration** section above, the raw data was preprocessed once and converted to csv with a set of python commands that ultimately generate the question.csv and answer.csv. Once completed the creation of the train and test data is done using the same approach as with the SEB data above. Pandas is leveraged heavily to read and write CSV files and to manipulate data in memory. Featurs such as adding columns to an existing Dataset read from a CSV and splitting data are often leveraged.

Manual Steps

1. The needed data must be extracted from multiple files:
    - Questions: data/ShortAnswerGrading_v2.0/data/sent/questions
    - Correct Answers: data/ShortAnswerGrading_v2.0/data/sent/answers
    - Scores: data/ShortAnswerGrading_v2.0/data/scores/1.1/ave. (one file per question)
    - Answers: data/ShortAnswerGrading_v2.0/data/sent/1.1 (one file per question)
2. Combine the questions and regerence answers into a single file: /data/sag2/questions.csv
    - Manually convert questions and answer files which were spaced seperated into a csv format using a text editor.
    - Use Pandas to merge the two files into one.
    - Note: the id field provided in SAG is in a format that will be misinterpreted as a number. This would result in the loss of information for ids like 11.10 and 12.10. To handle this we insure that pandas treats it as a string.

questions = pd.read_csv('data/sag2/questions.csv', dtype={'id': str}) questions.set_index('id') answers = pd.read_csv('data/sag2/answers.csv', dtype={'id': str}) answers.set_index('id') qanda = pd.merge(questions, answers) Deleted the working questions and answers separate files and replaced with combined file qanda.to_csv('data/sag2/questions.csv', index=False)

3. Create a single data_file for answers. This includes questions_id, answer responses (sentences) and scores.
    - Run a script to read and concatenate the answers and scores using the names of the files (data/ShortAnswerGrading_v2.0/data/docs/files)
    - The complete list of all answers is stored in a single file *data/sag2/sent*
    - All the scores are read for each question and concatenated in the order of the answers.
    - The scores are then appended as a series to the text answers.

    files = pd.read_csv('data/ShortAnswerGrading_v2.0/data/docs/files', dtype={'id':str}) scores_df = pd.DataFrame(columns = ['id', 'score']) for id in files['id'].values: scores['score'] = pd.read_csv(f"data/ShortAnswerGrading_v2.0/data/scores/{id}/ave")[0] scores['id'] = id scores_df.append(scores, ignore_index=True)

    answers = pd.read_csv('data/sag2/sent', header=None, dtype={'id':str}) answers['score'] = scores_df['score'] answers['correct] = answers['score'] >= 4 answers.to_csv('data/sag2/answers.csv', index=False)

XGBoost Data

The data used for XGBoost needs to be in the format of [labels, features...].

- XGBoost requires a validation.csv which is draw from the train.csv used in the LSTM models.
- A simple slice was done in the Jupyter Notebook to generate the train_xgb.csv and validate_xgb.csv from the train.csv file generated locally.

Implementation

The implementation for local testing is a set of short driver python scripts that all share the comon code in the utility file. Each driver script is used to configure the hyper parameters and call utility functions to load data and execute the model training and evaluation.

All local LSTM models are based on SkLearn with Keras on top of Tensorflow. The Keras Sequential model is used to build the model layers. Local XBGoost uses XGBCLassifier which conforms to the SKLearn api.

All Sagemaker models are based on SageMaker classes.

The Keras/Tensorflow approach used Locally for LSTM is not diretly supported on SageMaker so the SageMaker Tensorflow class is used and it calls a custom train.py file (located in /source/train.py) which builds and trains the Keras LSTM model. Thr train.py generates text output that can be parsed by SageMaker to obtain the metrics needed for Hypertuning.

Local Testing

Most of the initial testing is done locally on a laptop to debug and get initial tuning parameters. The goal is also to eliminate low performing configurations before using SageMaker for Hypertuning.

Local Tests Driver Scripts

1. SEB Data with LSTM (*seb_tf_train.py*)
2. SEB Data with XGBoost (*seb_xgb_train.py*)
3. SAG Data with LSTM (*sag_tf_train.py*)
4. SAG Data with XGBoost (*sag_xgb_train.py*)
5. SAG Data with XGBoost and added ngram features (*seb_ng_xgb_train.py*)

*1 Ngrams are tested to provide a feature that compares the student answer to reference answer.

The **local mode** code as well as the Sagemaker code passes in the tuning parameters as a dictionary. For LSTM models, the local driver scripts call a common shared utility function. This builds the model, fits the data and then uses the test data with the model's predictor to evaluate the model accuracy.

A similar apporach is used for XGBoost with the slight variation because the persistence of models is different between Keras and XGBoost. To handle this some of the processing is moved to the driver scripts which then calls lower level utility functions evaluate the model but build the simpler XGBoost model in the driver script. The resulting model is then saved using another unility fuction for that leverages the Joblib library to save model objects.

A common set of code is used by all tests to evaluate and display the results. The evaluation iterates through all the test data, predicts the result and calculates the metrics. The results are printed with the ability to list every answer and it's prediction or just the summary metrics.

Local LSTM

Tunning Parameters for **local mode** testing:

```
model_params = {'max_answer_len': max_answer_len,   # use by the embedding layer
                'vocab_size': vocab_size,           # used by the embedding layer
                'epochs': 20,
                'pretrained': pretrained,           # used by build model to load petrained embeddings if true
                'embedding_dim': 50,                # Size of embedding dimension
                'flatten': True,                    # The embedding dimension is flattened and shaped if true
                'lstm_dim_1': 100,                  # Size of the first lstm dimension (> 0)
                'lstm_dim_2': 20,                   # Size of the second lstm dimension (>=0, 0 means none)
                'dropout': 0.3}                     # The dropout layer percent used (>0).
```

SageMaker Testing

For LSTM/Tensorflow SageMaker requires access to a user devleoped Python scrip that builds and trains the model. The custom file for LSTM processing on SageMaker is stored in **/source/train.py** and can be referenced in the setup of Tensorflow models in the Jupyter notebook. The custom code also persists the model as expected by SageMaker and calculates the needed metrics for parsing by SageMakers Hypertuning jobs.

The XGBoost model is build like any standard SageMaker built in model.

SageMaker Tests

1. SEB Data with LSTM/Tensorflow - (*Capstone 1 SEB.ipnyb*)
2. SAG Data with LSTM/Tensorflow - (*Capstone 2 SAG.ipnyb*)
3. SAG Data with XGBoost - (*Capstone 2 SAG-XGB.ipnyb*)

Each of these Notebooks reads the data already prepared by in local processing and stored in the git repo under /data/seb and /data/sag in train.csv and test.csv.

