Voice to Text Documentation

By Reiss Pikett and Megan Stevens

Contents

Introduction

Calls that include private information in all industries must comply with security regulatory standards. The current principle way of ensuring calls are compliant is to manually listen to and check the audio call recordings. Our aim for this project was to develop a system to perform this check automatically, using voice to text transcription services and Machine learning algorithms in application with natural language processing. This essentially split the project into two main tasks: finding a voice to text engine that would work within the manual checking process and training a machine learning algorithm to read the transcription and determine a call's level of compliance.

Initial Set-Up

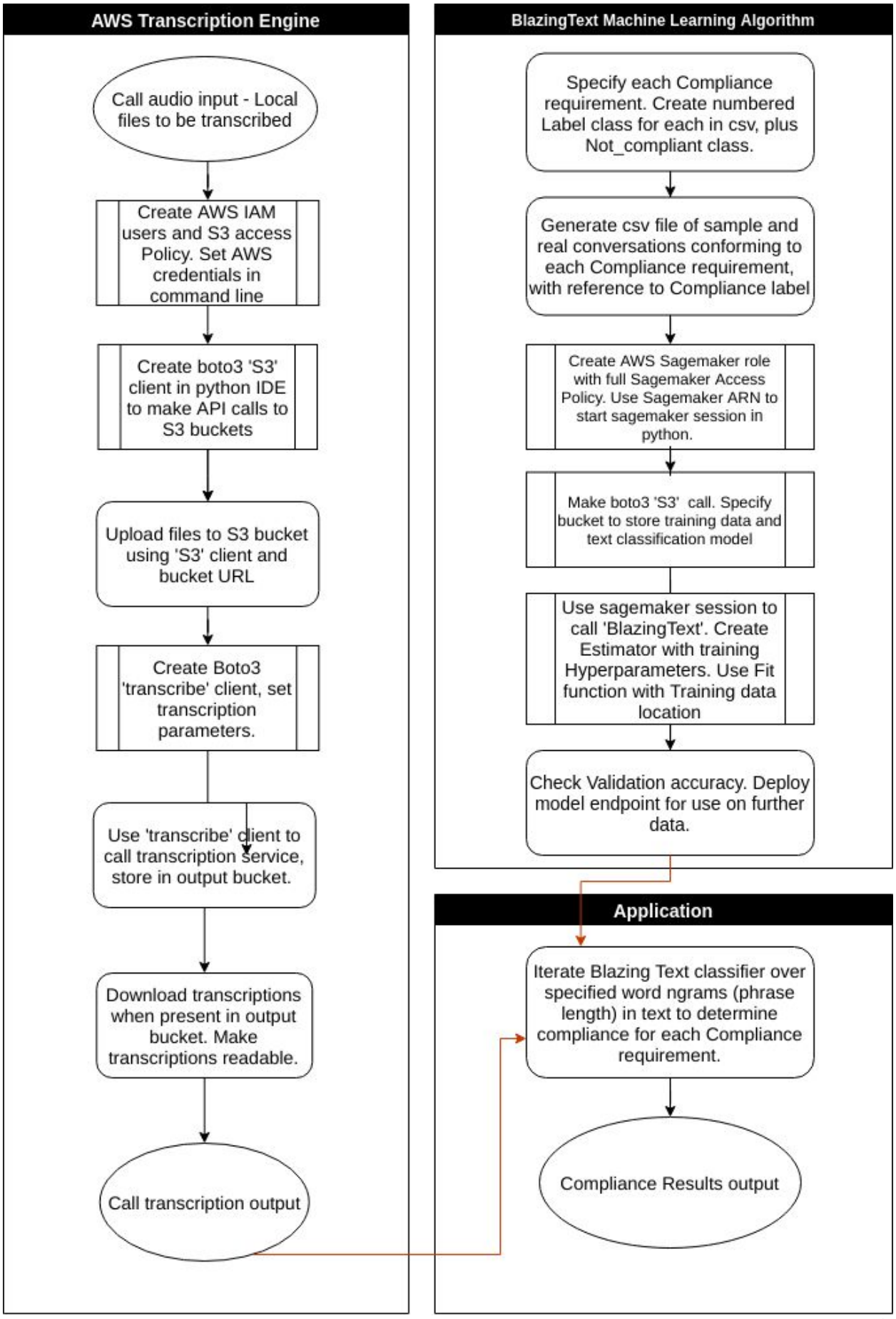
In order to invoke the necessary Amazon resources, certain users and access rights are required. Once these have been granted, Amazon Resource Names (ARN) are generated in order to make the Policy and Application Programming Interface (API) calls to Amazon Sagemaker and S3 bucket storage. The users, roles, and policies needed are as follows:

- IAM user
- Sagemaker Role
- Policy for password change access, attached to IAM user
- Policy for full S3 access, attached to IAM user and Sagemaker role. Download access keys and set AWS credentials in the command line.
- Policy for full Sagemaker access, attached to Sagemaker Role

To create a Sagemaker role, go to the AWS console IAM section. In roles, create a new role. Select the option called 'Sagemaker' and on the next page ensure you have the 'full access' policy. Leave any other fields blank or default. Once the role has been created add a new policy to it to give full S3 access. You will also need to ensure you have Amazon Transcribe Full Access.

Once access rights have been granted, install boto3 in the command line and in python IDE. Boto3 is the python Software Development Kit (SDK) for Amazon, and is used to make API and Policy calls to the necessary Amazon resources.

You will need to install punkt separately in python by typing into a blank python file: nltk.download('punkt'). You may also need to include lines of code that set up unique session environments in order to access the permissions.


AWS Transcription Engine
Call audio input - Local files to be transcribed
Create AWS IAM users and S3 access Policy. Set AWS credentials in command line
Create boto3 'S3' client in python IDE to make API calls to S3 buckets
Upload files to S3 bucket using 'S3' client and bucket URL
Create Boto3 'transcribe' client, set transcription parameters.
Use 'transcribe' client to call transcription service, store in output bucket.
Download transcriptions when present in output bucket. Make transcriptions readable.
Call transcription output
BlazingText Machine Learning Algorithm
Specify each Compliance requirement. Create numbered Label class for each in csv, plus Not_compliant class.
Generate csv file of sample and real conversations conforming to each Compliance requirement, with reference to Compliance label
Create AWS Sagemaker role with full Sagemaker Access Policy. Use Sagemaker ARN to start sagemaker session in python.
Make boto3 'S3' call. Specify bucket to store training data and text classification model
Use sagemaker session to call 'BlazingText'. Create Estimator with training Hyperparameters. Use Fit function with Training data location
Check Validation accuracy. Deploy model endpoint for use on further data.
Application
Iterate Blazing Text classifier over specified word ngrams (phrase length) in text to determine compliance for each Compliance requirement.
Compliance Results output

Amazon Speech To Text

After completing our benchmark for multiple speech to text engines we decided that Amazon Web Services (AWS) Speech to Text was the best for our needs. This was based on the general consistency and high accuracy of the service. Utilising Boto3 transcription client, the transcription process is fairly simple. All files stored in a local folder are uploaded to a specified S3 bucket, the transcription job is called for each file in the S3 bucket, and these transcriptions are then downloaded to a local folder to be made into a more readable form and analysed. The code that transforms the original AWS transcript into a more readable format does not require any manual input to run. It should be noted that no two transcription jobs may have the same name, though this should not be a problem in production assuming each audio file has a unique name.

Parameters

When making the transcription call certain parameters are specified to give the best transcription accuracy and to format the output in a required way. Most of these parameters are self-explanatory. The MediaFormat is set to mp3, the Language Code is set to 'en-GB', and Show Speaker Labels is set to True. The input and output locations for the audio and transcriptions are also set in the parameters but are defined elsewhere in the code. Parameters that have not been included thus far but may be useful include Media Sample Hertz and Job Execution Settings. Specifying the Hertz Rate of the transcribed media may improve the output however this parameters was manipulated for other engines we investigated, like Google, and did not seem to offer much of an improvement. Specifying the Job Execution Settings (which include the sub parameters Allow Deferred Execution and Data Access Role ARN) may be necessary if large batches are needed to be transcribed at once as there is a limit on the number of transcription jobs sagemaker can run at any one time and this parameter allows more transcription jobs than allowed to be called and will simply wait to do them.

Training the Amazon BlazingText Machine Learning Algorithm

Before we can analyse the transcriptions with a machine learning algorithm we must train a model. The AWS BlazingText text classification algorithm uses SageMaker, a fully managed Amazon service, to match arbitrary labels to phrases (instances) by forming connections between all phrases assigned the same label.

Training a particular model requires, at minimum, hundreds of sample phrases for each label, split into 80% training data and 20% validation data. For this project the training data included 5 labels: 'compliant - recorded', 'compliant - registration', 'compliant -

postcode', 'compliant - birthdate' 'not - compliant' to match the compliant questions needed to be included in calls. The sample phrases included both sentences we thought may be used and also real sentences from recorded calls. This data must be saved as csv files before running the training program.