The data is loaded into a notebook and then uploaded to S3 buckets for use by the SageMaker models. The first step is to test the model an estimator to verify it was working and then a use a Hypertuning model to optimize the parameters. The last step is to launch an endpoint for the predictor and use the test data to evaluat the model using an *evalutate()* function in a cell.

Note: Pretrained embedding are not tested on SageMaker due to time limitations and because they did not prove helpful in local testing.

Containment - Ngrams

One of the experiments mentioned was to try adding Ngrams to the features. Containment is the degree to which one sentence matches another based on comparision of word or word sequences.

As a quick experiment, this was tested with XGBoost. The ngrams were calclated using and SKLearn function called CountVectorizer (sklearn.feature_extraction.text CountVectorizer). Once instantiated, a CountVector object take the answer and reference answer and calculates the number of words or phrases in common basd on the input value for n.

Average containment for multiple values of n were calculated and only n1 and n2 were signifiant for the short answers. A cross correlation betwen n1, n2 and the label (correct/incorrect) were completed to see how well each correlated. A test and train data set was manually created by appending these values to the existing generated train.csv and test.csv.

SageMaker Tensorflow Custom Parameters

SEB

```
    hyperparameters = {
     'epochs': 200,
     'embedding_size': 30,
     'flatten': 0,
     'lstm_dim_1': 100,
     'lstm_dim_2': 0,
     'dropout': 0.2
```

SAG

```
    hyperparameters = {
     'epochs': 20,
     'embedding_size': 50,
     'flatten': 1,
     'lstm_dim_1': 100,
     'lstm_dim_2': 20,
     'dropout': 0.3
```

HyperTuning Setup for Tensorflow: The selection of parameters to hypertune was based on the experience in local testing.

SEB

```
tf_hyperparameter_tuner = HyperparameterTuner(estimator = estimator,
       objective_metric_name = 'Validation_accuracy', # The metric used to compare trained models.
       objective_type = 'Maximize', # Whether we wish to minimize or maximize the metric.
       metric_definitions = [{'Name': 'Validation_loss',
                              'Regex': 'Validation_loss:(.*?);'},
                             {'Name': 'Validation_accuracy',
                              'Regex': 'Validation_accuracy:(.*?);'}
                            ],
       max_jobs = 18, # The total number of models to train
       max_parallel_jobs = 6, # The number of models to train in parallel
       hyperparameter_ranges = {
            'dropout': ContinuousParameter(0.1, 0.4),
            'embedding_size': IntegerParameter(20, 100),
            'lstm_dim_1': IntegerParameter(50, 150)
       })
```

SAG

```
tf_hyperparameter_tuner = HyperparameterTuner(estimator = estimator, # The estimator object to use as the basis for
the training jobs.
       objective_metric_name = 'Validation_accuracy', # The metric used to compare trained models.
       objective_type = 'Maximize', # Whether we wish to minimize or maximize the metric.
       metric_definitions = [{'Name': 'Validation_loss',
                              'Regex': 'Validation_loss:(.*?);'},
                             {'Name': 'Validation_accuracy',
                              'Regex': 'Validation_accuracy:(.*?);'}
                            ],
```

```
        max_jobs = 18, # The total number of models to train
        max_parallel_jobs = 3, # The number of models to train in parallel
        hyperparameter_ranges = {
             'dropout': ContinuousParameter(0.2, 0.4),
             'embedding_size': IntegerParameter(50, 250),
             'lstm_dim_1': IntegerParameter(100, 250),
             'lstm_dim_2': IntegerParameter(10, 50)
        })
```

XGBoost Parameters

XBGoost was only used to test SAG data as a reference to see how a powerful machine learning algorithm compared to deep learning approaches. The basic starting parameters were dervied from local testing and then Hypertuning ranges were adjusted based on initial runs.

SAG

Estimator Parameters:

```
max_depth=5
eta=0.2
gamma=4
min_child_weight=6
subsample=0.8
objective='binary:logistic'
early_stopping_rounds=50
num_round=4000
```

Hypertuning Parameters:

```
'max_depth': IntegerParameter(3, 12)
'eta'      : ContinuousParameter(0.05, 0.5)
'min_child_weight': IntegerParameter(2, 8)
'subsample': ContinuousParameter(0.5, 0.9)
'gamma': ContinuousParameter(0, 10)
```

Refinement

The refinement process consisted of three major phases: (See the **Model Evaluation and Validation** section for additonal refinement details)

LSTM

Since the baseline is based on the LSTM model, all initial tests were with variations of that model.

1. Start testing locally to get some idea of the parameters and ranges of values that have a positive impact.
2. Get a baseline model to converge during training (accuracy of 100%) using overfitting as required.
3. Adjust the basic parameters individually to see the directional impact they have on accuracy. These include epochs, dropout, LSTM size, and Embedding size.
4. Try some of the model changes proposed bu Riordan to see if they improve on those obtained with the basic model tuning. If they are competitive consider them for further testing. These include, flattening, pretrained embeddings, multiple LSTM layers.
5. Take the most promissing values and model variations and build a SageMaker impelemtation for validation, testing and then Hypertuning.
6. Validate that the local results can be replicated.

7. See if better results can be obtained with Hypertuning than from local testing.

XGBoost

1. Using the same data from the LSTM runs, test an XGBoost model locally.
2. Do some basic tuning tests and compare the results to the LSTM model results.
3. Tranlate the local XBboost model into a SageMaker XGBoost model and build, train, Hypertune and evaluate the results.

IV. Results

Model Evaluation and Validation

The benchmark for this testing was an to meet or exceed an accuracy in the mid 70's. It is important to note that since the dataset is not perfectly balanced, predicting all incorrect for SEB would result in 64% accuracy while all correct for SAG data restuls in a 72.5% accuracy. It is critial, therefore, that the test results have a good distribution, predictions of correct and incorrect. To the degree that that solution is not much better than the trivial predicion of all correct/incorrect we likely will conclude that the model is not successful. The starting assumption of this project is that the outcome will match the baseline and that maybe some new improvement will be found. The ultimate goal remains to replicate the resutls and gain insight into what next steps and ares of exploration might lead toward a better solution.

Outcome

1. *The Deep Learning baseline was slightly exceeded at 77.1%, or 2% higher than the best variation test in the baseline result of 75.1%. The results show how difficult the deep learning model is to optimize with these datasets. The best results were slightly better than the baseline often with simpler solutions than the optimal ones found in baseline.
2. *XGBoost - The XGBoost result also slightly exceeded the baseline at 76.7%. This is not significantly different from the LSTM model which suggests that considering the extra effort of the deep learning model it is not performing that well. The main benefit of deep learning model is the insights it gives and the flexibilty is provides to improve as new insights are obtained.
3. *Similarity* - While only briefly tested, intutiion suggests that some use of the correct answer compared to the student answer would add value. Given the limitated the scope of this project, this was only lightly tested. It showed a some promiss but was not enought to warrant dedicating extensive testing. A seperate project to combine embedding features with similarity features in various way with different models would be valuable.
4. *Question Difficulty* - Analysis showes that question difficulty varies greatly. The correct/incorrect ratios that avergage to 72.5% correct are not the case for individual questions which vary from 20% to 100% correct answers. This makes the results intesting when looked at by individal questions. We see a strong inverse corelation between the difficulty and accuracy of predictions based on the question. This warants deeper examination and modeling in followup work.

1. SEB Data with LSTM

The first tests match the basline provided in the Riordan for the LSTM/Embedding model. Their best results are accuracies in the low to mid 70's. The approach used here was to first overfit to insure that training accuracy was at 100%. Once this was achieved, reduce the overfitting to imporove the test results. A variety of the major model adustments suggested as effective in the Riorden paper were also tested . An additional test was done to use a flattening and shaping layer which converts the Embedded dimension into a single vector from the vector per word representation. This was recommended on some on-line resources as a possible improvement when using the Embedding layer.

Best Results

- Epocs: For this dataset and model, 200 epochs generally achieved the necessary convergence and was mostly used for all tests.
- Embedding: The size of the embedding layer was an important variable. However, pretraining proved to be unproductive.

    - Since the results obtained match or exceeded the baseline, and Pretraining was far below, this approach was not tested extensively.
    - One note is that 50 dimensions were used here where 100 were used in the baseline. This may be a reason for the difference but the accuracy obtained without it still matched the baseline.
- Flatening/Shaping: This layer had a surprisingly negative impact for this dataset. It would appear that due to the small size of the data set the added information provided by keeping the word vectors seperate may have helped converge during training.
- LSTM: A single LSTM layer seemed sufficient for this data and the size was important.
- Dropout: The dropout percentage was a very powerful way to fine-tune the results and to counter the overfitting during training.

Results Table:

Each test is listed in the order they were performed with notes on the changes on the results. Changes for each test are highlighted in the cell. The best result is highlighted in green.

Test #	Note	Embedding	Pretrained	Flattening	LSTM-1	LSTM-2	Dropout	Epochs	Tr. Converg.	recall	precision	accuracy	F1-Weighted	Fbeta(0.1)	Outcome
1	Initial testing	30	No	No	100	0	0.2	200	No	58.8%	83.3%	78.6%	**77.7%**	79.3%	Improved
2	Reduce epochs to see impact of reduced training	30	No	No	100	0	0.2	100	No	52.9%	81.8%	76.2%	**74.9%**	77.2%	Overfitting incorrect
3	Original Epochs, lower dropout	30	No	No	100	0	0.2	200	No	23.5%	80.0%	69.0%	**68.4%**	68.5%	Overfitting incorrect, data skus incorrect so precision is overstated
4	Increase LSTM size	30	No	No	200	0	0.2	200	Yes	47.1%	47.1%	57.1%	**57.1%**	57.1%	Reduced accuracy
5	Decrease LSTM size	30	No	No	50	0	0.2	200	Yes	58.8%	47.6%	57.1%	**57.5%**	58.9%	Overfitting correct
6	Increase Embedding Size. Looking at range as proxy for good spread of correct/incorrect	50	No	No	100	0	0.2	200	Yes	52.9%	64.3%	69.0%	**68.4%**	68.5%	Improved convergence and fitting
7	Add flattening/shaping	50	No	Yes	100	0	0.2	200	Yes	64.7%	57.9%	66.7%	**66.9%**	67.5%	Reduced accuracy. Overfitting correct
8	Remove Flatening, Increase Dropout to reduce overfitting	50	No	No	100	0	0.3	200	Yes	52.9%	75.0%	73.8%	**72.7%**	74.0%	Improves accuracy back to resonable level
9	Increase Dropout more	50	No	No	100	0	0.4	200	Yes	52.9%	81.8%	76.2%	**74.9%**	77.2%	Improved precision and accuracy
10	Increased Interations	50	No	No	100	0	0.4	300	Yes	52.9%	90.0%	78.6%	**77.1%**	80.9%	Improved Precision and F1, F1Beta
11	Increased Iterations more	50	No	No	100	0	0.4	500	Yes	53%	90.0%	78.6%	**77.1%**	80.9%	No improvement
11	Increase Dropout more	50	No	No	100	0	0.5	200	Yes	52.9%	60.0%	66.7%	**66.3%**	66.2%	Dropout too high, results decline substantially
12	Try Flattening again with previous best	50	No	Yes	100	0	0.4	200	Yes	52.9%	42.9%	52.4%	**52.8%**	54.2%	Overfitting incorrect
13	Try Pretrained Embedding	50	Yes	No	100	0	0.4	200	No	88.2%	37.5%	35.7%	**21.3%**	15.3%	Non converging, overfitting correct severely
	Add a second layer to LSTM	50	Yes	No	100	20	0.4	200	No	5.9%	100.0%	61.9%	**49.6%**	71.4%	Non converging, overfitting incorrect severely
14	Add Flattening/shaping w/ pretrained embedding. Drop extra LSTM and dropout rate to help convergence.	50	Yes	Yes	100	0	0.2	200	Yes	35.3%	37.5%	50.0%	**49.7%**	49.6%	Model converges but results seem random with very low accuracy.
15	Increase epocs to see if we can imporove accuracy	50	Yes	Yes	100	0	0.2	500	Yes	29.4%	55.6%	61.9%	**58.7%**	60.3%	Improvement
16	Increase epocs to see if we can imporove accuracy	50	Yes	Yes	100	0	0.2	1000	Yes	29.4%	50.0%	59.5%	**56.8%**	57.4%	Starting to decline

Sagemaker Hypertuning

Starting with the local testing results, a Jupyter Notebook was used to run Hypertuning for the LSTM/Tensorflow model. A total of 18 Hyptertuning jobs were run. The final results of the best training job were about the same as from the local training best restults. At a minimum this validated the local modeling was repeatable.