The code to train a model requires the desired S3 bucket and the Sagemaker role ARN to be specified. It then creates a sagemaker session which analyses the labels and data given and deploys an endpoint of the trained model for later use. We have, up to this point, had a new endpoint created every time we retrained our model although it looks like it's possible to simply update the existing endpoint instead of creating a whole new one. This would be very useful as the name wouldn't need to be changed in the code when invoking the endpoint.

Application of the Trained Blazing Text Model

Once a model has been trained it can be easily invoked and applied to tokenized instances made from the transcriptions. The instances we used were made of overlapping short phrases that covered the entire text. The endpoint name will need to be specified and boto3 is used to access the model. Assuming an endpoint is updated when the model is retrained, rather than a new one being created, the endpoint only needs to be specified once. 'Predictions' are then made of which label matches which instance.

The k configuration value can be varied between 1 and the number of labels to show either just the most confident label or the other labels in order of confidence also. It should be noted that the code for method 1 may need to be edited if k is higher than 1 due to the way information is accessed and that method 2 requires k to be set to max in order to produce graphical representation.

In method 1 compliance is awarded for the individual labels based on a percentage confidence threshold determined by looking at the results. These thresholds will need to be modified as the algorithms get trained.

Parameters and Hyperparameters

The parameters and hyperparameters set when training the model require further understanding of Natural Language Processing (NLP) principles in order to fully understand how to optimally set them. Blazing text runs through the data multiple times, updating the hyperparameters by minimising the error in the validation data prediction accuracy. These parameters have been mostly unchanged from the default recommended by Amazon although few were adjusted slightly to hopefully give more accurate results. These included epochs, min_count and vector_dim:

- Epochs sets the number of complete passes through the training data in order to update the remaining parameters, and is defaulted to 5, though we found 10 worked better.

- Min_count sets the number of times that words must appear in the entire training set in order to be included in the model, in order to reduce training time and pattern fitting by discluding words that are not useful. We adjusted this from 5 to 6 though a higher number determined as a fraction of the total number of sample conversations may prove to be better.
- Vector-dim sets the dimension of the word vectors that the algorithm learns, and is analogous to the Principal Components, or most useful patterns, that the algorithm can determine in the dataset. For example, for the words 'dog', 'cat' and 'cheetah', we may have two vector dimensions to quantify them, "fluffy" and "domestic", with the three words scoring middle to high for 'fluffy', yet 'dog' and 'cat' scoring highly for 'domestic' and cheetah scoring low, an additional vector of "animal" would prove less useful as there would be little or no difference in the score between the words. Vector-dim is defaulted to 100 though we reduced this to 10 in the belief that less complicated patterns between the phrases and the labels would be required.

Parameters that have been left at their default values that we believe may be worth adjusting are word_ngrams and learning_rate. Word_ngrams specify the number of continuous words data is separated into. e.g. ngrams=2 All_that_is_gold -> (All_that),(that_is),(is_gold). The default Word_ngrams is set to 2, though we believe setting it at 4 or 5 (the max) may prove more useful. Learning_rate is the step size used for updating parameters each epoch and requires further investigation as to how it affects the results.

The parameters in the sagemaker estimator are to do with storage and speed. We had no need to change them while testing but they may need to be adjusted once the program is used in production.

Next Steps

Our current methodology has taken into consideration the current limitations of transcriptions services and machine learning algorithms, but there is the possibility to improve both by taking the next steps:

- The speech to text algorithm may be improved using the custom dictionary. This is a function that Amazon, and many other speech to text engines, offers which allows the transcription engine to favour key words over other similar sounding words. This could allow for compliance words to be transcribed more accurately, although there is the uncertainty that this could cause compliance words to be transcribed when they have not been said.
- Our machine learning algorithm should only get more accurate as time goes on with constant training. It would be great if there was a way to automatically update training data using erroneous labeling.

- Consider Microsoft Azure's speaker identification enrollment for potentially improved accuracy.

Other Applications

This project was started in the hopes of being able to automate the process of checking conversations for compliance, but it has many other promising possibilities.

- Conversation type - It would be very easy to train another BlazingText model with sentences that allow the conversations to be categorised in different ways. For example, deciding whether or not a phone call was made to book an appointment or was just to enquire into the customer's general happiness with the product.
- More complex patterns with two speakers - It may also be possible with further tuning to train an algorithm to not only find compliance questions but also take note of the relevant answers.
- E-mails - The text classification methodology could also be applied to verifying the compliance of e-mails.