The basic estimator was tested first based on successful local parameters and the results were consistent:

```
hyperparameters = {
 'epochs': 20,
 'embedding_size': 30,
 'flatten': 0,
 'lstm_dim_1': 100,
 'lstm_dim_2': 0,
 'dropout': 0.2
```

Hypertuning was tested using ranges around the successful local values:

```
max_jobs = 18, # The total number of models to train
max_parallel_jobs = 6, # The number of models to train in parallel
hyperparameter_ranges = {
     'dropout': ContinuousParameter(0.1, 0.4),
     'embedding_size': IntegerParameter(20, 100),
```

```
        'lstm_dim_1': IntegerParameter(50, 150)
    })
```

```
* dropout: 0.22199686351998094
* embedding_size: 54
* lstm_size 97
```

```
predictions  0.0  1.0
actuals
0.0           23    3
1.0            7   10

Recall:     0.588
Precision:  0.769
Accuracy:   0.767
```

2. SEB Data with XGBoost

Testing XGBoost both locally and on SageMaker was much easier and straight forward than LSTM. The results after hypertuning actually matched or maybe were slightly better than tests with LSTM locally. The downside is that you do not learn much from the resutls about the underlying data and how variations to the model might be done to ultimately improve the restuls given insights based on our learnings. XGBoost provides a good validation of the LSTM results but also highligted the big challenges that remain in predicting short answer data. There were issues for the larger SAG data using XGBoost on SageMaker which could not be explained. Early tests seemed consistent but could not be repeated and the final results were substantially worse than XGBoost done locally.

Note: Two of the best results had the same accuracy. I chose the result with the higher precisions becuase as stated earlier, it's preferable to mark a correct score incorrect and later adjust upward than the reverse.

Test #	Note	eta	max_depth	min_child_weight	gamma	subsample	estimators	objective	recall	precision	accuracy	F1 Weighted	Fbeta(0.1)	Outcome
1	First clean test	0.01	4	6	1	0.8	1000	reg:squarederror	29.4%	62.5%	64.3%	**60.6%**	63.6%	Seriously overfitting incorrect
2	Try chaning objective to binary:logistic	0.01	4	6	1	0.8	1000	binary:logistic	29.5%	45.5%	57.1%	**54.8%**	54.9%	More balanced but still overfitting incorrect
3	Increase epochs significantly	0.01	4	6	1	0.8	10000	binary:logistic	35.3%	50.0%	59.5%	**57.9%**	57.9%	Overfitting incorrect
4	Adjuist Gamma to reduce overfitting	0.01	4	6	6	0.8	10000	binary:logistic	11.8%	100.0%	64.3%	**54.4%**	75.0%	Overfitting incorrect
	Try objective to binary:hinge which predicts only 0 or 1	0.01	4	6	1	0.8	1000	binary:hinge	64.7%	68.8%	73.9%	**73.6%**	73.6%	Corrected for the incorrect overfitting. Presumbably handled the imbalance of more incorrect better.
5	Increase epochs significantly	0.01	4	6	6	0.8	10000	binary:hinge	64.7%	73.3%	76.2%	**75.9%**	76.0%	Improved accuracy
6	Reduce epocs to see if we are overtraining.	0.01	4	6	6	0.8	1000	binary:hinge	58.8%	71.5%	73.8%	**73.5%**	73.5%	Silghty reduced accuracy
7	Increase Max Depth	0.01	10	6	6	0.8	1000	binary:hinge	58.8%	71.4%	73.8%	**73.3%**	73.5%	Unchanged
8	Increase iterations to handle depth	0.01	10	6	6	0.8	10000	binary:hinge	64.7%	73.3%	76.2%	**75.9%**	76.0%	Improved precision and accuracy But same result as lower max depth
9	Adust min child weight for more aggressive model	0.01	4	2	6	0.8	10000	binary:hinge	64.7%	68.8%	73.8%	**73.6%**	73.6%	Better recal worse precision
10	Adust sub sample for more conservative model	0.01	10	2	6	0.8	10000	binary:hinge	70.6%	70.6%	76.2%	**76.2%**	76.2%	Improved F1 value with same accuracy as earlier model
11	Adust sub sample for more agressive model	0.01	10	2	6	0.5	10000	binary:hinge	58.8%	58.8%	66.7%	**66.7%**	66.7%	Worse with lowe subsample.
12	Raise Subsample	0.01	10	2	6	0.9	10000	binary:hinge	64.7%	68.8%	73.8%	**73.6%**	73.6%	Reduced F1

3. SAG Data with LSTM

Based on the SEB tests as a starting point, the initial SAG testing proved very confusing. With a similar model to the best performing for SEB, convergence was not acheivable. Monitoring the minimum and maximum prediction probabilities showed they were essentially the same for a variety of initial tests. The model was not converging on a solution. It was not until flattening was tryied that convergence suddenly became easily achieavable. The opposite was the case for the SEB data for which flattening did not lead to convergence. The SAG is a signficantly larger and more complex dataset and contains longer answers. It has 80 questions rather than 4. Flattening here may simplify the feature vector in an essential that helps convergence.

Two LSTM Layers were required to achieve the best results which also suggest the complexity of the dataset requires different tunning from the SEB data.

Pretrained Embedding was not helpful. As with the SEB data, pre-trained embedding was not helpful and did not provide a converging model. It should be noted that a decent percentage of the data could not be found in the Glove dictionary. This was due to misspellings, accidental concatinations and technical jargon. About 18% or 466 out of 2648 of the vocubulary was not found.

- Examples:
  - arrary (array)
  - fasterbecause (faster because)
  - nonconstant (jargon)
  - enqueue (jargon)

Each test is listed in the order they were performed with notes on the changes (also highlighted in the cell changed) and notes on the results. The best result is highlighted in green. The best restult is consistent with the best result for the SEB data.

Test #	Note	% of dataset	Embedding	Pretrained	Flattened	LSTM-1	LSTM-2	Dropout	Epochs	Converge	min-pred	max-pred	recall	precision	accuracy	Outcome
1	Initial Testing	100%	30	No	No	100	0	0.2	20	No	0.7408	0.7417	58.8%	100.0%	76.2%	Overfitting incorrect. No convergence at all. After 2nd epoch.
2	Test a balanced dataset with 50% correct/incorrect (data/sag2/balanced_answers.csv')	100%	30	No	No	100	0	0.2	10	No	0.2888	0.4995	100.0%	49.0%	49.1%	Overfitting incorrect
3	Drastically increase layer sizes	100%	400	No	No	400	0	0.2	10	No			100.0%	49.0%	49.1%	No change
4	Work with 10% of data set to speed up testing. Try changing objective from linear to sigmoid activation and binary:crossentropy .	10%	400	No	No	50	0	0.2	10	No			100.0%	58.5%	58.5%	No improvement
5	Switch back to linear/mean_squared_error. Tried adding a masking layer to mask 0's after the Embedding layer	10%	400	No	No	50	0	0.2	10	No			0.0%	0.0%	41.5%	Just flipped overfitting to incorrect.
6	Added flatening and shaping after embedding layer	10%	50	No	Yes	100	0	0.2	20	Yes	0.187	0.671	62.5%	75.0%	65.9%	Flattening made all the difference with this data set. Convergence in 20 epochs
7	Trying increasing Epochs	10%	50	No	Yes	100	0	0.2	50	Yes	0.02	0.708	79.2%	65.5%	63.4%	No improvement
8	Move to 20% of data	20%	50	No	Yes	100	0	0.2	50	Yes	0.0336	0.919	51.2%	58.3%	56.8%	Better distribution of probabilities. Worse accuracy
9	Increase embedding 100 and add a second LSTM Layer	20%	100	No	Yes	100	20	0.2	50	Yes	0.0002	0.983	53.7%	71.0%	65.4%	Improved accuracy
10	Increase imbedding to 200 and reduce epocs to 20, remove extra LSTM layer	20%	200	No	Yes	100	0	0.2	20	No	0.475	0.486	58.5%	68.6%	65.4%	Adding Embedding hurts convergence
11	Try increasing LSTM layer	20%	50	No	Yes	200	0	0.2	20	Yes	0.149	0.993	36.0%	62.5%	56.8%	Decreased accuracy
12	Swicth back to normal data (rather than balanced)	20%	50	No	Yes	200	0	0.2	20	Yes	-0.0734	1.24	74.0%	79.4%	68.0%	Improvement in probability distribution using non-balanced data.
13	Reduce LSTM back to normal	20%	50	No	Yes	100	0	0.2	20	Yes	-0.0122	1.21	74.0%	81.1%	69.4%	Slist improvement
14	Add second LSTM Layer, increase epocs	20%	50	No	Yes	100	20	0.2	50	Yes	-0.022	0.987	90.4%	75.8%	72.8%	Improved accuracy
15	Increase seond LSTM layer size	20%	50	No	Yes	100	50	0.2	50	Yes	-0.012	1.078	87.5%	75.8%	71.4%	Decreased accuracy
16	Use 100% of data	100%	50	No	Yes	100	20	0.2	30	Yes			86.6%	82.1%	76.7%	Increase accuracy with more data
17	Increase dropout to reduce overfitting , remove LSTM second layer	20%	50	No	Yes	100	0	0.4	50	Yes	0.0095	1.003	75.0%	76.0%	66.0%	Decreased accuracy
18	Moderate dropout add back 2nd LSTM layer	20%	50	No	Yes	100	20	0.3	50	Yes	0.0189	1.017	91.3%	75.4%	72.9%	Improved accuracy
19	Use 100% of data	100%	50	No	Yes	100	20	0.3	50	Yes	0.025	1.01	84.7%	82.7%	76.1%	Improved accuracy
20	Try Pretrained Embedding	100%	50	Yes	Yes	100	20	0.3	50	No	0.5775	0.678	99.8%	71.4%	72.3%	Non converging. All predicted correct.
21	Try Pretrained without Flattening, remove 2nd LSTM	50%	50	Yes	No	100	20	0.3	15	No	0.7089	0.7089	80.3%	72.8%	62.9%	Non converging. Results not meaningful
22	Return to flattening non pretrained and reduce epochs to the minimum converging.	100%	50	No	Yes	100	20	0.3	20	Yes	-0.062	0.998	86.4%	82.7%	77.1%	Improved accuracy with fewer epochs

Sagemaker Hypertuning

A total of 18 Hyptertuning jobs were run with the results of the best training job ending about the same as from the local training best restults. The SageMaker results are consistent with the local results but show no improvement after Hypertuning. The F1 result is 76.7% compared to the local 76.6% with a FBeta(0.1) of 76.6% compared to local 76.3%.

```
    hyperparameters = {
          'epochs': 20,
          'embedding_size': 50,
          'flatten': 1,
          'lstm_dim_1': 100,
          'lstm_dim_2': 20,
          'dropout': 0.3
    })
```

```
tf_hyperparameter_tuner = HyperparameterTuner(estimator = estimator, # The estimator object to use as the basis for
the training jobs.
       objective_metric_name = 'Validation_accuracy', # The metric used to compare trained models.
       objective_type = 'Maximize', # Whether we wish to minimize or maximize the metric.
       metric_definitions = [{'Name': 'Validation_loss',
                              'Regex': 'Validation_loss:(.*?);'},
                             {'Name': 'Validation_accuracy',
                              'Regex': 'Validation_accuracy:(.*?);'}
                            ],
       max_jobs = 18, # The total number of models to train
       max_parallel_jobs = 3, # The number of models to train in parallel
       hyperparameter_ranges = {
            'dropout': ContinuousParameter(0.2, 0.4),
            'embedding_size': IntegerParameter(50, 250),
            'lstm_dim_1': IntegerParameter(100, 250),
            'lstm_dim_2': IntegerParameter(10, 50)
       })


Best Job

dropout: 0.34991949487349483
embedding_size: 184
epochs: 20
flatten: 1
lstm_dim_1: 188
lstm_dim_2: 16

predictions  0.0  1.0
actuals
0.0           93  110
1.0           52  479

Recall:      0.902
Precision:   0.813
Accuracy:    0.779
F1 weighted: 0.767
FBeta(0.1):  0.766
```

4. SAG Data with XGBoost

Starting with the local testing results, a Jupyter Notebook was used to run Hypertuning for the XGBoost model. A total of 18 Hyptertuning jobs were run with the results of the best training job ending about the same as from the local training best restults.

The basic estimator was tested first based on successful local parameters and the results were consistent with SEB. After initial tests, the testing was moved quickly to SageMaker for Hypertuning.

A total of 18 Hyptertuning jobs were run with the results of the best training job ending about the same as from the local training best restults. The table below includes both local and the SageMaker results. The Sagemaker Hypertuning results were not as expected compare to the local results. An earlier run was consistent but I believe it was with a subset of the data. It's not clear at this point why the results didn't match local XGBoost results since the tests were run in an identical manor to the SEB data. This will require investigation in the future.

Test #	Note	eta	max_depth	min_child_weight	gamma	subsample	estimators	objective	recall	precision	accuracy	Outcome
1	Initial Test using SEB optimal params	0.01	4	6	1	0.8	1000	binary:logistic	98.1%	75.1%	75.0%	Overfitting correct
2	Increase epochs	0.01	4	6	1	0.8	2000	binary:logistic	97.0%	77.1%	76.9%	Overfitting correct but improving
3	incrose epochs	0.01	4	6	1	0.8	4000	binary:logistic	95.1%	79.0%	78.2%	Overfitting correct but improving
4	increase epochs significantly	0.01	4	6	1	0.8	10000	binary:logistic	93.6%	79.1%	77.5%	No improvement
5	After SageMaker Hyptertuning											
6	Sagemaker best job result	0.184	12	3	1.173	0.807	4000	binary:logistic	66.7%	72.7%	57.7%	Surprisingly poor results. Some tests early in the process resulted in high results as high as 81% but I was unable to repeat them suggesting some issue with this test.
7	Rerun local with best SageMaker values	0.184	12	3	1.173	0.807	4000	binary:logistic	94.0%	77.7%	76.1%	Better results locally with same parameters. Not clear why and future exploration is required. Results were matched with some SageMaker estimations.

Sagemaker Hypertuning

```
estimator.set_hyperparameters(max_depth=15,
        eta=0.01,
        gamma=1,
        min_child_weight=6,
        subsample=0.8,
        silence=1,
        objective='binary:logistic',
        early_stopping_rounds=50,
        num_round=10000)

xgb_hyperparameter_tuner = HyperparameterTuner(estimator = estimator, # The estimator object to use as the basis
for the training jobs.
   objective_metric_name = 'validation:rmse', # The metric used to compare trained models.
   objective_type = 'Minimize', # Whether we wish to minimize or maximize the metric.
   max_jobs = 18, # The total number of models to train
   max_parallel_jobs = 6, # The number of models to train in parallel
   hyperparameter_ranges = {
        'max_depth': IntegerParameter(12, 20),
        'eta'      : ContinuousParameter(0.005, 0.5),
        'min_child_weight': IntegerParameter(2, 8),
        'subsample': ContinuousParameter(0.5, 0.9),
        'gamma': ContinuousParameter(0, 10),
   })
```

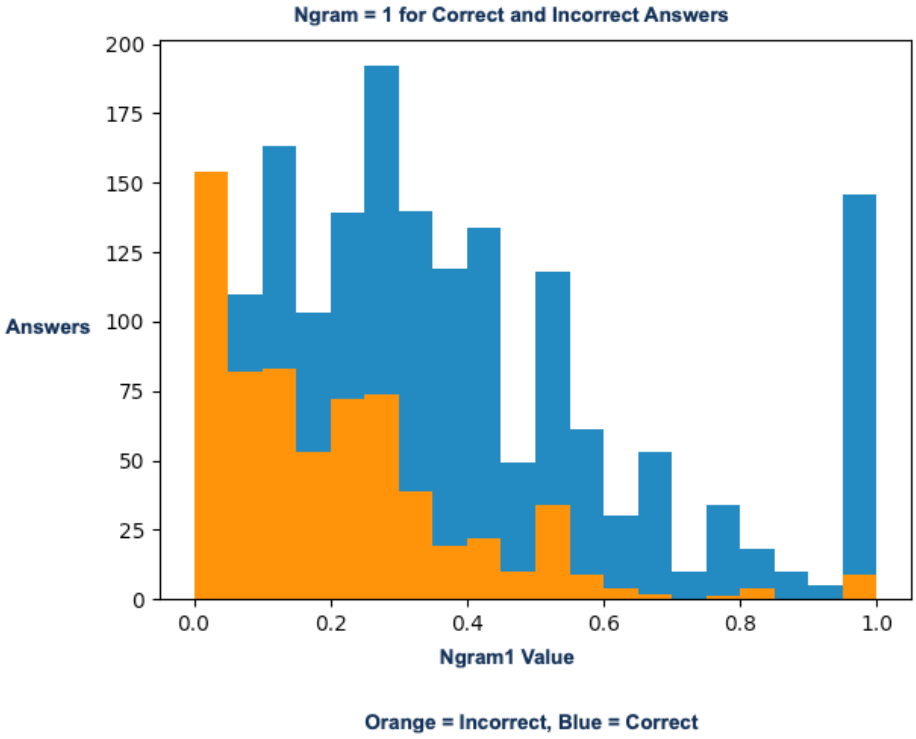
5. Ngrams: SAG Data with XGBoost and added Ngram features.

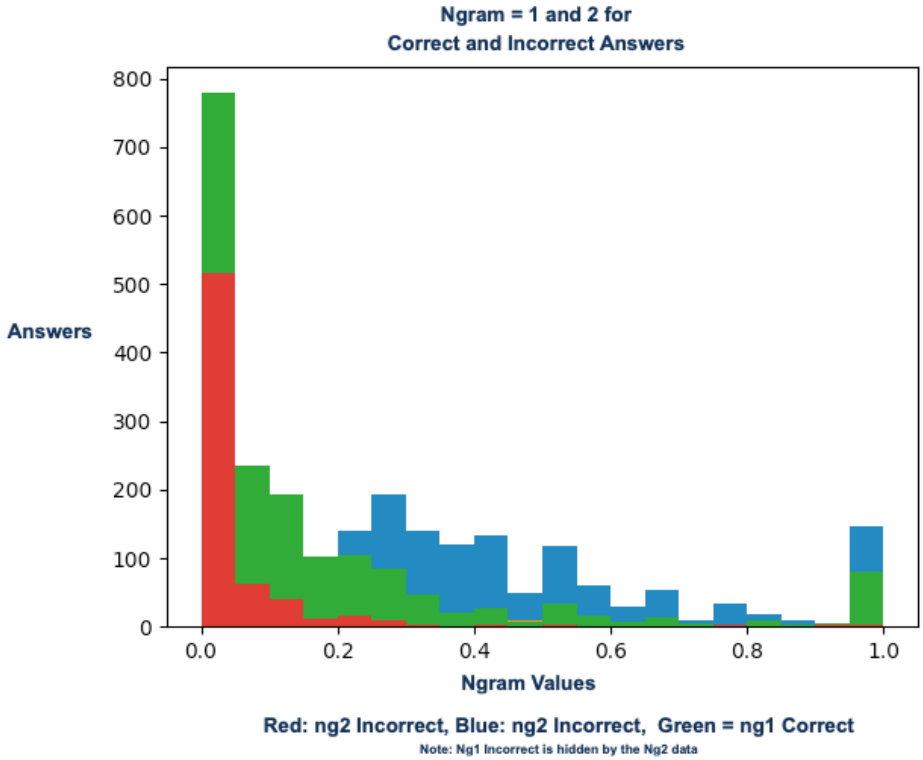
For this test ng1 and ng2 were determined to be the most useful. These two ngrams were found to correlation to correct vs. incorrect answers at about 20%. This suggests that for a given a student answer, about 30% of the correct/incorrect grade is predicted by the ng1 or ng2. The two ngrams have a 72% correlation to each other.

The results showed no gains over the encoded answers without ng1 and ng2. The result F1, **69.3%** , lower than XBGoost without ngrams so further testing was not waranted with this apporach at this time. More testing and that better factors in the reference answer compared to the student answer is needed. It is possible that a different modeling approach will be needed to reap the advantages of these similarity features combined with word vectors.

```
Correlation between Ngrams and Correct. (20 – 30%)
Only 72%  correlated together.
corr_matrix = ngrams.corr().round(2)
display(corr_matrix)
             ng1            2       correct
ng1         1.00           0.72      0.29
ng2         0.72           1.00      0.23
correct     0.29           0.23      1.00
```

The two histograms below show the values for ng1 and ng2 which range from 0 to 1 and how they compare for correct and incorrect answers. Ideally, the ngram values for correct answers would skew higher than the ngram values for the incorrect answers. This was not very evident and would show up in a histogram as two humps where ngram for incorrect would have low values and ngrams for correct answers would have higher values. What we see below is that except for very high values of ng1 and ng2, there is not a very strong corelation with correct answers. What we see is some small signs of skew as expected for the 30% correlation.





```
estimator.set_hyperparameters(max_depth=12,
    eta=0.01,
    gamma=6,
    min_child_weight=6,
    subsample=0.9,
    silence=1,
    objective='binary:logistic',
    early_stopping_rounds=5000,
    num_round=10000)
```

Before Hypertuning

```
Predictions  0.0  1.0
actuals
0.0           46  156
1.0           39  492

Recall:      0.927
Precision:   0.759
Accuracy:    0.734
F1 weighted: 0.693
FBeta(0.1):  0.698
```

Hypertuning

We see that Hypertuning improved on accuracy but since it's not optimizing on F1 the F1 result is slightly lower. The FBeta(0.1) is higher so in some sense they are not that different.

```
{'_tuning_objective_metric': 'validation:rmse',
'early_stopping_rounds': '5000',
'eta': '0.01',
'gamma': '1.2276608440869838',
'max_depth': '15',
'min_child_weight': '2',
'num_round': '10000',
'objective': 'binary:logistic',
'silence': '1',
'subsample': '0.8294398452847527'}

Predictions  0.0  1.0
actuals
0.0           38  164
1.0           26  505

Recall:      0.951
Precision:   0.755
Accuracy:    0.741
F1 weighted: 0.688
FBeta(0.1):  0.708
```

Justification

The resuls have been extensively covered and explained in previous sections. The LSTM model works as expected and an XGBoost model performs almost as well suggesting that the deep learning model still has a way to go to solve this type of problem. There are multiple areas for additional testing and study as a result of this analysis but none point to a fully automated solution. The next steps would be to find a solution that automates the grading of questions that are more predictable with additonal preprocessing to prepare and clean the data and predict which questions can be auto graded.

The final results suggest the following model best fits this problem and could be used with questions that are suitable for automated grading:

Convert the answers to word vectors of unique numbers based on a generated vocabulary.

LSTM Model:

1. Embedding layer with approximatly 50 dimensions per word.
2. Flatening and Shaping layers to simplify and compact the results from the Embedding layer.
3. Two LSTM layers of approximatly 100 (L1) and 20 nodes (L2).
4. Dropout Layer at 0.3 to reduce overfiting.
5. Activation Layer using Linear.

V. Conclusion

Free-Form Visualization

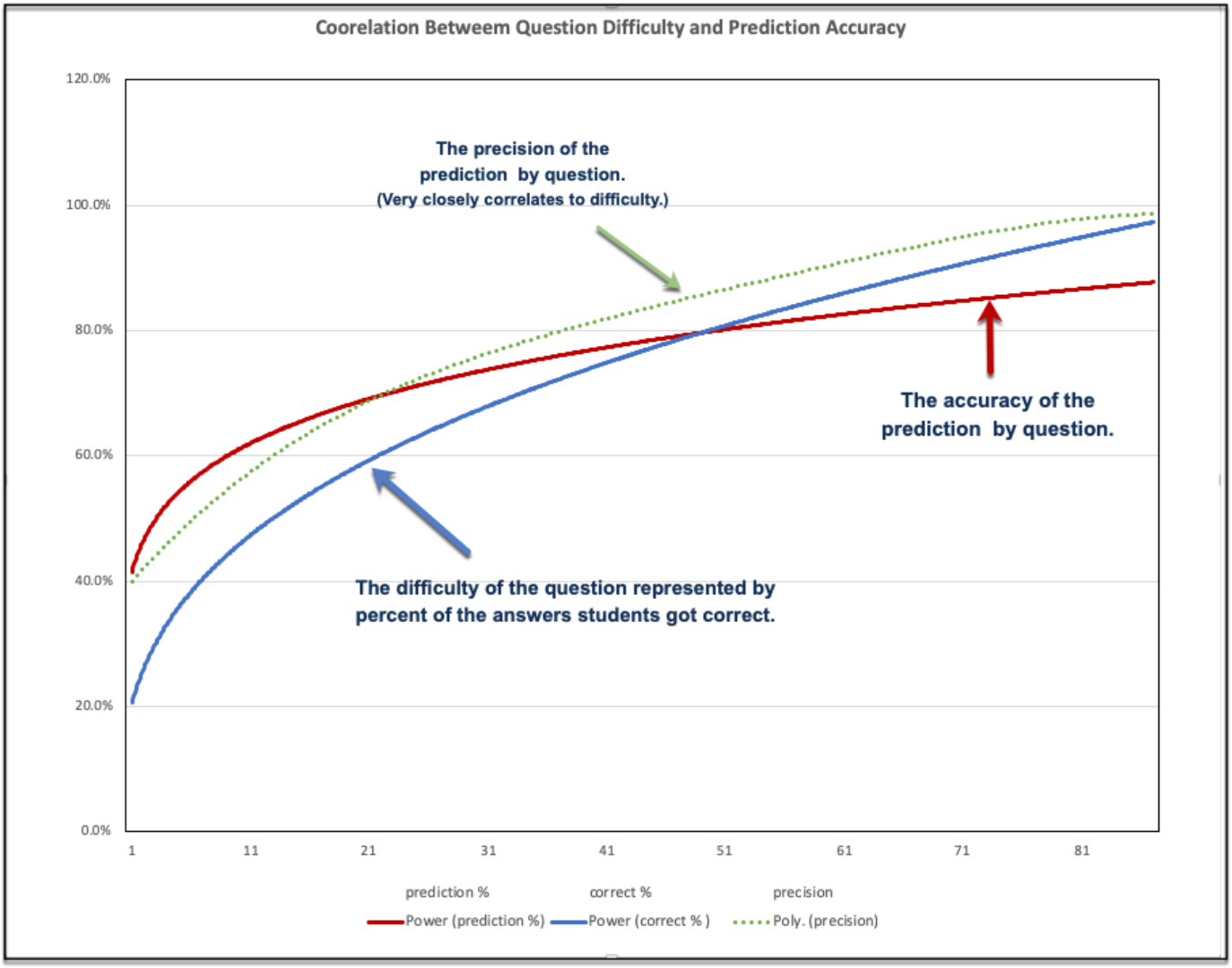
One of the concerns with modeling this data is the distirbution of correct vs. incorrect student responses. The correct vs. incorrect can also be though of as the relative difficlty of the questiosns and the tests in general for it's students. The higher the ratio of incorrect to correct, the more difficult the test is compared to a test with a higher correct percentage. It was observed that avererage test ratios of 2/3 correct to incorrect (or vice versa) performed better than a randomly balanced data set based on selecting equal numbers. This is likely becuase of the small dataset to start with. Every datapoint is critical. The SEB dataset had the reverse percentages to the SAG dataset but the results for accuracy were about the same.

Since the SAG dataset has a very large number of questions with a decent count of results for each question, some analysis at the question level is possible. When this was done it becomes evident that the difficulty of questions varied greatly. Looking at the quality of predictions both from an accuracy and precision point of view shows a strong correlation between the difficulty of the question and the accuracy of the prediction. The easier the question the better the prediction. This could be interpreted as harder questions results in more random or inchorent answers which are harder to predict. This also impacts the quantity of correct results. When there are very few examples of correct answers on difficult questions it is not surprising that predictability for these would suffer.

The conclusion is that tests with very imbalanced difficulty of questions are a likely be a problem for deep learning models. This should be further explored to see if there is a way to adjust for the question level imbalance but still model the entire test. Alternatively, we could use this information asa means of eliminating questions from autograding when there are insuficient correct answers.

Percentage of correct v.s. incorrect by Question

The chart orders the question from lest percentage correct to most. This builds on the chart from the Exploritory Visualization section by adding predictions.



Reflection

A solution to short answer granding using machine learning still remains to be discovered. This project replicated the results and the findings of the latest research and also identified paths (see Improvements below) to move towards a usable solution to augment human graders.

The steps taken in the project were:

1. Take the identified data sets and transform the answers into bag of words used to generate embedding vectors.
2. Build a custom LSTM model with additional layers based on the latest approaches from the literature.
3. Replicate the baseline results through adjustments to the model layers and hypertuning:
    - Embedding: size, calculated or pretrained. Pretrained was not helpful given the percent vocabulary misses in the Glove dataset.
    - Flattening/Shaping: include or exclude these layers. The results suggested that flattening is helpful for large datasets.
    - LSTM Layers: At least two layers are required for complex datasets.
    - The adam optimizer was used throughout these tests. Some others were tried but no obvious improvements were found.
    - Objective: Literature suggested that linear optimizate outperformed binary. This was proven to be the case for the LSTM type of model. For XGBoost binary:logistic was superior.
        - Linear apperas to help converge on an solution much better than softmax.
        - One possible explaination is that linear activation results in a wider range of probabilities providing more inforamation to the training.

4. Hypertune the models in SageMaker.
    - This validated the test done with local models using a completely different codebase and processing platform.
    - Hypertuning was not able to achieve better results. This is not surpising given the difficulty of this problem.
5. Compare the LSTM model to XGBoost.
    - Find out if deep learning models outperform a powerful machine learning algorithm.
    - XGBoost does a pretty good job nearly matching in most cases the deep learning model.
    - The deep learning model helps to expose and understand the nature and solution which adds value over more brute force machine learning approach.
    - The deep learning apporach leaves us with many avenues to better understand and improve on the soltuion in the future.

Finally, the solution to this problem remain elusive but it is a very interesting problem and is very similar in some way to understanding human speach. If an AI assistant could respond to any question with a useful answer, it would understand the complex and often sloppy way we express ideas. This is the same problem with short answer grading. Humans answer with all kinds of mistakes and variations that even other human grader are challenged to handle.

Improvement

Throughtout the testing the goal was to look for improvements on the baseline which itself is not an adequate solution to short answer grading using machine learning. Before knowing which direction to persue for future testing, it is necessary to fully understand the prior work and get a sense of how deep learning and machine learning solutions perform on short answer grading.

After testing a variety of approaches, several areas or possible paths should be explored.

1. Adust for the variying difficulty of questions. Balance the correct and incorrect ratios at the question level rather the dataset level where there is sufficient data. This suggests that some questions in a given dataset should be discarded and left for manual grading while those with sufficient data to balance difficulty could be modeled for machine learning.

2. Similarity: Leverage the knowlege of the reference answer. This may add value for some questions depending on how consistent the answer are. Where the answers can be expressed in may different ways, it might be necessary to provide multiple reference answers and pick the one whith the highest similarity scores compared to a given student answer.

3. Leverage Pretrained Embedding: In order for this to be effective, fixing typos and spelling errors and finding a way to add jargon specific to the test is required so words are not omitted from the embedding vectors. Correcting data quality will never be perfect but is a problem that could be solved with algorithms and another machine learning preprocessing model.